SESSION

ERSA KEYNOTE

Chair(s)

Dr. Toomas P. Plaks

OpenCL for FPGAs: Prototyping a Compiler

Tomasz S. Czajkowski, David Neto, Michael Kinsner, Utku Aydonat, Jason Wong,
Dmitry Denisenko, Peter Yiannacouras, John Freeman,
Deshanand P. Singh and Stephen D. Brown

ABSTRACT

Hardware acceleration using FPGAs has shown orders of magnitude reduction in runtime of computationally-intensive applications in comparison to traditional stand-alone computers [1]. This is possible because on an FPGA many computations can be performed at the same time in a truly-parallel fashion. However, parallel computation at a hardware level requires a great deal of expertise, which limits the adoption of FPGA-based acceleration platforms.

A recent interest to enable software programmers to use GPUs for general-purpose computing has spawned an interest in developing languages for this purpose. OpenCL is one such language that enables a programmer to specify parallelism at a high level and put together an application that can take advantage of low-level hardware acceleration.

In this paper, we present a framework to support OpenCL compilation to FPGAs. We begin with two case studies that show how an OpenCL compilation could be done by hand to motivate our work. We discuss how these case studies influenced the inception of an OpenCL compiler for FPGAs. We then present the compilation flow and the results on a set of benchmarks that show the effectiveness of our automated compiler. We compare our work to prior art and show that using OpenCL as a system design language enables large scale design of high-performance computing applications.

1. INTRODUCTION

Modern FPGAs are some of the largest and most complex integrated circuits, and have become a defacto solution for many high-performance applications, such a network packet processing. They have also been successfully used to accelerate computation (medical imaging [1, 2], molecular dynamics [3]) in comparison to workstation computers.

Starting in the early 1990s through the present day, there has been an increasing interest in high-level synthesis (HLS), because describing a circuit at a high level can take a 10^{th} of the lines of code as compared to an equivalent Verilog or VHDL description [4]. If HLS tools could produce a good circuit, then they would significantly improve productivity, and allow designers to test their circuits at a higher level of abstraction, making the process much faster.

Traditional HLS tools implement a circuit from a software-like language, such as C. For a program, they implement a circuit in a single-instruction single-data (SISD) fashion, comprising a datapath that performs computation and a control circuit that schedules the flow of computation. Some parallelism is achieved through scheduling independent instructions at the same clock cycle. This approach, however, does not provide the best possible use of an FPGA. First, an FPGA can be a very large device, and it is possible many such circuits can simultaneously process data, increasing the overall throughput of a design. Second, pipelining is not explicit in programming languages such as C, and thus it is not always possible to generate a circuit that has a comparable performance to a hand-coded Verilog/VHDL design.

Recently, a new computing paradigm called OpenCL (Open Computing Language) [5] has emerged that is suitable for adoption as an FPGA design entry method and addresses the two aforementioned problems associated with HLS. In OpenCL computation is performed using a combination of a host and kernels, where the host is responsible for I/O and setup tasks, and kernels perform computation on independent inputs. Because of the explicit declaration of the kernel, and the fact that each set of elements processed are known to be independent, each kernel can be implemented as a high-performance hardware circuit. Based on the amount of space available on an FPGA, the kernel may be replicated to improve performance of the application.

In this paper, we first prototype two applications to determine how to build an automated compiler to efficiently synthesize applications from an OpenCL description into a complete system on an FPGA. After analyzing the results from case studies, we proceeded with the description of the architecture of the compiler, highlight some of its key features and present the results we obtained by using the compiler on a few key benchmark applications that we implemented on an Altera Stratix-IV based board (DE4).

2. BACKGROUND

In this section, we provide some basic background about OpenCL and our target platform, the DE4 board.

2.1 OpenCL Primer

OpenCL is a language, and a computing paradigm, to enable acceleration of parallel computing, targeting a wide variety of platforms [5]. OpenCL was developed to standardize the

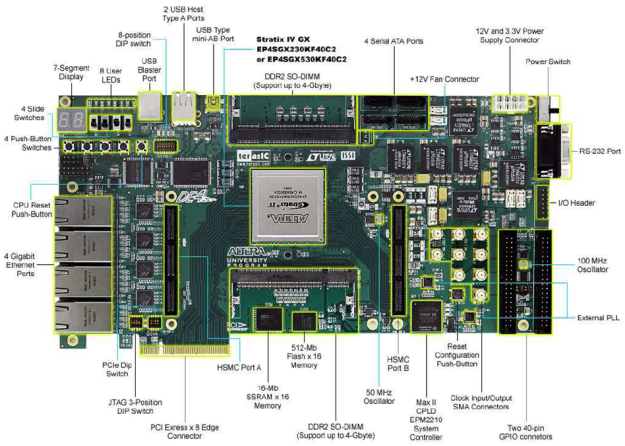


Figure 1: DE4 Research Board

method of parallel computation using GPGPUs, for which proprietary languages such as CUDA already existed.

An OpenCL application comprises a host and kernels. The *host*, is responsible for processing I/O requests and setting up data for parallel processing. When the host is ready to process data, it can launch a *kernel*, which represents a unit of computation to be performed. A kernel executes computation by loading data from global memory as specified by the host, processing it, and then storing the results back into global memory so that they can be read by the host. In OpenCL terminology a kernel and the data it is executing on comprise a *thread*. Results are computed for a group of threads at a time. Threads may be grouped into *work groups*, which allow data to be shared between the threads in a work group; however, no constraints are placed on the order of execution of threads in a work group and they are thus required to be independent.

2.2 DE4

The DE4 FPGA board, manufactured by Terasic Technologies, features an Altera Stratix IV 230/530 GX device. It provides two DDR2 memory SO-DIMM slots, four GigE ethernet ports, four SATA connections, a PCIe header, three USB ports, SD Card slot, and other peripherals. Figure 1 shows a photograph of the board.

3. INITAL CASE STUDIES

The first step towards an OpenCL-to-FPGA compiler was to study two benchmarks that could be implemented in OpenCL. These benchmarks are Black-Scholes option pricing and a Bloom filter. In both cases, we implemented each application using a host/kernel model, to generate a circuit in the OpenCL style. We found that in each case we were able to steer the final implementation towards a pipeline-oriented design, replicating the kernel hardware several times to achieve high performance.

We implemented each design to have a Nios II processor [6] to run the host program and interact with kernels, two DDR2-800 memory controllers to store data, and a set of hardware accelerators that implement the kernels in OpenCL. The host program communicates with the kernels using memory-mapped I/O to set up each kernel, launch it, and monitor its progress. The kernels run independently of the host, loading and storing data in memory using dedicated memory-mapped master interfaces and local memory buffers for temporary data storage.

3.1 Black-Scholes Option Pricing

Black-Scholes equations are used for modeling pricing of European-style options. The equations computed in this model are as follows:

$$\begin{aligned} d_1 &= \frac{\log \frac{S}{X} + T\left(R + 0.5V^2\right)}{VT^{0.5}} \\ d_2 &= d_1 - VT^{0.5} \\ \text{call} &= SN\left(d_1\right) - N\left(d_2\right) Xe^{-RT} \\ \text{put} &= Xe^{-RT}\left(1 - N\left(d_2\right)\right) - S\left(1 - N\left(d_1\right)\right) \end{aligned} \tag{1}$$

where R is the riskless rate of return, V is stock volatility, T represents option years, X represents a strike price and S the current price. The N function is an approximation of a cumulative normal distribution. The outputs *call* and *put*, represent the *call* and *put* prices. Usually, the above computation is performed for a set of options and a fixed volatility and rate of return, which requires an input of three floating-point numbers per computation.

In the OpenCL paradigm, a thread is responsible for producing one or more results. Each thread loads inputs X, S and T from memory and computes the *call* and *put* values that are then stored in the corresponding location in memory.

On a GPU, the threads are executed in a SIMD fashion on a multiprocessor. A similar level of parallelism can be achieved on an FPGA by using pipelining. A pipelined design allows a long and intensive computation to be performed over a series of clock cycles. During each clock cycle, a new set of data can be accepted for computation. Thus, if a computation pipeline has a depth of 200 cycles, results will be produced at every clock cycle after some latency.

3.1.1 An Efficient Pipelined Computation Engine

Building a hardware implementation of this kernel begins with a data-flow graph (DFG). The DFG consists of three parts. The first part performs computation of inputs to a cumulative normal distribution (CND) function approximation, the second part is the CND function itself, and the final part computes the *call* and *put* values. Each part of the computation produces data at each clock cycle, allowing the hardware to produce a new result at every clock cycle. To demonstrate how this is accomplished, we show the implementation of the first part in Figures 2 and 3.

Figure 2 shows the DFG, where circles represent single-precision floating point operations, implemented using hardware cores. The latency of each block varies, as it takes a different number of clock cycles to multiply, invert, or take a square root of a number. To balance the latencies on each path, we insert shift registers as shown in Figure 3. The resulting latency of this computational block is 53 clock cycles. When each part of the computation is similarly implemented, we put them together to create an engine that given a set of inputs produces a result 161 clock cycles later.

3.1.2 Memory Access

To attain a high performance from the above kernel implementation we need to sustain data throughput to and from

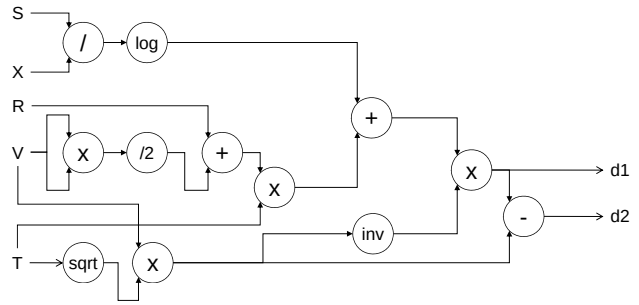


Figure 2: DFG for the first part of the computation

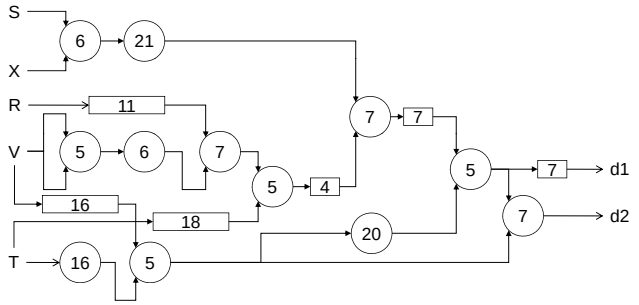


Figure 3: DFG with latency-balancing shift registers

memory. For this kernel, the data consists of three floating-point numbers as input, and two floating-point numbers as output. To facilitate the required memory bandwidth we use two separate high-speed memories (DDR2-800).

A DDR2-800 is a high-speed memory device that can be accessed via a DDR2 memory controller. The controller is capable of high-speed transfers of data in a burst fashion. It exchanges data with the external memory at a clock rate of 400 MHz, 64-bits at each edge of the clock. On the side of the FPGA, the controller facilitates a synchronous interface clocked at a rate of 200 MHz, providing 256-bits of data at the positive edge of the clock. To read and write data through this interface, a kernel needs to access data in 256-bit blocks at each edge of the clock. This is done by combining inputs and outputs into groups, such that each 256-bit read comprises several sets of inputs X, S, and T. Similarly, outputs are grouped together such that a single 256-bit write stores several *call* and *put* results.

To facilitate fast access to memory, we created bit-width adapting buffers for the kernel. The input buffer takes in 256-bit wide data and stores it in an array of eight 96-bit by n buffers, as shown in Figure 4.

3.2 Bloom Filter

In networking applications, Bloom filters [7] are used to scan the payload of a packet to help determine if a packet should be dropped or kept. For example, a packet router may wish to determine if a message it is about to forward contains unwanted data. If so, a packet can be dropped, protecting the destination computer from a potential hazard.

Bloom filters scan the payload of a packet by looking for signatures of interest. While a usual search could be slow, Bloom filters employ hash tables, where each signature is hashed using several hash functions. This allows the filter to compute a hash value for the data as it arrives, and perform a quick hash lookup to determine if a signature is of interest.

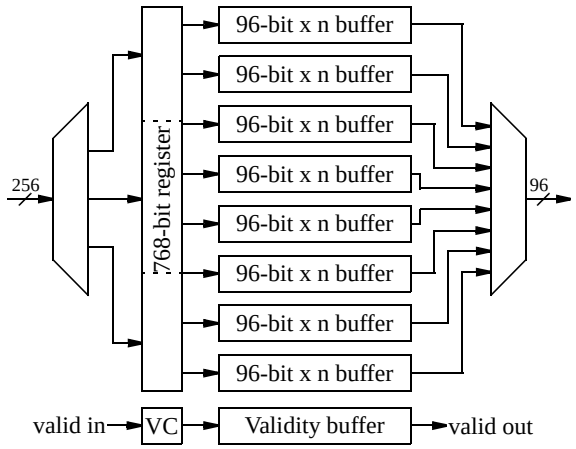


Figure 4: Bit-width adapting input Buffer

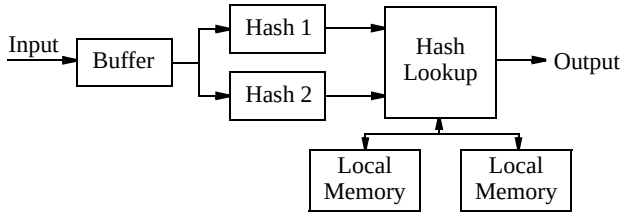


Figure 5: High-Level Diagram of a Bloom Filter

A high-level diagram of a Bloom filter is shown in Figure 5. In this design, a packet is first loaded into a buffer. The packet data is then sent through two parallel hash computation functions. Once each hash value is computed, a hash lookup is performed. If both values at addresses specified by the hash values are 1, then a Bloom filter reports it has found a signature of interest. To achieve a high throughput for this design, it may be necessary to replicate local memory that stores the hash values, as indicated in the figure.

This type of a search can be efficiently implemented on an FPGA. A recent paper [8] showed an implementation of such a filter that can sustain a throughput of approximately 3.2 Gbps. Implementation by Suresh et al. [9] provided a throughput of 18.6 Gbps on an FPGA.

Our filter is designed to scan the payload of a packet, expected to contain simple text. The text is parsed and words are identified and compared against a dictionary of approximately 150000 words. The result of the Bloom filter, is a pair of values that indicate how many words in every packet are spelled correctly and how many are not.

3.2.1 Bloom Filter Core

Pseudo-code for the Bloom filter is given in Figure 6. In this kernel, we process each packet one character at a time. For every character in a packet, we detect if it is a separator. If it is not, then we update the *hash_1* and *hash_2* values as per the else clause, updating the length of the word currently being processed. Once a separator is detected, we perform a lookup into a hash table, updating the *hit* and *miss* registers. This approach matches the high-level design presented in Figure 5.

In this kernel, the hash lookup is performed at the end of every word and hence can be executed in parallel with the rest of the kernel. However, it has to be able to stall the pipeline when the *perform_hash_lookup* function is executing

```
__kernel void bloom_filter(read_only int *d_lookup,
        unsigned int *d_result, unsigned char *d_packet_mem,
        int *d_packet_size, unsigned int packet_count) {
    for(packet_id = 0; packet_id < packet_count; packet_id++) {
        int packet_length = d_packet_size[packet_id];
        unsigned int offset = 1024*packet_id;
        unsigned int hash_1 = 0, hash_2 = 0, miss = 0, hit = 0;
        short int length = 0;
        for (index = 0; index < packet_length; index++) {
            unsigned char packet_ch = d_packet_mem[offset+index];
            char is_end_word= is_separator(packet_ch);
            int temp_hash_1 = update_hash_1(hash_1, packet_ch);
            int temp_hash_2 = update_hash_2(hash_2, packet_ch);
            if ((is_end_word) && (this_word_length > 0))
                perform_hash_lookup(d_lookup,
                        hash_1 % 2500000, hash_2 % 2500000,
                        &hit, &miss);
            hash_1 = (is_end_word) ? 0 : temp_hash_1;
            hash_2 = (is_end_word) ? 0 : temp_hash_2;
            length = (is_end_word) ? 0 : length+1;
        }
        d_result[packet_id] = (miss << 16) | hit;
    }
}
```

Figure 6: Bloom Filter kernel

and another lookup is required. Such an implementation allows a lookup to be concurrent with the processing of the next word, increasing the throughput of the circuit.

3.2.2 Shared Hash Table(s)

A key aspect of performance in this design is the hash table lookup. The lookup takes several cycles, which makes it possible for another word to be processed before the lookup is completed. In such a case, it is important to design the lookup circuitry with minimal latency. Also, to enable higher throughput, we need to be able to share the hash table between a set of kernels.

We implemented a 2.5Mbit multi-ported shared hash table using dual-ported on-chip memory. To create more than two ports for the hash table, we divided it into five segments, each implemented as a dual-ported memory with 16384 32-bit words. Each segment can simultaneously read data from two distinct memory locations, making a total of 10 ports on the hash table available for simultaneous access.

4. STUDY ANALYSIS

In this section, we discuss the performance results we obtained in our case study of two example benchmarks, in an effort to determine the key aspects that we will need to address when creating an OpenCL-to-FPGA compiler.

4.1 Area and Performance

We implemented each of the case studies on an Altera DE4 board, comprising an Altera Stratix IV 230GX device and two DDR2 memory interfaces. In each case, we began with an OpenCL description of an application, manually created a DFG for each kernel, and coded each kernel hardware module in Verilog HDL. We created a system comprising many instances of hardware kernels on which threads could be executed, as well as a Nios II processor to run the host program. Table 1 presents the clock frequency, logic utilization and throughput results for each circuit in our case study.

The Black-Scholes option pricing circuit comprised four accelerators, where floating-point cores were set for minimum latency. To obtain the throughput results, we ran the system at a clock rate of 150 MHz, allowing each instance of the accelerator to produce two results (*call* and *put*) per cycle. To exchange data with the external DDR2 memory, we connected each pair of accelerators to a single DDR2 memory controller. Because of the limited memory bandwidth in comparison to the required number of load and store operation, four accelerators fully utilized the memory bandwidth when producing 885.6 million results per second.

Table 1: Performance and Area Results

Name	Freq. (MHz)	Utilization (%)	Throughput
Black-Scholes	150	76	885.6 M results/s
Bloom Filter	150	46	25.559 Gbps

To implement the bloom filter, we instantiated 48 kernels to process packets in parallel. Each kernel was connected to DDR2-800 memory to load data packets, and used local on-chip memory for intermediate data storage. The system was clocked at a rate of 150MHz and occupied 46% of the available device resources. The resulting throughput of the circuit was 25.6 Gbps.

4.2 Observations from Case Studies

The initial case studies yielded both compelling results as well as a number of interesting challenges that need to be overcome when developing an OpenCL-to-FPGA compiler. We summarize the key results in three categories: memory access, pipelining, and hardware replication.

4.2.1 Memory Access

The first challenge we encountered was one of effectively utilizing the available memory bandwidth. This was particularly an issue with the Black-Scholes case study, as the kernel itself was able to process a lot of data very quickly. To address this challenge we introduced buffers to store input and output data. While these buffers were not explicitly defined in OpenCL, they could be inferred. With the help of input and output buffers we were able to group memory transfers into sizable batches, such that the kernel could be kept occupied 100% of the time.

One of the less obvious tasks from the point of view of the programmer was to ensure that the memory interface, in our case DDR2-800 memory, and the kernel hardware needed to have matching bus widths. This is something a usual high-level language programmer does not need to think about, as the underlying architecture does not change. In our case, the process of synthesizing a circuit from scratch requires such details to be handled. To overcome such obstacles, our compiler will have to take care of low-level implementation details, while being mindful of the external memory it is interfacing with.

The key lesson we learnt here was that our compiler will have to be able to generate efficient memory interface architecture to external RAM on a per application basis. This will be a key consideration for many applications.

4.2.2 Pipelining

The second challenge dealt with using pipelining to implement high-performance hardware for kernels. While in the case of the Black-Scholes, this can be done easily (by hand or in an automated tool), implementing a Bloom filter as a pipelined circuits was more difficult.

To implement a Bloom filter as a pipelined circuit, we had to break the kernel up into two parts: the hash computation, and the hash lookup, each of which could run independently. The hash computation was responsible for processing packet data, one character at a time, and computing a hash value for every word in the packet. When a word was found, the hash lookup was executed. At the same time another set of characters were being processed and a hash value was computed for them. If the computation of a hash value for the next word completed before the hash lookup, it was necessary for the hash lookup to stall hash computation, until the lookup was completed.

The key lesson we learnt was that a pipelined architecture for many kernels is possible, especially in an OpenCL context where explicit parallelism is defined. The pipeline would have to allow sections of the circuit to stall when a long memory access is required and thus a flexible design will be required. This is particularly important when dealing with loop constructs where it is possible for the loop to be only able to handle a limited number of threads simulteneously.

4.2.3 Hardware Replication

The third challenge in the implementation of both designs was the level of hardware replication needed to achieve high throughput. While in the case of Black-Scholes an entire kernel was replicated, in the case of the Bloom filter it turned out that a major bottleneck was the hash table.

We noted earlier that the hash table had to be replicated to achieve high throughput, allowing only half of the kernels (24) to use each hash table. One way to define such replication in OpenCL is to provide a kernel with several hash tables as inputs. A better approach would be to be able to model a kernel to be able to detect such problems, and replicate resources shared by kernel instances.

The lesson we learnt here was that some level of control over the architecture of the system needs to be provided, to allow the designer to analyze the performance results and then affect the architecture of the kernel hardware to improve throughput. While the ultimate goal is to be able to automatically determine the correct level of replication required for an application, a reasonable first approach is to allow user some control over the compiler output.

5. OPENCL-TO-FPGA COMPILER

In this section we present the compiler we devised based on the lessons we learnt from initial case studies.

Figure 7 presents the flow of our compilation framework, based on an LLVM compiler infrastructure [10]. The input is an OpenCL application comprising a set of kernels (.cl files) and a host program (.c file). The kernels are compiled into a hardware circuit, starting with a C-language parser thath produces an intermediate representation for each kernel. The intermediate representation (LLVM IR) is in the form of instructions and dependencies between them. This representation is then optimized to target an FPGA platform. An optimized LLVM IR is then converted into a Control-Data Flow Graph (CDFG), which can be optimized to improve area and performance of the system, prior to RTL generation that produces Verilog HDL for a kernel.

The compiled kernels are instantiated in a system with interfaces to the host and off-chip memory. The host interface allows the host program to access each kernel to specify workspace parameters and kernel arguments. The off-chip memory serves as global memory for an OpenCL kernel. This memory can also be accessed via the host interface, allowing the host program to set data for kernels to process and retrieve computation results. The complete system can then be synthesized, placed and routed on an FPGA using Altera Complete Design Suite (ACDS) [11].

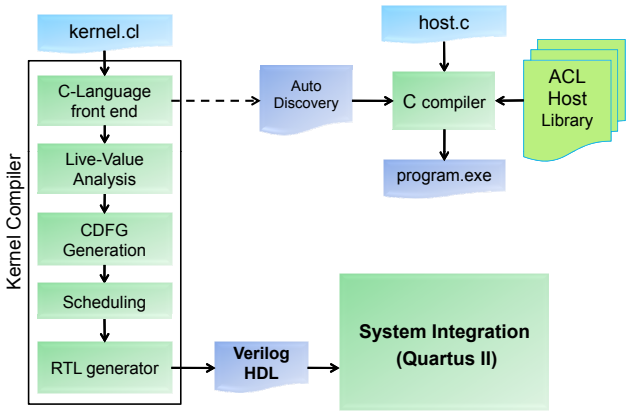


Figure 7: OpenCL-to-FPGA framework.

Finally, we compile the host program using a C/C++ compiler. There are two elements in the compilation of the host program. One is the Altera OpenCL (ACL) Host Library, which implements OpenCL function calls that allow the host program to exchange information with kernels on an FPGA. The second is the Auto-Discovery module which allows a host program to detect the types of kernels on an FPGA. The Auto-Discovery module is embedded in the system by the kernel compiler, and stores the information pertaining to the kernels in a given design.

6. IMPLEMENTATION DETAILS

Our framework comprises a Kernel Compiler, the ACL Host Library, and System integration. The Kernel Compiler implements OpenCL kernel functionality as a circuit described in Verilog HDL and produces a description of the generated circuit for the host program in a form of an Auto-Discovery module. The C compiler then takes that description as well as the ACL Host Library and compiles the host program. The host executable as well as the Verilog HDL are then put together using a System Integration tool (Qsys) and compiled using ACDS [11].

6.1 ACL Host Library

The Altera OpenCL (ACL) host library implements most of the Platform and Runtime APIs of the OpenCL 1.0 specification [5]. The Platform APIs allow the host program to discover the FPGA accelerator device and manage execution contexts. The Runtime APIs are used to manage command queues, memory and program objects, and to discover precompiled kernels and invoke them.

The ACL Library comprises two layers: a platform independent layer that performs device independent processing, and a hardware abstraction layer that adapts to platform specifics. The platform independent layer provides the user-visible OpenCL API functions and performs generic bookkeeping and scheduling. The hardware abstraction layer pro-

```
__kernel void triangle(__global int *x, __global int *y) {
    int i, t = get_global_id(0), sum=0;
    for (i=0; i < t; i++) sum += x[i];
    y[id] = sum;
}
```

Figure 8: Example OpenCL Kernel Program

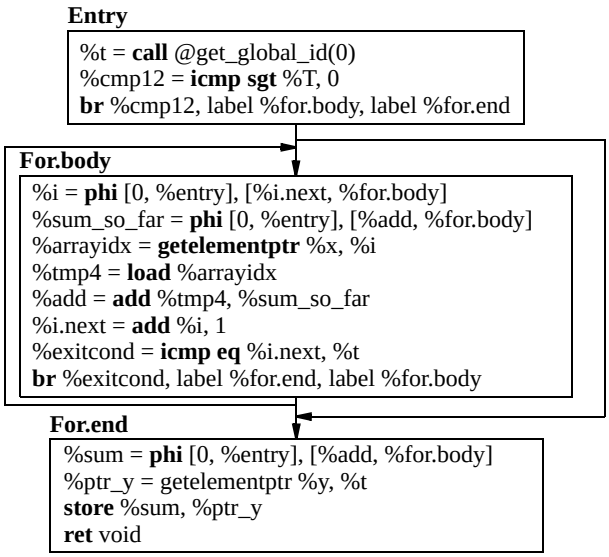


Figure 9: Intermediate Representation Example

vides low-level services to the platform independent layer. These services include: device and kernel discovery, raw memory allocation, memory transfers, kernel launch and completion. In particular, all communication between the host and the kernels goes through this layer.

6.2 Kernel Compiler

To compile OpenCL kernels into a hardware circuit, we extended the LLVM Open-Source compiler [10] to target an FPGA platform as shown in Figure 7. The LLVM compiler represents a program as a sequence of instructions, such as load, add, subtract, store. Each instruction has associated inputs and produces a resulting value that can be used in computation downstream. A group of instructions in a contiguous sequence constitutes a *basic block*. At the end of a basic block there is always a terminal instruction that either ends the program or redirects execution to another basic block. The compiler uses this representation to create a hardware implementation of each basic block, which are then put together to form the complete kernel circuit.

The above abstraction allows us to implement a kernel from basic block modules. Each basic block module comprises an input and an output interface with which it talks to other basic blocks. In the cases of a first and a last basic block, their interfaces have to be exposed at the top level, forming primary inputs and outputs of a kernel module.

6.2.1 C-Language Front-End

The first step in the conversion of a high-level description to a hardware circuit is to produce an intermediate representation (IR). To illustrate the IR, consider a program in Figure 8. In this example, each thread reads its ID using the *get_global_id(0)* function and stores it in variable *t*. It then sums up all elements of array *x* beginning at the first and ending at *t-1*. Finally, the result is stored in array *y*.

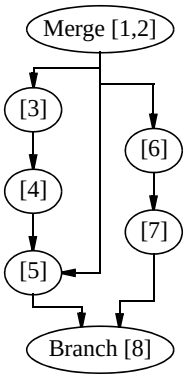


Figure 10: Basic Block Module

C-Language front-end parses a kernel description and creates an LLVM Intermediate Representation (IR), which is based on static single assignment [12]. It comprises basic blocks connected by control-flow edges as shown in Figure 9. The first basic block, **Entry**, performs initialization for the kernel and ends with a branch instruction that decides if a thread should bypass the loop. The second basic block represents the loop body and the last basic block stores the result to memory. To determine the data each basic block consumes and produces, we perform Live Variable Analysis.

6.2.2 Live Variable Analysis

Live Variable Analysis identifies variables consumed and produced by each basic block. In our example, the **Entry** basic block contains only kernel arguments as input variables (x, y). At the output of the basic block, variables *sum*, t and i are also created. This tells us that each thread produces these values when it completes execution in this basic block. The **For.body** basic block includes all kernel arguments as well as the three arguments produced by the first basic block. It then produces y, t, *i.next* and *add* as output live variables. Notice that *i.next* and *add* effectively replace i and *sum* when the basic block loops back to itself, allowing the loop to function correctly. Finally, the last basic block has input live variables y, t and *add*, while no variables are live after the return instruction.

6.2.3 CDFG Generation

Once each basic block is analyzed, we create a Control-Data Flow Graph (CDFG) to represent the operations inside it. Each basic block module takes inputs either from kernel arguments or another basic block, based on the results of Live Variable Analysis. Each basic block then processes the data according to the instructions contained within it and produces output that can be read by other basic blocks.

A basic block module, shown in Figure 10, consists of three types of nodes. The first node is the *merge* node, which is responsible for aggregating data from previously executed basic blocks. This ensures that for each thread, its *id* as well as all other live variables are valid when the execution of the basic block begins. In addition, in cases such as loops, the merge node facilitates *phi* instructions that take inputs from predecessor basic blocks and select the appropriate one for computation within the basic block, based on which predecessor basic block the thread arrived from.

Operational nodes represent instructions that a basic block needs to execute, such as load, store, or add. They are

linked by edges to other nodes to show where their inputs come from and where their outputs are used. Each operational node can be independently stalled when the successor node is unable to accept data, or not all inputs of the successor node are ready. This resembles the idea of elastic circuits [13, 14], however our implementation is much smaller and simpler because each operation has fixed latency. In particular, in [14] a convergence of data flow from two operations feeding a single one required 2 FFs and 15 gates, where communication between elastic nodes requires four signals. In [13], the approach is simpler with two signals in each direction and a total of 4 gates in what they refer to as a join operation. In our implementation we use two signals and two gates. The valid signal into a node is an AND gate of its data sources (called *ready*). The stall to each predecessor node is computed as *!ready+stall_out*. The fanout splitting, or fork, logic in our implementation is also distinct from [13, 14]. In our case, each output of a node has an associated register called *consumed* that indicates if a specific successor already consumed the data being produced. If so, the register is set to 1. When all consumed register are, or are about to be set to 1, the functional unit producing a value is unstalled (its stall_in is cleared).

The last node in a basic block module is a *branch* node. It decides which of the successor basic blocks a thread should proceed to.

6.2.4 Loop Handling

Loops are handled at a basic block level. A simple example of a loop is a basic block whose output is also an input to it, such as shown in Figure 9. The loop itself presents a problem in that it is entirely possible that a loop can stall. To remedy the problem, we insert an additional stage of registers into the merge node, that allows the pipeline to have an additional spot into which data can be placed.

When loops comprise many basic blocks, it is possible that stalling can occur when loop-back paths are unbalanced. In such cases, we instantiate a loop limiter that allows only specific number of threads to enter the loop. The number of threads is equal to the length of the shortest path in a loop.

6.2.5 Scheduling

Once each basic block is represented as a CDFG, scheduling is used to determine the clock cycles in which each operation is performed; however, since no sharing of resources occurs, the key contribution of scheduling is to sequence the operations into a pipeline where independent operations occur in parallel. This is important because not all instructions require the same number of clock cycles to complete. For example, an AND operation may be purely combinational, but a floating point addition may take eight cycles. Thus if possible, we would like to schedule operations that add up to 8 cycles while the adder performs computation maximizing the throughput of the hardware circuit, while reducing the area at the same time. In some cases, it may be necessary to insert pipeline balancing registers into the circuit because one execution path is longer than another.

To solve the scheduling problem we apply SDC scheduling algorithm [15]. The SDC scheduler uses a system of linear equations to schedule operations, while minimizing a cost function. In the context of scheduling, each equation represents a clock cycle relationship between connected operations. For example, in implementing an equation $f = a * b + c * d$, the scheduler has ensure that both multiplications occur before addition. A secondary objective is the reduction of area, and in particular the amount of pipeline balancing registers required. To minimize the impact on area, we minimize a cost function that reduces the number of bits required by the pipeline.

6.2.6 Hardware Generation

To generate a hardware circuit for a kernel we build it out of basic block modules. To achieve high performance, we implement each module as a pipelined circuit, rather than a finite state machine with datapath (FSMD). This is because a potentially large number of threads need to execute using a kernel hardware, and their computation is largely independent. Hence, the kernel hardware should be able to execute many threads at once, rather than one at a time.

In a pipelined circuit, a new thread begins execution at each clock cycle. Thus, a basic block with pipeline depth of 100 executes 100 threads simultaneously. This is similar to replicating an FSMD circuit 100 times, except that subsequent threads execute different operations. In our design, the *valid_in-stall_out* pairs are used as a handshaking mechanism to synchronize subsequent operations.

Once each basic block is implemented, we put the basic blocks together by linking the stall, valid and data signals as specified by the control edge. We then generate a wrapper around a kernel to provide a standard interface to the rest of the system. In our case, we implement all load and store instructions as Avalon Memory-Mapped Master interfaces [16] that can access data from global or local memory. In addition, the wrapper keeps track of kernel execution, issuing workitems into the pipelined circuit and signals when the kernel has completed execution.

7. SYSTEM INTEGRATION

Once each kernel has been described as a hardware circuit, we create a design comprising the kernels, memories and an interface to the host platform, as shown in Figure 11. We utilize a templated design, where sections that do not change from one application to another remain locked down on an FPGA. These sections include memory interfaces and a host interface facilitated by a PCIe core. The sections that change, shown at the bottom of the figure, are attached at compile-time, synthesized placed and routed. This section can include many kernels, possibly replicated several times, where each kernel has a dedicated local memory segment associated with it.

8. MEMORY ORGANIZATION

OpenCL defines three types of memory spaces: global, local and private. The global memory space is designated for access by all threads. Read and write operations to this memory can be performed by any thread; however, OpenCL does not guarantee memory consistency between an arbitrary pair of threads, thus one thread may execute fully before the other one even begins running. Thus, this type of memory is usually used to store data threads will require for computation, as well as any results the threads produce.

In our implementation, the global memory space resides in

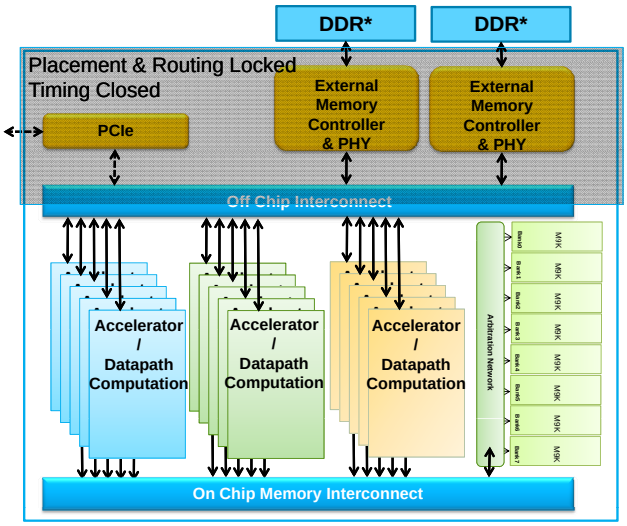


Figure 11: System Integration Details

off-chip DDR2-800 memory. It has large capacity allowing us to store data, but long access latency. Accesses to this memory are coalesced when possible to increase memory throughput. In the compiler we can detect when access to particular memory arrays are random or sequential and take advantage of this. When an array is determined to be accessed sequentially by consequtive threads, we create a special read or write module that includes a large buffer for burst transactions. This is very important, because if we know that the next set of threads will request consecutive data, we can request a large burst transaction the moment the first tread requests data. As a result we incur a small delay penalty loading/storing the data for the first thread, and no latency on the subsequent threads.

When memory accesses are random we create a coalescing unit that collects requests to see if the seemingly random memory accesses address the same 256-bit word in memory. If so, the requests are be coalesced into a single data transfer, improving memory throughput.

Local memory is used by work groups to enable synchronized exchange of data. To synchronize threads within a workgroup, barriers/memory fences are used to force threads to wait on one another before proceeding further. This allows complex algorithms that require collaboration of threads to be implemented (ex. bitonic sort).

Local memory is implemented using on-chip memory. It has short latency and multiple ports, allowing the kernel to access it efficiently. To do this we create a shared memory space for each load and store unit to any local memory. For example, if a kernel uses an array which it reads and writes a total of 4 times, then the compiler creates four memory ports for local memory. The four ports are logical (not physical), and are then mapped into a set of dual-ported on-chip memory modules. In a case where it is necessary to read more data in one cycle than the dual-port memory can provide we split memory into banks to enable faster data access, allowing for higher on-chip memory bandwidth.

Private memory is implemented with registers that store the data on a per-thread basis, and are pipelined through the kernel to ensure that each thread keeps the data it requires as it proceeds with the execution through the kernel.

9. OPENCL-SPECIFIC FEATURES

OpenCL defines features to allow synchronization of threads, such as barriers. In our framework, a barrier is a reordering First-In First-Out (FIFO) buffer. The buffer contains dedicated logic to force an entire workgroup to enter the FIFO before the first element out of a workgroup is allowed to exit. This ensures that all threads have stopped execution at a predefined location and performed all data accesses as required. This is essential, because with the exception of barriers and memory fences the OpenCL paradigm does not guarantee any ordering on thread execution. Thus, any data dependencies between threads must be guarded by a barrier.

The presence of barriers can help in many applications, especially when reordering is made possible. This is because a barrier that is followed by a global memory transation has the property that the threads exiting the barrier are in order. Thus, we can predict how they will access global memory and instantiate an appropriate memory access module as described in the previous section.

Because we use on-chip memory for barriers and local memory, it places a constraint on how many workgroups can simultaneously occupy the same kernel hardware. To ensure that this limitation is not violated, a workgroup limiter is included inside of a kernel. It permits only a specific number of workgroups to enter a kernel at any given time.

In addition to barriers, function calls to obtain global and local IDs, and other work-item intrinsics are provided. In our framework, these function calls are replaced by kernel arguments, whose value is filled by the kernel wrapper as each thread is issued.

10. EXPERIMENTAL RESULTS

To evaluate the quality of results produced by our OpenCL compiler, we implemented several OpenCL applications on a Terasic DE4 board, with the host program running on a Windows XP64-based machine. Each application was compiled to generate both the host program and the kernels. The kernels were synthesized, placed and routed using ACDS 12.0 and downloaded onto the DE4 board. We then ran the host programs to obtain performance results, and use ACDS 12.0 report files to obtain fmax and area metrics.

10.1 Benchmark Applications

The applications we implemented are: Monte Carlo Black-Scholes, matrix multiplication, finite differences, and particle simulation. The Monte Carlo Black-Scholes (MCBS) simulation is an option pricing approximation. It computes the same data as our initial case study, but it does so using a Monte-carlo approach. Each simulation in this benchmark requires a randomly distributed number generator, a Mersenne Twister, as well as floating-point computations, including exponent, logarithm and square root functions. Matrix multiplication is an application that exhibits easy-to-visualise parallelism. Finite differences (FD) is an application used in Oil and Gas industries to analyze sensor data and detect the presence of desired natural resources. It requires a large amount of memory bandwidth to run efficiently. Particle simulation is a demo distributed with NVIDIA's OpenCL package, that simulates collisions of particles in a cube. It is a broad test of language coverage.

The area and fmax measurements for each circuit are listed

in Table 2. The first column shows the circuit name, the second column shows clock frequency of the kernel, and the remaining columns show the area of each application including the memory and host interfaces. The circuit area is specified in terms of ALMs, FFs, number of DSP blocks and memory bits, as well as an overall logic utilization metric (*%Util*) on an Altera Stratix IV-530 device. The PCIe and DDR2 interfaces are included in the resource utilization, and comprise approximately 11% of resources. The throughput of each application is summarized in the last column of Table 2.

10.2 Discussion

Each application had its unique challanges that we needed to address to obtain high-quality results.

Monte Carlo Black Scholes simulation requires a Mersenne Twister to implement the random number generator. Describing it is easy in Verilog, but more challenging in a multi-threaded environment, because each thread must obtain a specific output value from a random number sequence and in turn generate the next random number for a subsequent simulation. While it implies a dependency between threads, it is possible to break that dependency by using barriers.

To do this, we synchronize the data accesses such that each workgroup accesses a subset of random numbers, while generating values for the next simulation. When a simulation completes, all other threads enter a barrier and wait. Once all threads enter a barrier, they are allowed to proceed further and compute the next result. This prevents any race conditions from occuring. To speed the process up, local memory is used and as such each workgroup uses a dedicated random number generator, which is initialized when the workgroup enters a kernel.

When we compare the circuit generated by our compiler to a hand-coded implementation [17], the compiler performed very well. We achieved a throughput of 2.2 billion simulations per second (Gsims/sec) in comparison to a hand-crafted design that achieved a 1.8 Gsims/sec [17] using two FPGAs. The difference in the number of FPGAs used can be omitted here, because Stratix IV device is larger than the Stratix III FPGAs used in [17]. It is expected that the circuit designed in [17] would fit successfully providing similar performance to our compiler-generated design if the same Stratix IV device was used.

The matrix multiplication application (1024x1024 floating point) uses on-chip local memory to store a part of a row and column in local memory. Each thread reads this data, performs a multiply-and-add operation and keeps track of the current sum. Once a thread finishes computing a matrix entry, it stores the result in global memory. To account for practical aspects of this application, we chose a sufficiently large matrix size such that external memory storage was required and the effects of communication between the FPGA and DDR2 memory were account for. A choice of 1024x1024 matrix size was sufficient as storing three 1024x1024 matrices requires 12 MBs of memory, which exceeds the on-chip memory capacity of the Stratix-IV device.

In this application, each thread is requires to access and process the same information other threads use. For example, when computing a column for a resulting matrix, each thread needs to access the same row of the first input matrix causing contention for memory access. To alleviate the problem, we vectorize the kernel to allow each thread to simultaneously perform a computation for four matrix entries. This increased our throughput by a factor of 4.

To utilize the high memory bandwidth, the inner loop is unrolled to perform 64 floating-point multiplications and addition simultaneously. This implementation permits a maximum throughput of 89.6 GFLOPS and we achieve 88.4 GFLOPS with some losses due to communication with the host. In comparison to [18], our compiler produces a faster circuit (16ms [18] to compute 64x64 matrix multiplication). A recent work using double-precision floating point on a Virtex5 device showed a performance of 29.8 GFLOPs [19]. A throughput of approximately 15.6, 15 and 8 GFLOPS was also reported in [20], [21] and [22] respectively. Integer matrix multiplication was implemented in [23], where 128 cores filled a Xilinx Virtex5 device. Our implementation comprises a pipeline of 500 clock cycles, still leaving a significant amount of the FPGA unused. This demonstrates that while an FSM-based approach can produce a small circuit for a single thread, over many threads a pipeline-based approach is superior.

The Finite Differences application is similar in nature to matrix multiplication in that the key operations happen in a tight loop comprising floating-point multiplication and addition. However, the data access pattern is irregular, thus lower bandwidth to global memory is achieved. Also, after each iteration of the loop, more preprocessing is required than in the case of matrix multiplication. While an FSM-based HLS compiler would attempt to apply loop pipelining in this case, it is not a necessary step in our flow. The key reason for it is that the design is already pipelined, so while each thread is takes several cycles to process the loop, several threads are being processed through the same loop simultaneously. Because we have many threads reading data from memory, we can detect when a series of threads accesses sequential memory locations and optimize memory accesses. In a case of loop pipelining, we would be accessing data in strides unless it was reordered in memory.

The final application is a particle simulation. The most time consuming part of this application is collision detection. It comprises several steps: partitioning, sorting and collision computation. Partitioning divides the cube in which particles collide into smaller subcubes. Each particle in a subcube is known to only be able to collide with particles within the same subcube, or one of the 8 adjacent subcubes. The particles are then sorted based on the subcube index they belong to. Once sorted, the collision detection is engaged independently for each subcube.

In this application, the constant exchange of data between the host, the FPGA and the GPU slows down the processing. The processing of a single frame on an FPGA takes only 9ms, while the remaining time is spent copying data from the FPGA to the GPU for rendering.

11. CONCLUSION

In this paper we presented the prototyping and development of an OpenCL-to-FPGA compiler. We showed through two case studies that the OpenCL-to-FPGA flow is not only feasible, but also effective in designing high-performance circuits. When discussing the initial case studies, we covered

Table 2: Circuit Summary and Application Throughput Results

Circuit Name	Fmax (MHz)	ALUTs	FFs	DSPs	Mem. Mbits	Util (%)	Num. copies	Throughput
MCBS	192.2	175687	261716	988	6.56	71%	12	2181 MSims/sec
MatMult	175	204894	254880	1024	5.12	80%	1	88.4 GFLOPS
FD	163.5	125662	180321	108	5.05	55%	4	647.6 Mpoints/sec
Particles	179.2	168527	206137	444	7.39	69%	1	62 FPS

major lessons we learnt from this process that helped us shape the architecture of the automated compiler. These lessons were incoporated in the framework, which we implemented and evaluated on a set of applications.

Our work shows OpenCL is well-suited to automatic generation of high-performance circuits for FPGAs. Our framework generated circuits for the benchmark suite that provide high throughput and has been shown to have a wide coverage of the OpenCL language, including synchronization using barriers, support for local memory, as well as floating-point operations. In addition, the framework includes a host library to communicate with kernels over PCIe interface. Finally, we have shown the ability to automatically implement complex OpenCL applications that comprise not only an FPGA-based computation engine, but also host processing that interfaces with peripherals such as a GPU.

The circuits generated by our compiler are very different from what a GPU-based implementation would look like. While at the high-level, the system components are similar, their architecture at the low level accounts for the difference. While on GPUs each processing core performs operations in a SIMD fashion, in our architecture each thread executes a distinct operation on a distinct set of data. Thus in a true sense, this kernel architecture is a multiple-instruction multiple-data style design. This accounts for its size, and also high performance.

12. REFERENCES

[1] M. Leeser, S. Corid, E. Miller, H. Yu, and M. Trepanier, "Parallel-beam backprojection: An FPGA implementation optimized for medical imaging," *Journal of VLSI Signal Processing*, vol. 39, no. 3, pp. 295–311, 2005.

[2] "Medical imaging implementation using FPGAs," *Altera Corporation - White Paper*, 2010.

[3] H. Guo, L. Su, Y. Wang, and Z. Long, "FPGA-accelerated molecular dynamics simulations system," *Inter. Conf. on Embedded Computing*, pp. 360–365, 2009.

[4] K. Wakabayashi, "C-based behavioral synthesis and verification analysis on industrial design examples," *ASPDAC*, pp. 344–348, 2004.

[5] Khronos OpenCL Working Group, *The OpenCL Specification, version 1.1.48*, June 2009. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.0.pdf

[6] *Nios II Processor Reference Handbook*, Altera, Corp., San Jose, CA, 2011.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *ACM*, vol. 13, pp. 422–426, May 1970.

[8] Y.-H. Yang, H. Le, and V. Prasanna, "High performance dictionary-based string matching for deep packet inspection," in *INFOCOM*, 2010, pp. 1–5.

[9] D. Suresh, Z. Guo, B. Buyukkurt, and W. Najjar, "Automatic compilation framework for bloom filter based intrusion detection," in *Reconfigurable Computing: Architectures and Applications*, ser. Lec. Notes in CS. Springer Berlin / Heidelberg, 2006, vol. 3985, pp. 413–418.

[10] LLVM, *The LLVM Compiler Infrastructure Project*, 2010. [Online]. Available: http://www.llvm.org

[11] A. Corp., "Altera complete design suite 12.0," *http://www.altera.com*, 2012.

[12] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Trans. on Prog. Lang. and Systems*, vol. 13, no. 4, pp. 451–490, Oct 1991.

[13] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *DAC*, July 2006, pp. 657–662.

[14] J. Cortadella and M. Kishinevsky, "Synchronous elastic circuits with early evaluation and token counterflow," in *DAC*, June 2007, pp. 416–419.

[15] J. Cong and Z. Zhang, "Activity estimation for field-programmable gate arrays," in *IEEE/ACM Design Automation Conference*, 2006, pp. 433–438.

[16] *Avalon Interface Specifications*, Altera, Corp., San Jose, CA, 2011.

[17] N. A. Woods, "FPGA acceleration of european options pricing," *XtremeData Inc. - White Paper*, 2008.

[18] M. Owaida, N. Bellas, K. Daloukas, and C. Antonopoulos, "Synthesis of platform architectures from OpenCL programs," in *FCCM*, may 2011, pp. 186 –193.

[19] V. Kumar, S. Joshi, S. Patkar, and H. Narayanan, "Fpga based high performance double-precision matrix multiplication," *International Journal of Parallel Programming*, vol. 38, pp. 322–338, 2010.

[20] Y. Dou, S. Vassiliadis, G. Kuzmanov, and G. Gaydadjiev, "64-bit floating point fpga matrix multiplication," in *In Proc. of the 13th Int. Symp. on FPGAs*, Feb 2005, pp. 86–95.

[21] L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs," in *Parallel and Distributed Processing Symposium*, Apr 2004, p. 149.

[22] M. Smith, J. Vetter, and S. Alam, "Scientific computing beyond CPUs: FPGA implementations of common scientific kernels," in *MAPLD*, 2005.

[23] A. Papakonstantinou, G. Karthik, J. A. Stratton, D. Chen, J. Cong, and W.-M. W. Hwu, "FCUDA: Enabling efficient compilation of cuda kernels onto fpgas," in *Proc. of the 7th Symp. on ASPs*, jul 2009, pp. 35–42.

SESSION

APPLICATIONS

Chair(s)

Dr. Franz Richter
FAU Erlangen-Nuremberg
Germany

Future Internet Infrastructure and its Hardware Support Technology for Smart Grid

Hiroaki Nishi
Department of System Design, Keio University, Japan.
e-mail: west@sd.keio.ac.jp

***Abstract*— This paper proposes a service-oriented communication platform enabling the provision of scalable smart community services. The proposed platform is managed and operated using Extensible Markup Language (XML) that has advantages in terms of flexibility, cost-effective implementation and affinity with databases. In practice, the platform is embedded into distributed communication nodes, e.g. routers, switches and home gateways, and both existing IP services and smart community services are provided over public networks. This platform should provide a lossless data correction for the application of smart grid because it is calculate average or total electric power consumption in order to maintain the stability of a power grid. To achieve the lossless data correction, database insertion architecture was implemented on FPGA and ASIC environment. Moreover, as an application, a building energy management system (BEMS) using the platform is demonstrated. The feasibility of the network architecture with the platform is discussed based on experimental results of data acquisition and control in the BEMS demonstration.**

***Index Terms*—Energy management, Smart grids, Coprocessors, Service-oriented Network, Optical communication equipment**

I. INTRODUCTION

THE situation around information is going through great change due to the improvement in its value. We are developing, implementing and evaluating energy management system in smart grid. Smart grid is shortly the integration of energy infrastructure and information and communication technology. This integration is not limited in an energy area but any kind of infrastructure is shifting to a smart system. In these smart infrastructures, the role of information systems becomes more important and is required to be more flexible to cache the great change. In this paper, though we focus on the smart grid area

Many different approaches are currently utilized in the field of integrated network control systems, known as smart grids. We implemented an energy management system with the aim of developing a future integrated infrastructure for this area. It is difficult to connect heterogeneous systems that contain different data structures and standards. Even if a system can correct data from the heterogeneous systems, the difference of required latency, jitter, reliability and privacy policy may cause a problem in a real installation. We considered distributed and shared operations in a large system containing various subsystems. This concept maintains interoperability between heterogeneous systems. The concept was demonstrated at the Fukue Port Terminal building in Goto City, Nagasaki Prefecture, and the area of Kurihara City, Miyagi Prefecture. A field demonstration of the energy management system was conducted. This system uses a variety of devices that are produced by seven different companies. An air-conditioning control system to reduce carbon dioxide emissions and achieve electric-load leveling with a battery was implemented and experimented with as an application of the proposed system. The battery was emulated by real-time system simulation as hardware-in-a-loop system.

In the future, it is expected that the smart grid will be developed into the smart community based on the broadband ICT infrastructure [9]. The smart community generates not only existing IP traffic but also real-time machine-to-machine (M2M) traffic including smart grid traffic [10]. Kim et al. [11] proposed a data-centric information infrastructure for the smart grid. Liu et al. [12] presented fundamental models to operate the smart grid in an integrated manner. The authors have been developed EMS applications using an Extensible Markup Language (XML)-based platform [13]. However, the communication platform to provide the smart community services with various requirements has not been studied. In the smart community, centralized and decentralized data management should be combined appropriately according to the requirements of each service [14], and therefore a service-oriented communication platform is needed.

This platform should provide a lossless data correction method for the application of smart grid because it is calculate average or total electric power consumption in order to maintain the stability of a power grid. To achieve the lossless data correction, database insertion architecture was implemented on FPGA and ASIC environment.

Data Stream Management Systems (DSMS) are a new class of management systems that handle enormous data streams [15][16][17]. DSMS makes it possible to look up all the data without accumulating the data in an external storage system. A feature of DSMS is the constant processing of data generated at a high bit rate by several sensors in real time. Thus, DSMS is required to produce high-speed data streams. DSMS assumes an application such as monitoring an RSS (RDF Site Summary) stream, stock trading, or RFID (Radio Frequency Identification) Management. For example, an online stock trading system requires enormous processing throughput of up to hundred thousand transactions per second. About one Kbyte of data per transaction is inserted when considering an average

size packet. Moreover, it is known that 25,000 transactions are concentrated in a peak period and the average packet size is about 1,300 bytes. In this case, the total required throughput can be calculated as 25,000 × 1,300 × 8 = 260Mbps [18]. Thus, the throughput required for DSMS is 260 Mbps at most. A database used on the Internet has to handle packets at the wire-rate without fail. This means the database also requires multi-gigabit-speed insertion. Therefore, it is difficult to perform the required throughput using existing software-based DSMS.

The Oracle TimesTen In-memory Database is a product for real-time data management. An In-memory Database (IMDB) achieves high-speed data processing by storing all the data in a main memory, which enables the application to access data directly. In addition, this makes it possible to perform rapidly and it can process a transaction in tens of microseconds. TimesTen allows the users of an application to access a database, select their data, or update information with the required high throughput execution. This technology can be widely applied to systems that need high-speed processing such as online stock trading or an information superhighway, which was difficult to apply with a general database before. According to the TimesTen IMDB white sheet, the average turnaround time is 30 µs when a tuple of data is inserted [19]. The average turnaround time is 11 µs when a tuple of data is read. If we capture traffic data over the internet using TimesTen, we can regard a tuple of data as a segment of packet data. The average packet size is about 1,300 B and we consider the size of tuple as 1,300 B. In this case, the required throughput of the reading process can be calculated as 1,000,000/11 × 1,300 × 8 = 950 Mbps, while the throughput of the writing process can be calculated as 1,000,000/30 × 1,300 × 8 = 350 Mbps. Network traffic flows at a multi-gigabit rate over the Internet and it is difficult for TimesTen to perform the necessary throughput. Therefore, TimesTen cannot achieve the required throughput for storing traffic in memory, although it achieves the required throughput for data capture to application users.

This means XML-based communication platform available for flexible data acquisition and control (DAC) is required in the smart community. XML-based communication platform has advantages in terms of flexibility, cost-effective implementation and affinity with databases. Firstly, the heterogeneity in the smart community and the need for protocol convergence are clarified. In addition, an Ethernet Passive Optical Network (EPON)-based network architecture [20] enabling the provision of scalable smart community services is proposed. In this research, we focus on a scalable EMS application providing home EMS (HEMS), building EMS (BEMS) and cluster EMS (CEMS) services. This paper discusses the feasibility of the proposed network architecture with the service-oriented communication platform based on experimental results of DAC in a BEMS demonstration.

Additionally, hierarchical cloud services by using passive optical network and contents-based routing on a special router with hardware designed database manager as an infrastructure of future smart grid. Our results demonstrate the effects of the proposed platform and its advantages.

This paper is organized as follows. The following section clarifies technical issues to provide the scalable smart community services. Section III proposes the XML-based communication platform. Section IV presents the configuration of the BEMS demonstration in Fukue Island, Nagasaki, Japan and the network architecture including EPON-based access networks. The experimental results in the BEMS demonstration are shown in this section. In section V, real-time data correction is discussed as an future application of smart grid infrastructure. Finally, our conclusion is described in Section VI.

I. Technical Issues in Smart Communities

This section describes heterogeneity in smart communities and the concept of protocol convergence to resolve the heterogeneity.

A. Heterogeneity in Smart Communities

The smart community is a kind of "system of systems," and includes many and various subsystems such as power systems, information/communication systems, intelligent transport systems (ITSs), electric vehicles (EVs) and information appliances. Currently, each subsystem is designed based on different existing standard technologies. Therefore, it is difficult to ensure the interconnectivity and interoperability among the subsystems. If available protocols are restricted by a new standard, the interoperability is ensured at the cost of the flexibility.

However, it is not realistic that available protocols are standardized for each interface in the smart community, since there are too many exchanged data and infrastructures. In such a heterogeneous environment, it is a better solution to develop a common platform which can flexibly operate and manage any existing protocols. In other words, we need to select and combine appropriate standard technologies according to requirements and data semantics when we construct a smart community system.

B. Protocol Convergence

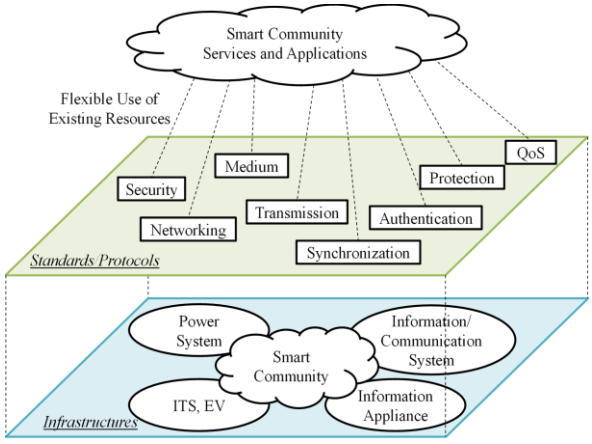


Fig. 1. Convergence of protocols related to smart communities.

From the viewpoint of applications, all infrastructures and protocols are considered as resources to manage and operate the smart community as shown in Fig. 1. For example, there are various standard technologies related to medium, quality of

service (QoS) and security. In practice, the infrastructures are connected by networks, and the standard technologies are available as resources beyond the range of each area. A service-oriented communication platform is expected to enable flexible management of the resources effectively. In addition, the platform should be flexible, cost-effective and compatible with cloud services, since the future smart community application is expected to work in collaboration with other IP-based services such as triple-play services.

In this research, the interface between the resource and the platform, and the interface between the application and the platform are designed so as to meet the above conditions. By using a formatted application-level message, a resource management server on the platform collects any information as configurations or specifications of each component of subsystems. This scheme enables flexible use of appropriate lower-layer protocols and standard technologies with fulfilling each requirement.

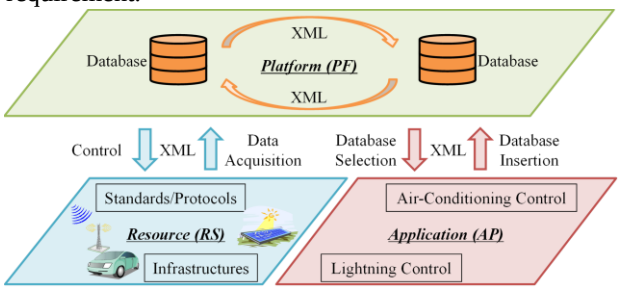


Fig. 2. XML-based communication platform for smart communities.

```
<?xml version="1.0"?>
<inms>
  <head>
    <version> 1.0 </version>
    <id name="device"> 12345abc </id>
    <timestamp timezone="JST"> 2011-02-24 23:13:56 </timestamp>
  </head>
  <body>
    <group name="A01">
      <timestamp timezone="JST"> 2011-02-24 21:45:43 </timestamp>
      <in name="temperature" type="room" id="ABC" action="write"> Data </in>
      <in name="humidity" type="room" id="DEF" action="write"> Data </in>
    </group>
  </body>
</inms>
```

(a) Message description in data acquisition

```
<?xml version="1.0"?>
<inms>
  <head>
    <version> 1.0 </version>
    <id name="device"> 56789def </id>
    <timestamp timezone="JST"> 2011-03-14 03:42:53 </timestamp>
  </head>
  <body>
    <group name="A01">
      <timestamp timezone="JST"> 2011-03-14 03:41:45 </timestamp>
      <out name="temperature" type="room" id="ABC" action="ack"> Data </out>
      <out name="humidity" type="room" id="DEF" action="ack"> Data </out>
    </group>
  </body>
</inms>
```

(b) Message description in database selection

Fig. 3. Examples of XML-based messages.

II. Service-Oriented Communication Platform

This section proposes a service-oriented communication platform enabling the provision of scalable smart community services. The proposed platform is managed and operated using Extensible Markup Language (XML). The message format described in XML is also explained.

A. Platform and Interface Design

Fig. 2 shows the system model of the proposed smart community. The system is comprised of "Resource (RS)" domain, "Application (AP)" domain and "Platform (PF)" domain. The role of each domain is described.

1) RS Domain: The RS domain includes available infrastructures and standard technologies/protocols. For example, storages, EVs, home appliances, sensors, actuators, communication networks and renewable energy sources are infrastructures. On the other hand, interfaces, QoS control protocols, authentication mechanisms, protection architectures are standard technologies/protocols. All the infrastructures and standard technologies are available for the smart community applications.

2) AP Domain: The AP domain includes any applications and services. For example, air-conditioning control algorithms, lightning control algorithms, energy management algorithms and any other algorithms are implemented as application software. In addition, visualization services of power consumption and home security services are provided as one of the smart community services. The smart community applications and services access the resource information through the platform.

3) PF Domain: The PF domain includes distributed databases for DAC. All information on available infrastructures and standard technologies/protocols are stored in the databases. The databases have management tables of the data collected from the RS domain. The collected data are classified according to the QoS level, security level, reliability level, and other basic levels. The advantages of the XML-based platform are summarized as follows.

Flexibility: The application-level platform should not be restricted by the lower-layer protocols or standard technologies. Thus, it allows the applications and services to flexibly operate and manage the resources. Using XML tags is a good solution for providing the flexibility of adaption and enhancement in exchanging any kinds of information concerning system resources because it is enough to designate the guidelines of tag naming and utilization.

Cost-effective implementation: The XML specification is released by the World Wide Web Consortium (W3C), and can be utilized by any users. Many applications and services are constructed on the XML based handler or driver. It is easy for users to create synergy effects by using mashup technology. The XML-based platform is suitable for open innovation. The XML description is also suitable for the applications which have the possibility to be used extensively in combination with other cloud services.

Affinity with databases: The smart community service is a wide area service, and it collects massive data sets such as sensor data, operation logs or operation commands from the subsystems. In other words, a data center in the smart

community has to manage a lot of data. Using the XML database is a direct solution of this requirement. In common use, general database management systems are enough to manage or store data elements implemented using XML-based and hierarchically-designed tables.

4) RS-PF Interface: The RS-PF interface is defined to realize DAC in the smart community. For example, distributed protocol converters collect sensor data from installed sensing devices, and send them to resource management servers on the platform using XML-based messages. In addition, the control data stored in the resource management servers are sent to the installed control devices by using XML-based messages.

5) AP-PF Interface: The AP-PF interface is defined to realize database selection and insertion. For example, the applications access resource management servers on the platform using XML-based messages, and obtain sensor data from the resource management servers. In addition, control data calculated by applications are sent to the resource management servers using XML-based messages.

6) PF-PF Interface: The PF-PF interface is defined to exchange stored data between databases. For example, distributed resource management servers on the platform share battery information of EVs using XML-based messages, and provide navigation services to drivers. In addition, database sharing makes it possible to provide virtualized cloud services.

B. Massage Descriptions

To exchange data through the RS-PF, AP-PF and PF-PF interfaces, it is necessary to define the format of XML-based messages. The message through the RS-PF interface is illustrated in Fig. 3(a), and the message through the AP-PF interface is illustrated in Fig. 3(b).

1) Overview: The message starts with the definition of XML version. The message description is compliant with XML 1.0 released by the W3C. The message is comprised of a header element and a body element. Both the header and body elements are described between start tag <inms> and end tag </inms>.

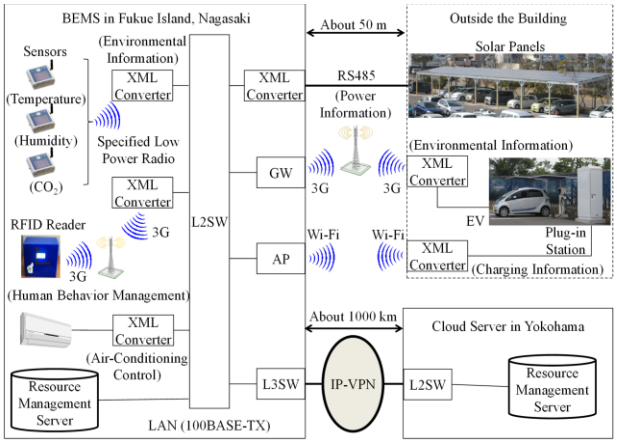


Fig. 4. System configuration of BEMS demonstration.\

2) Header Element: The header element is described between start tag <header> and end tag </header>. In the header element, the description of format version, device ID and time stamp is mandatory requirement. The format version, device ID and time stamp indicate the version of message format, ID of a message aggregator, e.g. an XML converter, and aggregation time, respectively. The format version is described between start tag <version> and end tag </version>. The device ID is described between start tag <id name="device"> and end tag </id>. The time stamp is described between start tag <timestamp name="timezone"> and end tag </timestamp>. In Fig. 4, JST means Japan Standard Time.

3) Body Element: The body element is described between start tag <body> and end tag </body>. The body element includes at least one group element, all user data are described between start tag <group name="group name"> and end tag </group>. The nested structure of the group elements is allowed. In the group element, the description of time stamp is mandatory requirement in the same way as in the header element.

Two description examples of the XML-based messages are also described in Fig. 3. Fig. 3(a) shows the situation that the sensor data are sent from the RS domain to the PF domain and written to the database on the platform. Fig. 3(b) shows the situation that the sensor data stored at the database on the platform are sent from the PF domain to the AP domain. The sensor data are described between start tag <direction name="data name" type="device type" id="device id" action="action mode"> and end tag </direction>. The tag <direction> is replaced by the tag <in>, which means input to the PF domain, or the tag <out>, which means output from the PF domain. The attribution "action mode" is replace by "read" (selection), "write" (insertion), "ack" (acknowledgment) or "transfer" (insertion and transfer). The description of all attributions is mandatory requirement.

III. System Configuration

This section presents the configuration of the BEMS demonstration using the proposed service-oriented platform. In addition, a future network architecture including EPON-based optical access networks is proposed for scalable smart community applications.

A. BEMS Demonstration

Fig. 4 shows the configuration of the demonstration system constructed in Fukue Island, Nagasaki, Japan. The system was developed in 2011 through the project supported in part by the Ministry of Internal Affairs and Communications (MIC) of Japan. The system includes various subsystems such as environmental sensors, photovoltaic generation system, power measurement system, air-conditioning control system, EVs, plug-in stands, resource management servers and communication networks. Each subsystem was developed by different vendors based on different standard technologies. Both pre-installed and newly-installed devices are mixed. Scalable and integrated control of the heterogeneous BEMS was achieved by using the proposed XML-based communication platform.

1) Environmental Sensors: The environmental sensors

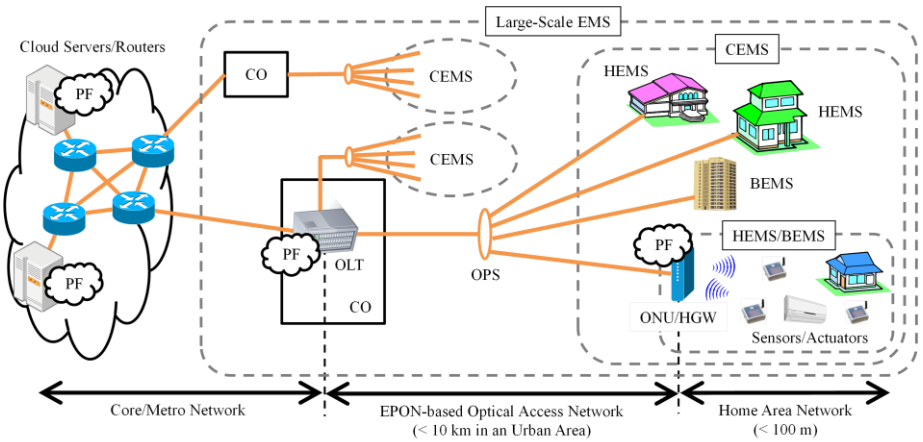


Fig. 5. System configuration of smart communities using EPON-based optical access networks.

measures temperature, humidity, lighting intensity, CO2 concentration, wind speed and the presence or absence of humans. All environmental sensors are powered by attached small photovoltaic panels. 88 environmental sensors were installed at restaurants, conference rooms, shops and shared spaces in the building, and connected to the XML converter. All the environmental data are sent to the resource management servers in any one minute using XML-based messages.

2) Photovoltaic Generation System: The 12 kW-class photovoltaic generation system includes 96 photovoltaic panels and a power conditioner. The photovoltaic panels were installed at the parking area of the building. The power conditioner converts the DC power generated by the photovoltaic panels to AC power at 60Hz. After that, the generated AC power combined with the AC power at 60 Hz is transmitted to the building. The environment and generation data such as temperature, solar radiation intensity, generated power, voltage, current and frequency are sent to the resource management servers every 10 seconds through the XML converter.

3) Power Measurement System: The power measurement system is comprised of 186 sensors, and measures power consumed by air-conditioning machines, lightning devices and outlets. The sensors are connected to the XML converter. The measured data, i.e. voltage, current, instantaneous power and power factor, are sent to the resource management servers in every 5 seconds using XML-based massages.

4) Air-Conditioning Control System: The air-conditioning control system achieves on/off control of the air-conditioning machines to reduce the power consumption. In the AP domain, the on/off scheduling of the air-conditioning machines is designed based on the static algorithm [13]. The static algorithm decides the on/off schedule in advance. In the PF domain, the scheduling tables on the platform are updated according the algorithm. In the RS domain, the relay switches connected to the air-conditioning machines are controlled according to the on/off scheduling. The air-conditioning control system using radio frequency identification (RFID) authentication was also installed in the building. The system turns on the air-conditioning machines when anyone exists in the room. The system turns off the air-conditioning machines when no one exists in the room. The personal data stored in the RFID tag are sent to the XML converter through 3G wireless communication, and then sent to the resource management servers using XML-based messages.

5) EVs and Plug-in Stands: In Fukue Island, 100 EVs are already installed by the Nagasaki EV&ITS Consortium, and quick chargers for the EVs are deployed for public use. This research used two EVs and quick chargers. The charging information is converted to XML-based messages, and sent to the resource management servers through wireless communication. The environmental data in the EVs measured by the environmental sensors are converted to XML-based messages, and sent to the servers through wireless communication. The driving history of the EVs obtained by the global positioning system (GPS) are also converted to XML-based messages, and sent to the servers through wireless communication.

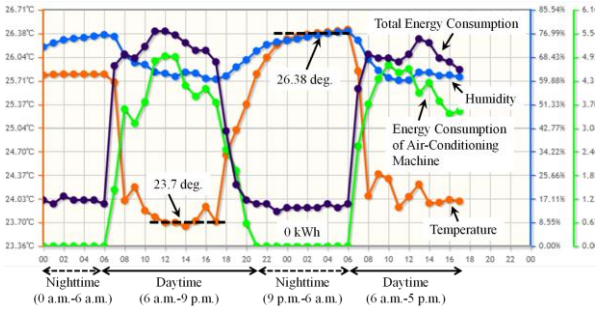


Fig. 6. Data visualization in BEMS demonstration.

6) Resource Management System: The resource management servers manage all data obtained from the BEMS and provide the service-oriented communication platform. In the servers, all collected data are converted to application-friendly format by the XML parser, and stored at the PostgreSQL-based database, which is a kind of open-source object-relational database management systems, on the platform. One of the servers is located at the building, and the other is located at Keio University, Yokohama, about 1000 km away from the building.

IV. EPON-BASED SMART COMMUNITY ARCHITECTURE

This section describes the architecture of EPON-based smart community architecture and its experimental results in the BEMS demonstration using the proposed communication

platform, and discusses the feasibility of the future smart community architecture from the viewpoint of the network delay. Fig. 5 shows a system configuration of the proposed smart communities we are currently planning to construct. The network architecture includes wide-area core/metro networks, EPON-based optical access networks and home area networks. The future smart community system should be managed and operated by the distributed and service-oriented communication platform (PF) in an integrated manner. The EPON-based optical access network is suitable to provide flexible, cost-effective and broadband services especially in an urban area.

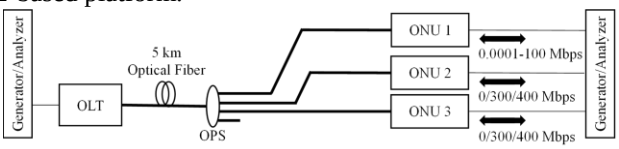
The EPON is comprised of one optical line terminal (OLT) located at a telecom central office (CO) and multiple optical network units (ONUs) located at user premises. The OLT and ONUs are connected by optical fibers and optical power splitters (OPSs). In the EPON system, the transmission of downstream signals to the ONUs is based on time division multiplexing (TDM), and the transmission of upstream signals from the ONUs is based on time division multiple access (TDMA). In addition, the upstream and downstream signals are multiplexed by wavelength division multiplexing (WDM) technology.

Currently, most of the HEMS and BEMS services are provided using a kind of home gateway (HGW). In the BEMS demonstrated in Nagasaki, the resource management server installed in the building has a function as the HGW. In future smart communities, large-scale EMS services may be provided using cloud servers located at distant data centers. However, mission-critical services including sensor/actuator networks and emergency services, which require several-millisecond delay, cannot be provided through the large-scale core/metro networks because the data are delayed for more than 100 ms.

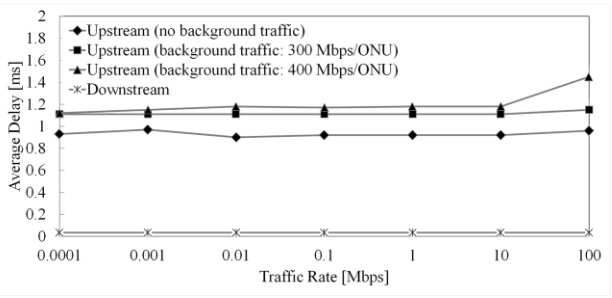
In the proposed architecture, any communication nodes, i.e. not only ONU/HGW but also OLT, switches and routers, have the service-oriented platform to analyze the data exchanged through them. For example, if the data are to be used only in the local area, the OLT does not transfer the data to core/metro networks and transfer them to the destination directly. The routing is performed not only using the destination address of packets but also based on the application-level data analysis. This technique enables the provision of scalable smart community services including the mission-critical services and the existing IP services. In addition, since unnecessary data are not transferred to core/metro networks, it leads to secure data management and traffic reduction of core/metro networks.

A. BEMS Demonstration

In the experiment, all the collected data were visualized on the web, and air-conditioning machines of the building were controlled based on the static algorithm [13]. Following services are implemented on the platform using the proposed XML-based platform through a common application program interface (API). The applications are not limited in the followings. We also developed similar systems at Kurihara, Miyagi, Japan, and at the campuses of Keio University, Yokohama, Japan. We are going to connect these different BEMS and construct a CEMS by using the API and the XML-based platform.



(a) Experimental setup for delay measurement



(b) Upstream and downstream delays of modeled traffic

Fig. 7. Experimental setup and results.

Fig. 6 shows an example of the data visualized on the web. Temperature, humidity, energy consumption of the air-conditioning machine and total energy consumption in a room are visualized. For example, we can confirm that the temperature varies from 23.7 degrees C (during the daytime) to 26.38 degrees C (during the nighttime). The energy consumption of the air-conditioning machine during the nighttime was 0 kWh, since the air-conditioning machine was turned off by users. This visualizer can depict all measured sensor data. The data are stored in the resource management servers located at the building and Keio University, distributed to the all visitors of the terminal building and used to voluntary energy saving and effective use of EV quick charger.

The air-conditioning control was also performed by the static algorithm [13]. The air-conditioning machines were turned off during 15 % of operating time. As a result, 13.42 % of total CO2 emission of the building was reduced by the air-conditioning control system. Questionnaires on comfort level were sent to 23 users of the building. If a user feels comfort-able as usual, the answer is 0. If a user feels hotter/colder than usual, the answer is +1/-1. If a user feels much hotter/colder than usual, the answer is +2/-2. The average value of the answers was about -0.33. When the air-conditioning control was not performed, the average value of the answers was about -0.37. Thus, it is confirmed that usual comfort level can be kept, even when the air-conditioning control is performed.

B. Feasibility of the EPON-based Architecture

The data collected in the BEMS demonstration is summarized in Table I. We collected approximately 2 million data per day. In the BEMS, the average data rate through the server was less than 1 Mbps. In the future CEMS using the proposed smart community architecture shown in Fig. 5, however, the data is collected at the OLT. The existing IP services such as triple-play services and the smart community services including mission-critical services have to be provided simultaneously. In this case, the traffic is estimated to grow to

approximately 100 Mbps.

To confirm the feasibility of the proposed architecture, we measured the upstream and downstream delays of the EPON. The experimental setup for delay measurement is illustrated in Fig. 7(a). One OLT was connected to three ONUs using an OPS. For the upstream/downstream directions, the background traffic data were sent from/to ONU2 and ONU3. The upstream and downstream delays through ONU1 were measured. The experimental results are shown in Fig. 7(b). The average upstream delay varies from 0.9 ms to 1.5 ms. The average downstream delay was around 35 µs regardless of the background traffic. It was confirmed that the average round-trip transmission delay in the EPON is approximately 1.5 ms. The proposed architecture can accept mission-critical services which cannot be provided by distant data centers.

C. Experimental Results

To confirm the feasibility of the proposed architecture, we measured the upstream and downstream delays of the EPON. The experimental setup for delay measurement is illustrated in Fig. 7(a). One OLT was connected to three ONUs using an OPS. For the upstream/downstream directions, the background traffic data were sent from/to ONU2 and ONU3. The upstream and downstream delays through ONU1 were measured. The experimental results are shown in Fig. 7(b). The average upstream delay varies from 0.9 ms to 1.5 ms. The average downstream delay was around 35 µs regardless of the background traffic. It was confirmed that the average round-trip transmission delay in the EPON is approximately 1.5 ms. The proposed architecture can accept mission-critical services which cannot be provided by distant data centers.

TABLE I
DATA COLLECTED IN BEMS DEMONSTRATION

Type	Number of Sensors	Collection Interval	Number of Data per Day
Air-Conditioning Control	15	5 s	259,200
Environmental Sensors	88	1 min.	74,880
Power Measurement	186	10 s	1,607,040
Photovoltaic Generation	1	10 s	8,640
Supplied Power	2	5 s	34,560
Total	292	-	1,984,320

V. REAL-TIME DATA CORRECTION

When network applications exchange large number of data including smart grid information over the Internet, to correct these data and achieve the selection process as a database management system becomes important to accomplish a quick response from critical status of power grids. The benefit of handling the content of packets mainly exchanged among electric devices at a router is great, but this benefit is not widely exploited by existing network infrastructures. This is because a router or other network device has to guarantee the full-cost wire-rate processing of packet stream reconstruction. Thus, the cost of high speed processing and high memory requirements is a major issue. To solve these problems, several studies have proposed the construction of streams from packets.

Not limited in this smart grid area, network streams contain useful information such as the creation time of a stream or a history of user behavior on the network, which is not exploited. We propose high-throughput insertion hardware to accelerate the process and prevent the correction of important packets including smart grid protocols.

A. Data correction on a router and DBINS Co-processor

The peak throughput of Internet traffic is always changing because it depends on time, events, and the network environment. In this kind of sever condition, lossless correction of required packet is indispensable for the future smart grid infrastructure on the Internet. In smart grid area, as described in EPON-based architecture, quick response is the most important requirement for maintaining the stability of a power grid. Nowadays, cloud based power grid control is well discussed. However, we can say to achieve this control on a cloud service is difficult because the required delay for the stability is up to several milli-seconds. Within this delay, communication, calculation and control have to be finished. To address this timing critical application, the intermediate network device, a router can help the real-time control on the Internet.

As the first step of this application, we design the DBINS co-processor that achieves lossless filtered packet correction and management. This is a core function of the database insertion hardware. The DBINS Co-processor consists of three hardware modules; DB Insertion Engine (DBINS Engine), Archiving Engine, and In-memory Reading Engine (Fig. 8). The DBINS Engine allows us to insert captured data into IMDB at a multi-gigabit throughput rate. The Archiving Engine migrates data from IMDB to a disk-based database, while the IMDB Reading Engine is a mechanism for preprocessing the basic functions of stream processing such as selection, projection, and joining. In this study, the DBINS Engine is focused on satisfying wire-rate database insertion.

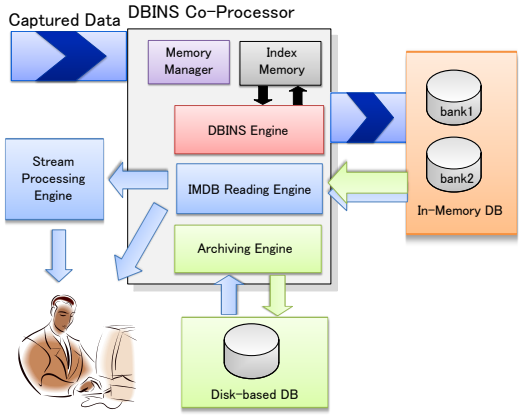


Figure 8. DBINS Co-processor

B. Implementation

When DBINS Engine stores network stream into an IMDB, it is inefficient to store all data, so it is necessary to extract the required data selectively according to a query issued by users. It is also desirable that the data is listed based on the

issued query, where the stored data can be loaded as a series of data by high-speed selection processing. Therefore, the DBINS Engine creates and index that is accessible to the stored data according to Leaf-ID, which is the ID query key. To make the stored data accessible by Leaf-ID, the DBINS Engine creates the index shown in Table II. Next.ID Pointer is the next pointer in the ID list. The Share Number is the number of owners of the captured data. When several queries match only one packet, the information is recorded with this value. After the stored data is loaded by the selection operation, the Share Number is decremented, but the data is deleted if the Share Number becomes zero. In addition, the DBINS Engine manages the index of the captured data so it is scalable with an increase in queries, because the resources of the DBINS Co-processor are limited.

TABLE II. KEYS AND POINTERS IN THE INDEX

Keys and pointers	Contents
ID	Leaf-ID
Timestamp	Packet arrival time
Packet Size	Packet Size
Share Number	The number of owners of captured data
Prev. T-Pointer	Previous pointer of Time-list
Next. T-Pointer	Next pointer of Time-list
Next. ID-Pointer	Next pointer of ID-list

To optimize the data structure, four different index structures are focused on Leaf-ID as the primary key. Part of a stream matches multiple queries issued by different upper layer applications, so it is preferable to combine the management information for each part. All four methods perform this optimization in a different way. Thus, four different index structures are implemented.

In the first approach, the DBINS Engine copies the overlapping parts of captured data and adds a Next.ID Pointer to all of the generated indexes. This approach facilitates the selection and deletion of data, because the DBINS Engine does not manage the Share Number of the index. However, when the number of overlaps is large, the index creation throughput is degraded because the DBINS Engine has to make multiple indexes for each part of a data stream. It also causes a loss of memory efficiency because the same data is stored in several segments of the memory. Furthermore, to support newly extracted data arrival the DBINS Engine copies data, but it must have a buffer that stores the data until the copying is finished.

In the second approach, the DBINS Engine adds new IDs based on a combination of matching queries. To insert the captured data, the DBINS Engine manages two tables. One manages the correspondence of the Leaf-IDs and the new IDs. The other manages the memory area of new IDs used in the IMDB. Using this approach, the DBINS Engine can generate an index at a constant rate and it has the advantage that there is no need to manage a large queue, unlike approach 1. However, the data management and index table management becomes complicated. Approach 2 has another drawback. If the number of issued query increases, the table size increase according to O(2n) in the worst case. Moreover, when a new query is issued, the DBINS Engine must refresh the entry table.

In the third approach, the DBINS Engine adds several Next.ID pointers to an index. Using this approach, the DBINS Engine can improve the efficiency of memory usage compared with approach 1, while it can decrease the cost of managing the index table compared with approach 2. However, the index generation rate depends on the Share Number of the captured data. Therefore, when the overlap of a query is large, the index generation throughput is degraded. Using this approach, the number of Next.ID Pointers is limited by the reserved space because the BDISN Engine only reserves enough space for a fixed number of Next.ID Pointers

The fourth approach also adds several Next.ID Pointers to an index. The difference from approach 3 is that the DBINS Engine stores the Next.ID Pointers in the IMDB. Using this approach, the DBINS Engine does not require a large space for the Next.ID Pointers in the index memory and it can support a larger Share Number than approach 3. However, DBISN Engine has to access the IMDB whenever it creates an index, so the throughput is worse than approach 3.

Each processing cycle for the above approaches is shown in Fig. 9. The requirements for throughput, selection speed, or the memory utilization ratio depend on the application or services. Therefore, the application or service providers must choose an index structure that meets their requirement. The requirements for throughput, selection speed, or the memory utilization ratio depend on the application or services. Therefore, the application or service providers must choose an index structure that meets their requirement.

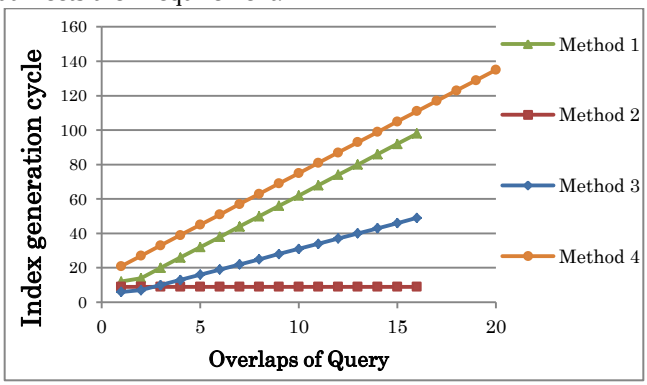


Figure 9. Processing Cycle

C. Implementation of DBINS Co-processor

The architecture of the DBINS Engine is shown in Fig. 10. The DBINS Engine has three blocks, Leaf-ID Module, Indexing Module, and IMDB Module. The Leaf-ID Module manages the Leaf-ID, which is added according to the query. The Indexing Module makes and manages the index. The IMDB Module stores the captured data in the IMDB. The procedure for storing the captured data is as follows. First, the header of the captured packet data is stored in the queue of the Leaf-ID Module while the body is stored in the queue of the IMDB Module. The Leaf-ID Module then checks the header information, which contains the Leaf-ID, timestamp, and index structure. The Leaf-ID Module creates a Leaf-ID Entry, which

contains the head address and tail address, or it loads the leaf-ID Entry information according to the header. Next, the Leaf-ID Module gets a free address from the Indexing Module, updates the Leaf-ID Entry, and sends the writing address to the IMDB Module. The Indexing Module generates an index using the Leaf-ID, timestamp, and index structure. The throughput of the Indexing module is mainly dominated by memory access so, to eliminate the memory access delay, the Indexing Module has three dedicated registers, i.e., old_ifreehead for Prev.T-Pointer, current_ifreehead for the address to write, and next_ifreehead for Next.T-Pointer. When the IMDB Module receives the writing address from the Leaf-ID Module, the IMDB Module stores it in the IMDB.

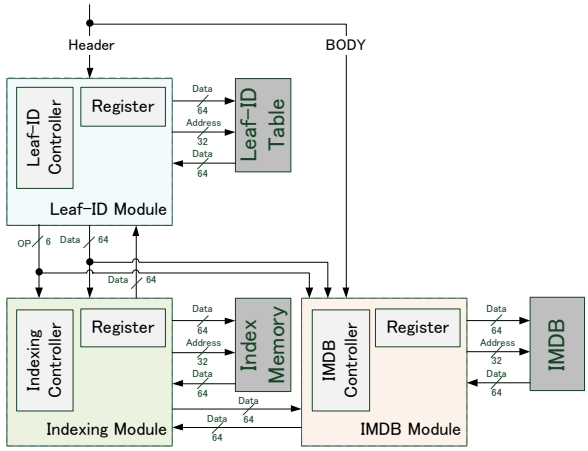


Figure 10. Architecture of the DBISN Engine

D. Evaluation of DBINS Co-processor

In this section, we evaluated the scale of the circuit and the throughput of the DBINS Engine. The DBINS Engine was implemented in Verilog HDL and it was synthesized using Xilinx ISE Design Suite 12.3 on FPGA (Vertex5 XC5VLX330T). For comparison, we used Synopsys Design Compiler 2005.09 by FREEPDK45n Technology as ASIC (Application Specific Integrated Circuit) [21]. The scale of the circuit, latency, and maximum frequency are shown in Table III. The throughput of the DBINS Engine was calculated based on the synthesis result. The memory parameters are shown in Table IV. The throughput of the DBISN Engine is shown in Table V (synthesized using ISE Design Suite) and Table VI (synthesized using Synopsys Design Compiler). Where an off-chip memory was used as the index memory, equation (1) and (2) were used to calculate the cycle time. Table V and Table VI show the available network throughput in two cases. In case 1, we assume the continuous storage of 50-byte packets, which was assumed to be the minimum size of network traffic. In case 2, we assume the storage of an average HTML packet size in the Internet backbone traffic captured by the WIDE MAWI project [22] between 14:00 and 14:15 on July 12th in 2009. The traces of the anonymous filter can be downloaded from the MAWI site. We used special trace data that was originally captured in the project and not filtered.

When the overlap of queries was four, this showed that the available throughput was 8.25 Gbps with a 50-byte packet size and 17.1 Gbps with 1306-byte packets, when the DBINS Engine was synthesized on FPGA. However, the throughput was 27.0 Gbps with a 50-byte packet size and 55.9 Gbps with a 1306-byte packets size, when the DBINS Engine was synthesized in ASIC. Therefore, the DBINS Engine could fulfill the metro network or edge network wire speed requirements when the overlap was sufficiently small.

TABLE III. RESULTS OF SYNTHESIS

1	Scale of Circuit	Latency (ns)	Max Frequency (MHz)
ISE Design Suite	# of Slices 475	3.73	318
Synopsys Design Compiler	(μm^2) 2.95 × 103	1.97	508

TABLE IV. MEMORY PARAMETERS

Memory standard	On-chip memory	Off-chip memory (QDR SRAM)	
		FPGA	ASIC
WL (cycle)	1	3	8
RL (cycle)	1	3	8

$$\text{Cycle time} = (\text{RL} \times \text{memory read count} + \text{WL} \times \text{memory write count} + \text{cycle time}) \times \text{Latency} \quad \cdots (1)$$

$$\text{Avairable network throughput} = \frac{\text{total packet size}}{\text{cycle time}} \quad \cdots (2)$$

TABLE V. AVAILABLE NETWORK THROUGHPUT USING FPGA

overlaps	Available Network Throughput (Gbps)			
	On-chip Memory		Off-chip Memory	
	50-byte	1306-byte	50-byte	1306-byte
1	13.4	4.3	17.1	17.1
2	13.4	3.8	17.1	17.1
3	10.7	3.2	17.1	17.1
4	8.25	2.8	17.1	17.1

TABLE VI. AVAILABLE NETWORK THROUGHPUT USING ASIC

overlaps	Available Network Throughput (Gbps)			
	On-chip Memory		Off-chip Memory	
	50-byte	1306-byte	50-byte	1306-byte
1	43.9	6.4	55.9	55.9
2	43.9	5.6	55.9	55.9
3	35.1	4.8	55.9	55.9
4	27.0	4.2	55.9	55.9

VI. CONCLUSION

This paper proposed a service-oriented communication platform for scalable smart community applications. In addition, the future smart community architecture including EPON-based optical access networks was also proposed. In the BEMS demonstration, we reduced 13.42 % of total CO2 emission of the building without sacrifice of the comfort level. The feasibility of the proposed smart community architecture is confirmed from the viewpoint of the network delay. The average round-trip delay of the EPON system was less than 2 ms, which is considered to satisfy the QoS requirement of mission-critical services.

Moreover, hardware-based DBINS co-processor is proposed.

This hardware supports high-speed database insertion into a router on the Internet. We implemented it in FPGA and ASIC. We showed that the throughput of the proposed architecture satisfied the wire-rate processing of a metro network with a throughput of 8.25 Gbps with 50-byte packets and 17.1 Gbps with 1306-byte packets. It also provided the requisite throughput of 27.0 Gbps with 50-byte packets and 55.9 Gbps with 1306-byte packets when the DBINS Engine was synthesized in ASIC. Based upon these results, the doors of the future network infrastructure for true real-time services like smart grid can be open.

ACKNOWLEDGMENT

This work has been supported by the VLSI Design and Education Center (VDEC), the University of Tokyo, Japan, in collaboration with Synopsys Inc., by National Institute of Communication Technology, by Low Carbon Technology Research and Development Program for "Practical Study on Energy Management to Reduce CO2 emissions from University Campuses" from Ministry of the Environment, by Special Coordination Funds for Promoting Science and Technology, and by Grant-in-Aid for Scientific Research (C) (22500069).

REFERENCES

[1] F. Li, W. Qiao, H. Sun, H. Wan, J. Wang, Y. Xia, Z. Xu and P. Zhang, "Smart Transmission Grid: Vision and Framework" *IEEE Trans. Smart Grid*, vol. 1, no. 2, pp. 168–177, 2010.

[2] IEEE Std 2030-2011, "Draft Guide for Smart Grid Interoperability of Energy Technology and Information Technology Operation with the Electric Power System (EPS), and End-Use Applications and Loads," 2011.

[3] NIST, "Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0," 2012.

[4] V.C. Gungor, D. Sahin, T. Kocak S. Ergut, C. Buccella, C. Cecati and G.P. Hancke, "Smart Grid Technologies: Communication Technologies and Standards," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 529–539, 2011.

[5] IEC 61850-SER ed1.0, "Communication Networks and Systems in Substations," 2011.

[6] C.H. Hauser, D.E. Bakken, L. Dionysiou, K.H. Gjermundrod, V.S. Irava, J. Helkey and A. Bose, "Security, Trust, and QoS in Next-Generation Control and Communication for Large Power Systems," *Int'l J. Critical Infrastructures*, vol. 4, no. 1/2, pp. 3-16, 2008.

[7] V.C. Gungor, B. Lu and G.P. Hancke, "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid," *IEEE Trans. Ind. Electron.*, vol. 57, no. 10, pp. 3557–3564, 2010.

[8] T. Sauter and M. Lobashov, "End-to-End Communication Architecture for Smart Grids," *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1218–1228, 2011.

[9] X. Li, R. Lu, X. Liang, X. Shen, J. Chen and X. Lin, "Smart Community: An Internet of Things Application," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 68–75, 2011.

[10] Z.M. Fadlullah, M.M. Fouda, N. Kato, A. Takeuchi, N. Iwasaki and Y. Nozaki, "Toward Intelligent Machine-to-Machine Communications in Smart Grid," *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 60–65, 2011.

[11] Y.-J. Kim, M Thottan, V. Kolesnikov and W. Lee, "A Secure Decentralized Data-Centric Information Infrastructure for Smart Grid," *IEEE Commun. Mag.*, vol. 48, no. 11, pp. 58–65, 2010.

[12] J. Liu, X. Li, D. Lu, H. Liu and P. Mao, "Study on Data Management of Fundamental Model in Control Center for Smart Grid Operation" *IEEE Trans. Smart Grid*, vol. 2, no. 4, pp. 573–579, 2011.

[13] T. Shen, T. Togoshi and H. Nishi, "Implementation and Substantiation of Energy Management Systems for Terminal Buildings," in *Proc. 16th IEEE Int'l Conf. on Emerging Technologies and Factory Automation , ETFA '11*, pp. 1–4, 2011.

[14] M. Shahraeini, M.H. Javidi and M.S. Ghazizadeh, "Comparison between Communication Infrastructures of Centralized and Decentralized Wide Area Measurement Systems," *IEEE Trans. Smart Grid*, vol. 2, no. 1, pp. 206–211, 2011.

[15] A. Arasu, S. Babu and J. Widom, "The cql continuous query language: Semantic foundations and query execution", VLDB Journal, 15, 2, (2006).

[16] A. Arasu, S. Babu and J. Widom, "The cql continuous query language: Semantic foundations and query execution", VLDB Journal, 15, 2 (2006).

[17] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World." In Proc. of SIGMOD, pp. 668 – 668, (2003).

[18] Yuan Wei, Sang H. Son, John A. Stankovic, "RTSTREAM: Real-Time Query Processing for Data Streams", Proc. of ISORC '06, pp. 141-150 (2006).

[19] Oracle TimesTen In-Memory Database - data sheet. Technical report, ORACLE JAPAN, July 2007.

[20] M. Qureshi, A. Raza, D. Kumar, S.S. Kim, U.S. Song, M.W. Park, H.S. Jang, H.S. Yang and B.S. Park, "A Survey of Communication Network Paradigms for Substation Automation," in Proc. IEEE Int'l Symp. on Power Line Communications and Its Applications, ISPLC '08, pp. 310–315, 2008.

[21] FreePDK, http://www.eda.ncsu.edu/wiki/FreePDK

[22] WIDE MAWI project, http://mawi.wide.ad.jp/mawi/

Hiroaki Nishi He received his Ph.D degree from Keio University, and currently he is serving as Associate Professor of Keio University, Japan and Visiting Associate Professor of National Institute of Informatics. He serves as IEEE-SA Japanese ambassador with Mr. Inoue, a member of his laboratory. He joins IEEE P2030 Standards Committee and a member of IEEE-SA ITS & Smart Grid Vision Project. He is also a member of IEEE Industrial Electronics Society (IES) Building Automation Control & Management (BACM) Technical Committee. He organizes Energy and IT special session in IEEE IES Industrial Informatics (INDIN) and Annual Conference of the IEEE IES (IECON) since 2006. He also manages several committee of Japanese ministry. He is conducting a project for a new generation network to achieve network based open innovation with a service oriented router as well as several smart community and smart island projects.

Software-Based Reconfigurable Computing Platform (AppSTAR™) for Multi-Mission Payloads in Spaceborne and Near-Space Vehicles

Dr. Edward R. Beadle and Dr. Tim Dyson
Harris Corporation
ERSA '12 Industrial Keynote

Abstract—We present an on-demand rapidly reconfigurable (~seconds) software-defined payload (SDP) architecture called AppSTAR™ with a core in-situ re-programmable processing capability that supports communications, radar, signal analysis and other missions. At the heart of Harris' AppSTAR™ SDP concept is a Virtex-based FPGA and interconnect fabric architecture that provides for a modular, flexible, scalable core capable of supporting a broad spectrum of missions with capabilities that can be customized for size, weight and power (SWaP) challenged platforms. Illustrating some of the capabilities evolving from this work, we present two real-world space qualified/qualifable SDPs, 1) a 100 Mbps-capable Ka-band software defined radio (SDR) for NASA and 2) a space-ready SAR/ISAR X-band RADAR based on the AppSTAR™ core. We also present an application of this core in a payload for an reconfigurable multi-mission space payload. The work described herein primarily leverages our space qualified V4 Processor employing several FPGAs in excess of 1 million gates each. Related work offering dramatically increased integration, reducing the V4 Processor card to 1 cubic inch package suitable SWaP challenged near-space and terrestrial applications is also discussed.

I. INTRODUCTION

Software defined radios (SDRs) have gained popularity due to the rapid reconfigurability of devices enabling the radio system to dynamically adapt to varying operational scenarios during a deployment or mission. Modern adaptation techniques include autonomous [1, 2] or cognitive methods [3], while traditional methods require external intervention (e.g. user entry or network control). Regardless of the adaptation method, the clear benefit is a single hardware platform that provides robust services over varying channel conditions and network connectivity, for example via adaptive modulation and/or waveform selection [4]. A key to the versatility of the SDR is that it relies on programmable computational hardware resources (e.g. DSP processors, FPGAs) rather than the dedicated circuitry typical in limited function MODEMS. A further attractiveness of SDRs is that they can be efficiently implemented using a high performance computing engine. Further, the commonality of the engine to support a spectrum of applications requiring advanced signal processing is supportable using a common tool suite and interfaces that enables dramatically lower non-recurring engineering costs, dramatically shortens development cycles, and improves the overall life cycle costs. This paradigm has been exploited in the development and real-world application Harris' V4 Reconfigurable Space Processor (V4 RSP) and highly integrated version the SiP-100 in the AppSTAR™ strategy.

The application of software-based dynamically reconfigurable DSP technology in the satellite domain has been limited. This is due to a number of interrelated factors such as

- space environment survivability
- limited SWaP envelope
- reconfiguration requirements
- software deployment/management complexity

However, it is possible to overcome these barriers and provide a highly flexible capability using an in-situ dynamically programmable hardware core (figure 1-1) we term the software-defined payload (SDP) AppSTAR™ concept which is a more general view than an SDR.

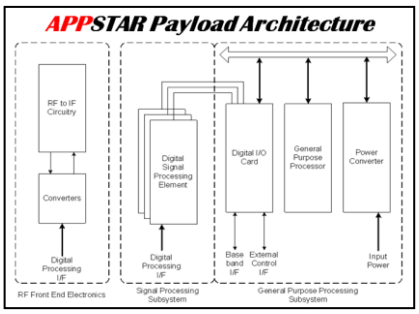


Figure 1-1: Software defined payload block diagram.

An SDR is a specialized case of adapting just one sub-system of a typical multifunction payload. The SDP on the other hand, is a concept where the "personality" and capabilities of a payload are quickly (~ seconds), and possibly autonomously, reconfigured on-orbit (figure 1-2) to meet subscriber service demands or a changing business model for commercial/civil space in remote sensing and communication.

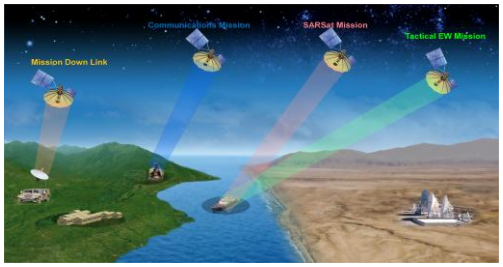


Figure 1-2: Notional use case for reconfigurable payload

"This work has been reviewed IAW the ITAR, 22 CFR part120.11 and the EAR, 15 CFR 734(3)(b)(3) and may be released without export restriction. Harris Corporation may hold specific patent rights pertaining to techniques disclosure herein.

This paper presents current work on a space qualified multi-mission payload compute-engine supporting the SDP concept based on an FPGA hardware architecture that supports complete or partial in-situ on-demand reconfiguration of the processing elements. We chose FPGAs as the computing platform to achieve a balance in the cost, power, performance trade-space between ASICs and fully programmable processors. Our approach utilizes advances in radio management software, signal processing hardware, and adaptable systems to meet not only radio systems needs, but those of other missions as well such as signal collection/processing, communication, RF imagery remote sensing (i.e. SAR/ISAR) and could be extended to other missions such as MOVINT (e.g. GMTI). The paper will also present the capability of the hardware elements operating in various sensor and communication instantiations for real-world applications.

The paper is organized as follows. Section II discusses SDP concept (section II.1) and two hardware variants of the same core, the V4 processing element (section II.2) and SiP-100. We also present some of the issues and mitigations for a reconfigurable spaceborne platform for operation and survivability in various radiation environments (section II.3). Sections III and IV present application of the V4 Space Processor in a communication payload and a remote sensing payload. Assuming the existence of various RF support elements, the applications could be melded into a single payload. Presently we leverage the reconfigurability of the hardware platform by merely loading different application software/firmware to customize the processing resources to the target application. The paper concludes with some closing commentary in section V.

II. FPGA-Based Software Defined Payload

II.1 AppSTAR™ SDP Concept

Harris has been delivering multi-processor based Software Defined Payloads (SDP) since 1998 resulting in a development path from general purpose RISC processor-based architectures to today's modern SDP architectures based on Xilinx Virtex FPGAs. With the advent of high performance FPGA processors, and techniques [6] which allow them to operate in space environments, the performance and reconfigurability necessary to tackle more complex processing scenarios has become increasingly viable.

The high-level block architecture of a generic AppSTAR™ SDP is illustrated in Figure 1-1. The basic elements of the system are the General Purpose Processing (GPP) Subsystem, the Signal Processing Subsystem, and the RF Front End Electronics Subsystem [6]. The GPP controls the payload operations and includes functions to load waveforms or configuration data. The GPP includes a common set of software infrastructure components that provide essential system management functions regardless of payload function through a consistent set of interfaces to configure, manage, and control the system hardware resources. The digital I/O cards provide standard interfaces to the vehicle, and can be customized for unique needs of a particular point application. The RF electronics are generally a separate package from the digital subsystems so that the RF functionality may be placed close to the aperture which tends to benefit overall system performance. The signal processing subsystem is a prime focus of this paper. The configuration of the signal processing subsystem dictates the function of the payload. This is the core of the AppSTAR™ concept.

A fundamental tenet of the Harris approach has been the incorporation of standards and leveraging commercial off the shelf (COTS) products to the greatest extent possible into the radio architecture. Operation of the Harris SDP utilizes a mix of commercial-off-the-shelf (COTS) hardware and software. A commercially available real-time operating system (RTOS) provides the key interface between the hardware and operating environment (OE). The OE is based on the NASA Space Telecommunications Radio System (STRS) infrastructure. The system backplane uses the compact Peripheral Connect Interface (cPCI) standard. As a physical interface standard, Spacewire using the Remote Memory Addressing Protocol (RMAP) is employed to provide the intra- and intercomponent communications.

In prior work, limited to SDRs, it was found that requiring knowledge of the underlying physical architecture was a significant impediment to waveform porting and third party development in FPGA implementations [7]. Therefore, a key design objective in SDP using the V4 Space Processor is to enable third party development, while ensuring that the application will not harm the hardware or other payloads on the spacecraft. To achieve these opposing objectives, a set of Hardware Description Language (HDL) modules have been designed and implemented to abstract the hardware details of the SDR and provide a standard interface for payload developers and mission planners [7, 8]. Hardware abstraction is a key aspect of developing a software radio that promotes extensibility and new application development [8]. Additional work is needed to create abstractions of processing elements representing an entire spacecraft bus that uses a single or set of SDPs in order to promote the inclusion of third parties in evolving applications for reconfigurable spacecraft. We have taken some first steps in this direction with hardware abstractions of the SDR [8]. These abstractions have led to the ability to easily use the V4 Space Processor in several remote sensing applications in addition to the communication applications it was envisioned for.

II.2 The V4 Reconfigurable Space Processor

The V4 Reconfigurable Space Processor (V4 RSP) is shown in figure 2-1 and is a core element of the SDP concept (figure 1-1) and provides a flexible compact Digital Signal Processor (DSP) module that delivers throughput rivaling dedicated ASIC-based solutions while providing an optimal balance in SWaP, cost and performance between dedicated function ASICs and fully programmable DSPs.

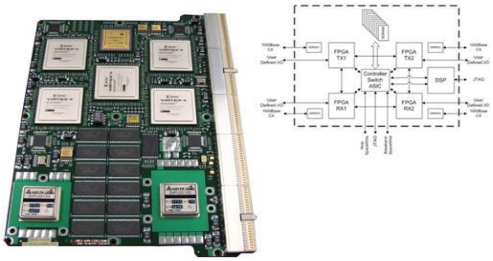


Figure 2-1: V4 RSP and block diagram.

The V4 RSP is latest variant of a line of sustained R&D into radiation resistant reconfigurable computing elements, complete with signal interfaces and data conversion, suitable for space environments. It integrates high performance Virtex-4 FPGAs (256 GOPS), 1 GLOP general purpose DSP (SMJ320C6701), 256 MB RAM, flexible standard IO (IEEE 1149.1 JTAG, Dual Spacewire, four 1000 base CX 2.5 Gbaud serial ports, and SEU Circumvention logic along with demonstrated total dose hardness > 30 Krads, into a single conductively cooled 6U compact PCI (cPCI) form factor consuming ~ 30W. The SDRAM shown is used to hold the application data, and the SDRAM is portioned amongst several possible applications (e.g. typically 16) for a mission. The SDRAM can also be reprogrammed "at will" to provide updates during a mission as warranted. Also, provisions have been made in the design such that larger fabrics of multiple processing cores can be easily woven together to form even more capable designs. Lastly, we are currently considering the system-level impacts to size, weight, power and speed of the new variants of Virtex radiation resistant/tolerant FPGAs, such as the Virtex 5Q family, as possible upgrades to our existing designs [5].

The V4 RSP has been used as a core element in several space systems research and development programs such as the NASA CoNNeCT SDR (section III) and two Imaging RADAR efforts (section IV). The V4 RSP (figure 2-1) will be undergoing in-situ testing as part of the NASA CoNNeCT SDR on the International Space Station in late 2012 (planned).

A major issue impacting the design and deployment of reconfigurable hardware in a space or near-space environment is radiation. Both prompt and long-term effects must both be considered. The V4 RSP addresses this issue by design. Elements impacting the design of this processing element are outlined in section II.3. However, for applications not requiring the radiation resistance of a space-deployable design, Harris has continued evolution of re-configurable processing engines into the highly integrated SDR SiP-100 (figure 2-2).

The SiP provides similar core functionality to that of a board-sized V4 space processor (figure 2-2), but has dramatically increased levels of integration. At a top-level, the Harris SDR SiP is composed of 25 million programmable gates over 5 FGPAs, one DaVinci DSP (which includes separate DSP and ARM-9 processors), two 125 Msps 14-bit ADCs, two 500 Msps DACs, 1 GB RAM, 1 GB Flash memory, USB 2.0 and Ethernet interfaces, an embedded operating system, on-module power management controls, and has equivalent processing capacity as a 160 GHz desktop computer. The small size and short bus lengths allow lower drive current, exhibiting low power and negligible digital noise.

The SDR SiP-100 is currently at technical readiness level (TRL) 7, having completed low-rate initial production (LRIP), passing MIL-STD-883 for 500 temperature cycles (-40°C to 100°C) and 100 g shock and vibration, and being demonstrated in multiple operational prototypes. The SDR SiP-100 is ideally suited for small systems, including man-pack, handheld, UAV's and other SWaP limited systems.

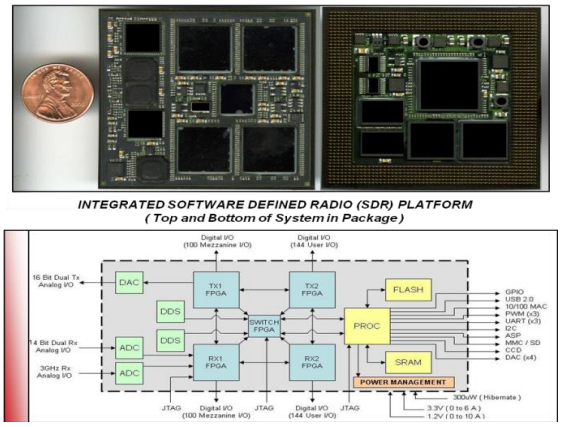


Figure 2-2: System in Package (top) & SDR diagram (bottom).

Similar to the V4, at a "flip-of-a-switch," the SDR SiP-100 can be dynamically repurposed to a new communications mode/waveform or recommit the entire computational fabric to perform functions for remote sensing or other desired operation. The FPGAs are typically loaded via the DaVinci's ARM-9 processor from on-module memory or off-chip resources.

The Harris SiP has already found use in miniature terrestrial software-defined radios [11], and is a step towards producing radios on a chip. An example of a packaged SiP-100 and tested in an SDR is shown in figure 2-3. The figure illustrates the evolution of the board-level

architecture V4-like system, into a deployable µTCA-compliant SiP-100 SDR platform.

In lab demonstrations the SiP-100 has shown that it provides high signal quality of -43 dB EVM for 64 QAM OFDM, demonstrated at over 200 Mbps [11]. Also, using the same hardware in the µTCA-compliant SiP-100 SDR platform we have implemented a novel digital chaotic communications system supporting a 100 kbps simplex link using a 10 MHz chaotically spread QPSK signal, with robust communications performance down to -13 dB below the intended receiver's noise floor. BER performance has been measured within 1 dB of theory, down to $8.5x10^{-6}$ [12].

The SiP provides an enabling technology for the emerging class of high altitude surveillance platforms called near-space vehicles (NSV) [13]. One potential application of low-altitude NSVs is to provide a rapidly deployable aerostat-based multi-standard cellular base station for disaster support. The dynamic reprogrammable capability of the SiP-100 enables the platform to be field reconfigured to support any of a variety of cellular or push-to-talk waveforms, and also offers the option to dynamically partition the payload processing capabilities in response to cellular usage/demand patterns or other sensing/payload processing functions such as MIMO, adaptive equalization, beamforming, or STAP. Given the anticipated CONOPs of NSVs, a SWaP efficient multi-mission payload will likely be the preferred design option.

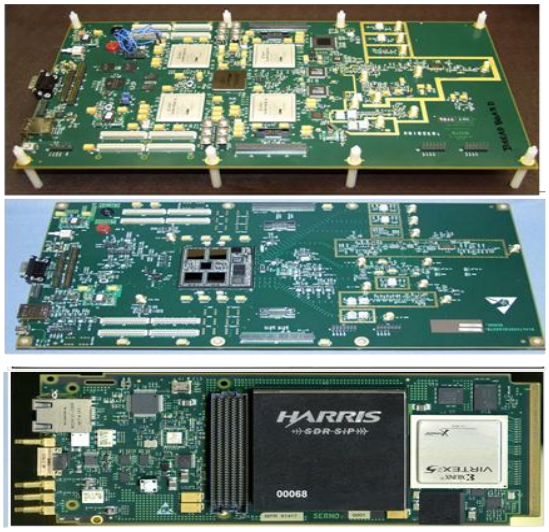


Figure 2-3: Evolution of the SiP-100 multi-mission platform from packaged parts development card (top), prototype SiP development card (center), and SiP deployment in an integrated µTCA implementation (bottom).

2.3 V4 Space Processor Radiation Hardness

For operation in a space environment the total ionizing dose (TID) and single event effects (SEE) require consideration. Additionally, the space environments are very different depending on the orbit (i.e. LEO, MEO, GEO, Molinya) and inclination. Shielding provides part of the solution, but shielding alone is often not practical. Hence, the V4 RSP is radiation hardened by design and part selection. The space processor, to meet the cost/performance objectives of the design, is specified to sustain a minimum total dose of 100 Krads before any performance degradation occurs, and 100Krad is feasible for most long duration LEO missions. However, for GEO mission life (e.g. 5 – 10 years), additional shielding is likely part of the solution.

The primary issue concerning this design is SEEs. SEEs can be extremely problematic for programmable devices, and are divided into three events classes

- Single Event Latchup (SEL)
- Single Event Upset (SEU)
- Single Event Transient (SET)

Our design philosophy has been to evaluate each part to ensure that they are immune from SEL (i.e. latch-up free), and to also characterize the SEU and SET events.

To mitigate the possible SEU and SET upset vulnerability the FPGAs V4 RSP uses two "on-board" methods:

- A "Watchdog ASIC" monitors each FPGA configuration memory during operation. The configuration is compared against the correct configuration (stored as a checksum in protected memory) and any detected errors are immediately corrected or "scrubbed". The scrubber power consumption depends on the scrubbing rate, which is user selectable to optimize power consumption for a given environment.
- The logic within the FPGA is augmented by selected Triple-Modular Redundant (TMR) storage and logic. A special development tool is used to triplicate the appropriate logic in the application. Voters are used to isolate an error and "vote" it out of the result providing corrected data on the fly.

These two techniques together deliver very high (>99%) availability, verified by radiation testing conducted at the Berkeley and Texas A&M accelerators while the V4 RSP performed as a high data rate MODEM.

The primary mechanism employed to mitigate SEE in the SDRAM is Error Detection and Correction (EDAC) coding. EDAC is used to protect the memory contents and it can correct any error that exists in any single nibble of a 32-bit word. A dedicated scrubber, housed in the "watchdog ASIC", reads every location in the SDRAM and validates the memory contents. The memory scrubbing rate is also programmable for possible power savings in various environments. Two spare SDRAM columns are provided in the memory array to allow the "watchdog ASIC" to "hotswap" these spare in for any other columns in the memory array as needed. We have used this memory architecture on several spacecraft.

Combined, these mechanisms reduce the cost of the system by allowing the use of commercial parts that meet Class S reliability and this memory architecture has been validated on several spacecrafts [6]. Any additional components have been screened to ensure that they meet the requirements for outgassing, total dose, and latchup.

III. V4 Space Processor in a Ka-band SDR

In support of the NASA Communications, Navigation, and Networking re-Configurable Testbed (CoNNeCT) program, Harris leveraged its existing SDP architecture to develop a high data rate Ka-band SDR. The CoNNeCT program is unique in that it will enable an SDR demonstration testbed on the International Space Station (ISS). Three different radios covering L-band, S-band, and Ka-band will be installed onboard the ISS and demonstrated as part of this program. The goal of this mission is to provide an operational testbed to demonstrate and exploit the features and benefits of SDRs on-orbit. Many different datarates, waveforms, frequencies, and coding techniques will be required, hence the need for reconfigurability in-situ and on-orbit.

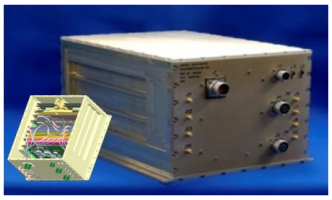


Figure 3-1: CoNNeCT Ka-band SDR Chassis.

The STRS compliant Ka-band radio delivered for this mission was based on the SDP architecture described above. Figure 3-1 shows the flight chassis designed around a 6U compact PCI open standard chassis which housed the majority of the payload elements. The Ka-band RF front end electronics is an external module to allow mounting at the antenna to optimize system performance. Figure 3-3 illustrates the overall payload architecture at the block level. A main feature of the design is the MODEM implemented with the Harris SDR based on the V4 processor. Again the common hardware architecture exploitable via software abstraction was a key enabler for the short design cycle and high performance.

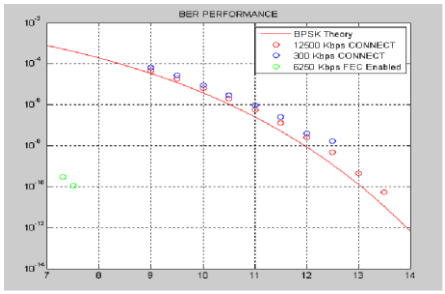


Figure 3-2: Sample CoNNeCT BER performance curve.

The Harris SDR has the ability to transmit up to 100 Mbps of user data rate coded with ½ rate error correction coding on the return link to the NASA tracking and data relay satellite system (TDRSS), and receive up to 25 Mbps on the uplink from TDRSS at Ka-band. Sample BER curves from lab tests are shown in the figure 3-2 versus theory. The FPGAs of the V4 processor were used to provide as much of the radio's functionality through the digital processor as possible. The radio digitally employed direct sampling of the intermediate frequency (IF) waveforms, a digital gain control algorithm, Doppler tracking filters, fine-tune frequency adjust, as well as the error coding, randomization, and other modem functions.

The CoNNeCT SDR is currently at a technology readiness level (TRL) of 8, system flight qualification through test and demonstration, and is scheduled to arrive at the ISS in 2012, where upon successful operation it will achieve a TRL of 9, system flight proven through successful mission operations. The the V4 RSP will be undergoing in-situ testing as part of the NASA CoNNeCT SDR on the International Space Station sometime in 2012 (planned).

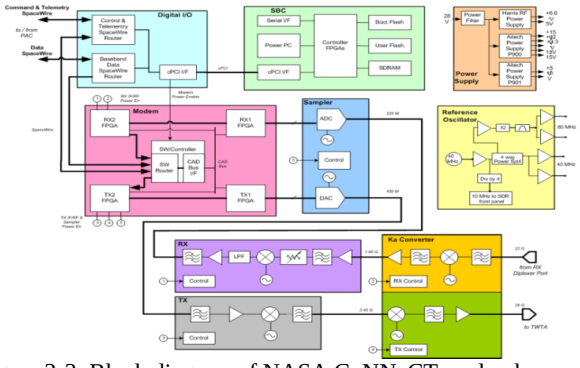


Figure 3-3: Block diagram of NASA CoNNeCT payload.

IV. The V4 Processor in Imaging RADARs

In order to demonstrate the simple re-purposing of the SDP architecture and core processing architecture reconfigurability to realize multi-mission capability, Harris embarked on a path to demonstrate an X-band radar capability in both lab demonstrations [9,12] and on-orbit demo of the AppSTAR^{TM} concept. In both designs, Harris heavily leveraged the V4 processing architecture as a core processing component of a chirp RADAR payload. The re-purposing of the V4 platform into the lab RADAR waveform generator was completed, from inception to lab integration, in less than 10 weeks, owing again to the software tools and adherence to standards supporting straightforward re-configuring and managing of the on-board resources. For the spaceborne payload, the V4 processor framework has been initially loaded with waveform descriptors specific to that mission's objectives.

IV.1 V4-based Software Defined X-Band RADAR Lab Demo System

For the RADAR demo payload, the FPGA firmware was programmed to generate chirped linear frequency

modulated (LFM) transmit radar waveforms. This combined some external support RF X-band multiplier/frequency converter to provide a LFM waveform capability of up to 800 MHz of bandwidth (figure 4-1). Various parameters such as pulsewidth, chirp rate, start frequency, stop frequency, pulse repetition frequency (PRF), and number of pulses are easily modified given the software defined nature of the payload. The ability to dynamically alter the RADAR waveform so completely could provide a basis for cognitive RADAR capabilities, as well as supporting the dynamic mission needs for specific collection scenarios (e.g. resolution, EMI avoidance, spectral allocation, target/scene phenomenology). These features are currently under consideration in our research on remote sensing.

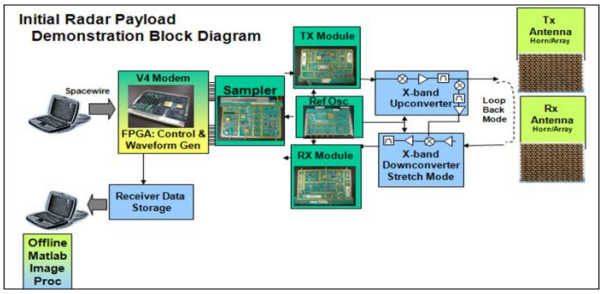


Figure 4-1: X-band software defined RADAR payload.

For initial lab demonstrations, a transmit pulse (400 MHz bandwidth) was digitally generated, frequency converted and multiplied up to X-band, and then looped back into the X-band receiver, pulse compressed by a swept stretch mode LO and digitized. The range Impulse Response (IPR) and Multiplicative Noise Ratio (MNR) were analyzed from the measured results [10]. The raw uncalibrated measured data shows nearly ideal main lobe shaping and acceptably low non-ideal sidelobe artifacts (figure 4-2). The only significant sidelobe energies of interest are the near-in lobes adjacent to the main lobe. These are due to deterministic phase errors and are easily removed through simple radar calibration techniques. However, even with these effects unmitigated the raw data exhibited an -23 dB MNR which meets typical allocations for RF radar electronics hardware.

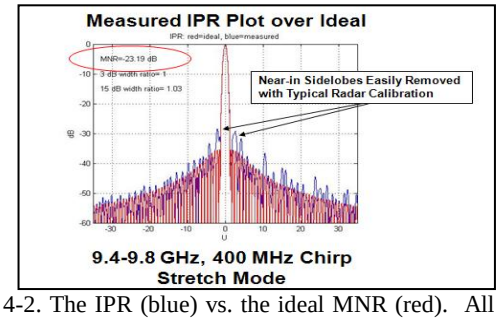


Figure 4-2. The IPR (blue) vs. the ideal MNR (red). All curves use Taylor weighted -35dB, nbar=5, per typical methodology.

To put the preceding result in a system context, the system and chirp waveform was tested as an inverse SAR (ISAR) RADAR. A simulated ISAR collect was performed on a ¼ scale model P-38 aircraft in an anechoic chamber (Figure 4-3). The RADAR waveform parameters were varied during the measurements with ultimate resolution capability of 0.25 m demonstrated which is typically 3x better than many commercial SAR systems. The ISAR images of the P38 scale model were formed by varying the RF chirp bandwidth using a constant chirp rate and varied pulse length. For the measurements shown, the frequency chirp slope rate was set to 40 MHz/usec and the pulse width was changed from 5 to 20 usec. At the highest resolution of 0.25m, the proportionate details of the plane's engines and fuselages, pilot cockpit, tail section, and wing-mounted fuel tanks were clearly recognizable.

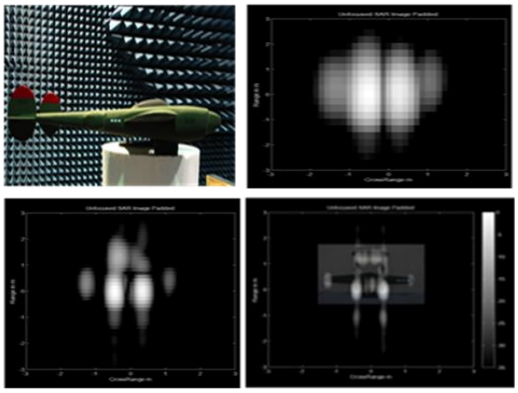


Figure 4-3: The lab mock-up of the P-38 and ISAR images at 1 m (top right), 0.5 m (bottom left), and 0.25 m (bottom right).

Plans are underway to build on the success of the lab demo system to dynamically switch payload from RADAR-only to a shared simultaneous RADAR/Comm system with waveforms dynamically loaded to meet real-time operational remote sensing and communication needs, again using the V4 re-configured in-situ.

IV.2 V4-based AppSTAR™ Space RADAR Payload

The V4 Space Processor card has been designated as the Reconfigurable Computing Element during the AppSTAR™ development to better reflect its multi-function usage and in response to a few enhancements incorporated to better match its true multi-mission role in AppSTAR™ concept. For the initial application configuration, as has been discussed on the RADAR lab demonstrations, an entire Synthetic Aperture RADAR (SAR) data collection application has been developed. This encompasses transmit waveform generation, including digital pre-correction of the wideband pulses (> 500 MHz) for IF/RF line-up imperfections such as gain slope, ripple and phase deviations from ideal linearity, and precise timing control of transmit and receive processing functions for both stripmap and spotlight modes as well as for the stretch (de-chirped) and matched filter operations [10]. Simultaneous receive processing, all hosted on the V4 fabric, includes digital quadrature down-conversion, mission selectable filtering and adaptive re-quantization of

returns to accommodate limited downlink resources while maintaining image quality.

Digital output compensation follows the Harris joint RF and digital system design philosophy of meeting operating requirements in a joint system-oriented fashion. This allows the RCE, coupled with the RF up-conversion electronics, to meet the expected mission MNR and IPR error budget allocations while minimizing overall system design and development cost.

The entire application load for the SAR mission easily fits within the four V4 FPGA fabric with significant margin and includes the extensive IO Wrapper designs used to abstract all hardware level IO interfaces. These consist of: A/D and D/A interfaces, CAD Bus, SERDES, Fiber Channel, chip-to-chip parallel IO, clock and enable high-speed transport and multiplexed Test Ports on each FPGA. The VHDL implementation follows a model-based design flow starting from a bit-exact behavioral model developed in MATLAB/Simulink to provide bit-true test vectors for all processing functions and data conversion interfaces. The model-based design approach along with the extensive use IO abstraction minimizes development and intergration time for an application by as much as 30%.

One of the enhancements to the RCE was to expand the on-board storage capacity to support four full application images for all four FPGAs. This allows the AppSTAR™ platform to support four concurrent missions with the ability to execute any two within a single orbit. Re-configuration and initialization can be accomplished in < 5 seconds once an uplink re-configuration command is received.

5. CONCLUSIONS

The adaptability and configurability of this space-qualified AppSTAR™ SDP architecture quickly enables the payload development of communications, radar and other missions. The AppSTAR™ concept, with the V4 and SiP-100 as processing cores, has proven valuable by supporting new and changing mission objectives, additional waveforms, and signal processing algorithms. Going forward SDPs can be assembled from processing elements and dynamically programmed in –situ to adapt to rapidly changing mission profiles, thereby extending mission life and increasing the value of platforms. Additionally the commonality of the processing core reduces mission life-cycle cost by decreasing the NRE incurred in new system designs.

Operational experience has shown that the V4 processor can be completely reprogrammed through industry standard interfaces and begin operating in a new configuration in < 15 seconds. This level of adaptability yields unprecedented dynamic mission flexibility for a multi-mission capable payload. It also opens the possibility to consider on-board autonomous controllers to manage the sensor timelines to address prioritized mission elements when configured in a multi-mission payload.

Work is planned to continue evolving AppSTAR™ payload concepts with the SiP-100 and V4 Space Processor to incorporate more complex capabilities to address the emerging areas of cognitive radios and dynamic spectrum access in conjunction with advanced remote sensing capabilities, signal analysis/recognizers, and other specialized needs.

6. REFERENCES

[1] A. Recio, J. Suris, P. Athanas, *"*Blind signal parameter estimation for the Rapid Radio framework*"*, in *IEEE MILCOM 2009 Proceeding, 2009.*

[2] *Autonomous Software-Defined Radio Receivers for Deep Space Applications*, NASA DECANSO, J. Hamkins and M. Simon (eds), 2006.

[3] T. Rondeau, Application of Artificial Intelligence to Wireless Communications, *Ph.D. Thesis, Virginia Tech, 2007.*

[4] J. Xu, Wei Su, M Zhou, "Software-Defined Radio Equipped with Rapid Modulation Recognition", in *IEEE Trans on Veh Tech*, vol. 59, No. 4, May 2010, pp. 1659 – 1667.

[5]*www.xilinx.com/esp/aerospace-defense/space/index.htm (accessed May 9, 2012).*

[6] *V.* Kovarik, Jr., S. McDonald, "Adaptable Architectures for Advanced Space-Based Communications Systems", in *International Communication for Satellite Systems Conference*, Edinburgh Scotland, June 2009.

[7] V. Kovarik, "The impact of hardware architecture on waveform portability" in *Software Defined Radio Forum Technical Conference*, Orlando, FL, November 2006.

[8] V. Giddings, T. Kacpura, and V. Kovarik. "Methods and approaches for abstraction of hardware dependencies in software radios" in *Software Defined Radio Forum Technical Conference - SDR07*, Denver, CO, November 2007.

[9] A. Mast, "Software defined payload architecture reduces cost and risk for various missions", in *IEEE Aerospace Conference Proceedings*, March 2011.

[10] *Spotlight Synthetic Aperture Radar: Signal Processing Algorithms*, W. Carrar, R. Goodman, R. Majewski, Artech House(1995).

[11] C. Moffat, J. Schroeder, R. Lilley, "Anti-Jam OFDM Waveform Design for Tactical Communications, in *IEEE MILCOM 2010 Proceedings, 2010.*

[12] E. Beadle, A. Michaels, J. Schroeder, "New Alternatives for Interference Tolerant Waveforms Hosted on a Software Programmable Multi-Mission Platform", in *2012 Waveform Diversity Conference Proceedings*, Lihue Hi.

[13] W. Wang, "Near-Space Vehicles: A Gap Between Satellites and Airplanes for Remote Sensing", in *IEEE A&E Systems Magazine*, April 2011, pp. 4-9.

Acknowledgements

The NASA CoNNeCT SDR payload described in this paper was partially supported through contract #NNC09AA01A with NASA Glenn Research Center. The author would like to thank the NASA CoNNeCT program teams both at Glenn Research Center and at Harris Corporation whose efforts contributed greatly to the successful demonstration of this platform.

Implementation of a Hardware Architecture to Support High-speed Database Insertion on the Internet

Yusuke Nishida[1] **and Hiroaki Nishi**[1]
[1]A Department of Science and Technology, Keio University, Yokohama, Kanagawa, Japan

Abstract - *In recent years, highly functional Web pages have been designed easily using free web services via web-based API. Google, Amazon, Facebook, YouTube, and other web services provide web APIs that supporting the design of rich web content. As the Internet develops, more useful services will be provided. In this environment, it is considered that new services can be made by storing network streams into databases on a router because a router can passively correct the data that is widely distributed over the Internet. The utilization of network streams is facilitated by implementing DBMS (Database Management System) as a hardware circuit. In this study, we examined the structure of an index for a database on router, which we implemented on FPGA. This showed that the throughput of the proposed architecture satisfied the wire-rate processing of the network with a throughput of 12.7 Gbps with 50-byte packets and 20.3 Gbps with 1,306-byte packets.*

Keywords: In-memory Database, Insertion Hardware, Memory Management, Service-oriented Router

1 Introduction

Web services are increasingly attractive because of the rich content provided by flexible and powerful APIs. Mashup is a technology that combines useful information from several sources on the Internet, which allows the creation of added value and services. The demand for more rich and valuable services will rise with the development of the Internet. If we can exploit the content of network transactions at the heart of the Internet, it will be possible to create highly functional and comprehensive services.

When network applications exchange large data over the Internet, the data is divided into several or many packets. These packets are known as the network stream. For example, a bundle of TCP/IP packets is known as a TCP stream. When a service is provided over the Internet, data should be managed and analyzed as a network stream rather than a bundle of packets. This means that packets must be reconstructed to form a stream to provide a service. Although there are many benefits in utilizing information as a stream, the use of a stream is mainly achieved by end-hosts and routers do not exploit the benefits. The benefit of handling the content of packets at a router is great, but this benefit is not widely exploited by existing network infrastructures. This is because a router or other network device has to guarantee the full-cost wire-rate processing of packet stream reconstruction. Thus, the cost of high speed processing and high memory requirements is a major issue. To solve these problems, several studies have proposed the construction of streams from packets.

Network streams contain useful information such as the creation time of a stream or a history of user behavior on the network, which is not exploited. If this information could be used effectively, an application provider could provide richer services by using the data captured at routers compared with data captured by existing end-hosts. This is because routers or gateways are better located. Thus, we propose a new service platform to provide services based on the utilization of a network stream at a router or gateway. To utilize network stream content, we need to extract the requisite part of a stream, store this in a database, and manage an index of this partial data based on selective throughput. If this function was implemented on a high-end router, it would allow processing throughput up to multi-gigabits, which cannot be achieved using only a software-based function. In this paper, we propose high-throughput insertion hardware to accelerate the process.

The remainder of this paper is organized as follows. Section 2 describes work related to this study, which focuses on database insertion and stream databases. Section 3 describes the concept of our Database Insertion Co-processor (DBINS Co-Processor) and its key technology. In Section 4, we present the design and implementation of our proposed mechanism. The evaluation is presented in Section 5 and our conclusions are stated in Section 6.

2 Related Work

In this section, we briefly survey the underlying technologies that can achieve high-throughput data insertion.

Data Stream Management Systems (DSMS) are a new class of management systems that handle enormous data streams [1][2][3]. DSMS makes it possible to look up all the data without accumulating the data in an external storage system. A feature of DSMS is the constant processing of data generated at a high bit rate by several sensors in real time. Thus, DSMS is required to produce high-speed data streams.

DSMS assumes an application such as monitoring an RSS (RDF Site Summary) stream, stock trading, or RFID (Radio Frequency Identification) Management. For example, an online stock trading system requires enormous processing throughput of up to hundred thousand transactions per second. About one Kbyte of data per transaction is inserted when considering an average size packet. Moreover, it is known that 25,000 transactions are concentrated in a peak period and the average packet size is about 1,300 bytes. In this case, the total required throughput can be calculated as $25{,}000 \times 1{,}300 \times 8 = 260$Mbps [4]. Thus, the throughput required for DSMS is 260 Mbps at most. A database used on the Internet has to handle packets at the wire-rate without fail. This means the database also requires multi-gigabit-speed insertion. Therefore, it is difficult to perform the required throughput using existing software-based DSMS.

The Oracle TimesTen In-memory Database is a product for real-time data management. An In-memory Database (IMDB) achieves high-speed data processing by storing all the data in a main memory, which enables the application to access data directly. In addition, this makes it possible to perform rapidly and it can process a transaction in tens of microseconds. TimesTen allows the users of an application to access a database, select their data, or update information with the required high throughput execution. This technology can be widely applied to systems that need high-speed processing such as online stock trading or an information superhighway, which was difficult to apply with a general database before. According to the TimesTen IMDB white sheet, the average turnaround time is 30 µs when a tuple of data is inserted [5]. The average turnaround time is 11 µs when a tuple of data is read. If we capture traffic data over the internet using TimesTen, we can regard a tuple of data as a segment of packet data. The average packet size is about 1,300 B and we consider the size of tuple as 1,300 B. In this case, the required throughput of the reading process can be calculated as $1{,}000{,}000/11 \times 1{,}300 \times 8 = 950$ Mbps, while the throughput of the writing process can be calculated as $1{,}000{,}000/30 \times 1{,}300 \times 8 = 350$ Mbps. Network traffic flows at a multi-gigabit rate over the Internet and it is difficult for TimesTen to perform the necessary throughput. Therefore, TimesTen cannot achieve the required throughput for storing traffic in memory, although it achieves the required throughput for data capture to application users.

3 Database on a Router

An enormous amount of network traffic flows over the Internet with a multi-gigabit throughput rate, so a database on a router has to capture network traffic selectively. To prevent leakage during capture, high-throughput data insertion functions are required by the databases compared with the selection function in the database. This requirement depends on the application a user wants to execute. Thus, the requisite throughput for database insertion should be more than multiple 10 G bps over a core, metro, and edge network. In a local area network, 1 Gbps throughput is required when it is used in a section of a business enterprise. As noted previously, TimesTen is an In-memory software-based database management system, which can improve the performance of database insertion up to the speed of several hundred Mbps. However, it is still difficult for TimesTen to meet the requirements of storing traffic data in a database. Therefore, we focus on hardware based wire-rate insertion into an in-memory database and we describe the proposed architecture of this insertion hardware.

3.1 Database Architecture

In a commercial database system, a disk-based database is widely used to provide a reliable database system, such as financial institutions, academic organizations, and hospitals. Generally, a disk-based database system writes user data onto the hard disk and RAID technology is typically usually for this. This technology can be applied widely to devices that require high I/O throughput, as well as high reliability, which is guaranteed by its redundancy function. However, this disk-based database lacks processing throughput. In-memory Databases (IMDB) have become popular because they can provide a new application with better throughput. IMDB allows us to achieve the high throughput processing required by today's real-time applications, but it has disadvantages because of its small capacity memory devices. Given this limited database capacity, it is better to filter captured data according to the user's query and to store packet data into IMDB selectively, like a DSMS, rather than to store captured data into IMDB directly. Moreover, the database also requires a disk-based database as an archiving device to maintain high reliability and fault tolerance.

For a database on a router, the hierarchical architecture of IMDB and a disk-based database is appropriate for reliable and high-throughput insertion. The top throughput of IMDB is almost equal to the throughput of memory I/O, which is enough to store network stream data selectively. We planned to use on-chip or off-chip memory as a temporary database, which is used as a primary database. At the same time, disk-based database is used as an archiving device. A data migration function is also implemented to move the captured data into appropriate storage, i.e., an IMDB or Disk-based database. This hierarchical architecture eliminates the throughput requirements of disk-based databases.

3.2 DBINS Co-processor

The peak throughput of Internet traffic is always changing because it depends on time, events, and the network environment. If all the traffic data from the Internet is captured on a router and stored into a database, the IMDB will be soon filled with an enormous volume of data in a busy networking environment. It is desirable to filter the network traffic using a query operation before an insert operation. High-throughput database insertion is required to ensure the optimum performance of database during the peak throughput period of the target network without any discards of packets.

The insertion process requires several complex processes such as data filtering, indexes creation, and memory management. It is difficult to obtain optimum memory I/O throughput performance because these complex processes are a bottleneck for overall performance. Thus, we proposed a DBINS Co-processor as a core function of the database insertion hardware. The DBINS Co-processor consists of three hardware modules; DB Insertion Engine (DBINS Engine), Archiving Engine, and In-memory Reading Engine (Fig. 1). The DBINS Engine allows us to insert captured data into IMDB at a multi-gigabit throughput rate. The Archiving Engine migrates data from IMDB to a disk-based database, while the IMDB Reading Engine is a mechanism for preprocessing the basic functions of stream processing such as selection, projection, and joining. In this study, the DBINS Engine is focused on satisfying wire-rate database insertion.

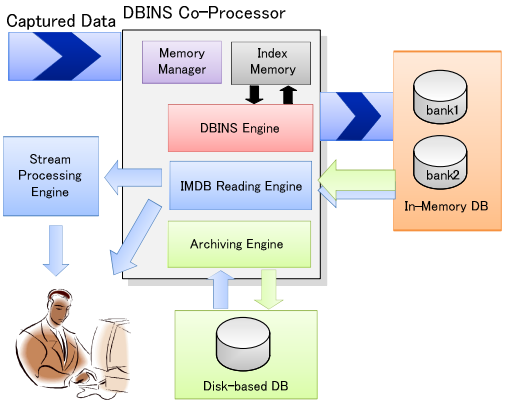


Figure 1. DBINS Co-processor

4 Implementation

In this section, we describe how the DBINS Engine manages the data captured on the Internet and the architecture of the DBINS Engine.

4.1 Index Structure

When DBINS Engine stores network stream into an IMDB, it is inefficient to store all data, so it is necessary to extract the required data selectively according to a query issued by users. It is also desirable that the data is listed based on the issued query, where the stored data can be loaded as a series of data by high-speed selection processing. Therefore, the DBINS Engine creates and index that is accessible to the stored data according to Leaf-ID, which is the ID query key. To make the stored data accessible by Leaf-ID, the DBINS Engine creates the index shown in Table 1. Next.ID Pointer is the next pointer in the ID list. The Share Number is the number of owners of the captured data. When several queries match only one packet, the information is recorded with this value. After the stored data is loaded by the selection operation, the Share Number is decremented, but the data is deleted if the Share Number becomes zero. In addition, the DBINS Engine manages the index of the captured data so it is scalable with an increase in queries, because the resources of the DBINS Co-processor are limited.

Table 1. Keys and Pointers in the Index

Keys and pointers	Contents
ID	Leaf-ID
Timestamp	Packet arrival time
Packet Size	Packet Size
Share Number	The number of owners of captured data
Prev. T-Pointer	Previous pointer of Time-list
Next. T-Pointer	Next pointer of Time-list
Next. ID-Pointer	Next pointer of ID-list

In this study, four different index structures are used. To optimize the data structure, these four index structures are focused on Leaf-ID as the primary key. Part of a stream matches multiple queries issued by different users, so it is preferable to combine the management information for each part. All four methods perform this optimization in a different way. Thus, four different index structures are implemented.

In the first approach, the DBINS Engine copies the overlapping parts of captured data and adds a Next.ID Pointer to all of the generated indexes (Fig. 2). This approach facilitates the selection and deletion of data, because the DBINS Engine does not manage the Share Number of the index. However, when the number of overlaps is large, the index creation throughput is degraded because the DBINS Engine has to make multiple indexes for each part of a data stream. It also causes a loss of memory efficiency because the same data is stored in several segments of the memory. Furthermore, to support newly extracted data arrival the DBINS Engine copies data, but it must have a buffer that stores the data until the copying is finished.

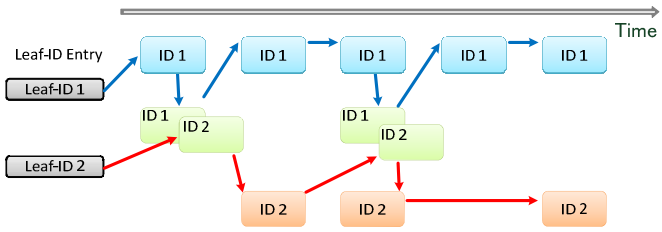


Figure 2. Approach 1

In the second approach, the DBINS Engine adds new IDs based on a combination of matching queries (Fig. 3). To insert the captured data, the DBINS Engine manages two tables. One manages the correspondence of the Leaf-IDs and the new IDs. The other manages the memory area of new IDs used in the IMDB. Using this approach, the DBINS Engine can generate an index at a constant rate and it has the advantage that there is no need to manage a large queue, unlike approach 1. However, the data management and index table management becomes complicated. Approach 2 has

another drawback. If the number of issued query increases, the table size increase according to $O(2^n)$ in the worst case. Moreover, when a new query is issued, the DBINS Engine must refresh the entry table.

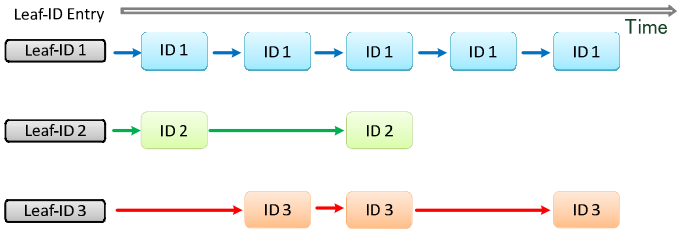


Figure 3. Approach 2

In the third approach, the DBINS Engine adds several Next.ID pointers to an index (Fig. 4). Using this approach, the DBINS Engine can improve the efficiency of memory usage compared with approach 1, while it can decrease the cost of managing the index table compared with approach 2. However, the index generation rate depends on the Share Number of the captured data. Therefore, when the overlap of a query is large, the index generation throughput is degraded. Using this approach, the number of Next.ID Pointers is limited by the reserved space because the BDISN Engine only reserves enough space for a fixed number of Next.ID Pointers.

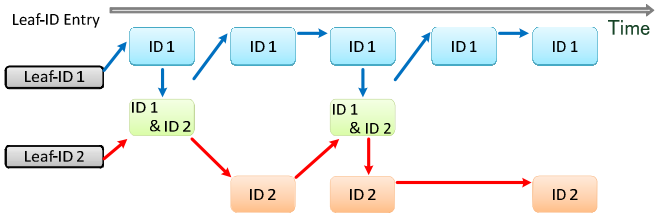


Figure 4. Approach 3

The fourth approach also adds several Next.ID Pointers to an index. The difference from approach 3 is that the DBINS Engine stores the Next.ID Pointers in the IMDB. Using this approach, the DBINS Engine does not require a large space for the Next.ID Pointers in the index memory and it can support a larger Share Number than approach 3. However, DBISN Engine has to access the IMDB whenever it creates an index, so the throughput is worse than approach 3.

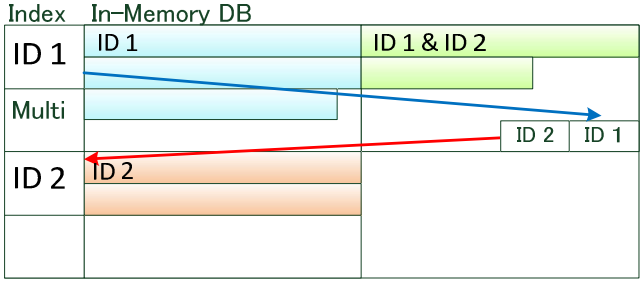


Figure 5. Approach 4

Each processing cycle for the above approaches is shown in Fig. 6. The requirements for throughput, selection speed, or the memory utilization ratio depend on the application or services. Therefore, the application or service providers must choose an index structure that meets their requirement.

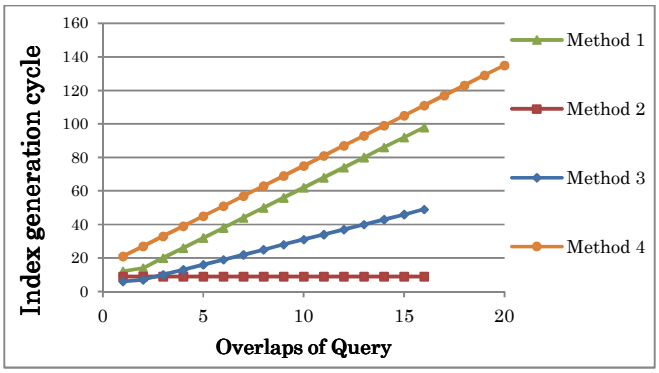


Figure 6. Processing Cycle

4.2 Implementation

The architecture of the DBINS Engine is shown in Fig. 7. The DBINS Engine has three blocks, Leaf-ID Module, Indexing Module, and IMDB Module. The Leaf-ID Module manages the Leaf-ID, which is added according to the query. The Indexing Module makes and manages the index. The IMDB Module stores the captured data in the IMDB. The procedure for storing the captured data is as follows. First, the header of the captured packet data is stored in the queue of the Leaf-ID Module while the body is stored in the queue of the IMDB Module. The Leaf-ID Module then checks the header information, which contains the Leaf-ID, timestamp, and index structure. The Leaf-ID Module creates a Leaf-ID Entry, which contains the head address and tail address, or it loads the leaf-ID Entry information according to the header. Next, the Leaf-ID Module gets a free address from the Indexing Module, updates the Leaf-ID Entry, and sends the writing address to the IMDB Module. The Indexing Module generates an index using the Leaf-ID, timestamp, and index structure. The throughput of the Indexing module is mainly dominated by memory access so, to eliminate the memory access delay, the Indexing Module has three dedicated registers, i.e., old_ifreehead for Prev.T-Pointer, current_ifreehead for the address to write, and next_ifreehead for Next.T-Pointer. When the IMDB Module receives the writing address from the Leaf-ID Module, the IMDB Module stores it in the IMDB.

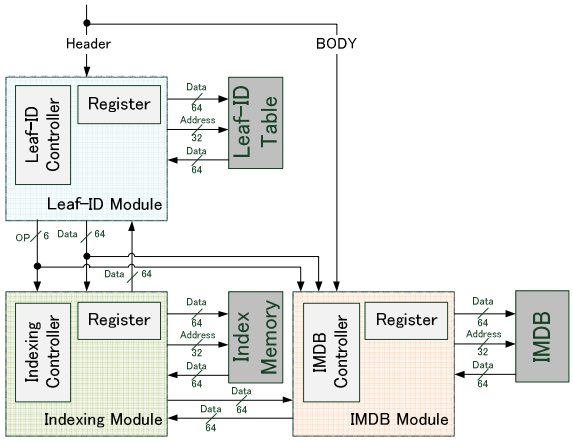


Figure 7. Architecture of the DBISN Engine

Table 2. Results of Synthesis

	Scale of Circuit	Latency (ns)	Max Frequency (MHz)
ISE Design Suite	# of Slices 475	3.73	318
Synopsys Design Compiler	(μm^2) 2.95 × 103	1.97	508

Table 3. Memory Parameters

Memory standard	On-chip memory	Off-chip memory (QDR SRAM)	
		FPGA	ASIC
WL (cycle)	1	3	8
RL (cycle)	1	3	8

5 Evaluation

In this section, we evaluated the scale of the circuit and the throughput of the DBINS Engine. The DBINS Engine was implemented in Verilog HDL and it was synthesized using Xilinx ISE Design Suite 12.3 on FPGA (Vertex5 XC5VLX330T). For comparison, we used Synopsys Design Compiler 2005.09 by FREEPDK45n Technology as ASIC (Application Specific Integrated Circuit) [6]. The scale of the circuit, latency, and maximum frequency are shown in Table 2. The throughput of the DBINS Engine was calculated based on the synthesis result. The memory parameters are shown in Table 3. The throughput of the DBISN Engine is shown in Table 4 (synthesized using ISE Design Suite) and Table 5 (synthesized using Synopsys Design Compiler). Where an off-chip memory was used as the index memory, equation (1) and (2) were used to calculate the cycle time. Table 4 and Table 5 show the available network throughput in two cases. In case 1, we assume the continuous storage of 50-byte packets, which was assumed to be the minimum size of network traffic. In case 2, we assume the storage of an average HTML packet size in the Internet backbone traffic captured by the WIDE MAWI project [7] between 14:00 and 14:15 on July 12th in 2009. The traces of the anonymous filter can be downloaded from the MAWI site. We used special trace data that was originally captured in the project and not filtered.

When the overlap of queries was four, this showed that the available throughput was 8.25 Gbps with a 50-byte packet size and 17.1 Gbps with 1306-byte packets, when the DBINS Engine was synthesized on FPGA. However, the throughput was 27.0 Gbps with a 50-byte packet size and 55.9 Gbps with a 1306-byte packets size, when the DBINS Engine was synthesized in ASIC. Therefore, the DBINS Engine could fulfill the metro network or edge network wire speed requirements when the overlap was sufficiently small.

$$\text{Cycle time} = (\text{RL} \times \text{memory read count} + \text{WL} \times \text{memory write count} + \text{cycle time}) \times \text{Latency} \quad \cdots (1)$$

$$\text{Avairable network throughput} = \frac{\text{total packet size}}{\text{cycle time}} \quad \cdots (2)$$

Table 4. Available Network Throughput using FPGA

overlaps	Available Network Throughput (Gbps)			
	On-chip Memory		Off-chip Memory	
	50-byte	1306-byte	50-byte	1306-byte
1	13.4	4.3	17.1	17.1
2	13.4	3.8	17.1	17.1
3	10.7	3.2	17.1	17.1
4	8.25	2.8	17.1	17.1

Table 5. Available Network Throughput using ASIC

overlaps	Available Network Throughput (Gbps)			
	On-chip Memory		Off-chip Memory	
	50-byte	1306-byte	50-byte	1306-byte
1	43.9	6.4	55.9	55.9
2	43.9	5.6	55.9	55.9
3	35.1	4.8	55.9	55.9
4	27.0	4.2	55.9	55.9

6 Summary

In this study, we proposed a hardware architecture to support high-speed database insertion into a router on the Internet. We implemented it in FPGA and ASIC. We showed that the throughput of the proposed architecture satisfied the wire-rate processing of a metro network with a throughput of

8.25 Gbps with 50-byte packets and 17.1 Gbps with 1306-byte packets. It also provided the requisite throughput of 27.0 Gbps with 50-byte packets and 55.9 Gbps with 1306-byte packets when the DBINS Engine was synthesized in ASIC.

7 Acknowledgments

This work was partially supported by the VLSI Design and Education Center (VDEC), the University of Tokyo, Japan, in collaboration with Synopsys Inc., National Institute of Information and Communications Technology (NICT), and a Grant-in-Aid for Scientific Research (C) (22500069).

8 References

[1] A. Arasu, S. Babu and J. Widom, “The cql continuous query language: Semantic foundations and query execution”, VLDB Journal, 15, 2, (2006).

[2] A. Arasu, S. Babu and J. Widom, “The cql continuous query language: Semantic foundations and query execution”, VLDB Journal, 15, 2 (2006).

[3] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. “TelegraphCQ: Continuous Dataflow Processing for an Uncertain World.” In Proc. of SIGMOD, pp. 668 – 668, (2003).

[4] Yuan Wei, Sang H. Son, John A. Stankovic, “RTSTREAM: Real-Time Query Processing for Data Streams”, Proc. of ISORC ’06, pp. 141-150 (2006).

[5] Oracle TimesTen In-Memory Database - data sheet. Technical report, ORACLE JAPAN, July 2007.

[6] FreePDK, http://www.eda.ncsu.edu/wiki/FreePDK

[7] WIDE MAWI project, http://mawi.wide.ad.jp/mawi/

SESSION

DEVELOPING HETEROGENEOUS COMPUTING SYSTEMS - MULTICORE, CPU + FPGA

Chair(s)

Prof. Pascal Benoit
University of Montpellier
France

From Streaming Models to FPGA Implementations

ERSA'12 Industrial Regular Paper

Hugo Andrade, Jeff Correll, Amal Ekbal, Arkadeb Ghosal, Douglas Kim, Jacob Kornerup, Rhishikesh Limaye, Ankita Prasad, Kaushik Ravindran, Trung N. Tran, Mike Trimborn, Guoqiang Wang, Ian Wong, Guang Yang
National Instruments Corportation, USA.

***Abstract*—Application advances in the signal processing and communications domains are marked by an increasing demand for better performance and faster time to market. This has motivated model-based approaches to design and deploy such applications productively across diverse target platforms. Dataflow models are effective in capturing these applications that are real-time, multi-rate, and streaming in nature. These models facilitate static analysis of key execution properties like buffer sizes and throughput. There are established tools to generate implementations of these models in software for processor targets. However, prototyping and deployment on hardware targets, such as FPGAs, are critical to the development of new applications. FPGAs are increasingly used in computing platforms for high performance streaming applications. Existing tools for hardware implementation from dataflow models are limited in their capabilities. To close this gap, we present DSP Designer, a framework to specify, analyze, and implement streaming applications on hardware targets. DSP Designer encourages a model-based design approach starting from a Parameterized Cyclo-Static Dataflow model. The back-end supports static analysis of execution properties and generates implementations for FPGAs. It also includes an extensive library of hardware actors and eases third-party IP integration. Overall, DSP Designer is an exploration framework that translates high-level algorithmic specifications to efficient hardware. In this paper, we illustrate the modeling, analysis, and implementation capabilities of DSP Designer. Through a detailed case study, we show that DSP Designer is viable for the design of next generation signal processing and communications systems.**

I. Introduction

Dataflow models are widely used to specify, analyze, and implement multi-rate computations that operate on streams of data. The Static Dataflow (SDF) model of computation is well-known for describing signal processing applications [1]. An SDF model is a graph of computational actors connected by channels that carry streams of data. The semantics require the number of data tokens consumed and produced by an actor per firing be fixed and pre-specified. This guarantees decidability of key execution properties, such as deadlock-free operation and bounded memory requirements [2].

Over the years, several extensions of SDF have been developed that improve the expressiveness of the model while preserving decidability, such as Cyclo-Static Dataflow (CSDF) [3], Parameterized Static Dataflow (PSDF) [4], Heterochronous Dataflow (HDF) [5], Scenario-Aware Dataflow (SADF) [6], and Static Dataflow with Access Patterns (SDF-AP) [7]. Complementing these modeling advances, algorithmic solutions for static analysis have been studied in depth. Viable techniques have been developed for computation of throughput, buffer sizes, and schedules [2] [8] [9].

The expressiveness of dataflow models in naturally capturing streaming applications, coupled with formal compile time analyzability properties, has made them popular in the domains of multimedia, signal processing, and communications. These high level abstractions are the starting points for model-based design approaches that enable productive design, fast analysis, and efficient correct-by-construction implementations. Ptolemy II [10], LabVIEW [11], and Simulink [12] are examples of successful tools built on the principles of model-based design from dataflow models.

These tools predominantly deliver software implementations for general purpose and embedded processor targets. However, ever-increasing demands on performance of new applications and standards have motivated prototyping and deployment on hardware targets, such as Field Programmable Gate Arrays (FPGAs). FPGAs are integral components of modern computing platforms for high performance signal processing. Surprisingly, few studies have been directed to the synthesis of efficient hardware from dataflow models.

The configurability of FPGAs and constraints of hardware design bring unique implementation challenges and performance-resource trade-offs. FPGAs permit a range of implementation topologies of varying degrees of parallelism and communication schemes. Fine-grained specification of actor execution at the cycle level enables execution choices between fully specified static schedules and more flexible self-timed schedules. Communication between actors could be through direct wires, handshake protocols, shift registers, shared registers with scheduled access, or dedicated FIFO buffers. Each mechanism poses different requirements on the interface and glue logic to stitch actors. Finally, a key requirement for hardware design is the integration of pre-created configurable intellectual property (IP) blocks. Hardware actor models must capture relevant variations in data access patterns and execution characteristics of different configurations.

We address these challenges with DSP Designer, a framework for hardware-oriented specification, analysis, and implementation of streaming dataflow models. The intent is to enable DSP domain experts to express complex applications and performance requirements in algorithmic manner and to auto-generate efficient hardware implementations. The main components of DSP Designer are: (a) a graphical specification language to design streaming applications, (b) an analysis engine to validate the model, select buffer sizes and optimize resource utilization to meet throughput constraints, and perform other pertinent optimizations, and (c) implementation support to generate an efficient hardware design and deploy it on Xilinx FPGAs. The specification is based on the Parameterized Cyclo-Static Dataflow (PCSDF) model of computation, which

is a sufficiently expressive model for wireless communications applications [13] [14]. DSP Designer provides an extensive library of math and signal processing functions that harness the resource elements on the FPGA. It also facilitates integration of custom-designed hardware blocks and third-party IP into the design. The back-end eases exploration of design trade-offs and translates a high level algorithmic specification to an efficient hardware implementation. Thus, DSP Designer simplifies the creation of complex streaming applications targeted for FPGA deployment.

In this paper, we highlight salient features of DSP Designer and illustrate a design flow to implement streaming applications. We then present a case study on the deployment of an Orthogonal Frequency Division Multiplexing (OFDM) wireless communication link from the Long Term Evolution (LTE) [15] mobile networking standard on a Xilinx FPGA.

II. Related Work

Synthesis flow from Register Transfer Level (RTL) logic and behavioral languages (typically C, C++, or SystemC) for hardware targets has been a popular topic of several studies. However, there is limited prior art on hardware generation from non-conventional high level models like dataflow. Ptolemy II is a prominent academic framework for graphical specification and analysis of dataflow models [10]. While these tools provide some support for RTL generation from restricted models, the focus is more on proof-of-concept and less on optimized hardware implementation.

On the commercial front, LabVIEW FPGA from National Instruments is a popular tool that supports FPGA deployment from dataflow models [16]. However, LabVIEW FPGA only supports the Homogeneous Static Dataflow (HSDF) model of computation, which does not allow native specification of streaming multi-rate computations. System Generator from Xilinx is another related offering that supports FPGA implementations of synchronous reactive and discrete time models of computation [17]. However, these models are not suitable for data driven streaming specifications. SystemVue ESL from Agilent supports more expressive dataflow models and provides libraries and analysis tools for the RF and DSP domains [18]. However, it primarily serves as an exploration and simulation environment, and does not offer a path to implementation in hardware.

The closest effort in synthesizing hardware from dataflow programs is the Open Dataflow framework [19]. The CAL actor language supported in Open Dataflow is an important step in formalizing actor and interface definitions. It has been adopted by the MPEG Video Coding group to develop codecs for future standards [20]. CAL builds on the Dynamic Dataflow model of computation but this model is undecidable and cannot be subject to static analysis. In contrast, the PCSDF model used by DSP Designer enables analysis of deadlock-free execution, and memory and throughput requirements. Also, CAL is a textual specification language, whereas DSP Designer provides an intuitive graphical design environment.

In summary, DSP Designer is an attempt to integrate the respective strengths of the previously discussed tools into a unified framework for hardware implementation. The graphical design environment is intended for algorithm designers who are generally not experts in hardware design. The framework supports analysis capabilities relevant to hardware implementation and includes an extensive library of common math, signal processing, and communications functions. It also enables easy integration of IPs from native and third-party libraries, like the Xilinx CoreGen library [21], which are essential to practical efficient hardware design.

III. Model Specification and Analysis

The foundation of DSP Designer is its models of computation – SDF, CSDF, and their parameterized extensions. We discuss the relevant characteristics of these models, and illustrate their suitability for specifying signal processing applications.

A. SDF and CSDF

A dataflow model consists of a set of actors inter-connected via channels. The actors represent computational units while the channels denote communication. The data is abstracted as tokens. In the *Static Dataflow* (SDF) model of computation, at each firing, an actor consumes a fixed number of tokens from each input channel, and produces a fixed number of tokens on each output channel. The channels store the tokens until an actor consumes the tokens.

Each actor is associated with an *execution time* and an *initiation interval*. Execution time is the time (in clock cycles) that the actor needs to process inputs, perform computation, and generate outputs. Initiation interval is the minimum time (in clock cycles) between consecutive firings of an actor. If initiation interval is less than execution time for an actor, then the actor may fire in an overlapping (pipelined) fashion.

Fig. 1 shows an SDF model for computing the standard deviation of non-overlapping blocks of 100 input samples each. Every actor in this model except `Sum` is single-rate or homogeneous, i.e. it consumes 1 token on every input, and produces 1 token on every output. The `Sum` actor consumes 100 input tokens and produces their sum as a single output token. Execution times of the actors vary with their implementations. `Square`, `Decrement`, `Subtract` execute in single cycle; `Divide`, `Square Root` take multiple cycles, and could be pipelined to have initiation interval of 1 cycle. `Sum` actor has execution time and initiation interval of 100.

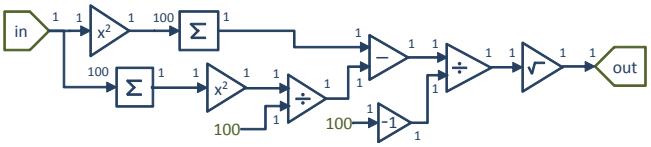


Fig. 1. Computing standard Deviations of input blocks of fixed size 100.

The SDF model of computation fits well with fixed-length computations. Such computations are abundant in signal processing standards, for example, the processing of 8×8 blocks of pixels during JPEG encoding. However, there are also computations that follow a fixed cyclic pattern in the number

of tokens processed. An example is the *normal CP* mode of LTE OFDM standard in which every slot has 7 symbols, with the first special symbol different in length from the other symbols. For such computations, the *Cyclo-Static Dataflow* (CSDF) model of computation generalizes SDF by allowing the number of tokens consumed or produced by an actor to vary according to a fixed cyclic pattern [3]. Each firing of a CSDF actor corresponds to a phase of the cyclic pattern. In Fig. 1, if we replace the input token count of `Sum` actor by a cyclic pattern $(100, 200, 300)$, then we get a CSDF model that computes standard deviation of input blocks whose lengths vary deterministically from 100 to 200 to 300 and back.

The SDF and CSDF models of computation permits efficient static analysis of key properties. The absence of deadlocks (i.e., the ability of each actor to fire infinitely often), and the consistency of execution rates (i.e., the ability to execute infinitely with bounded channels) can be verified efficiently [1] [2] [3]. Further, there are efficient algorithms for computation of throughput and buffer sizes [8], [9].

B. Parameterized Extensions

SDF and CSDF are static in nature. However, for many applications, the number of tokens processed needs to vary at run-time. For example, MP3 audio compression selects at run-time between long blocks of 576 samples and short blocks of 192 samples. Fig. 2 shows a variation of the model in Fig. 1, in which the `Sum` actors consume N tokens in each firing, where N is from the set $\{100, 200, 300\}$. This computes standard deviations of a mix of input blocks of lengths 100, 200 or 300 by varying N at run-time. This model of computation is called *Parameterized Static Dataflow* (PSDF) [4].

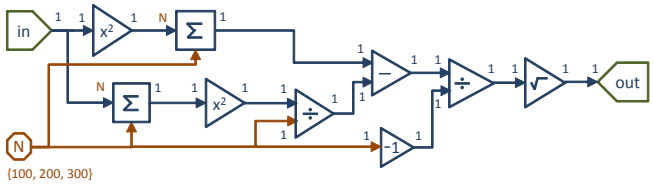


Fig. 2. Computing standard deviations of input blocks of varying size N.

The behavior of the PSDF model can be viewed as a composition of several SDF models, one for each possible value of the parameter (also referred to as configuration). Fig. 2 has 3 possible values of the parameter, hence 3 configurations. At any point in execution, the behavior of the PSDF model is the SDF model corresponding to the value of the parameter. To avoid non-determinism, a change in parameter value can take effect only at iteration boundaries. The analysis of a PSDF model accounts for the analysis for all possible configurations [4]. The CSDF model can similarly be parameterized to form the *Parameterized Cyclo-Static Dataflow* (PCSDF) model.

IV. Realizing Models in DSP Designer

DSP Designer is a graphical environment backed by the design and implementation flow of Fig. 3. In this section we describe how the user specifies applications using models, explores optimizations, and generates FPGA designs.

A. Design Flow

The user works in a graphical environment as shown in Fig. 4. The starting point is the *Application*, e.g. a DSP algorithm, which the user starts drawing by selecting actors from the *Actor Library* and placing them on the editor canvas. This begins the *Model Specification* step. The actor library consists of a rich set of primitive actors (add, square root, sine, etc.), stream manipulation actors (upsample, build stream, etc.), third-party actors (e.g. FFT and FIR blocks from Xilinx Coregen [21]), and user-defined actors that are either specified in the LabVIEW programming language or constructed using DSP Designer. This reuse of actors allows for hierarchical composition of designs within the tool.

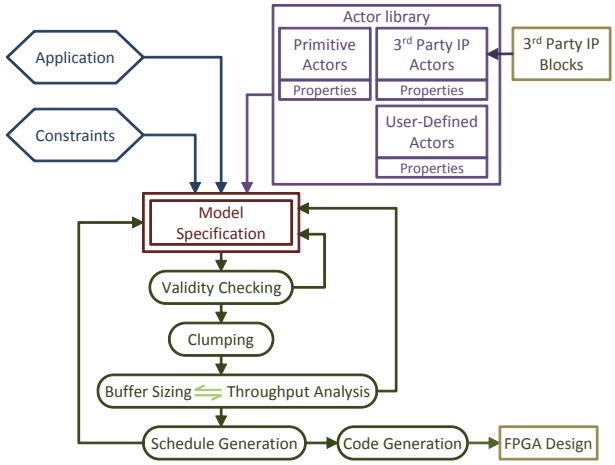


Fig. 3. Design and Implementation Flow in DSP Designer.

The user continues by connecting the actors, and optionally configuring their properties. Configurable properties of an actor include the data types and the number of tokens for its input and output channels. The number of tokens may vary at run-time for parameterized actors, depending on the current parameter value, resulting in a potentially distinct configuration for each parameter value. To ensure analyzability, the tool limits the value of each parameter to a finite set specified by the user. Some actors can also be configured for their throughput, pipeline depth, resource usage, or other implementation-specific options. The actor library includes cycle-accurate characteristics for each actor configuration, including the initiation interval and the execution time.

The second input from the user is the *Constraints*, which include minimum throughput requirements on input/output ports or internal channels of the design. Throughput is specified in engineering units, such as Mega-Samples per second (MSps).

The tool performs several types of analysis on the design in the background while the user is constructing it, with immediate feedback on the current state of the design. *Validity Checking* includes model consistency and deadlock checking. It also performs automatic type checking and type propagation across the design. Errors or warnings are immediately annotated on the offending nodes on the canvas and reported under the Errors & Warning tab in the tool. On a valid design, the

tool performs *Clumping* to identify regions that fit specialized implementation schemes. *Buffer Sizing* and *Throughput Analysis* are then performed on the design. This determines the buffer sizes required on the channels to satisfy user constraints such as minimum throughput. If the constraints cannot be met, the tool reports errors. *Schedule Generation* establishes a valid, cycle-accurate schedule for the design, given the determined buffer sizes and clumped regions. This schedule is viewable in the schedule view part of the tool (as shown at the bottom of Fig. 4), providing instant feedback on the run-time behavior of the design, including the achievable throughput.

The user can simulate the functional behavior on the development platform before invoking the hardware implementation stage. As a part of the simulation, the user can specify stimulus data and add graphical displays to the design to visualize the response on output ports or on any wire in the design.

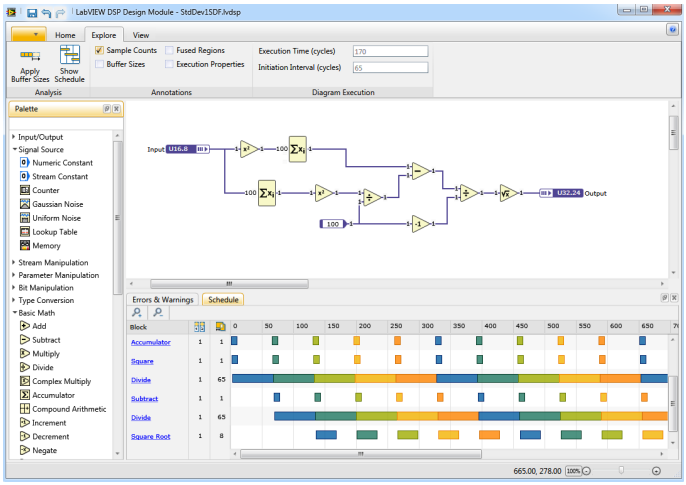


Fig. 4. DSP Designer Tool Implementing the Example in Fig. 1.

The final step is *Code Generation* that uses the results of analysis to emit an FPGA design in the form of synthesizable LabVIEW files. The tool can also generate a synthesizable testbench that allows the user to stimulate the design from the development computer and compare the response to validated signals. The testbench includes the necessary code for DMA communication between the FPGA device and the development computer. The LabVIEW files can be used to generate a bitfile used to implement the design on Xilinx FPGA devices or for timing-accurate hardware simulation. Currently the tool supports targeting Virtex 5 devices from Xilinx.

B. Implementation Strategy

DSP Designer uses a FIFO-based, self-timed implementation strategy to realize the designs on FPGA fabrics [8]. In the FIFO-based strategy every channel in a model is conceptually mapped to a hardware FIFO of appropriate size and every actor is mapped to a dedicated hardware block that implements its functionality. There is no resource sharing among two different channels or two different actors in the current state of the tool, but the model does not preclude this. In the self-timed execution strategy every actor instance is fired whenever it has a sufficient number of tokens on each of its input channels, sufficient number of vacancies on each of its output channels, and the initiation interval of the previous firing has expired. This evaluation is done on every clock cycle, allowing for a potentially more opportunistic execution than the conservative block-based model used in most SDF-based tools, where a downstream actor is not fired until a cycle after the one where its upstream actors write the last output token into their shared buffers. As a consequence, there is no global scheduling logic in this implementation strategy, reducing the complexity of the controller for each actor in the final design.

C. Actor and IP Stitching

The FIFO-based, self-timed implementation strategy is implemented using harness logic that surrounds every actor instance, providing a FIFO-based interface that realizes the SDF model and its extensions discussed in Section III. The generated code for all actors presents a standardized interface to the harnesses, based on designated lines for data and handshaking. This simplifies actor stitching since the tool can use generic harness wrapper templates. It also allows the tool to connect actors more directly and efficiently.

A faithful realization of the SDF model of computation requires extra resources for the harness logic and the FIFOs on each channel. In the synthesized design this overhead can be significant compared to the resource usage of the actors themselves. To reduce this overhead the tool applies a series of *clumping* transformations on the design to reduce both the number of harnesses and FIFOs in the design. These transformations preserve the observable flow of tokens on the input and output ports, while preserving or increasing throughput. The clumping activity is akin to the process of converting an asynchronous design, where all actors are connected by FIFOs, into a *GALS* [22] (Globally Asynchronous Locally Synchronous) architecture, where FIFOs connect regions of synchronously connected actors called clumps.

V. OFDM Transmitter & Receiver Case Study

In this section, we present a case study on the design and implementation of a real-time single antenna OFDM transmitter and receiver using DSP Designer.

A. System Specifications & Hardware Architecture

Our single antenna OFDM link design is based upon the LTE standard [15] with system specifications that include a transmission bandwidth of 5 MHz, 7.68 MSps sampling rate, 512 FFT length, 128 cyclic prefix (CP) length (extended mode), 250 data subcarriers, 50 reference subcarriers, and variable 4/16/64 Quadrature Amplitude Modulation (QAM). The proposed communication system is implemented on the National Instruments (NI) PXI Express platform shown in Fig. 6, where the transmitter (TX) and receiver (RX) consist of the following four main components.

- **PXIe-8133** Real-time (RT) controller equipped with a 1.73 GHz quad-core Intel Core i7-820 processor and 8 GB of dual-channel 1333 MHz DDR3 RAM.

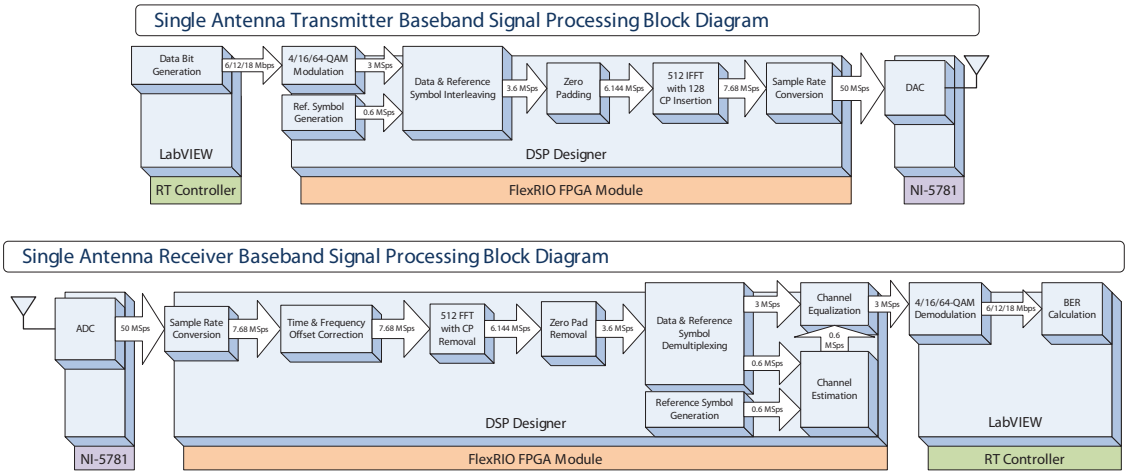


Fig. 5. Hardware and Software Mapping of Transmitter and Receiver Block Diagrams.

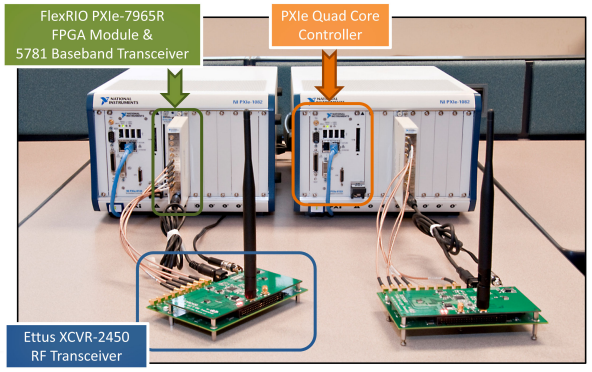


Fig. 6. National Instruments PXI Express Real-Time Signal Processing Platform with Ettus Research RF Front-End.

- **PXIe-7965R** FPGA module equipped with a Virtex-5 SX95T FPGA optimized for digital signal processing, 512 MB of onboard RAM, and 16 DMA channels for high-speed data streaming at more than 800 MBps.
- **NI-5781** 40 MHz baseband transceiver module equipped with dual 100 MSps 14-bit inputs, dual 100 MSps 16-bit outputs, and eight general purpose IO lines.
- **Ettus Research XCVR-2450** 802.11a/b/g compliant, 40 MHz, dual 2.4 GHz and 5.2 GHz band RF transceiver with 100 mW of transmit power.

Fig. 5 shows the TX and RX block diagram representations of the various signal processing blocks implemented in the devices. Also shown is a mapping of the various blocks to the underlying hardware targets and the respective design tools used in their implementation; e.g., the TX *Data Bit Generation* block (programmed using LabVIEW RT) executes on the PXIe-8133 RT controller, while the higher rate *512 IFFT with 128 CP Insertion* block (implemented using DSP Designer) executes on the PXIe-7965R FPGA module. The various data rates associated with the inputs and outputs of each block are also shown; e.g., the TX *Sample Rate Conversion* block up-samples input data streaming at 7.68 MSps to 50 MSps in order to meet the sample rate constraints of the NI-5781 DAC.

B. OFDM Transmitter Design Overview

Fig. 7 shows the DSP Designer implementation of the proposed transmitter. Random bytes of data generated by the RT controller are forwarded to the FPGA module for Multilevel QAM (M-QAM) [23]. Depending upon the modulation order value denoted by the parameterization port, Modulation, the bytes of data are unpacked into groups of 2, 4, or 6 bits corresponding to 4/16/64-QAM, respectively. Groups of bits are then mapped to their respective complex symbols and passed out of the output port of the sub-diagram.

After QAM modulation, 250 data symbols are interleaved with 50 reference symbols stored in a look-up table forming an array of 300 interleaved symbols which is then split into two equal groups and padded with zeros forming an array of 512 samples. The 512 samples are passed through a 512 point IFFT block translating the frequency domain samples into the time domain. A 128 point CP is also inserted such that the output of the block consists of 640 samples streaming at 7.68 MSps. Sample rate up-conversion is then performed through two sets of FIR filters, converting the 7.68 MSps signal to 50 MSps. The samples are forwarded to the NI-5781 for digital-to-analog conversion followed by RF up-conversion.

C. OFDM Receiver Design Overview

Fig. 7 shows the DSP Designer implementation of the receiver. The RX begins with two FIR filters that perform sample rate down-conversion taking the incoming 50 MSps signal from the ADC down to 7.68 MSps. Time and carrier frequency offset (CFO) estimation is performed using the blind estimation technique proposed in [24]. Because the first L samples (CP) and the last L samples of an $N+L$ length OFDM symbol are equal, the algorithm correlates the two, thereby eliminating the need for a priori knowledge of the transmitted signal. The correlation output is then used to estimate the start index of an OFDM symbol and the CFO thereof.

In order to meet throughput without loss of data during the estimation of the start index and CFO, the receive signal is buffered into a memory block while simultaneously being

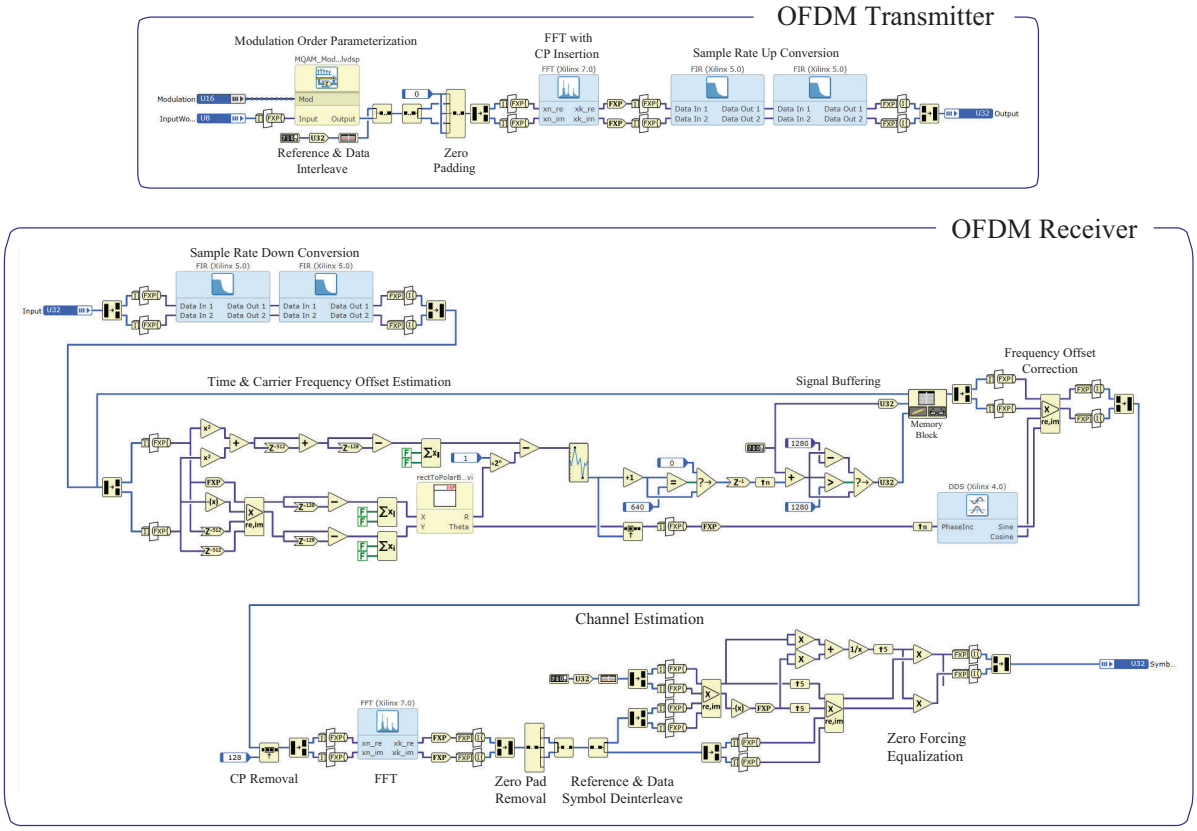


Fig. 7. DSP Designer Diagrams of OFDM Transmitter (top) and Receiver (bottom).

processed for time and CFO estimation. Once computed, the start index is used to calculate a corresponding read address pointer that indexes the beginning of an OFDM symbol stored in memory. When it is synchronized to the beginning of an OFDM symbol, the RX streams the signal out of memory and into the CFO correction block wherein the synchronized OFDM symbol is multiplied with a complex sinusoid generated by a direct digital synthesizer (DDS) block at a frequency defined by the CFO estimate present at its input.

After CFO correction, the received OFDM symbol is passed on for CP removal and FFT transformation returning the signal to the frequency domain. Zero pads are then removed and the reference and data symbols are separated in a deinterleave operation. As shown in Fig. 7, the received reference symbols are passed out of the first output of the deinterleave block for channel estimation while the received data symbols are passed out of the second for channel equalization.

In order to estimate the channel coefficients, we model the received reference symbols as $s_k = h_k r_k + z_k$ where $k \in \{0, \ldots, 49\}$. The reference symbol and channel coefficients are respectively modeled as $r_k = e^{j\theta_{r_k}}$ and $h_k = |h_k| \, e^{j\theta_{h_k}}$. Lastly, z_k represents additive white Gaussian noise.

The estimates of the channel coefficients, $\hat{h}_k$, are then calculated by multiplying the complex conjugate of the reference symbols, r_k^*, to the received reference symbol. Moreover, because only one reference symbol is allocated to every five data symbols, the 50 channel estimates, $\hat{h}_k$ for $k \in \{0, \ldots, 49\}$, are up-sampled by five generating a total of 250 channel estimates, $\hat{h}_i$ for $i \in \{0, \ldots, 249\}$.

In order to correct the effects of the wireless channel, zero forcing (ZF) channel equalization is employed where the received data symbols, y_i, are first multiplied by the complex conjugate of the channel estimates, $\hat{h}_i^*$, and then divided by their square magnitude, $|\hat{h}_i|^2$, effectively inverting the channel. The data symbol estimates, $\hat{x}_i$, are then transferred to the RT controller at a data rate of 3 MSps for QAM demodulation and bit error rate calculation.

D. FPGA Compilation & Run-Time Results

In addition to the portions of the design implemented in DSP Designer, the compilation results include nominal logic implemented in LabVIEW FPGA that manages data transfer across the NI-5781 baseband transceiver and PXIe-7965R FPGA module, and the PXIe-7965R FPGA module and PXIe-8133 RT controller. The results also include additional logic to control the NI-5781, such as ADC/DAC read/write operations, sampling frequency configuration, and clock select.

Table I is a summary of the compiled FPGA resource utilization. The first two columns show the various resources available on the PXIe-7965R's Virtex-5 SX95T FPGA and the total number of elements associated with each resource. The percentage utilization of the various resources for the TX and RX are listed in the last two columns. For instance, there are 14,720 slice elements available on each FPGA, 43.1% or 6,350 of which are used by the TX and 79.2% or 11,659 of

which are used by the RX. Due to significant differences in computational complexity between the two designs, the RX utilizes more than twice as many slice registers and LUT resources compared to the TX. With regard to timing, the TX and RX DSP diagrams are configured to be driven by 125 MHz clocks, and both successfully met timing during compilation.

Resource Name	Available Resource Elements	Transmitter Utilization	Receiver Utilization
Slices	14,720	43.1%	79.2%
Slice Registers	58,880	21.6%	54.6%
Slice LUTs	58,880	24.7%	57.3%
DSP48s	640	2.7%	8.3%
Block RAM	244	8.2%	19.7%

TABLE I
FPGA RESOURCE UTILIZATION SUMMARY.

Fig. 8 is a screen shot of the OFDM receiver front panel taken during an over-the-air test of the communications link. In addition to carrier frequency, modulation order, and LNA gain controls, a sample 16-QAM signal constellation plot is shown along with two average bit error rate (BER) curves, one taken on a single subframe basis (lower right hand plot), and the other taken over all received subframes (upper right hand plot). The average BER over all subframes converges to an approximate value of $8 * 10^{-4}$.

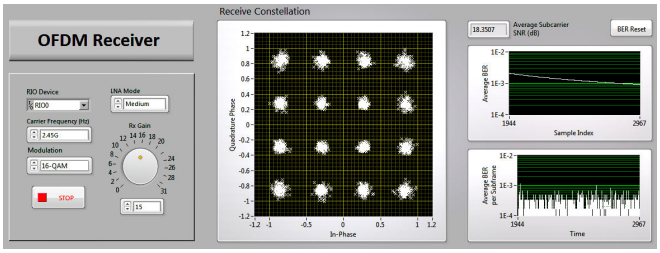


Fig. 8. Receiver Front Panel.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the DSP Designer framework to specify dataflow models, analyze them, and generate implementations for hardware targets. The PCSDF model of computation is sufficiently expressive in specifying complex streaming applications, while capturing characteristics specific to hardware design. The back-end performs key optimizations related to buffer sizing and scheduling. The actor library provides a rich set of building blocks to create complex signal processing and communications applications. It also facilitates easy integration of custom designed hardware IPs from native and third-party sources. Thus, DSP Designer serves as a design and exploration framework that enables algorithm experts to productively specify applications using high level models and still create efficient hardware implementations.

In the future we intend to (a) extend the DSP Designer modeling and analysis capabilities to support more expressive streaming models such as Heterochronous Dataflow (HDF); (b) enhance the analysis back-end to address problems related to dataflow pipelining and resource constrained scheduling; (c) study how intra-cycle timing optimizations for hardware, such as retiming and recycling, can be applied at the model level; (d) derive more resource-efficient hardware implementations through rate matching and clumping of multi-rate actors; and (e) enlarge the DSP Designer actor library and standardize the IP interface definition to ease third-party IP integration.

REFERENCES

[1] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept. 1987.
[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Norwell, MA: Kluwer Academic Press, 1996.
[3] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," in *IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, 1995, pp. 3255–3258.
[4] B. Bhattacharya and S. Bhattacharyya, "Parameterized Dataflow Modeling for DSP Systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408 –2421, oct 2001.
[5] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 6, pp. 742–760, June 1999.
[6] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk, "A Scenario-aware Data Flow Model for Combined Long-run Average and Worst-case Performance Analysis," in *Proceedings of MEMOCODE'06*, Jul. 2006, pp. 185–194.
[7] S. Tripakis, H. Andrade, A. Ghosal, R. Limaye, K. Ravindran, G. Wang, G. Yang, J. Kormerup, and I. Wong, "Correct and non-defensive glue design using abstract models," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '11. New York, NY, USA: ACM, 2011, pp. 59–68.
[8] O. M. Moreira and M. J. G. Bekooij, "Self-Timed Scheduling Analysis for Real-Time Applications," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 83710, pp. 1–15, April 2007.
[9] S. Stuijk, M. Geilen, and T. Basten, "Exploring Trade-offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs," in *Proceedings of DAC '06*, 2006, pp. 899–904.
[10] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - The Ptolemy Approach," in *Proc. of the IEEE*, vol. 91, no. 1, 2003, pp. 127–144.
[11] H. A. Andrade and S. Kovner, "Software Synthesis from Dataflow Models for G and LabVIEW," in *In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, 1998, pp. 1705–1709.
[12] The MathWorks Inc., "Simulink User's Guide," 2005, http://www.mathworks.com.
[13] H. Kee, C.-C. Shen, S. Bhattacharyya, I. Wong, Y. Rao, and J. Kornerup, "Mapping Parameterized Cyclo-static Dataflow Graphs onto Configurable Hardware," *Journal of Signal Processing Systems*, pp. 1–17, 2011.
[14] H. Berg, C. Brunelli, and U. Lücking, "Analyzing Models of Computation for Software Defined Radio Applications," in *International Symposium on System-on-Chip (SOC)*, Tampere, Finland, November 2008, pp. 1–4.
[15] "3GPP LTE: The Mobile Broadband Standard," Dec 2008, http://www.3gpp.org/.
[16] National Instruments Corp., "LabVIEW FPGA," www.ni.com/fpga.
[17] Xilinx Inc., *System Generator for DSP: Getting Started Guide*, www.xilinx.com.
[18] C.-J. Hsu, J. L. Pino, and F.-J. Hu, "A mixed-mode vector-based dataflow approach for modeling and simulating lte physical layer," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 18–23.
[19] J. W. Janneck, "Open Dataflow (OpenDF)," http://www.opendf.org/.
[20] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing Hardware from Dataflow Programs: An MPEG-4 Simple Profile Decoder Case Study," in *IEEE Workshop on Signal Processing Systems*, oct. 2008, pp. 287–292.
[21] Xilinx Inc., *Xilinx Core Generator, ISE 12.1*, Xilinx Inc., 2010.
[22] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford Univ., CA., October 1984.
[23] J. Proakis, *Digital Communications*, 4th ed. McGraw-Hill Science/Engineering/Math, Aug 2000.
[24] M. Sandell, J.-J. van de Beek, and P. O. Brjesson, "Timing and Frequency Synchronization in OFDM Systems Using the Cyclic Prefix," in *In Proc. Int. Symp. Synchronization*, 1995, pp. 16–19.

A Configurable VHDL Template for Parallelization of 3D Stencil Codes on FPGAs

ERSA'12 Distinguished Paper

Franz Richter, Michael Schmidt and Dietmar Fey
Department of Computer Science, Chair of Computer Science 3 - Computer Architecture,
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract— *2D and 3D stencil code applications are very common in scientific computing, but their performance is mostly limited by the memory bandwidth. Elaborate on-chip buffering techniques minimize memory transfers, but they cannot be directly realized on fixed general-purpose processors or GPUs.*

FPGAs instead offer flexibility regarding the processing scheme, the degree of parallelism and the numerical representation of values. This enables FPGA-based problem-solvers to perfectly scale from low-power embedded devices to high-performance accelerators with a much higher performance-to-power ratio than conventional processing nodes. To reach optimal performance, elaborate buffering techniques within the FPGA are necessary to avoid redundant memory access, first of all in 3D space.

We created a generic VHDL template to ease development of 3D stencil-based applications, using Full Buffering to minimize data transfers. The template allows a custom number format, together with variable parallelization in space and time. The parameters can be set according to the capabilities of the underlying hardware and the requirements of the application.

Keywords: Stencil Codes, FPGA, Full Buffering

1. Introduction

In numerical approximations a specific problem is often discretized in both, space and time. It is therefore mapped on a regular grid and computing one step in time for a single grid point at a certain position, referred to as *cell*, can be put down to a function of itself and its surrounding [1]. To perform one iteration, this function, also called *stencil*, is applied to every cell of the grid, while several thousand iterations are often needed for convergence of the results.

Applications of these stencil codes include 2D image preprocessing operations, which we already have analyzed in detail [2]. In high performance computing (HPC), 3D stencils are often used for physical simulation, like the heat dissipation with the *Jacobi Iterator* [3] or a particle behavior with *Lattice Gas* or *Lattice Boltzmann* methods [4].

These applications are very data-intensive, first of all for the 3D problem space. Using off-the-shelf hardware like CPUs and GPUs, stencil codes are mainly memory bound, which means that the external memory bandwidth is the bottleneck in the processing chain.

Fortunately, this limit can be shifted by using clever strategies for buffering and parallelization on FPGAs, because they can be strongly customized to the problem, instead of having to adapt the algorithm to a fixed architecture. Furthermore, even high-performance FPGAs have a moderate power consumption, in contrast to normal CPUs and GPUs, which allows HPC applications to be realized efficiently [5].

1.1 Buffering

To circumvent the performance limit on FPGAs, extensive use of on-chip memory is necessary to avoid redundant memory access and allow parallelization of computation. The high amount of resources of today's FPGAs enables hardware designers to use *Full Buffering (FB)* in favor of *Partial Buffering (PB)* [6].

For PB, only the data needed by the current computation is stored to minimize memory consumption. Fig. 1a shows a three-dimensional problem space of size $M \times N \times O$. Each block represents one value and its shade gives its current role in computation. The light gray blocks have neither been loaded nor processed yet. Values which are needed by the current stencil update are medium gray. Here, an $X \times Y \times Z$-wide area is used, which is called neighborhood. To perform the next update, $Y \cdot Z$ dark gray blocks are needed, and as much as data is now obsolete and can be evicted (the most left column of the neighborhood).

Hence, for a single computation, several values have to be loaded from external memory, limiting the resulting performance.

In contrast to that, the main idea of FB is to store data internally until all computation relying on it has been performed. In Fig. 1b, these additionally stored values are shaded differently, compared to PB. Data has still to be loaded to perform an update, but it depends no longer on the size of the stencil. Instead, only a single value is needed, since all other data is already present.

During computation, the stencil is applied row-wise, from the top to bottom and plane-wise from the front to the back. Thus, a value must have been used for computation at every

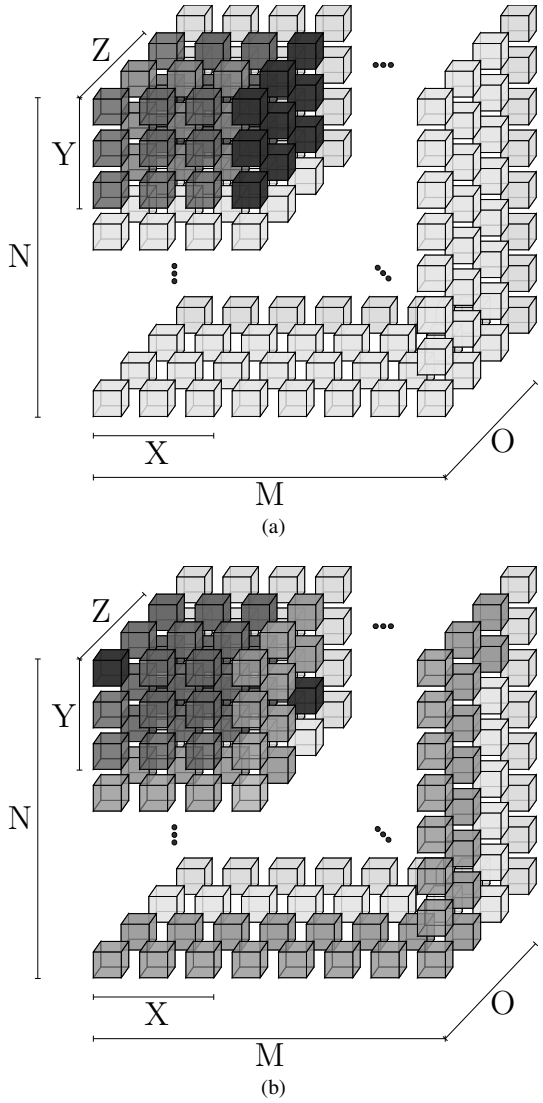


Fig. 1
3D PARTIAL BUFFERING (PB) (A) AND FULL BUFFERING (FB) (B)

position of the stencil, i.e $X \cdot Y \cdot Z$-times, until is can be evicted.

It is obvious that this is optimal regarding the utilization of external memory, at the expense of a high demand for local buffer space. However, this paper will show that FB is worth on modern FPGAs, since it is the only way to optimally minimize redundant memory access and it is furthermore predestined for stream processing, making it perfectly compatible with typical stencil codes.

The following sections give an overview of prior work concerning stencil codes on FPGAs and explain the motivation for our generic approach, before the template itself is explained.

1.2 Related Work

Stencil code applications on FPGAs are well discussed and several different architectures have been proposed.

Ref. [7] uses finite-difference schemes for real-time sound synthesis. They use a two-dimensional array of processing elements (PE), each containing an independent controller, internal memory (distributed or Block RAM) and a fixed-point arithmetical unit. Parallelism is achieved by using several PEs, which in turn can contain multiple and pipelined arithmetical units. All input data is spread over of the PEs, if enough on-chip space is available. Otherwise, external memory has to be used, limiting the total amount of PEs available since all PEs have to communicate simultaneously. For large inputs, it is not analyzed further how the memory bandwidth affects the overall performance.

The different degrees of parallelization are similar to ours, but at a different level of abstraction. We do not see any advantage of implementing multiple PEs, each with separate control logic and memory access, on a single FPGA. Instead, enlarging a single PE to the architectural limits significantly reduces the overhead for maintenance since a single controller is sufficient.

The authors of [8] [9] claim to have developed the first hardware accelerator card for the 3D finite-difference time-domain method. They build a memory hierarchy consisting of on-chip BlockRAM, coupled with on-board SRAM and DRAM, connected to PCI-bus of the host system. The BlockRAM is mainly used as input and output buffer to the external DRAM to allow bursts of data. Each arithmetical unit, of which more than one may exist for parallelization, accesses disjoint on-chip buffers and performs computation on single precision floating point numbers. They also independently exchange data with external memory, but we do not see multiple memory accesses in parallel with totally different addresses to be feasible.

In [10], an architecture for star-shaped stencil codes of size $n \times (n+1) \times n$, for even n, with single precision floating point data is proposed. It consists of the front-end, which is mainly responsible for external memory access and buffering of the data already loaded, and the back-end, implementing the actual update logic, running at twice the clock frequency of the front-end. The engines are controlled by the input FIFO and stalled if necessary. They rely on an extended PB, minimizing memory access along one axis. The paper emphasizes the speed of on-chip data transfers to maximize stencil throughput. This throughput depends on the clock frequency, the amount of BlockRAMs used for parallel data access and the degree of parallelization of computation, but we do not expect any of the three to be a bottleneck, despite of the external memory bandwidth, which is not primarily minimized in this approach.

The most recent publication [11] shows a similar approach to ours. They use an FB scheme for the 3D Reverse Time Migration algorithm, with the same argument as ours, that

the limitations by the external memory bandwidth is the main bottleneck in today's applications. They also cover different shapes of stencils, coming to the conclusion that compact ones are more suitable in FB-based applications due to less memory requirements, even if they have a higher computational intensity. Despite of our work, they do not give a detailed description of the hardware structure and the resource consumption for different configurations. Furthermore, they only consider 2D blocking to allow arbitrary problem spaces, though 1D blocking involves less redundancy, compared to 2D, as it is explained in Sec. 2.2.

1.3 Template

In contrast to the work listen above, our goal was to provide a flexible framework to ease the development of future 3D stencil-based streaming-applications on FPGAs, but offering the performance and transparency of a customized solution. This general approach is new to out knowledge. It is an advancement of our prior work on 2D stencil codes, for which we already have developed a VHDL template, firstly presented in [2].

The 3D template is based on the results of [12]. It can be adapted by different parameters, configuring the size of the problem space, the size of the stencil or the amount of bits per cell, to fit the application's needs. Furthermore, the degree of parallelization in both, space and time, can be adapted, according to the memory bandwidth and the FPGA-resources available. If the problem space is too large, partitioning may be necessary, imposing a certain degree of overhead. This scalability allows the template to be used in all ranges of complexity, from low-power embedded devices to HPC-accelerators, without having to bother with data handling.

The paper is structured as follows. The next section explains the components and parameters of our template. The third section presents some mathematical background, together with estimations and measurements of processing time and resource consumption. In the last section, we present a demonstrator design.

2. Template Design

The main goal of the template is to allow an optimal use of resources on an FPGA for a given problem, while taking full advantage of the external memory bandwidth. Stencil codes are usually very unbalanced regarding the amount of data required for a single update and the computational intensity. Therefore, they are often limited by memory bandwidth, first of all for 3D problems. Hence, an optimal buffering scheme is needed to optimize the overall performance.

2.1 Full Buffering for 3D Stencil Codes

FB reduces memory transfers to a minimum by keeping data inside of much faster on-chip memory as long as it is required for computation. On FPGAs, a limited amount of on-chip SRAM with a latency of a single clock cycle is available for buffering.

The total amount of values A_{FB}, which have to be buffered internally for an FB scheme, is given by (1) (see Fig. 1).

$$A_{\text{FB}} = M \cdot N \cdot (Z-1) + M \cdot (Y-1) + X \qquad (1)$$

Note that each value of the grid is represented by a constant amount of bits. In scientific computing, single- and double-precision floating point numbers are very common, though the wide range of values is often not needed by the application. In fact, a fixed-point number format may allow a much more hardware-efficient implementation of arithmetical units, and even raise the accuracy of values within the interval. Therefore, the parameter D is used for the width of a single value, instead of a fixed data type.

As mentioned above, the size of buffer space required A_{FB} depends on the stencil itself since each value has to be stored until is has been streamed through the stencil, starting at the bottom-right of the last dimension and ending at the top-left of the first dimension, if it is applied in a row- and plane-wise fashion. As a result, the space required is independent from the parameter O.

More complex problems or higher requirements on accuracy often lead to larger stencils, which results in a strong increase of buffer space. High order compact schemes[13][14] reduce the size of the stencil at the expense of computation, which is not regarded to be critical on FPGAs, despite of the capacity of on-chip memory.

In a PB approach, solely the processed stencil of size $X \times Y \times Z$ is stored which requires much less data to be buffered by the FPGA, with the drawback that $X \cdot Y$ values have to be loaded to update a single value. This leads to a lot of redundant memory access in a PB scheme, which is therefore unfavorable if enough on-chip storage is available, as it is on current FPGAs.

Apart from PB and FB, we think it is not worth to implement a CPU-like memory hierarchy since caches are intended to speed up applications with an unpredictable memory access pattern, which is not necessary for streaming applications, since data is consumed sequentially.

Elaborate techniques like multi-buffering, well known from cluster computing to hide memory latency, are also not able to speed up the FB-approach any further since the external memory bandwidth remains as bottleneck. In general, the actual data transfer between the template and external memory should make excessive use of sequential burst transfers, but since the exact conditions are user- and application-specific, data transfer is not to be handled by the template itself.

2.2 Blocking

Depending on the size of the stencil and the FPGA used, it is likely that an FB approach is not possible for

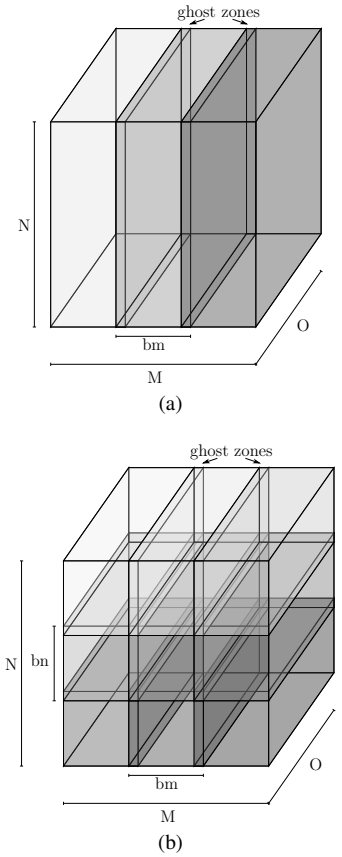


Fig. 2
SCHEME FOR 1D (A) AND 2D BLOCKING (B)

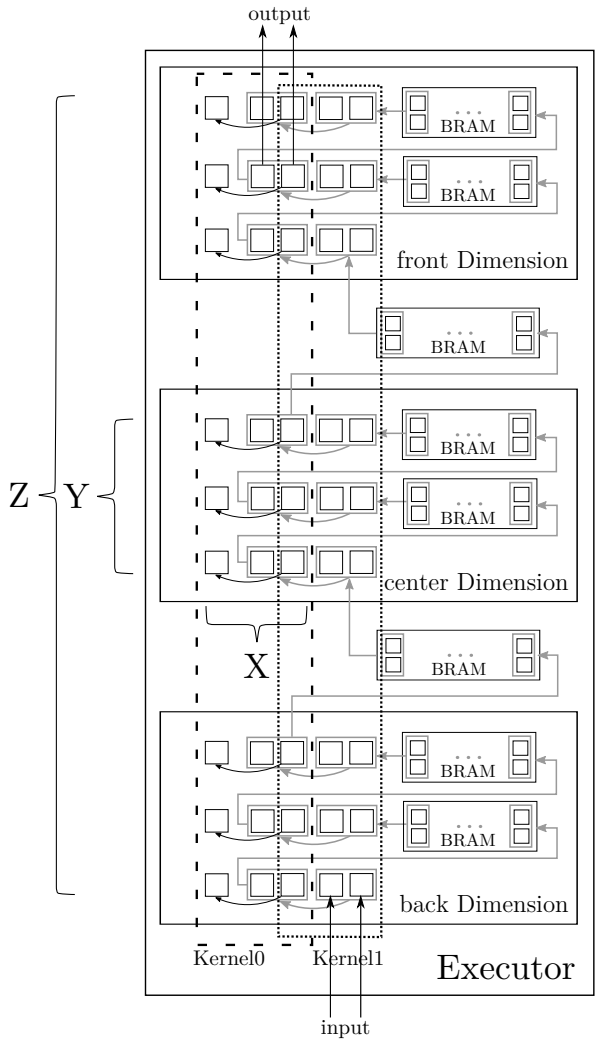


Fig. 3
SCHEMATIC OF FB TEMPLATE. EXECUTOR CONSISTS OF SEPARATE DIMENSIONS WITH INTERMEDIATE BUFFERS AND TWO KERNELS ($P = 2$) FOR A STENCIL OF SIZE $X = Y = Z = 3$

the complete input space. A sufficient compromise for the buffering scheme has to be found. The input space has to be split into blocks. FB is then applied to a block and all blocks are processed consecutively for each iteration. This is called *blocking*.

There are two main blocking methods. For 1D blocking as in Fig. 2a, the input space is split into slices, each spanning two whole dimensions. It is recommended for a non-square input space to choose O as the largest dimension to fully exploit the streaming character, because as shown in the section before, A_{FB} is independent of the parameter O. This maximizes the throughput and hides the initial wind-up overhead to fill the buffer. Using N for the smallest dimension reduces the size of the buffer and thus allows less blocks at all.

If the input space is even too large for 1D blocking to be feasible, which would result in many slices with a limited width bm, blocking in two dimensions is applicable (see Fig. 2b). Again, the largest dimension of the input space should be used for O.

On the other hand, blocking introduces redundancy because neighboring blocks must contain overlapping values in order to update the boundaries, called ghost zones.

This blocking mechanism can also be used to efficiently distribute the stencil computation of the input space to separate devices, if a multi-FPGA platform is used for example, which even does not have to be homogeneously due to the configurability of the template. For such a solution, the synchronization overhead is regarded to be negligible due to the deterministic character of the template. To minimize communication among separate FPGAs, it is applicable to enlarge the overlapping, even though it raises redundancy in memory transfers. That way, multiple iterations can be computed without having to communicate off-chip. These techniques are also well-known from the area of cluster computing using MPI and can be adapted to FPGAs.

2.3 Implementation

We implemented a generic VHDL design based on FB, allowing different degrees of parallelism, and supporting arbitrary problem sizes by spatial blocking. To be flexible, no vendor-specific library components are used but only behavioral descriptions, to make full usage of the synthesizer's inference capabilities. The main components of the template are *Kernel*, *Dimension* and *Executor*. Their dependence is illustrated in Fig. 3 and is explained in the following.

The Executor encapsulates the buffering and computation of the template. For each update, new data is streamed into the Executor, and results computed by the Kernel are streamed out. To be able to embed the template in different environments, it is controlled by a simple clock-enable signal which has to be activated if new data is available.

The Kernel implements the computation of the stencil and depends on the application. A Kernel has access to all data required for the realization of a stencil operation on a cell of the input space. It has to be implemented by the user and may contain either combinational or sequential logic, depending on the complexity of the algorithm. Updates of a cellular automata, e.g. for the realization of the *Lattice Gas* model, may be performed in a single clock cycle, while a lot of other stencil codes are more complex, leading to considerable critical paths. Adding intermediate registers allows pipelining of operations in order to increase the system frequency. Though this raises the latency of a single update operation, it does not limit the application's performance, because with each clock cycle, a new result is available, similar to pipelining of a microprocessor. The size of the kernel depends on the X, Y, and Z of the stencil and can be configured. Nevertheless, it is recommended to use minimal stencils if possible, even if they imply a higher computational intensity, to save buffer space.

Based on registers and FIFOs, an instance of the component Dimension holds the rows of an $M \times N$-wide plane which are currently covered by the Kernel. The remaining data of each plane is stored in a separate FIFO, except for the last plane, which directly consumes data from the Executor's input (see Fig. 1b).

In detail, each row within a Dimension contains X times D-wide registers and a separate $M - X$ values wide BlockRAM-FIFO. The registers are used as input to the Kernel, since FIFOs can not handle multiple reads simultaneously if they are realized with BlockRAM. On FPGAs, one cell of BlockRAM usually only has two ports. These registers are queued, allowing data to be shifted from one end to the other. The end of a queue is connected to the output port of a FIFO, which contains the remaining data of the row. On each update, the registers and FIFOs are shifted by one, allowing the next values to be processes by the Kernel. If a value is shifted out of the register queue, it is either sent to the row spatially above, the next plane, or, if it is not needed any more, it is discarded.

The explained buffering scheme allows to perform one update with a single load and store. This can be increased by introducing different degrees of parallelism, which is explained in the next section.

2.4 Parallelization

Depending on the amount of bits per value D and the external memory bandwidth, the degree of spatial parallelization P can be adapted. With single precision floating point numbers, 64 bit wide dual-port ram, and an the same clock frequency for FPGA and memory, $P = 2$ allows to exploit the available bandwidth. Two ports are needed to read and write data simultaneously.

By setting P, several neighboring values are updated simultaneously with only $P - 1$ additional values that have to be buffered. The demand for processing logic increases due to multiple Kernels, but this is not a limiting factor for common stencils, so P is only limited by the external memory bandwidth.

Especially for applications with small D, it leads to a significant rise of performance. This mechanism also benefits from the fact that the data format used on the FPGA is fully customizable.

If P is set properly, the only way to improve performance is to reduce the amount of data to be transferred by temporal parallelization of computation, called pipelining. Multiple iterations are computed per sweep by queuing up I Executors. Each Executor has its own FB of the current region and enhances the iteration by one. It's output is used as input to the next one, again enhancing the iteration. In theory, it is possible to compute all iterations with only a single load and store per value, but multiple Executors raise the demand for on-chip memory by factors. Pipelining also raises the latency of the template, but it is tolerable if a high amount of data is processed.

If blocking is required due to the size of the problem space, the demand for memory even grows. For every iteration to be processed within one sweep, additional values surrounding the current block are required, called . Because the block size of $bm \times bn \times O$ has to be set with regard to the available internal memory, the actual data block is smaller if more than one iteration is realized.

Several ways to configure the template have been introduced by now. All of them, the variable blocking mechanism, i.e. 1D or 2D, the block size, the amount of P neighbors computed in parallel or the depth of the Executor-pipeline I either depend on the on-chip resources available or the external memory bandwidth or both. It is an optimization problem to find optimal parameters for an actual problem. For 2D stencil code applications, we have already analyzed such optimization in more detail [15] and developed an analytical model. At the moment, we are working on an advanced version of our model to get an optimal parameter set also for 3D.

Table 1
PARAMETER DEFINITION

Identifier	Description
$M \times N \times O$	Width, height, depth of input with border
$m \times n \times O$	Size of block with data to be processed
$bm \times bn \times O$	Size of block, with ghost zones
k	Number of blocks
$X \times Y \times Z$	Size of the stencil
D	Bits per value (per cell)
P	Spatial parallelization
I	Temporal parallelization
A_{FB}	Buffered cells for Full Buffering
A_{I}	Buffered cells with pipelined Executors
c_{block}	Total memory accesses with blocking
c_{FB}	Total memory accesses without blocking
PF_{BL}	Performance of blocking
PF_{PB}	Performance of Partial Buffering

As a rule of thumb, P is set as mentioned above. The problem space is split into equal blocks that barely fit into the FPGA. If resources are still available, I is increased.

3. Evaluation

In this section, the mathematical background of the template is analyzed. Furthermore, syntheses results and the performance are presented. Tab. 1 gives an overview of the parameters of the template. Most of them have been introduced by now.

3.1 Overhead of Blocking

Based on 1, we determine the performance of blocking in relationship to an FB of the complete input space and compute the amount of redundant memory access.

The size of the ghost zone depends on the stencil size. For simple $X = Y = Z = 33$ stencils the width of the ghost zone is one for a single iteration. The block size $bm \times bn$, which also has to contain the required ghost cells, has to be determined by D and the available on-chip memory, according to (1). The size of the block $m \times n$ of actual processed values can then be determined as shown in (2).

$$\begin{aligned} m &= bm - X + 1 \\ n &= bn - Y + 1 \end{aligned} \tag{2}$$

Depending on the size of the input space, the number of blocks k, which have to be processed consecutively, can be determined by (3).

$$k = \frac{M \cdot N}{m \cdot n} \tag{3}$$

The total amount of memory accesses for FB on the whole input space, c_{FB}, can be determined by (4), and with blocking, denoted as c_{block}, by (5). Note that for 1D blocking, bn becomes N, reducing redundancy.

$$c_{\text{FB}} = k \cdot m \cdot n \cdot O \tag{4}$$

$$c_{\text{block}} = k \cdot bm \cdot bn \cdot O \tag{5}$$

For FB, all redundant memory accesses are eliminated, leading to an optimal performance. The ratio between c_{FB} and c_{block}, giving the fraction of performance that remains due to redundancy of the ghost areas, is ascertained by (6).

$$PF_{\text{BL}} = \frac{c_{\text{FB}}}{c_{\text{block}}} = \frac{m \cdot n}{bm \cdot bn} = \frac{(bm - X + 1) \cdot (bn - Y + 1)}{bm \cdot bn} \tag{6}$$

Eq. (6) proves our expectations. A greater internal memory allows a greater block size $bm \times bn$ and, hence, a higher performance because the amount of data needed for the ghost area is strongly dominated by the data enclosed. Furthermore, a greater stencil size implies a greater ghost zone width, leading to an increase in redundancy and, thus, to a lower performance.

Another interesting issue at this point is the performance of blocking with regard to PB, where (7) values have to be loaded.

$$PF_{\text{PB}} = \frac{M \cdot N \cdot O}{M \cdot N \cdot O \cdot X \cdot Y} = \frac{1}{X \cdot Y} \tag{7}$$

From the equations it follows that if FB is not possible, blocked FB is still to be favored over PB since $PF_{\text{PB}} < PF_{\text{BL}}$ for all $bm > X$ and $bn > Y$.

3.2 Pipelining

It was already mentioned above that for pipelined Executors, the actual block size $m \times n \times O$ has to be wider to be able to update border values correctly. For I iterations, a block has to contain $(m + I \cdot (X-1)) \times (n + I \cdot (Y-1)) \times O$ values. For high I, it is feasible to use the template with different parameters for every pipeline stage and omit the outer border of the previous stage while streaming.

The amount of buffer entries of size D for a Executor pipeline of depth I is then given by (8).

$$\begin{aligned} A_{\text{I}} = \sum_{i=0}^{\text{I}-1} \Big(&(m + i(X-1)) \cdot (n + i(Y-1)) \cdot i(Z-1) \\ &+ (m + i(X-1)) \cdot i(Y-1) + i(X-1) + 1 \Big) \end{aligned} \tag{8}$$

This shows why increasing the pipeline depth highly increases the buffer space.

3.3 Synthesis

By now, only theoretical assumptions were made. To get real numbers, the template was synthesized with Xilinx ISE 11.5 for a Xilinx Virtex-5 XC5VLX110T FPGA. The FPGA contains 69120 Flip-Flops (FFs) and Lookup-Tables (LUTs) respectively, and 148 BlockRAM cells with a capacity of 2KiB each.

Table 2
SYNTHESIS RESULTS

$M \times N \times O \cdot P \cdot I$	FFs	LUTs	BlockRAM	MHz
$32 \times 32 \times 32 \cdot 1 \cdot 16$	11328	2113	80	317
$32 \times 32 \times 32 \cdot 1 \cdot 28$	19824	3697	140	317
$64 \times 64 \times 64 \cdot 1 \cdot 1$	728	157	11	303
$64 \times 64 \times 64 \cdot 2 \cdot 4$	4816	496	80	308
$64 \times 64 \times 64 \cdot 4 \cdot 4$	7088	416	128	316
$128 \times 128 \times 128 \cdot 1 \cdot 1$	748	209	35	301
$128 \times 128 \times 128 \cdot 4 \cdot 1$	1782	1860	53	282
$128 \times 128 \times 128 \cdot 1 \cdot 4$	2992	833	140	301
$256 \times 256 \times 256 \cdot 1 \cdot 1$	770	239	131	298
$256 \times 256 \times 256 \cdot 2 \cdot 1$	1228	208	140	306
$256 \times 256 \times 256 \cdot 4 \cdot 1$	1814	168	152	257

These BlockRAMs are predestined to be used for buffering data on-chip, but they are less flexible than Lookup-Tables. Large values for D and P could lead to a higher demand on blocks needed due to the limited interface-width of 36 bit of each port. If the target architecture is fixed, technology-dependent optimization of allocated resources may reduce the demand. For fixed parameters of the problem space and the stencil, even further optimization is possible, e.g. by implementing the shorter row-buffers as distributed RAM.

Tab. 2 shows the results for different parameters with $D = 32$. It can be seen that the main limitation for the template is the available on-chip memory, as already explained before. Even a small input space of $256 \times 256 \times 256$ almost exhausts the capacity, leaving no room to increase I but only P. Hence, for large input spaces, blocking is inevitable to achieve a notable speedup.

3.4 Performance

Tab. 3 shows some performance estimations. The runtime is based on a clock frequency of 400 MHz for a block size of $M = N = O = 128$, and a stencil of $X = Y = Z = 3$. Note that the actual frequency depends on the speed of the FPGA and the complexity of the Kernel. To get a reasonable runtime, 1000 iterations are assumed. The first column shows the total amount of memory transfers for reading and writing, followed by the on-chip space needed in multiples of D. The runtime is estimated for a dual-channel $64bit$ wide memory interface with $D = 32$.

It can be seen that an optimal buffering strategy like FB can greatly reduce the runtime for 3D stencil code applications. Furthermore, the speedup can be highly increased by spatial and temporal parallelization. Compared to regular FB, the asymptotic speedup with optimization is $P \cdot I$.

It is necessary to clarify that the accuracy of performance estimations of hardware descriptions differ from theoretical peak performance of CPUs and GPUs because it is totally deterministic and thus predictable what happens during one clock cycle. Therefore, the performance of the template can be exactly given without benchmarking. Of course, for an actual application, additional components are needed, but the memory interface remains the bottleneck.

Table 3
COMPARISON OF DIFFERENT CONFIGURATIONS

	r|w $\cdot 10^9$	Buffer	Speedup	Runtime $[s]$
No buffering	54.0| 2.0	27	1	140
PB	18.0| 2.0	27	2.8	50
FB	2.1| 2.0	33027	13.7	10
$P = 2$	1.1| 1.0	33028	26.7	5
$P = 2$, $I = 2$	0.6| 0.5	66056	50.9	3
$P = 2$, $I = 4$	0.3| 0.25	132112	180.0	1.4

Taking power consumption into account, the FPGA-based stencil solution shows its actual potential. The FPGA in our design needs about 1.1 W, independent of the actual problem.

An optimized implementation of the Jacobi-iteration for Laplace's equation on recent Nvidia Fermi and Geforce GTX GPGPUs delivers a performance of 10 GLUPS[1] per 250 W or 40 MLUPS/W [16] for double-precision floating point operations.

Based on the maximum clock frequency of approximately 100 MHz, 2.8 GLUPS for double precision can be reached on a small input space of size $32 \times 32 \times 32$ (see Tab 2, leading to an efficiency of 2.5 GLUPS/W, which is magnitudes higher than the GPGPUs, under the assumption that for each clock cycle, new data is available, which can easily be fulfilled due to the low frequency.

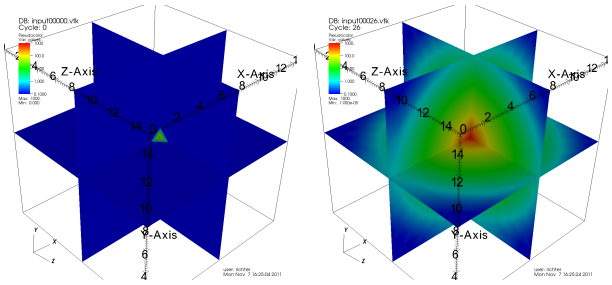
For larger problems, the performance drops, but even 0.2 GLUPS/W for $P = 2$ on a grid of size $256 \times 256 \times 256$ outperforms the GPUs by far. By using multiple FPGAs, power consumption scales linear with performance, allowing to build „green" HPC-systems. A very interesting example for the potential of FPGAs for high performance is [17], where a similar approach is used, but totally optimized for high-performance streaming applications and at a much higher degree of abstraction.

Furthermore, the FPGA used here is of medium size and two generations behind current versions. For greater applications, a larger FPGA offers even higher performance due to more on-chip memory, with only a limited increase of power consumption.

This analysis illustrates that FPGAs are able to outperform GPGPUs by factors with regards to power efficiency.

The comparison of the power consumption of the single FPGA-chip and the whole GPGPU-board is seen to be reasonable since all components used for computation are taken into consideration, e.g. the on-board RAM of the GPGPUs vs. the FPGA's on-chip BlockRAM.

[1] Giga Lattice Updates Per Second, conforms to single stencil applications per second



(a) $i = 10$ (b) $i = 26$

Fig. 4
3D JACOBI-ITERATION FOR CENTRAL HOT SPOT

4. Reference Design

We implemented a demonstration application based on the Jacobi-iteration to show the applicability and flexibility of our template-based approach.

FloPoCo [18], an open and free generator for arithmetic cores, was used to create the Kernel. The resulting FPU has a pipeline depth of 13 cycles, and allows a target clock rate of 100 MHz on the FPGA mentioned previously. If the FPU is split up into more stages, a higher frequency is possible. With only about 5 % of available resources of the FPGA, the Kernel allows a high degree of parallelism.

The results of the demonstrator can be seen in Fig. 4. With no loss of generality, an input space of only $16 \times 16 \times 16$ was used. *Visit*[2] was used to visualize the 3D volume.

5. Conclusion and Outlook

We presented a generic VHDL template for 3D stencil-based applications, exploiting Full Buffering to minimize external memory accesses and increase performance of memory bound applications.

To configure the template, several parameters can be set, according to the problem and target architecture. Execution can be accelerated by spatial and temporal parallelization, depending on the memory bandwidth and the amount of on-chip storage available.

The main advantage of the template is its flexibility. Due to minimized demands to the environment, it is possible to embed it in a large variety of settings. This may be, for example, a low-power embedded streaming application, or a PCIe-based multi-FPGA accelerator for high performance computations.

Together with the mathematical model we are developing, we will be able to find an optimal mapping of a given stencil-based streaming application to the resources of a system and to implement it without having to bother with data access, while preserving the advantage of efficiency of FPGA designs.

[2] https://wci.llnl.gov/codes/visit/

References

[1] M. Sadiku, *Numerical techniques in electromagnetics*. CRC Press, 2000.

[2] M. Schmidt, M. Reichenbach, A. Loos, and D. Fey, "A smart camera processing pipeline for image applications utilizing marching pixels," *Signal & Image Processing: An International Journal (SIPIJ)*, vol. Vol. 2, no. No. 3, pp. 137–156, 9 2011.

[3] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, ser. Computational Science. Taylor & Francis, 2010.

[4] D. Wolf-Gladrow, *Lattice-gas cellular automata and lattice Boltzmann models*. Springer, 2000.

[5] M. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, pp. 50 –57, march 2007.

[6] X. Liang, J. Jean, and K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," *J. Supercomput.*, vol. 19, pp. 77–91, May 2001.

[7] E. Motuk, R. Woods, S. Bilbao, and J. McAllister, "Design methodology for real-time fpga-based sound synthesis," vol. 55, no. 12, pp. 5833–5845, 2007.

[8] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, M. S. Mirotznik, and D. W. Prather, "Hardware implementation of a three-dimensional finite-difference time-domain algorithm," vol. 2, pp. 54–57, 2003.

[9] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, P. F. Curt, and D. W. Prather, "FPGA-based acceleration of the 3D finite-difference time-domain method," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 156–163.

[10] M. Shafiq, M. Pericas, R. de la Cruz, M. Araya-Polo, N. Navarro, and E. Ayguade, "Exploiting memory customization in FPGA for 3D stencil computations," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, dec. 2009, pp. 38 –45.

[11] H. Fu and R. G. Clapp, "Eliminating the memory bottleneck: an FPGA-based solution for 3d reverse time migration," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 65–74.

[12] F. Richter, "A full buffering based template for 3d stencil code applications on FPGAs," Diploma Thesis, Department of Computer Science, Chair of Computer Architecture, Friedrich-Alexander-University Erlangen-Nuremberg, 2011.

[13] W. F. Spotz, "High-order compact finite difference schemes for computational mechanics," Ph.D. dissertation, University of Texas at Austin, dec, see http://www.cisl.ucar.edu/css/staff/spotz/papers/Dissertation/Spotz_Dissn.pdf.gz, visited 2011-09-22.

[14] G. Sutmann and B. Steffen, "High-order compact solvers for the three-dimensional poisson equation," *Journal of Computational and Applied Mathematics*, vol. 187, no. 2, pp. 142 – 170, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042705001500

[15] M. Reichenbach, M. Schmidt, and D. Fey, "Analytical model for the optimization of self-organizing image processing systems utilizing cellular automata," in *SORT 2011: 2nd IEEE Workshop on Self-Organizing Real-Time Systems*, Newport Beach, 2011, pp. 162–171.

[16] A. Schäfer and D. Fey, "High performance stencil code algorithms for GPGPUs," *Procedia Computer Science*, vol. 4, no. 0, pp. 2027 – 2036, 2011, proceedings of the International Conference on Computational Science, ICCS 2011.

[17] O. Lindtjorn, R. Clapp, O. Pell, H. Fu, M. Flynn, and O. Mencer, "Beyond traditional microprocessors for geoscience high-performance computing applications," *IEEE Micro*, vol. 31, pp. 41–49, March 2011. [Online]. Available: http://dx.doi.org/10.1109/MM.2011.17

[18] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *Design Test of Computers, IEEE*, vol. 28, no. 4, pp. 18 –27, july-aug. 2011.

Distributed Approaches for Self-Adaptive Embedded Systems

ERSA'12 Academic Invited Paper

Pascal BENOIT
LIRMM, UMR 5506, CNRS University of Montpellier, 161 rue Ada, 34095 Montpellier Cedex 5, France

Abstract - *In the recent years, there has been a growing interest in self-adaptive embedded systems. Compared to the conventional approach, they require a control loop based on a three-step process: (1) observation, handled by a set of sensors/monitors, (2) diagnosis, which analyzes observed data to adapt and optimize the system, and (3) action, which tunes system parameters accordingly. Putting an additional intelligence into the circuit so that it is capable of modifying itself a set of parameters is not a new idea. But today, it seems that the conditions have been met to build such circuits. Firstly, self-observation has been made feasible with different kind of monitors, like activity counters, temperature sensors, critical path-monitors, etc. Secondly, it is possible to tune the voltage/frequency pairs, to migrate the code of a given task from one processing element to another, to adapt the routing of data in the interconnection network, etc. So what is the real challenge today? Achieving a complex but realistic unified self-adaptation mechanism, which strikes the balance between the introduced overhead, power consumption, performance and area. Given the increasing complexity of embedded systems, our approach is to consider a regular distributed architecture, with a set of identical Processing Elements, interconnected with a network on chip. Thus, all the hardware/software building blocks required for self-adaptation, are the same for each PE, which simplifies the scalability for future technologies. During this talk, we will present an open experimental platform and original approaches for the control loop based on the three-step adaptation process; we will analyze the cost of their implementation and will draw the perspectives offered by such techniques.*

Keywords: MPSOC, Distributed Architectures, Self-Adaptability, Game Theory, Consensus Theory

1 Introduction

For many years, the evolution of silicon technologies has pushed the emergence of new high-tech products, leading to new applications and new uses. Today, the application needs seem to clearly lead the technological developments. The ever-increasing performance and low-power requirements continuously brings new challenges in the semiconductor research and industry communities. In the 2011 edition of the ITRS [1], it was reported that in the next 10 years, the processing power will increase by 1000x for SOC Consumer Portable devices. As depicted in figure 1, the number of processing engines, logic and memory size will exponentially grow. Due to the short time-to-market and reduced lifecycles, the design efforts cannot be increased: it is assumed that in 2020, 90% of a SOC will be a reused design (58% in 2012).

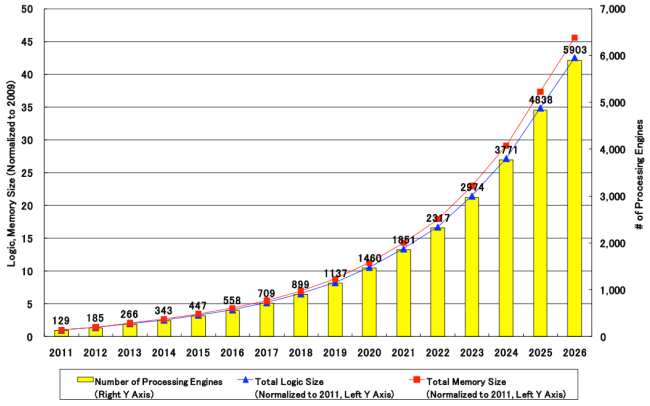


Fig. 1. SOC Consumer Portable Design Complexity Trends from ITRS

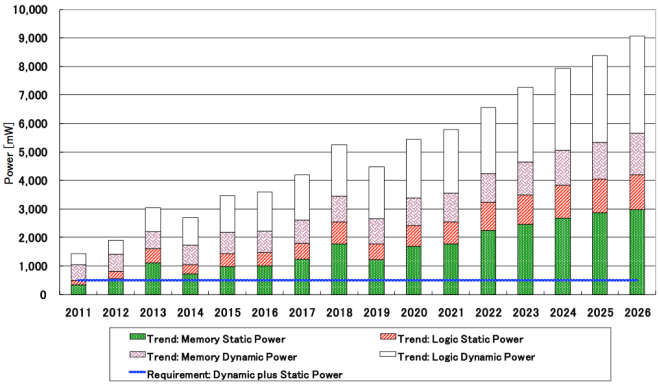


Fig. 2. SOC Consumer Portable Power Consumption Trends from ITRS

In this context, energy is also a critical issue. The power budget for a portable device ranges from 0.5 to 2W. It is clear from figure 2, that the current trend will not be acceptable for

future circuits. The reduction of the power consumption of electronic products must be addressed at all the stages: technology, design and applications.

From a technological point of view, variability has become a major issue. While process variations impact process, supply voltage and internal temperature [2], chip performances are also dependent on environmental and on applicative changes that may further influence chips behavior [3].

Performance, energy efficiency, technological variability and application versatility motivate the need of self-adaptability. The objective is to develop a system able to adapt itself to new applicative requirements as well as changes in the chip itself, or in its environment. As an example, the available energy for a telecom application is strongly reduced under weak battery conditions on mobile terminals, while real-time constraints depend on the telecom configuration. Compared to the classical approach, a self-adaptive system requires a control loop based on a three-step process: (1) observation, handled by a set of sensors/monitors, (2) diagnosis, which analyzes observed data to adapt and optimize the system, and (3) action, which tunes system parameters accordingly. The real challenge is to achieve a complex but realistic unified self-adaptation mechanism, which strikes the balance between the introduced overhead, power consumption, performance and area. Given the increasing complexity of embedded systems, our approach is to consider a regular distributed architecture, with a set of identical Processing Elements, interconnected with a network on chip. Since 2005, we develop our own MPSOC platform and investigate different distributed strategies for the monitoring and the optimization at run-time.

The paper is structured as follows. In the next section, we will give a definition of a distributed self-adaptive system. Then, we will present the platform developed at LIRMM called Open-Scale. In section 4, we will summarize our research works on monitoring techniques. Optimization techniques handled by distributed controllers will be reported and analyzed in section 5. Finally, we will conclude this article and give some future research directions.

2 A definition of a distributed self-adaptive system

The objective of this work is to find coherent solutions for the design of integrated systems that are efficient, low power, insensitive to technological process variations, reliable, scalable, with capabilities enabling them to adapt to their environment, and to the various applications they must support. We believe that simplicity and regularity of the system architecture are two key aspects that should guide the design choices. We present in this section on the one hand the concept of self-adaptability applied to embedded systems, and on the other hand, the specifications of the target architecture and the means to implement this system to make it adaptive.

2.1 Self-adaptability concept

Self-adaptability is the faculty attributed to an entity that can be self-sufficient and act within its environment to optimise its functions. Self-adaptability also describes a system that can manage itself using its own rules.

In order to apply this concept to the reality of technological systems, this study will start with the abstract view of a system architecture while applying the notion of self-adaptability. Figure 3 gives a synthetic view of self-adaptability: the activator creates the physical state of the system and the diagnosis motivates it. Self-adaptability can be used to lower energy consumption in microelectronics. In robotics, the challenge is to increase performance in different environments. In the artificial intelligence domain, self-adaptability is the consequence of a life cycle where the sensors observe the system, the diagnosis gives the direction, the language of command orders (decisions) and actuators act.

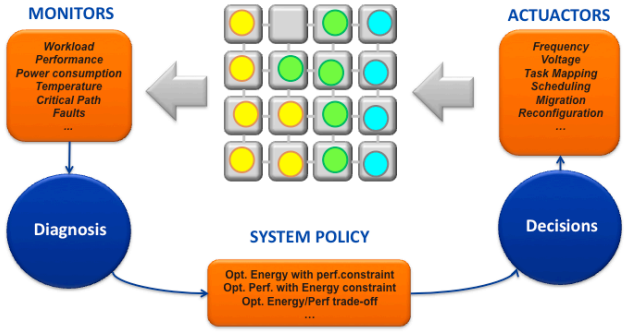


Fig. 3. Self-adaptability

2.2 System Specifications

When referring to current integrated systems, it is often about MPSOC or multi-core system. This is in fact for both integrated systems consisting of several processing units. When these ones are identical programmable general-purpose processors, MPSOC are said to be homogeneous. When dealing with heterogeneous architectures, there are different kinds of computing resources: general-purpose processors to support the system management, but also DSP, dedicated accelerators, etc. The current trend is finally the same for over 40 years: the integration density doubles approximately every 2 years, we then expect in the longer term Many Core systems, or MP2SOC (Massively Parallel MPSOC), *i.e.* systems with hundreds or thousands of processing units. Obviously, this complexity poses a number of questions about the evolution of design methods, verification, manufacturing, test, programming, debugging, etc.

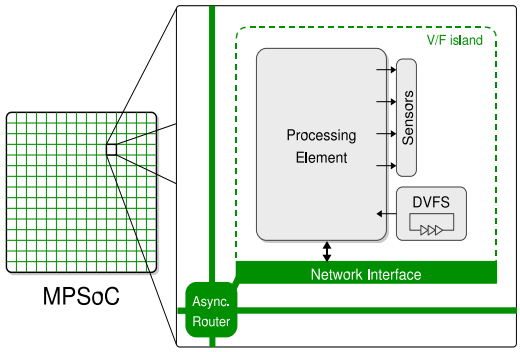


Fig. 4. System overview.

Given the huge design space due to the increasing number of transistors, it is not possible to build a 10 billion-transistor circuit from scratch, for each new product. During the 2000s, IP reuse and platform based design appeared as new methodologies to accelerate the Time To Market for SOCs. Due to the ever-increasing complexity, we start hearing about (Sub-) System IPs Reuse to build the future many core systems.

By observing these changes, we decided in 2005 to devote our research efforts on designing a regular and homogeneous MPSOC architecture. The basic idea is to have a simple subsystem, composed of a programmable processing element, memory, and a generic interconnection module, that we can instantiate theoretically to infinity. The primary version of our architecture called HS-Scale was first published in [4]. The thrust of this project is to define a generic and regular software and hardware support, to facilitate scaling. This very flexible model is originally not designed for specific applications, but with the addition of programmable and/or dedicated accelerators to the building block, one can think advantageously taking advantage of the same principle of regularity for specific products.

The regular architecture of our MPSOC is described in Figure 4. The PEs are interconnected by a Network-on-Chip (NOC) [5-8]. A NOC is composed of Network Interfaces (NI), routing nodes and links. NI implements the interface between the interconnection environment and the PE domain. It decouples computation from communication functions. Routing Nodes are in charge of routing the data between the source and destination PEs through links. Several network topologies have been studied in the literature [9, 10]. Our approach is based on a 2D mesh interconnect. We consider that the offered communication throughput is enough for a general purpose application set. Our NOC fulfills the "Globally Asynchronous Locally Synchronous" (GALS) concept, by implementing asynchronous nodes and asynchronous-synchronous interfaces in NIs [11, 12]. As in [13], GALS properties allow MPSOC partitioning into several Voltage Frequency Islands (VFI). Each VFI contains a PE clocked at a given frequency and voltage. This approach allows real fine-grain power management.

Dividing the circuit into different power domains using GALS must facilitate the emergence of more efficient designs that take advantage of fine-grain power management [14]. As in [15, 16], the considered MPSOC incorporates distributed Dynamic Voltage and Frequency Scaling (DVFS): each PE represents a VFI and includes a DVFS device. It consists of adapting the voltage and frequency of each PE in order to manage power consumption and performance. A set of sensors integrated within each PE must provide information about the power consumed, the local temperature, the performance or any other metric needed to manage the DVFS.

3 Open-Scale Prototyping Platform

Since 2005, we develop an MPSOC architecture based on the principles described in the previous section, *i.e.* a regular and distributed architecture for the implementation and the evaluation of self-adaptability principles. This architecture published for the first time in [4] has undergone a number of developments and changes. The current version of this architecture is called Open-Scale: it is therefore an MPSoC governed by the basic principles outlined above. We have different models of the architecture: a version based on ISS (Instruction Set Simulator) and SystemC, and a second based on the fully synthesizable RTL code. The first one allowed making high level explorations, and the development of the RTOS. Although the architecture is the same for a system standpoint, we are interested especially here in the RTL version, validated on multiple FPGA prototypes, as the one depicted in figure 5. In this section we describe the hardware and software building elements of the Open-Scale platform.

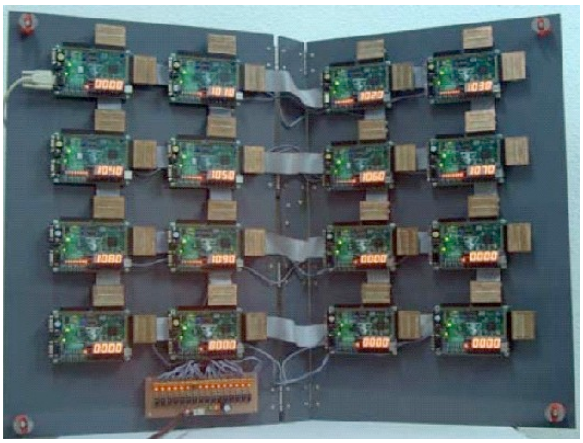


Fig. 5. MPSOC prototype

3.1 NPU, the Network Processing Unit

The Open-Scale system is an architecture that employs a distributed memory/message passing approach and a 2D-

mesh NOC for connecting different PEs. As the PEs are connected through a network, they are called Network Processing Units (NPUs) [4].

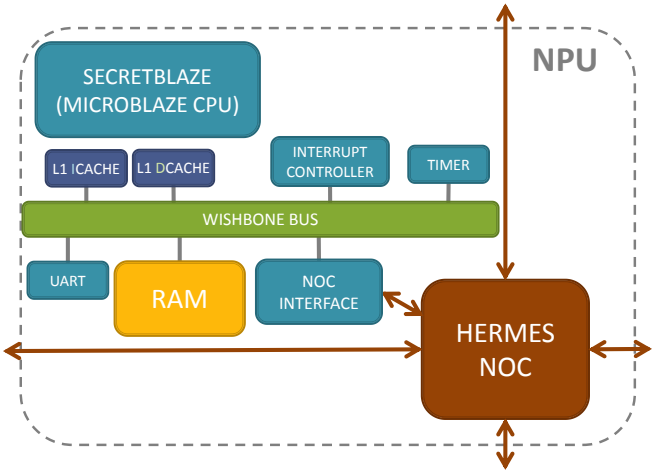


Fig. 6. NPU architecture overview

The figure 6 depicts the internal architecture of the NPU, which includes: (*i*) a general purpose processor, the SecretBlaze [17,18], (*ii*) an embedded RAM, (*iii*) an interrupt controller, (*iv*) a timer, a UART, (*v*) a NI, a (*vi*) HERMES-based router [19] and a (*vii*) Wishbone v4 bus [20].

Originally designed with the Plasma processor [21], the NPU is now based on the SecretBlaze [17,18], which is a highly configurable open-source RISC soft-core processor developed by our research group. It implements the MicroBlaze instruction set architecture with a MIPS five-stage pipeline, where most of the instructions require one clock cycle to be executed. As reported in [17], it achieves the same performance level as the MicroBlaze for FPGA implementations. This processor was developed with a modular approach, not only to ensure reliability and efficiency across the whole design, but also to provide better design reuse opportunities in various research and educational projects. It is available as a single open-source processor at *http://www.lirmm.fr/openscale*

The flexibility is one of the driving aspects of the Secret- Blaze design. On the one hand, the core provides several optional logical and integer instructions such as multiplication, division, and pattern operations, which balances computing performance and area cost to meet embedded system requirements. On the other hand, the SecretBlaze is a MMU less processor with a simplified memory sub-system that offers optional configurable data and instruction caches. It implements the pipelined Wishbone protocol for memory interfaces. However, no global cache coherency between NPU is provided.

The SecretBlaze uses an embedded RAM as local memory. The interrupt controller can handle up to 8 interrupts with masking, arming, and poling mechanisms. The timer is a 32-bit counter that can generate an interrupt according to a configurable time window. Besides, a UART interface, which is adjustable via software, can be used for debugging purposes. These components are interconnected by a Wishbone v4, which is a standard and an open-source bus [20]. The communication between the NPU and the NOC router is implemented in the NI (Network Interface), which defines HW/SW integration (*e.g.* bus width, bandwidth), as well as packing/unpacking the packets from/to the NOC.

The adopted NOC router is a small XY router based on HERMES [19]. The NOC employs packet switching of wormhole type: the incoming and outgoing ports used to route a packet are locked during the entire packet's transfer. The routing algorithm is an XY engine that allows deterministic routing. Each router processes one incoming FIFO per port. The size of FIFOs can be chosen regarding the desired communication performance.

3.2 Open-Scale RTOS

Open-Scale is a set of programmable Network Processing Units. Programming (and debugging) complex applications tp make an efficient use of multiple computing units is a real challenge. It is necessary to provide a coherent middleware layer to simplify these two fundamental aspects of embedded systems. In order to keep a distributed memory structure and to preserve the scalability of the system, each NPU operates asynchronously and communicates with each other by means of a Messaging Passing Interface (MPI) protocol. The global operation is performed in a distributed fashion and no global shared-memory is used.

Each NPU runs a tiny preemptive RTOS that was further extended from the Plasma RTOS [21]. The global structure of the operating system is depicted in figure 7. The RTOS provides a multi-threaded preemptive execution, using a scheduler based on thread priorities that is executed periodically according to a fixed *timeslot*. A round robin scheduling algorithm is executed when all tasks have the same priority. The structure of this system is divided into 4 categories: (*i*) services that provide the basic operating system requirements, (*ii*) communication, (*iii*) drivers, and (*iv*) libraries. The RTOS allows the use of semaphores and mutexes, communication between local and remote tasks, and dynamic memory allocation, as well. Further, it also provides the standard C library together with a compact math library that allows floating point operations as well as software multiplications/divisions. Different kind of drivers, e.g. timer, UART, etc., are also available.

Due to the distributed memory characteristic of the system, the applications are described using Kahn Process Network (KPN) formalism [22].

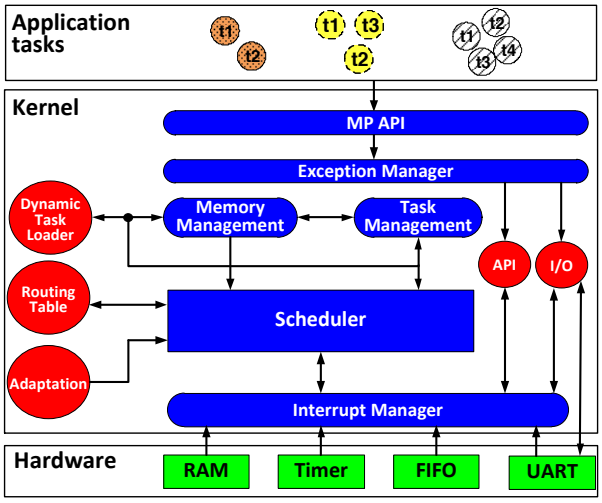


Fig. 7. Structure of the Open-Scale RTOS

MPI provides a comprehensive number of primitives that relate to general-purpose distributed computing; a number of works have devised lightweight implementations supporting only a subset of MPI mechanisms for embedded processors and systems. This makes sense since KPN formalism offers a sufficient support that requires only blocking read operations, which are necessary to model, for instance, data flow (*e.g.* video and audio) applications. Some MPI implementations are layered, and advanced communication synchronization primitives (*e.g.* collective) found in the upper layers make use of the simple point-to-point primitives such as *MPI Send()* and *MPI Receive()*. This enables using these collective mechanisms in an application-specific basis in case they prove necessary. The KPN computation model allows deterministic behavior of the application in an asynchronous way. Furthermore, tasks placement can be optimized depending on the user requirements (*e.g.* computation time, energy consumption).

The Open-Scale RTOS was implemented in such a way that users can easily choose which features are needed in their implementation in order to either save memory or meet performance requirements. In this scenario, new services and features were implemented in order to be compliant with the SecretBlaze architecture, while providing more efficiency in terms of management and QoS support (Table 1).

Table 1 summarizes some services that were included in the Open-Scale RTOS. As mentioned before, one of the goals of Open-Scale is to explore adaptive mechanisms (*e.g.* monitoring techniques, distributed control, dynamic voltage and frequency scaling, task migration, etc.). For instance, to enable dynamic load balancing, the system has to be able to move running tasks from one NPU to another. For that reason, a run-time loading mechanism was included to allow compiled separately applications from the RTOS being dynamically uploaded at run-time.

Table 1. Services Included into the Open-Scale RTOS

Plasma RTOS Services	**Open-Scale Services**
1- generation of a single object file; 2- preemptive round-robin sheduler based on thread priorities; 3- intra-NPU communication based on local FIFOs; 4- Extra-NPU communication (e.g CP protocol) through ethernet; 5- interrupt and exception handling; 6- dynamic memory allocation and deallocation; 7- queues, semaphores, mutexes.	1- run-time dynamic applications loading; 2- preemptive round-robin scheduler based on thread credits; 3- intra-NPU communication based on messages exchanged by software FIFOs; 4- Extra-NPU communication (RAW protocol was included), as well as *MPI_Send* and *MPI_Receive*; 5- run-time monitoring support; 6- decision-making mechanisms; 7- a run-time control system used for regulating NPU frequency; 8- API with new primitives, etc; 9 – development of new drives; 10- dynamic mapping heuristics.

Besides, a preemptive round-robin scheduler based on thread credits has been implemented, avoiding task execution starvation. Moreover, intra/extra-NPU communications were extended to provide more flexibility and system performance. For example, the RAW protocol was implemented in order to achieve better performance when compared to TCP/UDP (as shown in [23]). Further, online system-monitoring mechanisms were included, in order to access hardware monitors available, or software defined monitors (CPU load, (*ii*) FIFO load, etc.). Once monitored information is provided, online decisions can be taken by decision-making mechanisms, like a run-time control system used for regulating NPU frequency. All the middleware support for self-adaptation (*e.g.* local DVFS control, optimization, etc.) was included to provide a complete infrastructure for self-adaptation strategies investigations.

4 Distributed Monitors

We have defined a set of software and hardware components, based on distributed resources and a principle of regularity of the architecture that has led us to the design of the Open-Scale platform. In this section, we will focus on the monitoring aspect. To achieve a self-adaptive system, it is necessary to provide a set of sensors allowing the system to observe both its internal behavior and its environment. We focus here on internal monitoring.

We seek solutions to evaluate strategies of self-adaptation that are also compatible with an FPGA prototyping environment. We therefore propose in this section to explore several types of approaches to monitoring in this context, by focusing on hardware sensors, such as PVT sensors or activity counters, on software monitors directly integrated into the middleware of Open -Scale, and finally an integrated database enabling a coherent organization for a subsequent operation for system optimization.

4.1 PVT sensors

Monitoring with trustable and valuable information systems made of a billion transistors at a reasonable area and performance cost is a real challenge, because of the increasing random nature of some process parameters, the spatial dependence of process (including aging), voltage and temperature variations, but also the broad range of time constants characterizing variations in these physical quantities.

There is a vast literature on this topic, and different kind of approaches. Several PVT sensors commonly used for post fabrication binning have been investigated [24-29] for global variability compensation. They are based on specific structures like Ring Oscillators or Replica Paths to measure, at runtime, the physical and electrical parameters required to dynamically adapting the system, *e.g.* operating frequency, the supply voltage, threshold voltage, substrate biasing, etc. Another approach is to directly monitor sampling elements of the chip (Latches or D-type Flip Flop) to detect delay faults. This can be achieved by inserting specific structures or using ad-hoc sampling elements [28-29] to detect a timing violation by performing a delayed comparison or by detecting a signal transition within a given time window.

In the Open-Scale platform, one objective is to design a set of sensors in order to monitor locally the Temperature, Voltage, and Process. The basic idea is to use digital hardware sensors designed with internal resources of the FPGA, *i.e.* configurable logic blocks and switch matrices. In this context, we proposed, designed and compared two structures: a Ring Oscillator and a Path Delay sensor.

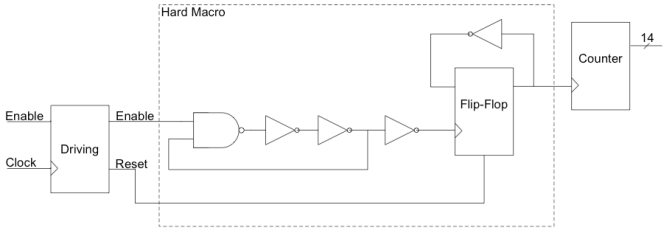


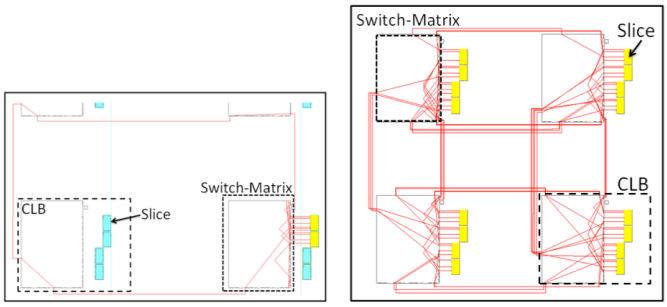
Fig. 8. Ring Oscillator sensor

The Ring Oscillator sensor is based on the measurement of the oscillator frequency. For instance in [30], this structure was used as an internal temperature sensor for FPGA. The frequency of a ring oscillator is measured and converted into temperature. We have developed a new version of this kind of sensor in Open-Scale. Its structure is depicted in Figure 8. The main part of the sensor is a *2p+1* inverter chain. The oscillation frequency directly depends on the FPGA process performance capabilities, for a given voltage and temperature. The first logic stage enables the oscillator to run for a fix number of periods of the main clock. The ending flip-flop is used as a frequency divider and allows filtering glitches from the oscillator. The final logic stage counts the number of transitions in the oscillator and transmits the count result. Then, the count result is used to calculate the oscillator frequency as follows:

$$F = \frac{count * f}{p}$$

where F is the ring oscillator frequency, *count* is the 14-bit value of the counter, *f* is the operating frequency of the clock and *p* is the number of enabled clock periods for which the sensor is active.

In order to use this sensor for resource monitoring in FPGA, a three-inverter ring oscillator was implemented. With this configuration, the core of the sensor (ring oscillator + first flip-flop) only consumes 4 LUTs. A Hardware Macro was designed so that the very same sensor structure can be mapped at each FPGA location (Fig. 9(a)). It possibly allows characterizing separately each CLB of an FPGA.



(a) Ring Oscillator (b) Path Delay

Fig. 9. Hard-Macro implementation of both PVT sensors

It exists a lot of techniques [31] to manage Critical Path but very few are used in FPGAs as PVT sensors. The Path Delay Sensor proposed here is directly inspired by CPM. Its structure is depicted in Figure 10. The idea of the Path Delay Sensor is to adapt CPM to FPGA. Indeed, the regularity of the FPGA structure enables to create more easily a critical path replica in FPGA than in ASIC.

The Path Delay Sensor is composed of *n* LUTs and n flip- flops (FF). The LUTs are chained together and a FF is set at the output of each LUT. A clock signal is applied to the

chain and is propagated into the LUT. At each rising edge, a *n-bits* thermometer code is available at the output of FFs. This thermometer code is representative of LUTs and interconnects performances.

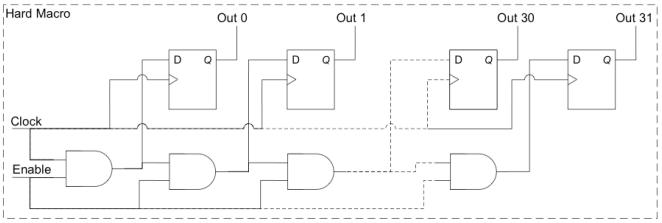


Fig. 10. Path Delay sensor

When the sensor is running, a thermometer code is stored in the FF, and then analyzed. Assuming the code is "11111111111111000000000000001111", then the position Nz of the last 0 is identified. The time T required to go through one LUT and the associated interconnect is approximated as follows:

$$T = \frac{Nz + 2}{f}$$

where f is the frequency of the clock signal applied to the sensor. In order to obtain relevant information, the size of this sensor must take into account the FPGA family in which it operates. For example, for a Spartan-3 device, this sensor is composed of 32 stages. It allows propagating a complete period of the Spartan-3 reference frequency clock (50MHz). The figure 9(b) shows the Hard Macro integration of this sensor.

Our study in [32] has proved that both structures were efficient for fast PVT local monitoring in FPGA devices. The ring oscillator is an interesting structure for fine grain monitoring, whereas the Path Delay Sensor will be a preferred structure to allow rapid performances estimations with a minimal area overhead.

4.2 Activity Counters

In some cases, PVT sensors might be not enough accurate and fast to reflect the exact load of the core, especially to estimate the power consumed locally by the Processing Elements. In-situ events counters have been proved to be efficient solutions for monitoring at real-time the activity of a core. They were first developed to better understand application behavior, facilitate debugging and so to improve high-end processors execution flow at run-time. For this usage, they are also called performance counters. These probes are a set of in-situ registers used to record the activity (*i.e.* number of events on some selected signals) in the processor. Significant events having an architectural meaning are typically used, such as instruction execution, floating point operation, memory transactions, pipeline stalls, etc. For instance, the super processor BlueGene/P from IBM is providing 52 counters [33], which can be configured and read through a dedicated API. The ARM processor architecture contains two event counters with an extending instruction set to configure their inputs. Moreover, a dedicated monitoring co-processor can be added through this interface. As the required precision becomes more and more important, the number of events tends to increase. Temporal multiplexing has been proposed to reduce the hardware overhead when applications show periodic execution time [34]. However, gathering information is problematic and may penalize applications with hard performance constraints. As a result, the number of monitors as well as the size of their associated counters must be reduced.

The use of activity counters was extended to embedded system run-time management in [35,36]. In such systems, area constraints hardly limit the possible number of monitors compared to general-purpose processors. Activity counters were generally selected manually and their usage was quite limited. Moreover, assuming that the NPU of Open-Scale is composed of different kind of resources (the processor, memory, peripherals, network interface) that could also be further augmented with accelerators or specialized DSP, it is necessary to provide other activity counters to estimate at run-time the power consumed by the different blocks. In such a context, the selection process of the signals to monitor might become a difficult and time-consuming work.

We tackled this issue by proposing an automated selection of events to be monitored starting from a power model, and usable for any IP. Moreover, our selection process allows striking the balance between the area taken by the monitors and the precision of the model.

Power consumption P_T can be approximated, over the period T, by a linear function of the following form:

$$P_T = c + \alpha_1 N_{e_1} + \ldots + \alpha_i N_{e_i} + \ldots + \alpha_n N_{e_n}$$

where :

- $\{e_i\}_{i=1:n}$ denotes events to which counters will be connected;
- N_{ei} is the number of occurrences of event e_i during the sampling period T;
- α_i is the regression coefficient describing power contribution of event e_i;
- c is a constant term representing static consumption;
- n measures the power model complexity.

Our goal is to minimize the cardinal of the set $\{e_i\}_{i=1:n}$ of events, so that we can limit the model complexity. We

propose to use a step-by-step selection technique to detect influential events on power.

Fig. 11 depicts an overview of the proposed modeling flow based on four steps: instrumentation, data processing, model extraction and physical placement of monitors. Our flow starts with a post-synthesis simulation to generate a VCD (Value Change Dump) file containing the transitions at each net over time. At this stage, we try to track highly power consuming blocks, so for the synthesis process we preserve the hierarchical structure of the design. The VCD is used as input to the *Xpower* tool from Xilinx to extract instantaneous power values. In parallel to this, the RTL model of the design is simulated. Events occurring on each controlling signal are time-stamped and then reported, this can be easily done with some options in *Modelsim*.

Our method based on stepwise process was evaluated to select events able to estimate the power consumption. This approach helps to explore cost and accuracy trade-offs when designing such probes. Sampling impact can be easily analyzed along with the number of counters that should be deployed. A control unit extracted from a dedicated VLIW accelerator for Open-Scale was used as a case study. The model was validated with less than 5% error. The implementation results show a reading latency around 500 clock cycles, at a very cost (only 12 monitored signal) regarding the accelerator area.

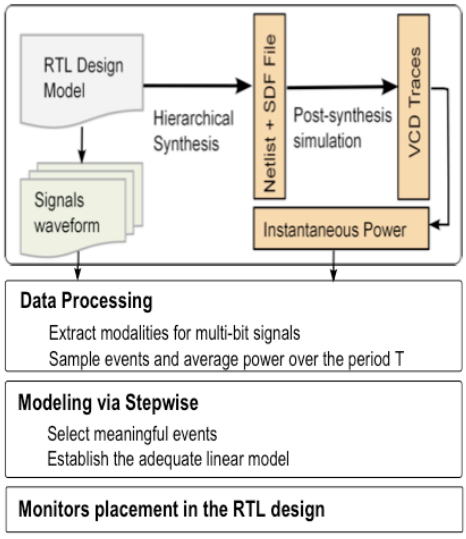


Fig. 11. Events profiling methodology flow

4.3 Software monitors and distributed database

Another complimentary approach to observe the system behavior is to use software monitors. As reported in section II, monitoring services are implemented into the Open-Scale RTOS to measure performance at the software level. The software threads implemented into the microkernel allow measuring the processor load, the communication load, application throughput, etc.

Applications suited to be run on Open-Scale are based on the traditional KPN (Kahn Process Network) paradigm. The application can be described as a finite set of tasks interconnected by software FIFOs. A given task can either be performed in hardware (with a coprocessor) or in software, in which case it becomes a thread executed by a processor on-chip. The performance of the system is directly related to the quality of the application mapping, *i.e.* the placement of threads on processors and the placement of software FIFOs on on-chip memories.

The processor load is simply calculated as the ratio between the number of cycles the PE is executing a set of threads, and the total number of cycles on a given time period. Indeed, due to local and remote thread dependencies, thread priorities, data availability, NOC traffic, etc., the processor may be more or less idled during the considered time period. This kind of information is very important to evaluate the efficiency of the application mapping for instance. For the same reasons, the software FIFOs may be filled more or less during application execution. Consequently, a specific service is implemented to measure the filling rate of the FIFOs.

It is mandatory for self-adaptive MPSOC architectures to be as reactive as possible to critical events, and to keep an accurate vision of the system behavior. A memory containing collected values from the different sensors is therefore needed, as well as appropriate and low cost means to store, handle, and retrieve these accumulated data. For this purpose, we have developed an in-memory database engine that fulfills these needs, as well as its associated API.

The DRET (Distributed Raw Events Table) is a distributed in memory database that is physically located in a specific part of the RAM memory of each NPU. Its purpose is to contain the monitoring data extracted from both hardware and software. The DRET of a tile may contain several tables like traditional database systems and each table contains several formatted events retrieved from the HW/SW sensors. The database has been designed to be an efficient data structure that can be used to store and retrieve information easily from the monitoring process. The used structure and the API are designed to keep the memory footprint and performance overhead as small as possible. The DRET can be seen as a uniform repository for the events whatever their origin (hardware sensor, software probe, local or external to a given processing element). While a DRET is mainly related to a given NPU, it can also be used to store a synthetic view of the state of neighboring NPU. In order to prevent the DRET from occupying too much memory space, a maximum size is associated to each table created in a DRET. A DRET table can also be cyclic. In that case, it behaves like a fixed-size cyclic buffer: inserting a new event in a full table discards the oldest one. The DRET and its API were

evaluated in [37]. Although there is a slight memory overhead in the system imposed by the usage of the DRET, the overall gain can be very beneficial for the system. It allows the system to handle some decisions through the analysis and diagnosis of such stored events. This brings to the system the possibility to better decide the application mapping tuning it according to the requirements, and thus to reach the objective of chip self- adaptability.

5 Distributed Controllers

Prior works addressing system optimization (performance, power consumption, temperature, etc.) are generally based on centralized schemes. Application mapping, task placement and scheduling, voltage and frequency are tuned to adapt the design dynamic settings to its runtime constraints, according to a solution provided by a centralized algorithm. For instance, authors of [38] present a method to select the frequencies and voltages based on non-linear Lagrange optimization. The work presented in [39] proposes a centralized system inspired by Kirchhoff's current law to decide on the optimal power/performance configurations. Conventional centralized approaches become problematic when the number of cores increases. First, the complexity of the optimization problem explodes as the dimension of the system increases. Secondly, communication latencies induced after collecting on-line information have to be considered; in [40], authors use a time-stamped monitoring mechanism to ensure the reliability of their system, but this solution leads to overload the network traffic for designs connecting more than a few units. Some recent proposals focus on distributed techniques to optimize embedded systems subject to online information. In [41], the problem of multi-core systems thermal control is posed as a convex optimization problem and solved in a distributed manner using dual decomposition technique. Stochastic methods are introduced in [42] to manage power consumption over the chip lifetime. The complexity of the used Markov model increases with the number of possible power/performance configurations, making this technique inconvenient for designs embedding hundreds of cores.

In this work, we study a new approach in order to minimize the energy consumption of MPSOC at run-time taking into account different application constraints. As depicted in figure 12, we assume that each core is able to monitor its resources as described in the previous section (*i.e.* sensors to measure the system performance, and energy consumption), and a DVFS engine to tune the Voltage/Frequency couples. We propose to achieve Power/Performance tradeoff by distributed schemes based on 3 different approaches: PID based controllers, Game Theory inspired controllers, and Consensus Theory inspired controllers.

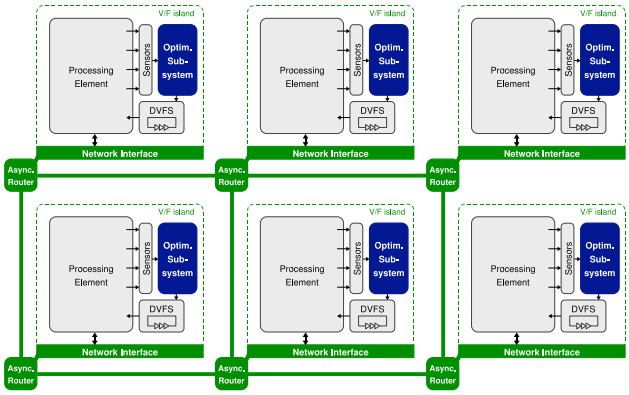


Fig. 12. Distributed Control overview

5.1 PID based Controllers

Due to the dynamic variations in the workload of MPSOC and its impact on energy consumption, adaptation techniques such as PID (Proportional-Integral-Derivative)-based control have been used to dynamically scale the voltage and the frequency of processors [43][44], and recently, of networks- on-chip [45][46]. These techniques differ in terms of adopted control parameters (*e.g.* task deadline, temperature) and response times (period necessary to stabilize a new voltage/frequency).

In [47], we have presented the following contributions: (i) power and energy consumption considered when tuning processor frequency; (ii) a PI-only controller proposed and compared to a PID-controller; and (iii) three perturbation scenarios with different application performance impact factors. The figure 13 illustrates an overview of the proposed approach. As it can be observed, one PID controller is devoted to each task in the system that must ensure soft-real time constraints. In this example, there is one task per NPU, so one PID controller for each processor is required. In the case where multiple tasks are executed in the same NPU, a system with multiple PID controllers in the same NPU could be proposed. The PID controller is implemented as a service in the Open-Scale RTOS. It only represents an overhead of less than 1% in terms of total memory required by the OS.

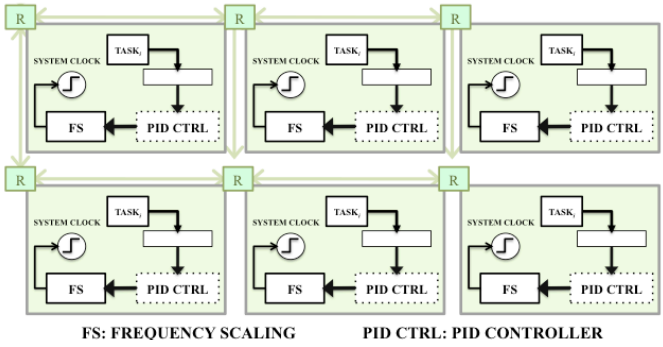


Fig. 13. MPSOC platform with PID controller

Monitoring information such as application throughput is fed into the PID controller module. It will match the actual throughput with the desired throughput (*setpoint*) and will then calculate an error value (*e*). This *e* value is used for calculating P, I and D parameters and as result, the PID controller will indicate a frequency in which the processor can reach a given *setpoint* according to reactiveness factor initially set. It is important to observe that the PID management and calculation are performed dynamically at run-time. In cases where current throughput is lower than the *setpoint,* the PID controller will select a frequency greater than the current one in order to reach the *setpoint* and to satisfy application performance constraints. On the other hand, when the current throughput is much higher than the expected one, the controller will sign to a lower frequency in order to reduce the power consumed by the NPU.

The proposed strategy consists in deciding controller parameters on a task basis. To this purpose, a SystemC/TLM-based Open-Scale simulation is executed in order to obtain the step-response. In this scenario, processor frequency is changed from 55MHz to 1005MHz and application throughput is monitored. The system is linear, that means the system's behavior remains the same under such frequency changes. Based on the high-level model, a number of different configurations of controllers can be explored. Each one exhibits different features such as speed, overshoot and static error. Once the process is modeled, PID parameters are fine tuned by using Simulink and the values of P, I and D are fed as input to the Open-Scale platform.

The PID strategy does not rely only on preventing application deadline-misses, but it also attempts to save energy, once the processor frequency is adjusted at real-time according to application requirements. Our approach can be easily integrated to linear systems and, platform resources utilization can therefore be optimized. By using PID controllers, we have shown in our simulation scenario that it was possible to save up to 32%, in terms of energy by tuning processor frequency according to application needs. As it can be noticed in [47], PID-controllers are intended to react faster under disturbing conditions compared to PI-only controllers. However its power consumption is higher. The good choice of which controller to use in multiprocessor system-on-chip platforms will be a trade-off between power consumption and the desired system's reactiveness.

5.2 Game-Theory based Controllers

Game theory involves a set of mathematical tools that describe interactions among rational agents. The basic hypothesis is that agents pursue well-defined objectives and take their knowledge and behaviors of other agents in the system into account to make their choices. In other words, it describes interactions of players in competitive games. Players are said to be rational since they always try to improve their score or advance in the game by making the best move or action. Game theory is based on a distributed model: players are considered as individual decision makers. For these reasons, game theory provides a promising set of tools to model distributed optimization on MPSOC and, moreover, this is an original approach in this context.

The second dynamic optimization proposed for Open-Scale is therefore inspired by game theory, where a *non-cooperative game* is a scenario with several players interacting by actions and consequences [48]. Basically, players individually choose an action within a defined set, resulting in consequences. Each player tries to maximize its outcome according to its preferences, leading to global optimization. If this sequence is repeated, under certain conditions, the game finds a solution formalized as *Nash Equilibrium*. These principles provide strong concepts to model the behavior of reactive systems where decisions are taken in a distributed and dynamic way, justifying the choice of the game theory for our approach.

We model the NPUs as players, the application latency and power consumption as a local objective function that depends on the global state of other NPUs. Then, a distributed algorithm selects the best solution. The objective functions are built by using different terms: the energy contribution of each PE to the whole energy consumption, the applicative latency contribution to the total latency and penalty functions modeling energy and latency constraints. We consider a MPSOC, composed of *n* NPUs interconnected by an asynchronous Network-on-Chip, such as Open-Scale. Each NPU integrates a DVFS engine that regulates the local voltage and frequency couple among a finite number of solutions. We denote T_i the clock period corresponding to NPU_i and $Ti- = (T_1,...,T_{i-1},T_{i+1},...,T_n)$ the periods of all other NPUs in the MPSOC. We assume that an external mechanism has mapped the application on the MPSOC, each NPU handling a unique task. The task assigned to NPU_i takes N_i cycles of T_i to be processed. We denote as L_{max} the application latency constraint and we consider that each task is scheduled every T_0 seconds. Energy, latency contributions and energy and latency constraints are modeled with this formalism. Then, two objective functions are built according to the formulated scenarios, *i.e.* energy or latency minimization under conditions.

The proposed method, based on Game Theory, optimizes the system while fulfilling dynamic constraints. A telecom test-case has been studied in [49,50] to demonstrate the effectiveness of this approach. For the evaluated case, the proposed technique has obtained up to 20% of latency gain under energy constraints, and 40% of energy gain under latency constraints.

Our studies have also shown in [51,52] that our method scales with the number of processors without excessive convergence times. For a 100-processor platform, our technique has required an average of 20 game cycles to reach the solution. A game cycle requires around 2500 cycles to collect monitoring data from other NPUs, minimize the

objective function, and transmit its data to other NPUs [53]. The few calculation cycles needed to converge make this technique a feasible approach to optimize consumption and performance at run time. Furthermore, we have demonstrated that the achieved optimization is about 89% in average compared to a global offline method, which proves the quality of the results obtained with this method.

5.3 Consensus Theory based Controllers

Although the Game Theory approach is easy to implement, it may lead sometimes to local unstable minima, which necessitates extra resources to detect oscillations in the algorithm execution. Besides, application constraints are modeled as a penalty function, so real-time deadlines are not always guaranteed. Finally, global information must be shared between many NPUs, which could lead to undesired traffic in the interconnection network. To overcome these limitations, we attempt to apply gradient methods with consensus concepts in order to implement a cooperative and dynamic approach for Open-Scale.

Consensus is derived from the research on cooperative control theory. It was developed mainly for data processing in sensor networks and for multi-agents coordination. Consensus is defined as an iterative process utilizing a predefined message-passing protocol, leading a set of communicating elements to an agreement on a value or a common behavior [54]. Similarly to [55], we intend using the consensus to reach an agreement on one state optimizing a global interest in networked system.

A mathematical framework has been proposed for distributed optimization using hybrid approaches. This framework is a combination of subgradient methods with the consensus formalism to handle distributed optimization. In this section we summarize the key aspects of this theoretical framework before reporting afterwards its benefits.

In the context of consensus/subgradient optimization, each NPU is considered as an agent. The interconnection graph G is built with respect to the NPUs' NOC connections, and the state vector of the system is proportional to operating frequencies in each PE. Distributed models and algorithms such as Consensus are naturally adapted to Open-Scale.

Within a distributed cooperative scheme, each unit adjusts its local frequency, so that power consumption of the whole system is reduced without degrading performance. Considering our benchmark application, the proposed technique provides up to 45% energy gain under latency constraint changes, and up to 80% when different standards are applied. Our experiments have also shown in [56] that our distributed model is scalable, and can handle energy efficiency in future many core platforms; the number of communicating units can be sized to increase convergence speed and optimization quality. Indeed, for a 100-processor platform, our technique has required an average of 270 consensus cycles to reach the solution. One consensus cycle requires 500 cycles to collect monitoring data from other NPUs, compute an iteration of the consensus, and transmit its data to neighboring NPUs. We have estimated that the achieved optimization is about 82% in average compared to a global offline method.

6 Conclusion and Research Perspectives

After having exposed our vision of future self-adaptive embedded systems, we have presented our open-source MPSOC called Open-Scale. It is based on a building-block, a Network Processing Unit, mainly composed of a RISC processor, its local memory, peripherals, a network interface, and a set local sensors and actuators. The Open-Scale RTOS provides classical scheduling services, task, memory and interrupt management, but also a Message Passing Interface and dedicated services to support self-adaptability. This distributed prototyping platform has been used to investigate self-adaptation mechanisms. We have reported our researches in the design of distributed monitors, PVT sensors in FPGAs, activity counters, software monitors and a distributed embedded database to collect monitored data and its API. High-level distributed control approaches based on PID, Game Theory and Consensus Theory have been implemented and simulated. Our experimental results have shown that they are able provide 40% energy savings in average under application performance constraints (throughput or latency) in a distributed manner, at run-time. The induced overhead is low and scales well thanks, to their inherent distributed nature.

There are still several research challenges to reach the goal of self-adaptability. First, an intelligent management of monitored data from hardware and software is necessary, in order to correlate and select the most relevant approaches. The fine-tuning of mixed software and hardware actuators is a second research topic, since there are many ways to adapt the system to strike the balance between performance and energy consumption. Finally, the learning capabilities of such distributed systems must be explored, in order to achieve certainly in a near future, real autonomous chips.

7 Acknowledgement

This work was supported in part by the French National Research Agency (ANR), in collaboration with CEA LETI and LIP6. The author gratefully acknowledges the contributions of many people, who helped in different ways to complete this work, the LIRMM colleagues, Lionel Torres, Gilles Sassatelli, Michel Robert, Luciano Ost, current and former PhD candidates, especially Gabriel Marchesan Almeida, Diego Puschini, Imen Mansouri, Nicolas Hébert, Nicolas Saint-Jean, Florent Bruguier, Lyonel Barthe and Rémi Busseuil.

8 References

[1] The International Technology Roadmap for Semiconductors, System Drivers Chapter, 2011, [online] http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, *"Parameter variations and impact on circuits and microarchitecture",* Design Automation Conference, 2003. Proceedings, pp. 338–342.

[3] O. Unsal, J. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, *"Impact of Parameter Variations on Circuits and Microarchitecture"*, *IEEE Micro,* vol. 26, no. 6, pp. 30–39, 2006.

[4] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, *"HS-Scale: a Hardware-Software Scalable MP-SOC Architecture for embedded Systems"*, IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07), IEEE, 2007, pp. 21-28

[5] P. Guerrier and A. Greiner, *"A generic architecture for on-chip packet-switched interconnections"*, in DATE '00: Proceedings of the 2000 Design, Automation and Test in Europe Conference and Exhibition, pages 250–256, 2000.

[6] William J. Dally and Brian Towles, *"Route packets, not wires: on-chip inteconnection networks"*, In DAC '01: Proceedings of the 38th conference on Design automation, pages 684–689, New York, NY, USA, 2001. ACM.

[7] L. Benini and G. De Micheli, *"Networks on chips: a new SoC paradigm"*, IEEE Computer, 35(1):70–78, Jan 2002.

[8] Tobias Bjerregaard and Shankar Mahadevan, *"A survey of research and practices of Network-on-chip"*, ACM Computing Surev, 38(1):1, 2006.

[9] Partha Pratim Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, *"Performance evaluation and design trade-offs for network-on-chip interconnect architectures"*, Computers, IEEE Transactions on, 54(8):1025–1040, Aug. 2005.

[10] D. Bertozzi and L. Benini, *"Xpipes: a network-on-chip architecture for gigascale systems-on-chip"*, Circuits and Systems Magazine, IEEE, 4(2):18–31, 2004.

[11] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, *"An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework"*, In ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pages 54–63, Washington, DC, USA, 2005. IEEE Computer Society.

[12] J. Pontes, M. Moreira, R. Soares, and N. Calazans. *"Hermes-glp: A gals network on chip router with power control techniques"*, In Symposium on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual, pages 347–352, April 2008.

[13] Umit Y. Ogras, Radu Marculescu, Puru Choudhary, and Diana Marculescu, *"Voltage-frequency island partitioning for GALS-based Networks-on-Chip"*, In DAC '07: Proceedings of the 44th Annual Conference on Design Automation, pages 110–115, New York, NY, USA, 2007. ACM.

[14] James Donald and Margaret Martonosi, *"Techniques for multicore thermal management: Classification and new exploration"*, In ISCA '06: Proceeding of the 33rd International Symposium on Computer Architecture, pages 78–88, 2006.

[15] Edith Beigne, Fabien Clermidy, Sylvain Miermont, and Pascal Vivet, *"Dynamic voltage and frequency scaling architecture for units integration within a gals NOC*", In NOCS, pages 129–138, 2008.

[16] Edith Beigne, Fabien Clermidy, Sylvain Miermont, Alexandre Valentian, Pascal Vivet, S Barasinski, F Blisson, N Kohli, and S Kumar, *"A fully integrated power supply unit for fine grain DVFS and leakage control validated on low-voltage SRAMs"*, In ESSCIRC'08: Proceeding of the 34th European Solid-State Circuits Conference, Edinburg, UK, Sept. 2008.

[17] Barthe L., Cargnini L. V., Benoit P., Torres L., *"Optimizing an Open-Source Processor for FPGAs: A Case Study"*, IEEE FPL'11: Field Programmable Logic and Applications (2011), Greece, pp. 551-556

[18] L. Barthe, L. V. Cargnini, P. Benoit and L. Torres, *"The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor"*, 25th IEEE International Parallel & Distributed Processing Symposium, May 16-20, 2011, Anchorage (Alaska) USA, pp. 310-313

[19] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, *"Hermes: an infrastructure for low area overhead packet- switching networks on chip"*, Integration VLSI Journal, vol. 38(1), 2004, pp. 69–93.

[20] O. Richard Herveille, *"Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores"*, Revision B.4, OpenCores, 2010. [Online]. Available at: http://www. opencores.org/

[21] S. Rhoads, *"Plasma - most mips i(tm)"* [Online]. Available at: http://www.opencores.org/project,plasma

[22] G. Kahn and D.B. MacQueen, *"Coroutines and networks of parallel programming"*, In B. Gilchrist, editor, Information Processing 77: Proceedings of the IFIP Congress 77, Toronto, Canada, August 8-12, 1977, pages 993–998. North-Holland, 1977.

[23] Busseuil R., Barthe L., Almeida G. M., Ost L., Bruguier F., Sassatelli G., Benoit P., Robert M., Torres L., *"Open-Scale: A Scalable, Open-Source NOC-based MPSoC for Design Space Exploration"*, IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, pp. 357-362

[24] M. Nourani, A. Radhakrishnan, *"Testing On-Die Process Variation in Nanometer VLSI"*, IEEE Design & Test of Computers, Volume 23, Issue 6, June 2006 Page(s):438 – 451

[25] SB Samaan, *"Parameter Variation Probing Technique"* US Patent 6535013, 2003

[26] M. Persun *"Method and apparatus for measuring relative, within-die leakage current and/or providing a*

temperature variation profile using a leakage inverter and ring oscillators" US Patent 7193427 (2007)

[27] H-J Lee, *"Semiconductor device with speed binning test circuit and test method thereof "* US Patent 7260754

[28] Z. Abuhamdeh, B. Hannagan, Jeff Remmers, Alfred L. Crouch *"A Production IR-Drop Screen on a Chip"*, IEEE Design & Test of Computers, Volume: 24, Issue: 3, pp. 216-224, 2007

[29] A. Drake et al. *"A Distributed Critical Path Timing Monitor for A 65nm High Performance Microprocessor",* ISSCC 2007, pp.398-399.

[30] S. Lopez-Buedo, J. Garrido, and E. Boemo, *"Dynamically inserting, operating, and eliminating thermal sensors of FPGA- based systems",* IEEE Transactions on Components and Packaging Technologies, vol. 25, no. 4, pp. 561–566, 2002.

[31] A. Drake, *"Adaptive Techniques for Dynamic Processor Optimization",* Series on Integrated Circuits and Systems. Boston, MA: Springer US, 2008.

[32] Bruguier F., Benoit P., Torres L., *"Investigation of Digital Sensors for Variability Characterization on FPGAs"*, ReCoSoC'10: 5th International Workshop on Reconfigurable Communication-Centric Systems on Chip, France, pp. 95-100

[33] Ganesan, K.; John, L.; Salapura, V.; Sexton, J.; , *"A Performance Counter Based Workload Characterization on Blue Gene/P",* Parallel Processing, 2008. ICPP '08. 37th International Conference on , vol., no., pp.330-337, 9-12 Sept. 2008

[34] J.M.May, *"MPX: Software for multiplexing hardware performance counters in multithreaded programs",* in Parallel and Distributed Processing Symposium, Proceedings 15th International, p. 8, April 2001

[35] K. Jihong and K. Yongmin *" Performance Analysis and Tuning for a Single-Chip Multiprocessor DSP",* IEEE Parallel Distrib. Technol. , vol.5 , pp. 68-79, Jan 1997 . Los Alamitos, CA, USA.

[36] K. Hyun-min et al. *" Performance monitor unit design for an AXI-based multi-core SoC platform",* Proceedings of the 2007 ACM symposium on Applied computing, SAC '07,pp. 1565-1572, Seoul, Korea.

[37] E. Faure, G.M. Almeida, M. Benabdenbi, P. Benoit, F. Clermidy, F. Pêcheux, G. Sassatelli, L. Torres, *"An in-memory monitoring database for self adaptive MP2SoCs"*, Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on, 2010, pp. 97 – 104

[38] Niyogi, K. and Marculescu. *"Speed and voltage selection for GALS systems based on voltage/frequency islands",* ASP-DAC '05. ACM, New York, 292-297

[39] Deniz, Z.T.; Leblebici, Y.; Vittoz, E.A., *"On-Line Global Energy Optimization in Multi-Core Systems Using Principles of Analog Computation",* Solid-State Circuits, IEEE Journal of vol.42, no.7, pp.1593-1606, July 2007

[40] S. Madduri, et al., *"A monitor interconnect and support subsystem for multicore processors",* in the Proc. of the IEEE/ACM Design Automation and Test in Europe Conference, Nice France, pp. 761-766, 2009

[41] Mutapcic, A. et al., *"Processor speed control with thermal constraints".* Trans. Cir. Sys. Part I 56, 9 (Sep. 2009), 1994-2008.

[42] H. Jung and M. Pedram, *"Uncertainty-Aware Dynamic Power Management in Partially Observable Domains",* IEEE Trans. on VLSI Systems, 2009.

[43] Q.Wu,P.Juang, *et al.* , *"Formal online methods for voltage/frequency control in multiple clock domain microprocessors",* SIGARCH Comput. Archit. News, vol. 32, pp. 248–259, October 2004

[44] Y. Zhu and F. Mueller, *"Feedback EDF scheduling exploiting hardware- assisted asynchronous dynamic voltage scaling",* SIGPLAN Not., vol. 40, pp. 203–212, June 2005

[45] U. Y. Ogras, R. Marculescu, and et al., *"Variation-adaptive feedback control for networks-on-chip with multiple clock domains",* Proceedings of the 45th annual Design Automation Conference (DAC'08), pp. 614–619, June 2008

[46] A. Sharifi, H. Zhao, and et al., *"Feedback control for providing qos in noc based multicores",* Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10), pp. 1384–1389, March 2010

[47] G. Almeida, R. Busseuil, L. Ost, F. Bruguier, G. Sassatelli, P. Benoit, L. Torres, M. Robert, *"PI and PID Regulation Approaches for Performance-Constrained Adaptive Multiprocessor System-on-Chip"*, Embedded Systems Letters, IEEE, Volume: PP, Issue:99, ISSN: 1943-0663, DOI: 10.1109/LES.2011.2166373, September 2011, pp. 1-4

[48] M.J. Osborne and A. Rubinstein, *"A Course in Game Theory"*. MIT Press, 1994

[49] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Adaptive energy-aware latency-constrained DVFS policy for MPSoC"*, 2009 IEEE International SOC Conference (SOCC), IEEE, 2009, pp. 89-92

[50] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Dynamic and distributed frequency assignment for energy and latency constrained MP-SoC"*, Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09., 2009, pp. 1564-1567

[51] D. Puschini, P. Benoit, F. Clermidy, G. Sassatelli, *"A Game-Theoretic Approach for Run-Time Distributed Optimization on MP-SoC"*, International Journal of Reconfigurable Computing, Volume 2008, 403086 (2008), pp. 1-10

[52] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory"*, Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual, 2008, pp. 375-380

[53] I. Mansouri, P. Benoit, D. Puschini, L. Torres, F. Clermidy, G. Sassatelli, *"Dynamic Energy Optimization in NoC-based System-on-Chips"*, Journal of Low Power Electronics JOLPE – Vol. 6, N° 4, December 2010, pp. 564-577

[54] Olfati-Saber, J. A. Fax, and R. M. Murray. *"Consensus and cooperation in networked multi-agent systems,"* Proceedings of the IEEE, 95(1): 215233 (2007).

[55] Johansson, B. and al.,"Subgradient *methods and consensus algorithms for solving convex optimization problems"*. Decision and Control, CDC , Dec. 2008

[56] Mansouri I., Clermidy F., Benoit P., Torres L., *"A Run-time Distributed Cooperative Approach to Optimize Power Consumption in MPSoCs"*, SOCC'10: 23th IEEE International SOC Conference, United States, pp. 25-30

Identifying Data-Dependent System Scenarios in a Dynamic Embedded System

E. Hammari[1], F. Catthoor[2], P. G. Kjeldsberg[1], J. Huisken[3], K. Tsakalis[4] and L. Iasemidis[4]
[1]Dept. of Electronics and Telecom., Norwegian Univ. of Science and Technology, Trondheim, Norway
[2]imec and Katholieke Univ. Leuven, Leuven, Belgium; [3]imec/Holst Centre, Eindhoven, The Netherlands
[4]Dept. of Electrical Engineering, Arizona State Univ., Tempe, AZ, USA

Abstract— *System scenario-based design methodologies are applied to reduce the costs of dynamic embedded systems. At design-time, the system is optimized for a set of system scenarios with different costs, e.g., alternative scheduling of tasks. At run-time, certain parameters are monitored, scenario changes are detected, and mapping and scheduling are reconfigured accordingly. In this process, optimized identification of parameters and system scenarios is essential. Previously, the parameters have been limited to control variables, or variables with a limited number of distinct values. This paper presents a scenario identification approach based on polyhedral partitioning of the parameter space for systems where the parameters may have a wide range of data-dependent values. We evaluate our approach on a biomedical application. The results indicate that with 20 system scenarios the average execution time cost can be reduced with a factor 3 and brought within 15% of the theoretically best solution for the workload-adaptive designs.*

Keywords: Application-specific embedded systems, run-time reconfiguration, system scenario-based design

1. Introduction

Increasingly, modern applications are becoming dynamic resulting in input data dependent variations in system costs, e.g., execution time and energy consumption. An over dimensioned solution based on a few extreme workloads can be very costly, or even impossible to implement, and a workload-adaptive reconfigurable design will be necessary [14].

System scenario based design methodologies [5] provide a systematic way of constructing workload-adaptive embedded systems and have been successfully applied to multiple designs in multimedia and wireless domains [3], [11], [12], [13], [15], [17], [18]. Through structural analysis and profiling of the application code at design-time, a set of system scenarios with different costs is identified along with the parameters that determine the cost variations. The system is then separately optimized for each system scenario and augmented with a scenario predictor and switching mechanism. At run-time, the active system scenario is predicted up front from the actual parameter values and the system is switched to the most cost-optimal configuration for this system scenario.

Note, that system scenarios are conceptually different from the more common use-case scenarios. While both of them aim at reducing the total costs, use-case scenarios are extracted from the obvious system parameters, modes or usage pattern which can be detected without detailed knowledge of the algorithmic implementation. System scenarios are identified from the observed costs and then characterized in terms of implementation parameters. System scenarios do not depend on obvious parameters, modes or usage patterns and can hence be efficiently applied even if the application do not contain any of them.

This paper targets the scenario identification technique in system scenario based design methodologies, in particular for systems having parameters with widely varying data-dependent values. Existing techniques assume that the parameters are control variables and/or that they have a limited number of possible parameter values. They make use of enumeration and apply a bottom-up approach to cluster these values into system scenarios [6], [4].However when the parameters are data-dependent, they may have thousands or even millions of possible data values making bottom-up clustering and enumeration-based prediction impractical (see Section 3). Our method should then instead be used because it performs a scalable top-down polyhedral partitioning of the parameter space. This is our first main contribution.

Secondly, we apply our scenario identification technique to a real application and demonstrate the feasibility of our approach for different number of system scenarios.

The paper is organized as follows. Section 2 gives a motivating example for our work. In Section 3 the existing techniques for scenario identification are reviewed and the necessary terminology is introduced. Our proposed approach for scenario identification is detailed in Section 4. Experimental results are presented in Section 5, followed by our conclusions and plans for future work.

2. Motivational example

Recent biomedical applications for outpatient care have a dynamic nature and are at the same time subject to strict cost constraints. They continuously monitor patient's signals for

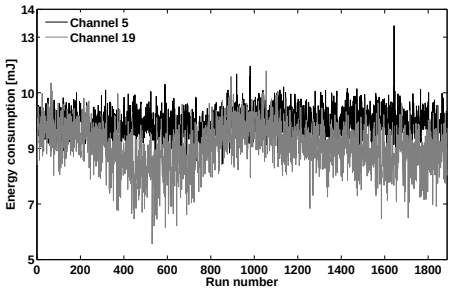


Fig. 1: Energy consumption of Lyapunov exponent calculation in 6 hours EEG recording.

an anomaly and perform specific tasks when it is detected. They may use complex signal processing on multiple channels and are required to be powered by a battery for months or even years. One such example is an epileptic seizure predictor, [7], [8], which tracks EEG signals from up to 32 channels and may warn patients of upcoming seizures hours in advance. A front-end part of this predictor performs calculations of Lyapunov exponents for each channel once every 10 seconds. Figure 1 shows the variations in the energy consumption of different runs of the Lyapunov exponent calculation of two channels over a six hour period. Due to different EEG input data, the energy consumption of one calculation can vary widely from 6 mJ to 13 mJ. The peak energy consumption for this application occurs only once in the 6 hours long EEG recording. A system designed based on this worst case energy consumption will consume 829 J/channel while processing the recording.

An ideal workload-adaptive system is able to reconfigure the system optimally in each run so that it consumes the minimum amount of energy possible. A heavily optimized thread-level workload-adaptive design for the Lyapunov exponent calculation will require only 567 J/channel for the same recording. However it can never be built in practice as the costs of reconfiguring such system, storing the different configurations and predicting them would be excessive.

A system scenario-based design methodology uses the same concept of adaptively reconfiguring the system, but allows only a limited set of possible configurations. A given system scenario has a fixed system cost corresponding to its system configuration. It contains the group of runs for which this configuration is better than any of the other configurations in the limited number of scenarios. For most runs the system will then require a small energy overhead compared with running on the optimal configuration. E.g., with 10 scenarios, the Lyapunov exponent calculator will consume 594 J/channel processing the EEG recording above. That is, somewhat more than the ideal workload-adaptive system, but far less than the system based on the worst case energy consumption. There will be an added energy consumption related to the scenario detection and reconfiguration, but this can be kept low if the guidelines for scenario based design is followed [5].

The Lyapunov exponent calculator is a good example of an applicatin were more traditional use-case scenarios can not be applied. It repeatedly performs the same calculation on equal-sized packages of input data. A system scenario approach can, however, be used to exploit the potential benefits of a reconfigurable platform.

The system scenario-based design methodology is a powerful tool that can also be used for fine grain optimizations at the task abstraction level and for simultaneous optimization of multiple system costs. The ability of handling multiple and nonlinear system costs differentiates system-based design methodologies from the dynamic run-time managers intended for DVFS type platforms [10]. DVFS methodologies concentrate on optimization of a single cost - the energy consumption of the system, that scales monotonically with frequency and voltage. They perform direct selection of the system reconfiguration from the current workload situation. This, however, cannot be generalized for costs that depend on the parameters in a nonuniform way. That makes the decision in one run-time step too complex. Scenario-based design methodologies solve this problem by a two-stage approach decided both at run-time: they first identify what scenario the working situation belongs to and then choose the best system reconfiguration for that scenario. Since the relationship between the parameters and the costs will in practice be very complex, the scenario identification is however performed at design-time.

This paper targets fine grain optimization of a single system cost.

3. Theory and related work

The term *Run-Time Situation (RTS)* is an important concept used in task level system scenario-based design methodologies [5]. Each instance of running a task has a corresponding cost (e.g., energy consumption). The run instance and its cost is treated as a unit denoted an RTS. One complete run of the application on the target platform represents the sequence of RTSs.

A scenario identification technique lies at the heart of any system scenario-based design methodology. It determines how the different observed RTSs should be divided into groups with similar costs - the system scenarios, and how the system scenarios should be represented to make their runtime prediction as simple as possible.

Two examples of techniques for task-level scenario identification are presented in [6] and [4]. Both of them split scenario identification into two steps. In the first step, the variables in the application code are analyzed, either statically, [6], or through profiling of the application with a representative data set, [4]. The variables having most impact on the runtime cost of the system are determined. These variables are called RTS parameters, denoted by $\xi_1, \xi_2, \ldots, \xi_k$, and are used to characterize system scenarios

and design the scenario prediction mechanism. Typically a small set of RTS parameters is selected to keep the runtime prediction overhead low.

The output of the first step is the set of the selected RTS parameters and, for each RTS i, its RTS *signature* is given by Equation 1 below:

$$r(i) = \xi_1(i), \xi_2(i), \ldots, \xi_k(i); c(i), \quad (1)$$

containing parameter values $\xi_1(i)$, $\xi_2(i)$, ..., $\xi_k(i)$ and the corresponding task costs $c(i)$. I.e., each run instance of each task will have its own RTS signature. The number, N, of RTS signatures will hence be very large. Depending on the number of RTS parameters and how many different values each of them can take, there will be a small or large number of *different* RTS signatures. This is important for the complexity of step 2 of the scenario identification.

In the second step, the RTS signatures are divided into groups with similar costs - the system scenarios. In [6] and [4] this is done by a bottom-up clustering of RTS signatures with a resulting multi-valued decision diagram (MDD) that is used as a predictor for the upcoming system scenario. The limitation for this technique is that the size of an MDD explodes for many-valued parameters making it infeasible for the runtime prediction.

The two techniques differ in how they evaluate the impact of RTS parameters. The first one [6] is based on pure static analysis of the code and do not take into consideration the frequencies of occurrence of different RTS parameter values. It may therefore produce system scenarios that almost never occur. The second one [4] extends the first one [6] with profiling information and forms a system scenario set that exploits runtime statistics. Our scenario identification technique uses the same approach for the selection of RTS parameters as the second technique [4]. This approach typically leads to only a limited amount of parameters being labeled as important enough to incorporate in the identification step. That is crucial to limit the complexity.

Identification of thread-level system scenarios has been studied in [16]. This is fully complementary to the focus of our paper which considers intra-thread-level system scenarios.

As we have seen the existing scenario identification approaches cannot be used in a system with many-valued RTS parameters, causing an explosion of states in the MDD and the associated runtime/implemenation costs. In the next section we will discuss a possible solution to this problem.

4. Proposed method

4.1 General overview

Figure 2 illustrates the theoretical concepts of our scenario identification technique. Given k RTS parameters and N profiled RTS signatures from Equation 1. If we assign one dimension to each RTS parameter, the resulting k-dimensional

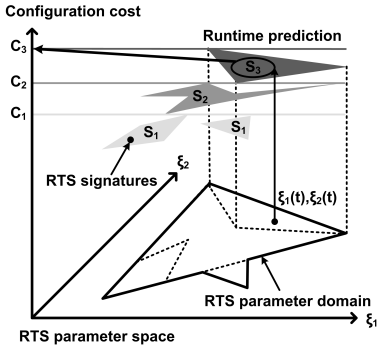


Fig. 2: System scenarios and runtime prediction.

space will define all theoretically possible values for the RTS parameters in the application. We will call such space an *RTS parameter space*. When static max and min constraints on the values are added, the space reduces to one or several k-dimensional domain(s).

Assuming a $(k+1)$-dimensional cost representation for each RTS, all signatures can then be plotted as points in a $(k+1)$-dimensional space. In the profiling sequence, several identical signatures may exist, giving coinciding points in the space. The number of times a point repeats itself is useful information as it quantifies the probabilities of occurence of each RTS.

With the representation above, the scenario identification task can be viewed as a distribution of points into S different groups, representing system scenarios, according to which the overall configuration cost is minimized. An RTS point i is assigned to scenario j whenever its cost $c(i)$ falls into that scenario's cost range $\{C(j)_{min}, C(j)_{max}\}$. The scenario cost ranges are determined by a balancing function that ensures that all scenarios have a near-equal probability to occur at run-time. In this way, rare system scenarios are avoided since their storage cost will exceed the gains of adding them. We measure this probability by the number of points, including the repeating ones, that each scenario contains and call it *scenario size*. The maximum scenario size equals the number of all RTS signatures N, divided by the number of system scenarios S. Given a list r of RTS signatures sorted by descending cost, the scenario cost ranges are given by the indices corresponding to the integral number of the scenario size:

$$C(j)_{max} = r\left((j-1) \cdot N/S + 1\right) \quad (2)$$
$$C(j)_{min} = r\left((j) \cdot N/S\right) \quad (3)$$

The *cost* of scenario j is defined as the maximum cost of any of the RTS signatures that it includes: $C_j = C(j)_{max}$.

The projection of scenarios onto the RTS parameter space (see Figure 2) will produce $M \geq S$ regions that characterize the system scenarios in terms of RTS parameter values. Each region can be described as a polyhedron, and the runtime scenario prediction can be done by checking which polyhedron contains the RTS parameter values of the next

RTS. Since we know which scenario the region belongs to, we can foresee that the next running cost will be no more than the cost of that scenario. Checking if a point lies inside a polyhedra is the classical point location problem from the computational geometry domain, and the advantage of using it for prediction instead of MDD is that it operates on/stores only the vertices of polyhedral regions, not all possible parameter values.

This top-down approach can handle arbitrary large domains, provided that the number of regions stays reasonably low. Otherwise prediction overhead will grow. The number of regions depends on the number of system scenarios and the underlying structure of the system - the relationship between the cost locality of RTS points and the value locality of their RTS prameters.

The desired number of system scenarios is best defined by the user according to the characteristics of the application domain. Typically this is limited to a few tens because beyond that the potential gains in better following the system dynamics are counterbalanced with the additional cost complexity of detecting and exploiting the (too) large set of possible system scenarios.

For the biomedical application that we are investigating, a strong correlation is present in the locality of the RTS parameter values and the locality of the corresponding costs on the target DSP platform, resulting in a single region per scenario (see Figure 5). The locality of parameters and the corresponding costs is an important prerequisite for the efficiency of the current scenario identification technique. Moreover, we assume that that there is a one-to-one correspondence between the cost of a scenario and the system configuration. In other words, we target the systems were scenarios are mostly defined by the application characteristics and not by the details of the platform.

4.2 Detailed algorithm

Our scenario identification algorithm, GENERATESCENARIOSET, is presented in Figure 3. Line 2 is a preprocessing step, where profiled RTS signatures are sorted by their costs starting from the maximum cost. In line 4 the worst case system scenario is created. In lines 6 to 8, the system scenario is filled in with signatures having the next costs in the sorted sequence. When the size of the system scenario exceeds the maximum allowable size, a new system scenario is created (line 17).

Finally, in lines 10 to 15, each completed system scenario is checked for overlap with previously calculated higher cost system scenarios. An overlap means that the scenario regions in the RTS parameter space are not disjoint (see example on Figure 6), and equals the intersection of the regions: *overlap* $\leftarrow$ *scenario1.paramRegion* $\cap$ *scenario2.paramRegion*.The intersections make prediction of scenarios ambiguous and have to be eliminated. This is done by subtracting the overlap region from the lower cost system scenario and moving all signatures in the overlap region to the higher cost system scenario. Given $C_1 > C_2$, these operation can be written as:
scenario2.paramRegion $\leftarrow$ *scenario2.paramRegion - overlap*
scenario1.paramRegion $\leftarrow$ *scenario1.paramRegion* $\cup$ *overlap*

The complexity of this algorithm is calculated below:

$$\mathrm{O}(N,S) = N\mathrm{O}(\textsc{AddSignature}) + (1/2)S(S-1)(\mathrm{O}(\textsc{Overlap}) + \mathrm{O}(\textsc{AdjustBorder})) + S\mathrm{O}(\textsc{NewScenario}) \quad (4)$$

Thus, it depends on the complexity of the underlying geometric algorithms in the labelled functions.

For the fixed number d of RTS parameters, the function NEWSCENARIO has a constant complexity O(1) as it only copies the value of each parameter in a single RTS signature to a scenario region.

The function ADDSIGNATURE performs a CONVEXHULL operation on the existing border of the scenario and the projection point of the new RTS signature onto the RTS parameter space. For a 2 or 3-dimensional RTS parameter space an incremental convex hull algorithm has the complexity O($n \log n$) [9], where n is the final number of processed points, which here equals to the scenario size, N/S. The convex hull of a polygon has the expected number of $v = \mathrm{O}(\log n)$ vertices and many of them may lie very close to each other. To limit the number of vertices in the hull for faster run-time prediction, we modify the algorithm, such that it calculates the distance between the points on the hull and removes those that are closer than L/v_{max}, where L is the perimeter of the hull, and v_{max} is a user defined constraint of the maximum number of vertices in the prediction polyhedra. For the application that we investigate a reasonable value of this parameter could be 10 (see Figure 5).

The functions OVERLAP and ADJUST BORDER apply boolean set operations for intersection, difference and union of two d-polytopes. For the case $d = 2$ and $d = 3$ these operations can be done in O($v_{max} \log v_{max}$) time [2], giving the total complexity of the algorithm:

$$\mathrm{O}(N,S) = S\mathrm{O}((N/S)\log N/S) + (3/2)S(S-1)\mathrm{O}(1) + S\mathrm{O}(1)$$
$$\mathrm{O}(N) = \mathrm{O}(N \log N) \quad (5)$$

Recall that this scenario identification algorithm is run only in the design phase of the embedded system. The run-time prediction of the next scenario is equivalent to a point location problem in the polyhedral partitioning of the parameter space. The time complexity of the point location problem is O($\log v_{tot}$), where $v_{tot} = S \cdot v_{max}$ is the total number of vertices in the partitioning. The required memory space is O($v_{tot} \log v_{tot}$). To compare, an MDD for d parameters with l distinct values has l^d states and a query time of O($d \cdot l$), where $l \gg v$.

For $d > 3$, i.e., for systems with more than 3 parameters, the complexity of convex hull, boolean set operations and the point location algorithm, increases exponentially in d [2], similar to MDD. It remains an open research area to find

```
GENERATESCENARIOSET(SET rtsSignatures, INT S)
 1  SCENARIO solutions ← ∅
 2  SORTBYCOST(rtsSignatures)
 3  wcSignature ← rtsSignature(1)
 4  currentScenario ← NEWSCENARIO(wcSignature)
 5  MAXSCENARIOSIZE := N/S
 6  for signature in rtsSignatures do
 7     if (currentScenario.size < MAXSCENARIOSIZE) then
 8        currentScenario.ADDSIGNATURE(signature)
 9     else
10        for scenario in solutions do
11           overlap ← OVERLAP(scenario, currentScenario)
12           if (overlap ≠ ∅) then
13              ADJUSTBORDER(currentScenario, overlap)
14           end if
15        end for
16        solutions.INSERT(currentScenario)
17        currentScenario ← NEWSCENARIO(signature)
18     end if
19  end for
20  return solutions

NEWSCENARIO(RTSSIGNATURE signature)
   SCENARIO new
   new.size ← 1
   new.cost ← signature.cost
   new.paramRegion ← signature.paramValues
   return new

ADDSIGNATURE(RTSSIGNATURE signature)
   thisScenario.size ← thisScenario.size + 1
   thisScenario.paramRegion ←
                  CONVEXHULL(thisScenario.paramRegion,
                             signature.paramValues)
   return
```

Fig. 3: Scenario identification algorithm

an efficient representation of polytopes in higher dimensions that will decrease the complexity of the algorithms operating on them. The complexity does not increase exponentially when the number of possible parameter values increase, however, the way it does for MDD. Thus, our algorithm is efficient for the systems with upto three important data-variables that determine the data-dependent behaviour of the system and have a significant number possible values.

4.3 Refining system scenarios

The function ADDSIGNATURE in Figure 3 produces convex scenario projections in the RTS parameter space. An example of convex scenarios is shown on Figure 5. However, in some situations concave scenario projections, representing a geometrically tighter envelop of a set of points, are preferrable. This is the case when: a) the inherent correlation between the RTS parameter values and the corresponding costs has a concave shape, b) the system scenarios overlap in the RTS parameter space and complete migration of the signatures to the higher cost system scenarios results in considerable reduction of run-time gain. An example of concave scenarios is presented in Figure 6.

A concave scenario projection reduces the overlap and can potentially improve the run-time gain. However, large overhead may incur since algorithms processing concave polyhedra are much more complex. A possible solution is to split the concave projection into a set of convex polyhedra at design-time and apply convex hull algorithms. The separate polyhedra require still additional storage and processing time that should be kept low. To achieve that, a restriction must be made on the number of reflex angles v_r in the concave projection, and also a careful consideration of the cost tradeoffs is required. The refinement step is performed before the OVERLAP and ADJUST BORDER functions in Figure 3 and currently includes only a geometric refinement of the border by manual selection of additional vertices. The cost tradeoff considerations are the goals of our future work.

It should be noted that scenario overlaps may also be produced by variables affecting the costs, but not selected as RTS parameters, or by nondeterministic properties of the underlying platform, resulting in different costs for the same RTS parameters. Such overlaps indicate either a faulty RTS parameter selection or the use of hardware components unsutable for scenario-based design.

5. Experiments and Conclusions

We have evaluated our scenario identification algorithm on two versions of the Lyapunov exponent calculator described in Section 2. Energy numbers in Section 2 are obtained using the CoolBio DSP platform, presented in [1], and have been extracted through layout back-annotated power simulations. In our case we use the high performance mode, running the DSP on 1.1V. For this voltage we reach 80MHz speed which is required for this application in the worst-case condition.

In this section we present results for execution time optimization using system scenarios. The improved execution time can be exploited for reconfiguration in several ways. DVFS can be applied, possibly in combination with rescheduling to allow other tasks to run in the idle time. On run-time reconfigurable multi-processor platforms, remapping of tasks is possible to achieve an overall optimized execution.

We have run tests on three different setups, displayed in Table 1. Throughout the tests we have varied: a) the version of the application, i.e. different settings for the Lyapunov exponent calculation, b) the platform, on which the execution time was measured, and c) the input database for application profiling. This results in different characteristics which represent distinct benchmarks to test our algorithm performance. Figures 5 and 6 show the results of the experiments. Prior to scenario identification, an RTS parameter selection step, similar to [4], was performed, and two internal application variables with the greatest impact on the execution time were identified - *Falsecount* and *Seqlength*. The same variables were identified in both application versions and they have upto a few thousands of distinct values. They were selected as RTS parameters, giving a two-dimensional RTS parameter space for our scenario identification algorithm.

Table 1: Experimental setup.

No	Application version	Platform	Database
1	I, settings: nsize=2048, dimm=7, evolv=12, idist=20, tau=4	CoolBio DSP	6 hrs continuous EEG w/seizures
2	II, settings: nsize=1000, dimm=4, evolv=6, idist=12, tau=4	CoolBio DSP	6 hrs continuous EEG w/seizures
3	I, settings: nsize=2048, dimm=7, evolv=12, idist=20, tau=4	General purpose	200 EEG samples from epileptogenic zone, no seizures

The results of the first two experiments are presented on Figure 5. Here both application versions are investigated on the potential target embedded platform, CoolBio DSP. A profiling of the application with a 6 hrs continuous EEG recording shows that there is a clear correlation between the RTS parameters and the execution workload (in clock cycles), and our scenario identification algorithm produces a set of non-overlapping system scenarios. The two application versions have different size of the RTS parameter domain, but the identified system scenarios appear to be very similar, approximately a scaled version of each other. A relatively small number of 5 to 20 system scenarios is generated as preferred by users in scenario-based systems.

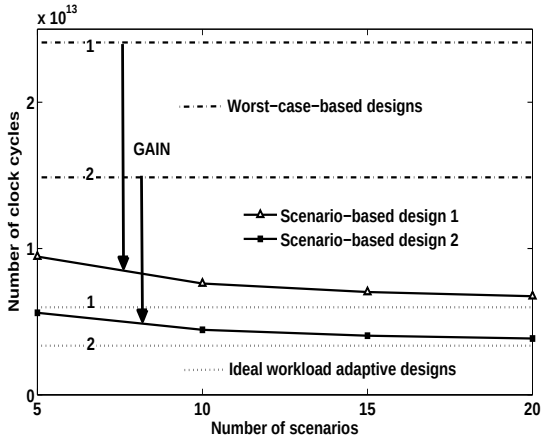


Fig. 4: Total execution time of the largest Lyapunov exponent calculation (versions I and II) with different number of system scenarios in the 6 hours continuous EEG recording.

The last experiment is run on a nondeterministic general-purpose desktop PC to demonstrate the scenario refinement step. Figure 6 presents the results. The top left subfigure shows the overlapping convex scenario projections produced by function ADDSIGNATURE on this nondeterministic platform. The remaining subfigures show the projection of each scenario separately along with its respective RTS point distribution. Concave refinement is performed on scenarios 2-5 in order to reduce the overlaps between the scenarios before the functions OVERLAP and ADJUSTBORDER are applied. Although a much smaller database is used for this experiment, the real scenario borders from Figure 5 can be discerned in the point distributions of Figure 6. The distortion in the scenario borders is caused by the noise from the nondeterministic platform. The dashed line indicates the convex hull of each scenario. It is determined by the extreme points in the distribution and causes strong overlap between the scenarios. In fact, the overlap is so big, that after application of OVERLAP and ADJUSTBORDER functions some of the scenarios would disappear totally. The solid line is the concave hull of the scenarios and comes significantly closer to the real scenario borders, reducing the overlap between them. If the point distributions here were the inherent point distributions for this application (i.e. not caused by the platform noise), the identified concave scenarios would improve the execution time of this system scenario-based design.

Figure 4 compares execution workload for running the system with and without system scenarios for the first two experiments. It also shows the execution workload of the theoretically optimal workload-adaptive design which is not realizable in practice. The results are presented for both application versions. Between 61% and 72% gain can be achieved with 5 to 20 system scenarios. With 5 system scenarios the total execution workload of the systems is still situated well above the theoretically best solution - 58% for system 1 and 67% for system 2. When the number of scenarios is increased, however, the total execution time of both systems reduces towards the theoretical limit and becomes at 20 scenarios less than 13% and 15% above the theoretically best solution for system 1 and system 2 respectively.

The results presented here demonstrate the feasibility of the proposed technique and show that it is possible to reach near optimal execution time with a limited number of scenarios. Future work includes optimization of scenario borders to realize a trade-off between overestimation and run-time prediction / switching complexity.

References

[1] M. Ashouei et al., "A voltage-scalable biomedical signal processor running ecg using 13pj/cycle at 1mhz and 0.4v," in *Proc. ISSCC'11*, 2011, pp. 332–334.

[2] M. J. Atallah and M. Blanton, Eds., *Algorithms and theory of computation handbook: special topics and techniques*, 2nd ed., NY, USA: Chapman & Hall/CRC, 2010.

[3] B. Geelen., "Low-power, Wavelet-based Applications in Dynamic Environments", PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Dec. 2008.

[4] S. V. Gheorghita, T. Basten, and H. Corporaal., "Scenario selection and prediction for dvs-aware scheduling of multimedia applications", *J. Signal Process. Syst.*, vol. 50, pp. 137–161, Feb. 2008.

[5] S. V. Gheorghita et al., "System-scenario-based design of dynamic embedded systems", *ACM Trans. Des. Autom. Electron. Syst.*, vol.14, paper 3, pp. 1–45, Jan. 2009.

[6] S. V. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal, "Automatic scenario detection for improved wcet estimation", in *Proc. DAC'05*, 2005, pp. 101–104.

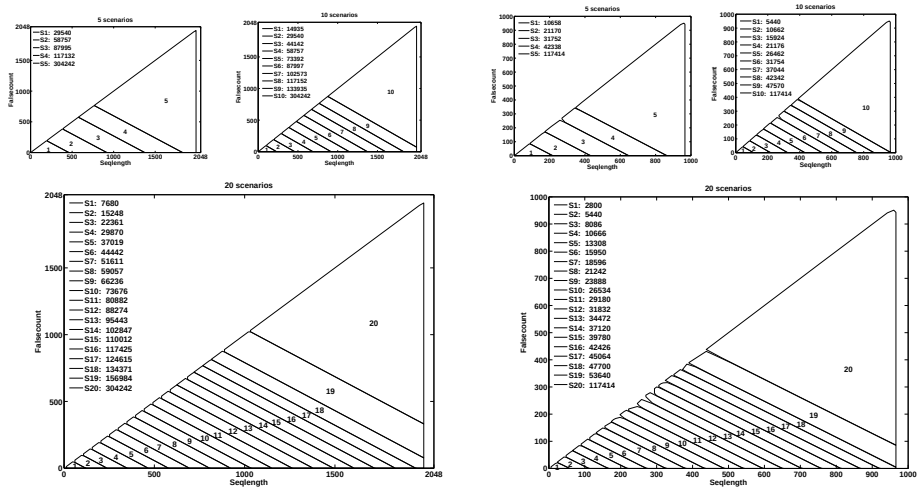


Fig. 5: System scenarios for the application version I(left) and II(right). Numbers in the top left corners are the execution times of scenarios in clock cycles

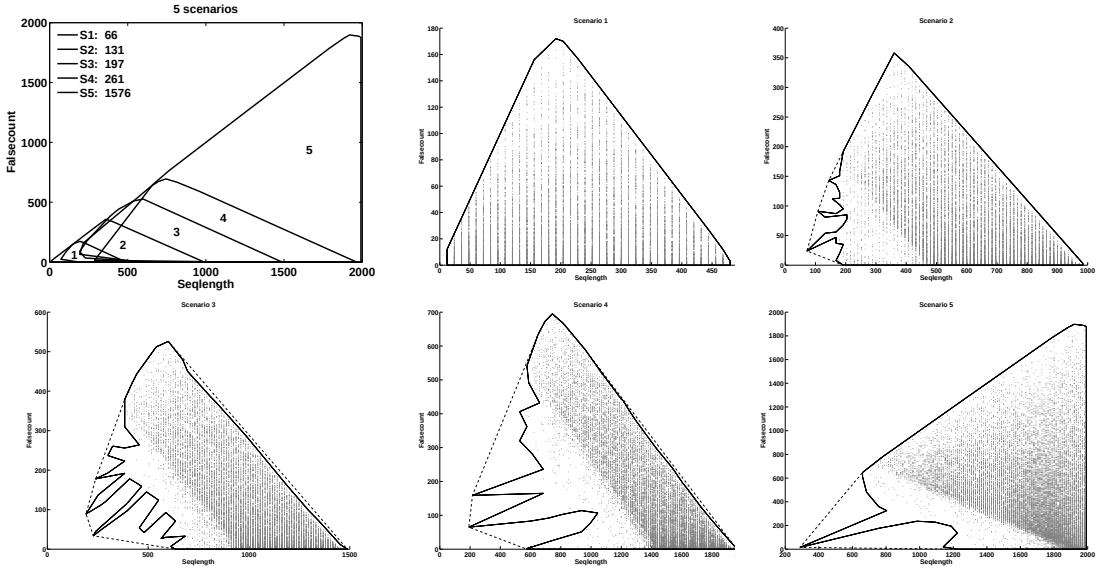


Fig. 6: Overlapping convex scenario projections and their concave refinement. Numbers in the top left corner are the execution times of scenario in μs.

[7] L. Iasemidis, "Seizure prediction and its applications", *Neurosurg. Clin. N. Am.*, vol. 22, pp. 489–506, Oct. 2011.

[8] L. Iasemidis et al., "Long-term prospective on-line real-time seizure prediction", *Clinical Neurophysiology*, vol. 116, pp. 532–544, Mar. 2005.

[9] M. Kallay., "The complexity of incremental convex hull algorithms in rd", *Information Processing Letters*, vol. 19, paper 4, p. 197, Nov. 1984.

[10] Y. Liu and H. Zhu, "A survey of the research on power management techniques for high-performance systems", *Softw. Pract. Exper.*, vol. 40, pp. 943–964, Oct. 2010.

[11] Z. Ma, "Interleaved Subtask Scheduling on Multiprocessor SoCs", PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Jan. 2006.

[12] S. Mamagkakis, D. Soudris, and F. Catthoor, "Middleware design optimization of wireless protocols based on the exploitation of dynamic input patterns", in *Proc. DATE'07*, 2007, pp. 1–6.

[13] N. R. Miniskar et al., "Scenario based mapping of dynamic applications on mpsoc: A 3d graphics case study", in *Proc. SAMOS'09*, 2009, pp. 48–57.

[14] D. Raskovic and D. Giessel,"Dynamic voltage and frequency scaling for on-demand performance and availability of biomedical embedded systems", *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, pp.903–909, Nov 2009.

[15] M. Steine et al., "Proactive reconfiguration of wireless sensor networks", in *Proc. MSWiM'11*, 2011, pp. 31–40.

[16] P. v. Stralen, "Scenario Based Design Space Exploration", PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, Sep. 2009.

[17] D. Stroobandt and K. Bruneel, "How parameterizable run-time fpga-reconfiguration can benefit adaptive embedded systems", in *Proc. ERSA'11*, 2011, pp. 184–194.

[18] N. Zompakis et al., "Enabling efficient system configurations for dynamic wireless baseband engines using system scenarios", in *Proc. SIPS'11*, 2011.

SESSION

HARDWARE SECURITY AND TRUST IN RECONFIGURABLE HETEROGENEOUS SYSTEMS

Chair(s)

Prof. Tim Guneysu
Ruhr-Universitaet Bochum
Germany

Tackling the Security Issues of FPGA Partial Reconfiguration with Physical Unclonable Functions

Yohei Hori[1,2], **Toshihiro Katashita**[1,2] **and Akashi Satoh**[1]
[1]National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Ibaraki, Japan
[2]CREST, Japan Science and Technology Agency, Chiyoda-ku, Tokyo, Japan

Abstract— *Protecting the confidentiality and integrity of data processed by field-programmable gate arrays (FPGAs) is a major concern in the field of electronics as the use of FPGAs is becoming increasingly widespread in various commercial, industrial and other products. Since the FPGA bitstream is essentially an electronic data stream, it is susceptible to eavesdropping and tampering during transport via a data bus or network. Such security issues clearly hinder the use of systems supporting partial reconfiguration, where users can design their own circuits or download and implement custom circuits from the Internet on demand.*

Although currently available high-end FPGAs feature cryptographic cores to counteract such security issues, in 2011 it was reported that a cryptosystem on an FPGA could be broken by means of a side-channel attack. The success of this type of attack indicates that, in the presence of state-of-the-art techniques, a fixed key in the memory constitutes a flaw in security-sensitive systems.

To address the problem of side-channel analysis, we have developed evaluation boards referred to as SASEBO, which is an acronym for Side-Channel Attack Evaluation Board and serves as a collective name for the entire range of evaluation boards developed thus far, namely SASEBO, SASEBO-G, -B, -R, -GII, -W, -RII and -GIII. Used in more than 30 countries, SASEBO is the world's most popular series of evaluation boards for side-channel analysis. The latest board (SASEBO-GIII) is equipped with the newest Xilinx Kintex-7 FPGA for cryptographic module evaluation and Spartan-6 for system control.

To resolve the security issue of embedded secret keys, we are also developing and evaluating Physical Unclonable Functions (PUFs) with SASEBO. A PUF is a circuit that generates a device-specific identifier by using the process variation of the device. Such variations are virtually unclonable, and thus the output of the PUF is considered to be a device fingerprint*, which is expected to be unique among devices. The generated identifier is used to communicate secret information, and there is no need to store that information in the device itself.*

In this paper, we outline security issues in the modern large-scale integration market, and we demonstrate the functionality of SASEBO boards. We also explain how PUFs can solve the abovementioned security problems concerning FPGAs.

Keywords: Dynamic Partial Reconfiguration (DPR), SASEBO, Side-Channel Analysis (SCA), Physical Unclonable Function (PUF), Authenticated Encryption

1. Introduction

Dynamic Partial Reconfiguration (DPR), or Partial Run-Time Reconfiguration (RTR) of Field-Programmable Gate Arrays (FPGAs), refers to the ability to replace a portion of a circuit with another module while the rest of the circuit remains fully operational. FPGAs of the Xilinx Virtex family are probably the most popular dynamically reconfigurable FPGAs, and recently Altera announced that their Stratix V FPGA also supports DPR. In a DPR system, a user can change the functionality of the system on demand by downloading a hardware module suitable for particular applications, performance requirements or environments. Similarly to downloadable software, such as JavaScript and ActiveX content, downloadable hardware services for reconfigurable hardware devices are expected to become available in the near future. The flexibility of DPR is expected to increase the versatility of such hardware systems as well as to improve their cost effectiveness and area efficiency. DPR also results in shorter configuration times and consequently makes reconfigurable computing more practical and operational. The application of DPR has been studied in the fields of content distribution [1], network processing [2], image processing [3], automotives [4], fault-tolerant and self-healing systems [5], and software defined radio [6] among others.

However, there are certain security issues to consider before DPR can be applied in practice. Since hardware configuration data (bitstreams) for FPGAs can be downloaded from the Internet, the bitstreams are always exposed to attackers on the network. As represented by *Side-Channel Analysis (SCA)*, which exploits power consumption measurement or electromagnetic emanation to obtain secret keys, technology which can be used for attacks is becoming more sophisticated every day, and simple encryption and authentication techniques might not always be sufficient for protecting confidential data. Indeed, according to recent reports, the bitstream security mechanisms of some FPGAs have been defeated by differential power analysis [7], [8]. In other words, secret information stored in the memory

can be extracted by state-of-the-art attacks. Therefore, we are currently addressing this security issue with *Physical Unclonable Functions (PUFs)*, which extract unclonable process variation of the devices and provide fingerprints which uniquely identify these devices.

In the following sections, we provide a brief introduction of the security issues concerning FPGAs, SCAs and PUFs.

1.1 Security Issues Concerning FPGAs

Since a bitstream is merely an electronic data stream, it is constantly exposed to threats such as illicit cloning, reverse engineering and other forms of tampering, and therefore bitstream encryption is essential for protecting FPGA IP cores. Encryption-only systems, however, are not sufficiently secure since they cannot prevent erroneous or malicious bitstreams from being used for configuration. Since DPR changes the hardware architecture of the circuit, an unauthorized bitstream can cause fatal, unrecoverable damage to the system or may cause secret information to leak through a network connection. Such a malicious bitstream is referred to as a *hardware virus* or a *hardware trojan*. Cryptographic schemes also provide DPR systems with solutions for preventing damage from being inflicted by such hardware viruses and trojans.

To use DPR systems in practice, mechanisms for bitstream protection, safe configuration and side-channel attack prevention should be implemented in accordance with the intended application of the specific system. In this regard, we have developed a secure DPR system using the Advanced Encryption Standard with the Galois/Counter Mode (AES-GCM) [9], [10], which is one of the latest Authenticated Encryption (AE) systems [11]. AE is a cryptographic algorithm that provides both message confidentiality and authenticity. Also, several studies on bitstream protection have been reported thus far [12]–[18].

Modern cryptography provides a reasonably good solution to the security issues associated with DPR systems. However, we must also take SCA into consideration in order to be able to counteract more sophisticated attacks since the secret key of the encryption core embedded into the FPGA can be revealed by SCA.

1.2 Side-Channel Analysis

SCA is a collective term for a range of non-invasive attacks targeting cryptographic modules which focuses on the power consumption, electromagnetic emanation, and the leakage of other information about the physical state of electronic devices. Using SCA, an attacker can extract secret information from inside the target without physically accessing the device. The cost of SCA attacks is usually low, requiring only basic equipment, such as a digital oscilloscope and a personal computer along with the target device. Therefore, SCA is a rather straightforward but powerful attack technique targeting cryptographic modules.

After Kocher et al. reported the first SCA (based on timing analysis) in [19] and subsequently *Simple Power Analysis (SPA)* and *Differential Power Analysis (DPA)* in [20], SCA has become widely recognized in both industry and academia as a serious problem concerning cryptographic modules. Many derivative attacks have been studied to date such as correlation power analysis [21], electromagnetic analysis [22], [23] and mutual information analysis [24].

In 2011, Moradi et al. successfully extracted the secret key of the encrypted bitstream from a Virtex-II Pro FPGA by recovering the three encryption keys of the Triple-DES algorithm from 25,000 power traces obtained during a single boot-up process [7]. It should be noted that the technique adopted by Moradi et al. required only 3 min to extract the key.

To facilitate the study of SCA at academic, industrial and governmental institutions, we have developed and distributed a standard experimental environment named SASEBO, or Side-channel Attack Standard Evaluation Board [25]. SASEBO is a collective name for a series of evaluation boards developed thus far, namely SASEBO, SASEBO-G, -B, -R, -GII, -W, -RII and -GIII. Used in more than 30 countries, the SASEBO boards are now the world's most popular SCA evaluation boards. It should be emphasized that SASEBO-GII and -GIII are also designed to support DPR system evaluation, where the target device can be (re)configured in various ways to study the feasibility and effectiveness of DPR systems. It is particularly important that one of the two FPGAs on these boards can be dynamically reconfigured under the control of the other FPGA. A detailed explanation of SASEBO will be given in Section 2.

1.3 Physical Unclonable Functions

In this context, storing a secret key in memory might not provide sufficient security with respect to sensitive information. Therefore, we look to PUFs as an effective solution to SCA attacks.

A PUF is an object that outputs a device-specific response based on its intrinsic physical characteristics. In this sense, the texture of paper can serve as a PUF, however here we consider PUFs in the context of semiconductors (silicon PUFs [26]). A silicon PUF (hereafter referred to simply as "PUF") is a circuit constructed on a semiconductor, and its purpose is to output a unique identifier (ID) based on variation in the device. By using a PUF for key generation, the secret key need not be embedded in the FPGA, which can protect the device against side-channel attacks. Another novelty associated with using PUFs for FPGAs is that different IDs can be generated from the *same* bitstream. Although bitstreams are common for all devices, device-specific data are generated as a result of physical differences between individual devices. Note that the bitstream itself does not necessarily include any secret information. As

a consequence, the bitstream of the PUF can be safely transferred over unsecured network channels.

Maes and Verbauwhede have categorized PUFs into non-electronic PUFs, analog electronic PUFs, delay-based intrinsic PUFs and memory-based intrinsic PUFs [27]. Among these, delay-based and memory-based PUFs can be applied to FPGAs. Examples of delay-based PUFs can be given with arbiter PUFs [28], ring oscillator (RO) PUFs [29], Glitch PUFs [30] and others, while examples of memory-based PUFs include SRAM PUFs [31], butterfly PUFs [32] and tri-state PUFs [33].

Our Pseudo-LFSR PUF (PL-PUF) [34], which is a delay-based type of PUF, was developed to eliminate certain shortcomings of existing PUFs. A conventional delay-based PUF outputs a response consisting of one or several bits from a challenge consisting of a long bitstream, and consequently has a low throughput. Additionally, some types of PUFs can be attacked by using machine learning to perform mathematical modeling of their signal delay characteristics. In contrast, PL-PUF efficiently outputs an N-bit response from an N-bit challenge, and the size of the PL-PUF circuit is reasonably small. Although the structure of PL-PUF is based on the Linear Feedback Shift Register (LFSR), in fact it does not contain a shift register; rather, it constitutes a large combinational logic. As a result of this structure, modeling its delay is considered to be exceedingly difficult. Furthermore, the challenge-response mapping of the PL-PUF can be varied depending on the active duration of the circuit, that is, a single PL-PUF behaves as though it consists of multiple PUF cores. The PL-PUF is explained in detail in Section 3.

1.4 Organization of this Paper

The remainder of this paper is organized as follows. Section 2 presents our SCA evaluation boards of the SASEBO family, where the details about the structure, functionality and various configuration mechanisms of the boards are explained. Section 3 introduces PL-PUF together with details of its structure and the results of its implementation on SASEBO-GII boards, and the effectiveness of PL-PUF is discussed on the basis of performance evaluation results. Furthermore, Section 4 presents a secure DPR system using authenticated encryption AES-GCM together with results regarding its structure and implementation. Finally, Section 5 summarizes the paper and the directions of future work.

2. SASEBO

SASEBO was our first version of an SCA evaluation board and is also the collective name for the entire series of evaluation boards developed thus far, namely SASEBO-G, -B, -R, -GII, -W, -RII and GIII. SASEBO is developed to provide an experimentation environment for SCA to researchers from various academic and industrial fields. SASEBO is currently the most common SCA evaluation board in the world, being used at more than 100 academic, governmental and industrial institutions in more than 30 countries. Images of the boards in the SASEBO family are shown in Fig. 1 through 6, and a summary of the functions of each board is given in Table 1.

After successful timing attacks and differential power analysis (DPA) were reported in 1996 and 1999 respectively, SCA attacks were recognized as a serious threat to the industry. However, at the time there was no common experimental environment for conducting SCA tests, so some research groups developed their own evaluation boards while others modified off-the-shelf boards to measure the power consumption of the chips. These experimental environments were drastically different from each other, which rendered the comparison of the experimental results obtained by different groups meaningless. Furthermore, even when a novel SCA experiment was conducted, performing independent confirmation experiments was virtually impossible since the original test environment in which the novel experiments were performed was unavailable to third-party research groups. The lack of a uniform and consistent test environment led to the development of SASEBO.

2.1 SASEBO-GIII

The latest board of the SASEBO family, SASEBO-GIII, is equipped with a Xilinx Kintex-7 FPGA as a testing cryptographic module and a Spartan-6 FPGA for implementing control logic. Its strongest advantage is in terms of expandability, with two standard FMC LPC[1] connectors. Therefore, off-the-shelf boards with an FMC connector, for example, HDMI cards, Ethernet cards and camera boards, can be connected to SASEBO-GIII. The configuration pins of Kintex-7 are connected to and controlled by Spartan-6, allowing the user to test complete and partial reconfigurations of the chip through the configuration pins.

Since SASEBO-GIII is not yet commercially available, in the following section we explain the functionality and configuration mechanisms of SASEBO-GII.

2.2 SASEBO-GII

The SASEBO-GII board is designed for developing secure DPR systems, as well as for improving the logic capacity and signal quality for advanced research on side-channel attacks. The FPGAs on SASEBO-GII can be configured via different interfaces: JTAG, SPI, SelectMAP and ICAP [35], and the designer can examine various configurations and evaluate the security of the developed configuration procedure.

In this section, first we explain the basic specifications of the SASEBO-GII board, after which we describe the various FPGA configuration patterns realized with the board.

2.2.1 Board Structure

The block diagram of the board is shown in Fig.7 and its basic features are summarized in Table 2.

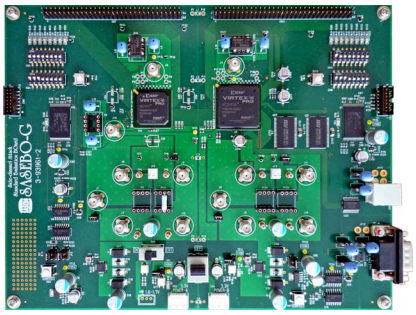


Figure 1: SASEBO-G

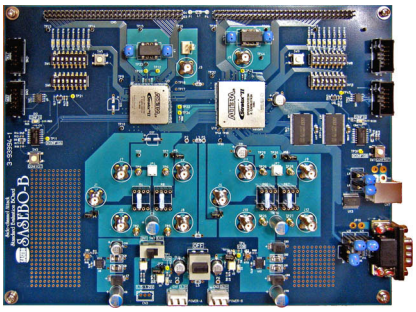


Figure 2: SASEBO-B

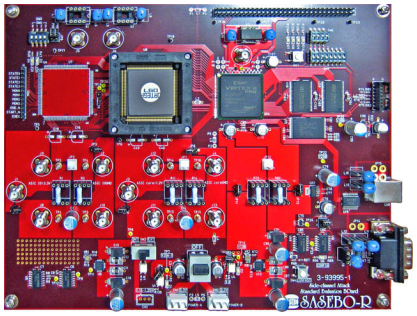


Figure 3: SASEBO-R

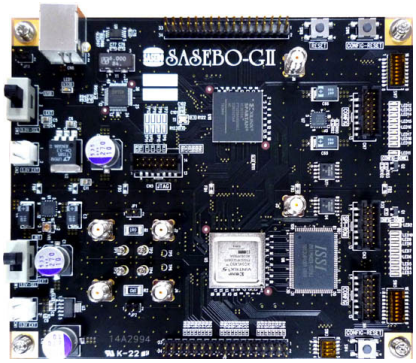


Figure 4: SASEBO-GII

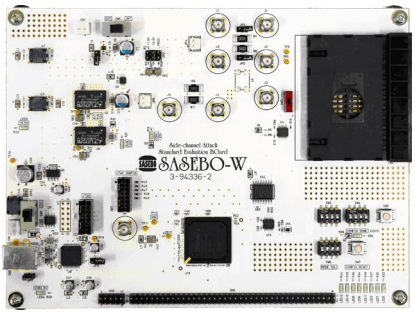


Figure 5: SASEBO-W

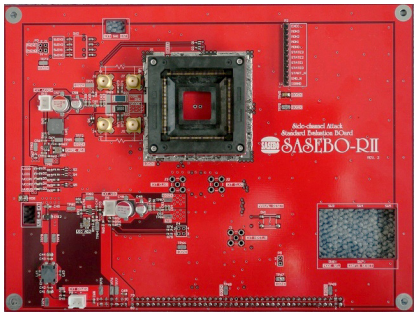


Figure 6: SASEBO-RII

Table 1: Summary of the SASEBO family.

Name	Year	Cryptographic Device	Control Device
SASEBO	2007	Virtex-II Pro (XC2VP7)	Virtex-II Pro (XC2VP30)
SASEBO-G	2008	Virtex-II Pro (XC2VP7)	Virtex-II Pro (XC2VP30)
SASEBO-B	2008	Stratix-II (EP2S15)	Stratix-II (EP2S30)
SASEBO-R	2008	LSI socket (QFP160)	Virtex-II Pro (XC2VP30)
SASEBO-GII	2009	Virtex-5 (XC5VLX30/50)	Spartan-3A (XC3S400A)
SASEBO-W	2010	Smartcard slot	Spartan-6 (XC6SLX150)
SASEBO-RII	2011	LSI socket (QFP160)	N/A
SASEBO-GIII	2012	Kintex-7 (TBD)	Spartan-6 (XC6S45LX)

Table 2: Basic specifications of SASEBO-GII

Size	$120x140x1.6$ mm^3, FR-4, six layers
Devices	xc5vlx30/50-ffg334 (for cryptographic circuit) xc3s50a-ftg (for control and interface circuit)
Power supply	5.0 V USB bus power / 5.0 V DC power supply 1.0 V internal regulators Alternative 1.0 V supply line for the FPGA
Monitoring points	Surface-mounted shunt resistors (1Ω) are inserted at the V_{CORE}, V_{IO} and GND lines
Local bus	38-bit bus between the FPGAs
I/F	USB
Clocks	24 MHz oscillator for control device JTAG, SPI-ROM, User-controllable SelectMAP, ICAP

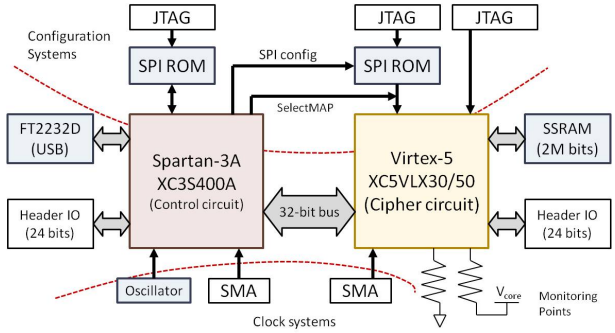


Figure 7: Block diagram of SASEBO-GII.

SASEBO-GII has two Xilinx FPGA devices—a Virtex-5 for cryptographic circuits and a Spartan-3A for interface and control circuits. Furthermore, there are two variants of Virtex-5, namely LX30 and LX50 for small and large logic circuits, respectively. Surface-mounted shunt resistors are soldered and SMA jumpers are inserted into the V_{CORE} and GND lines in order to improve the quality of power tracing. In addition, a surface-mounted device is chosen to reduce noise generated by the clock oscillator, whereas the previous SASEBO and SASEBO-G implementations use a PLL programmable crystal oscillator.

Power for operation can be supplied through the USB connector, or an external power source can also be used in cases where more stable power supply is necessary. The cryptographic device and the control device are equipped with their own V_{CORE} regulators, and the GND lines of the two parts are connected through inductors. This architecture also contributes towards further noise reduction. The size of SASEBO-GII has been reduced to 1/3 of that of SASEBO-G by removing the RS-232 interface, the monitoring points for power consumption for the control device, the FPGA configuration sequencer and the large header pins. As shown in the block diagram in Fig. 7, the wide local bus of SASEBO-G is emulated on the Virtex-5 FPGA. This simple and compact implementation also improves the quality of power tracing since it reduces the number of parasitic capacitances and resistances. In spite of its simplicity, SASEBO-GII provides high compatibility with SASEBO-G, and the same Verilog-HDL source code and control software [36], [37] designed for SASEBO-G can be used without modification for SASEBO-GII.

2.2.2 FPGA Configuration

SASEBO-GII allows for user-controllable configuration, where bitstreams are transmitted to the Virtex-5 SelectMAP interface or SPI-ROM through Spartan-3A. Thus, a JTAG cable is unnecessary for Virtex-5 configuration, although JTAG interfaces are still implemented for ordinary configuration and internal signal monitoring. Jumper pins on the board are used for selecting the configuration type.

Figure 8 illustrates the process of self-DPR of Virtex-5 through ICAP. In this case, a bitstream of a Partially Reconfigurable Module (PRM) is sent from the personal computer (PC) through Virtex-5. For the secure configuration, the integrity of the PRM bitstream should be checked, followed by decryption of the bitstream in the Virtex-5, after which the PRM is used to configure the device. The security of the DPR system with a single FPGA can therefore be examined with this configuration.

Figure 9 shows the configuration of Virtex-5 via the SelectMAP interface controlled by Spartan-3A. This type of configuration is useful for developing a device authentication protocol between the control logic and the FPGA. In addition to DPR, this configuration type can also be used for complete reconfiguration of the FPGA. Therefore, the security of completely reconfigurable systems can also be examined with this configuration setting.

Figures 10 and 11 show the configuration of Virtex-5 and Spartan-3A via the SPI, respectively. Configuration data are written to SPI-ROM through the JTAG interface or the FPGA. SPI-ROM is usually used for configuration during the booting process, in other words, the configuration data are automatically read from SPI-ROM after the system is powered on. If the FPGA writes configuration data to SPI-ROM, the function of the FPGA will be different the next time the system is booted. Additionally, SASEBO-GII can trigger an SPI-ROM configuration process while the system is operating, and therefore completely reconfigurable environments can be studied with this configuration.

2.3 Other Boards of the SASEBO Family

a) SASEBO and SASEBO-G/-B/-R: SASEBO, SASEBO-G, SASEBO-B and SASEBO-R are earlier members of the SASEBO family and were developed in collaboration with Tohoku University [37]. They have been discontinued and are currently unavailable on the market. SASEBO-G and SASEBO-R were replaced with SASEBO-GII and SASEBO-RII, respectively.

b) SASEBO-W: SASEBO-W was especially developed for studying and evaluating the security of smartcards. SASEBO-W is equipped with a card slot for a smartcard along with a Spartan-6 FPGA for implementing a relevant controller.

c) SASEBO-RII: SASEBO-RII is the latest version of the SASEBO-R series, and it was developed for ASIC evaluation. The architecture of SASEBO-RII is drastically different from that of SASEBO-R—the controlling FPGA is removed, and only an LSI socket is installed. SASEBO-RII is a daughter board of SASEBO-W, and SASEBO-W

[1]FPGA Mezzanine Card (Low-Pin Count).

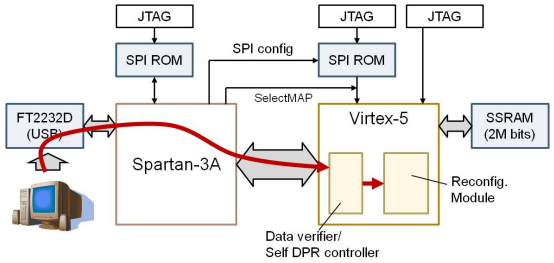


Figure 8: Dynamic partial reconfiguration (DPR) of Virtex-5.

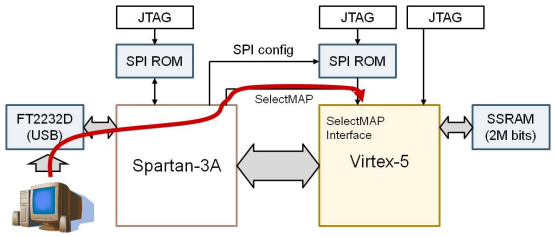


Figure 9: Virtex-5 configuration via the SelectMAP interface.

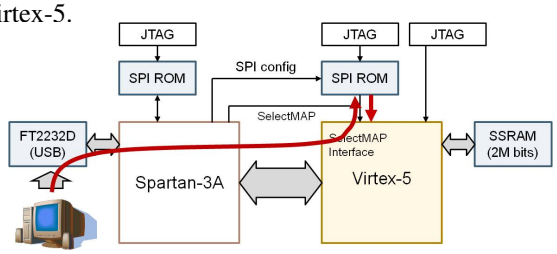


Figure 10: Virtex-5 configuration by implemented by updating SPI-ROM.

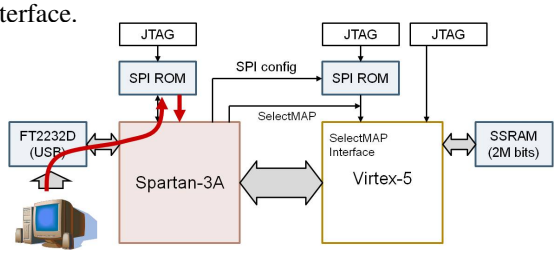


Figure 11: Spartan-3A configuration implemented by updating SPI-ROM.

is in charge of controlling the smartcard. The schematics of SASEBO-RII will be made available online under the condition that they be used for academic research, which would allow researchers to develop their own boards with LSI sockets of choice. This is expected to greatly reduce the cost of board development.

3. Pseudo-LFSR PUF

A PL-PUF is a delay PUF which is compact, efficient, multi-functional and resistant to attacks. PL-PUF does not contain a shift register; rather, it constitutes a large combinational logic based on the structure of LFSR. Figure 12 illustrates a 128-bit PL-PUF with the following primitive feedback polynomial [38]

$$x^{128} + x^{126} + x^{102} + x^{99} + 1. \quad (1)$$

Note that in PL-PUF the core logic (Fig 13) is not a register but an inverter, and thus PL-PUF constitutes a single combinational circuit. The output of PL-PUF oscillates since the output of the last core ($D_{out}(1)$) is fed back into the top core. The feedback signal is strongly affected by process variations in the device, and therefore the output of PL-PUF becomes sensitive to delays and consequently dependent on the device. The core logic does not necessarily have to be an inverter—it can be any combinational logic that efficiently extracts variations in the device.

PL-PUF realizes authentication based on a challenge-response pair (CRP). In the case of Fig. 12, the challenge is the 128-bit initial value supplied to the core logic, and the response (= ID) is the 128-bit output of the core logic. Note that the 128-bit ID is generated from a single 128-bit challenge, which is the remarkable novelty of PL-PUF realizing high throughput and high attack resistance.

After the initial value is set to each core logic, PL-PUF is activated for c clock cycles. This active cycle is referred to as an *active duration*, where the same PL-PUF can generate completely different outputs depending on the active duration c.

The features of PL-PUF can be summarized as follows.

- *Compactness*
  An inverter-based PL-PUF results in a small circuit. In the case of Fig. 12, it requires only 128 inverters and 3 XOR gates. By comparison, an arbiter-based PUF has two selector chains, and therefore a 128-stage arbiter PUF requires 256 multiplexers.
- *Efficiency*
  A PL-PUF efficiently outputs long IDs since all 128 bits of the ID are generated from a single 128-bit challenge. This is a notable advantage of the PL-PUF as compared to other PUFs, where only single-bit or several-bit output is generated from a long challenge. By comparison, an arbiter-based PUF usually requires 128 CRPs to obtain a 128-bit ID.
- *Multi-functionality*
  The output of the PL-PUF depends on the duration of the active clock cycles, and thus a single PL-PUF can be made to behave as multiple PUFs by changing the active duration. In other words, the challenge-response mapping of the PL-PUF can be easily changed without modifying its hardware structure. This property determines the unclonability of PL-PUF since cloning CRP mapping for all possible durations is considered impractical.

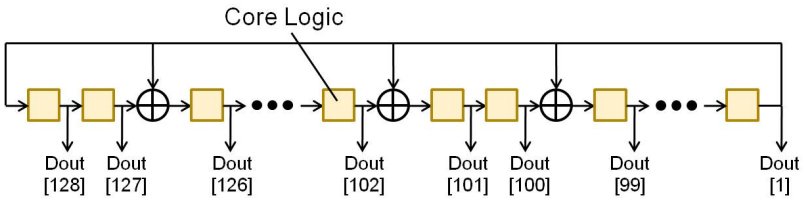


Figure 12: Structure of the PL-PUF.

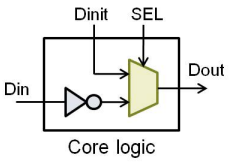


Figure 13: Structure of the core logic.

- *Reliability*
  A reliable PUF is expected to generate reproducible IDs which are unique to the device generating them. PL-PUF features both high reproducibility and uniqueness, as demonstrated below. In addition, the reliability of PL-PUF is configurable by changing the duration of the active clock cycles. Therefore, the user can choose a duration which corresponds to the preferred reliability.
- *Attack resistance*
  PL-PUF is expected to exhibit high resistance against attacks based on machine learning since modeling its delay would be exceedingly difficult. Furthermore, it outputs a 128-bit response at once from a single 128-bit challenge, in other words, the function of PL-PUF is

$$f : \{0,1\}^{128} \rightarrow \{0,1\}^{128} \quad (2)$$

  unlike the function of conventional PUFs

$$f : \{0,1\}^{128} \rightarrow \{0,1\}. \quad (3)$$

  Thus, learning the delay parameter from the 2^{128} output space would require an excessive number of CRPs, and thus it is considered impractical.

3.1 Experiments and Results

3.1.1 Quantitative and Statistical Analysis

First, we evaluated the performance of PL-PUF with respect to the quantitative indicators proposed in [34] (randomness, steadiness, correctness, diffuseness and uniqueness). The evaluation results are given in Table 3. Due to space limitations, only the results for Device 1 are given in the table. In the experiments, the active duration was varied between 1 and 16, and all performance indicators were in the range between 0 and 1, with 0 being the lowest and 1 being the highest.

Table 3: Performance of PL-PUF evaluated with respect to several quantitative indicators.

Active Duration	Randomness H	Steadiness S	Correctness C	Diffuseness D	Uniqueness U
1	0.984	0.982	0.979	0.988	0.656
2	0.975	0.966	0.960	0.987	0.728
3	0.964	0.954	0.947	0.985	0.746
4	0.967	0.925	0.913	0.989	0.755
5	0.966	0.878	0.859	0.990	0.766
6	0.944	0.804	0.775	0.988	0.772
7	0.969	0.726	0.686	0.989	0.776
8	0.960	0.622	0.572	0.988	0.772
9	0.967	0.516	0.460	0.985	0.773
10	0.964	0.415	0.357	0.978	0.771
11	0.966	0.324	0.269	0.974	0.760
12	0.964	0.253	0.203	0.958	0.756
13	0.964	0.200	0.155	0.950	0.744
14	0.962	0.165	0.126	0.929	0.739
15	0.965	0.145	0.109	0.914	0.738
16	0.963	0.131	0.097	0.900	0.734

As can be seen from the table, randomness and diffuseness are consistently high for all active durations. As a result, the entropy of PL-PUF is considered to be sufficiently high for cryptographic purposes. Also, the uniqueness of PL-PUF is markedly higher than that of the PUF in [39], and therefore PL-PUF is considered suitable for device identification as well. Furthermore, the steadiness and correctness are also high when the active duration is relatively short, although their values decrease as the active duration increases. This result indicates that PL-PUF can be suitable for device authentication when short active duration is used, while it can work as a high-quality random number generator in the case of long active duration.

3.1.2 Evaluation Results for Steadiness

Here, we assess the performance of PL-PUF with the biometric evaluation method, where the parameters *Fault Rejection Rate (FRR)* and *Fault Acceptance Rate (FAR)* are used as significant evaluation criteria. FRR represents the probability of a genuine input being rejected as a counterfeit one, while FAR represents the probability of a counterfeit input being accepted as a genuine one. FRR and FAR are derived from the intra-device Hamming distance (intra-HD) and the inter-device Hamming distance (inter-HD), where the former is the average HD between IDs generated by

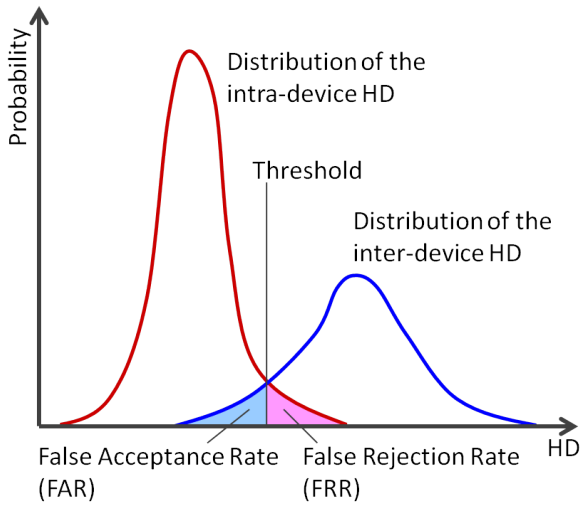


Figure 14: FRR and FAR of PUF.

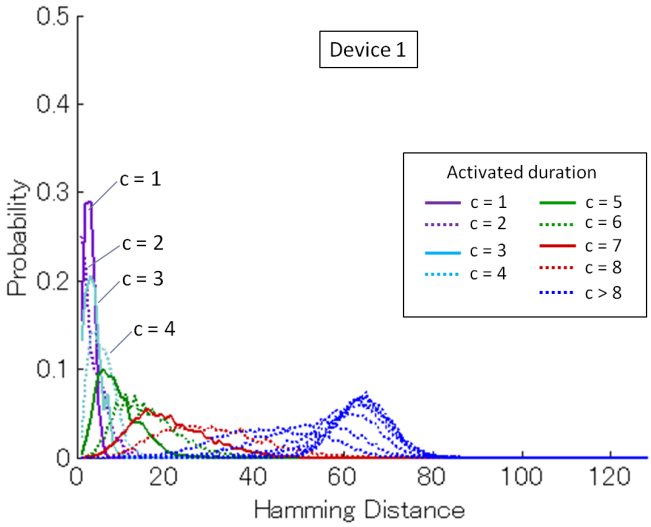


Figure 15: Distribution of intra-HD for Device 1.

the same device from the same challenge. If the intra-HD is small, the steadiness of the PUF is considered to be high. Furthermore, the inter-HD is the average HD between IDs generated by different devices from the same challenge. Since the ID length is 128 bits, the uniqueness of the PUF is considered to be high if the inter-HD is close to 64.

In Fig. 14, the curves on the left and right are the probability distributions of intra-HD and inter-HD, respectively. If the two curves cross, FAR and FRR take a non-zero value.

Figure 15 shows the probability distribution of the intra-HD for Device 1. As can be seen from the figure, when the active duration is short, the intra-device HD is rather small, and consequently the steadiness of the ID is high. On the other hand, the intra-device HD approaches 64 as the active duration increases, which indicates that the output of PL-PUF is almost purely random. This result shows that the active duration of PL-PUF should be reasonably short to obtain stable outputs.

3.1.3 Evaluation Results for Uniqueness

Figures 16-19 show the intra-HD for Device 1 and the inter-HDs between Device 1 and the other devices. The number of clock cycles for the active duration is set to 1, 4, 8 and 16, respectively. When the active duration is short, the shapes of the distributions of the intra- and inter-HD are sharp, and therefore FAR and FRR are both zero (Figs. 16 and 17). In Fig. 18, FAR and FRR become greater than zero but remain sufficiently low for Device 1 to be distinguishable from other devices. When the active duration becomes longer, Device 1 cannot be identified since its intra-HD and inter-HD distributions become indistinguishable from each other (Fig. 19).

As Figs. 16-19 show, the uniqueness of PL-PUF is high in the case of a sufficiently short active duration, although too short an active duration can fail to distinguish different PUFs.

4. Secure DPR Systems with PUF and AE

AE is a relatively new concept in cryptographic technology, providing both message encryption and authentication. Since both the confidentiality and the authenticity of bitstreams must be guaranteed, AE must be effectively applied to DPR systems. We developed a prototype of a secure DPR system using AES-GCM [14] and studied the relationship between the throughput and memory overhead of different AE modules [13]. As a result, we found that AE achieves high speed and area efficiency as compared with systems using separate encryption and authentication algorithms. However, the problem of the storage of the secret key remains since the secret key embedded into the chip can be extracted by an SCA attack. The use of PUFs is a promising approach for solving this problem.

The goal of our study is to build a secure DPR system by integrating AE and PL-PUF into the system. Such a DPR system is expected to be secure with respect to reverse engineering and hardware trojans since the bitstream of the system is protected by AE and therefore less vulnerable to SCA since PL-PUF eliminates the requirement that the secret key be stored in memory. Although there has been related work using PUF for protecting FPGA IP cores, PL-PUF is expected to realize higher throughput and considerably stronger protection against machine learning.

In this section, first we explain the AES-GCM algorithm, after which we show the implementation results and discuss the performance of our DPR system with AES-GCM. Fi-

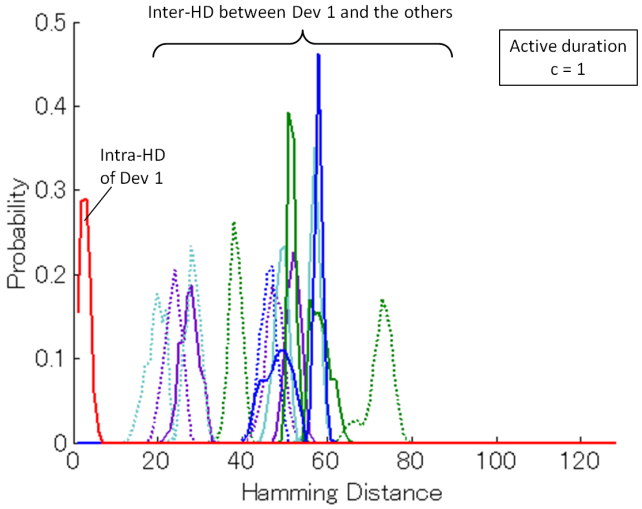


Figure 16: Distribution of the inter-HD for Device 1 for an active duration of 1.

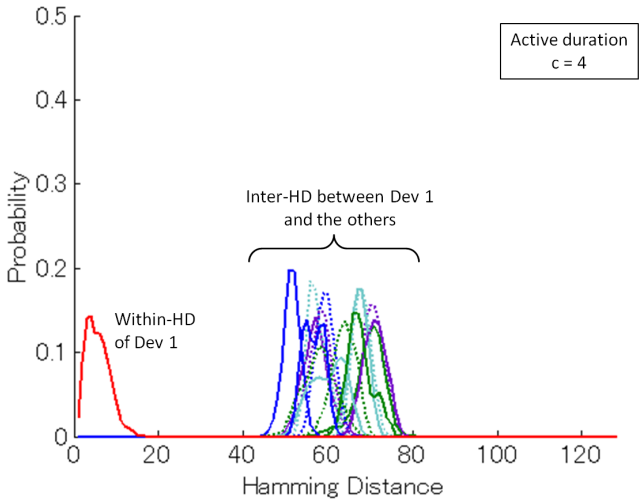


Figure 17: Distribution of the inter-HD for Device 1 for an active duration of 4.

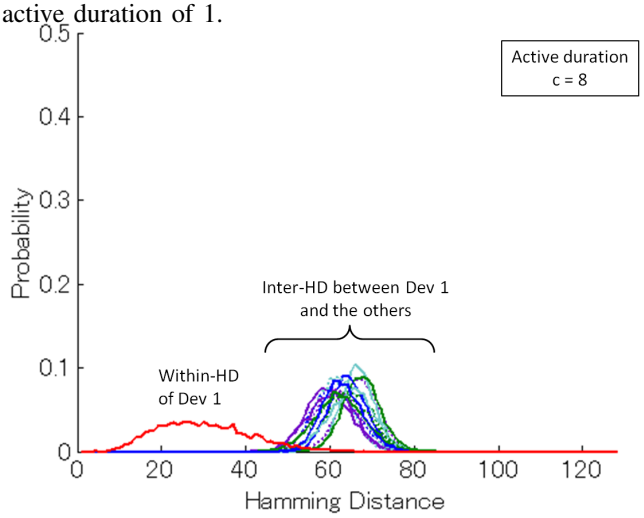


Figure 18: Distribution of the inter-HD for Device 1 for an active duration of 8.

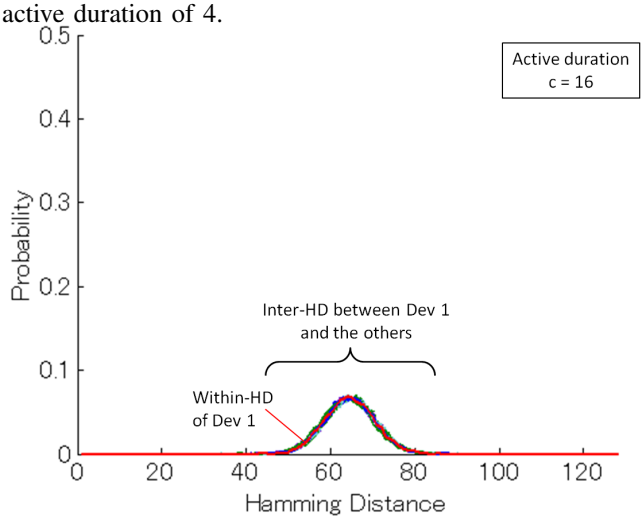


Figure 19: Distribution of the inter-HD for Device 1 for an active duration of 16.

nally, we present our ongoing project of a PUF-based secure video playback system.

4.1 AES-GCM

We chose AES-GCM [9], [10] as an AE algorithm for bitstream encryption and authentication. AES is a symmetric key block cipher algorithm standardized by the U.S. National Institute of Standards and Technology (NIST) [40]. While the previous standard (DES [41]) features a Feistel network architecture, AES employs a substitution-permutation network (SPN) architecture. The block length of AES is 128 bits, and the key length can be 128, 196 or 256 bits.

A block cipher algorithm can be applied to various modes of operation. GCM is one of the latest modes of operation standardized by NIST. Figure 20 shows an example demonstrating the operation of GCM.

The encryption and decryption scheme of GCM is based on the CTR mode of operation [42]. Thus, GCM can be highly parallelized and pipelined and is therefore suitable for hardware implementation, exhibiting a number of advantages ranging from compactness to high speed [43], [44]. There are other AE algorithms which are not necessarily suitable for hardware implementation as they cannot be parallelized or pipelined [45].

AES-GCM is an AE algorithm providing both message confidentiality and authenticity. GCM uses universal hashing in the finite field $GF(2^w)$ for generating a message authentication code (MAC). The additional merit of using $GF(2^w)$ is that the computational cost of multiplication under $GF(2^w)$ is lower than that for integer multiplication.

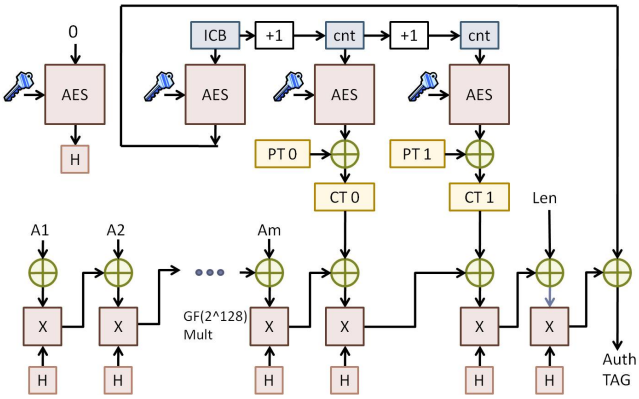


Figure 20: Example demonstrating the operation of the Galois/Counter Mode (GCM).

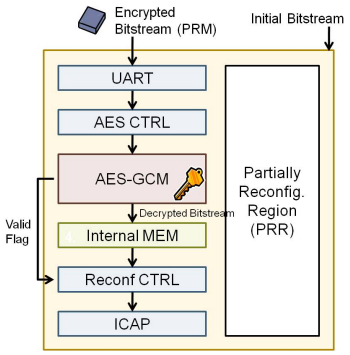


Figure 21: Overview of the proposed system with AES-GCM.

4.2 AES-GCM-based DPR System

Here, we introduce our AES-GCM-based DPR system [14]. Unlike other DPR systems, our system does not use an embedded processor to control partial reconfiguration. Rather, the input data and the ICAP control signals are directly connected to and controlled by the user logic. Thus, our system is free from the delay associated with processor buses. In Virtex-5, the maximum frequency of the ICAP interface is limited to 100 MHz, and thus the ideal throughput of the reconfiguration process is 3,200 Mbps.

Figure 21 shows a block diagram of the DPR system with bitstream encryption and authentication using AES-GCM. In this system, the lengths of the AES key and the initial vector are set to 128 bits and 96 bits, respectively.

As the main purpose of this study is to clarify the feasibility of AES-GCM for bitstream encryption and authentication, rather simple function blocks, for example, a 28-bit adder and a 28-bit subtractor, are used as PRM, which is connected to the static modules with two bus macros. The four most significant bits of the adder or the subtractor are output from PRM and connected to LEDs on the board. The PRR contains 80 slices, 640 LUTs and 320 registers. The size of the PRM bitstream is about 11KB.

The S-box of AES is implemented as a table using Block RAM. In AES-GCM, a 128-bit block is decrypted in 12 clock cycles. The last block of the message requires 12 clock cycles and an additional 10 clock cycles to calculate the authentication tag.

Table 4 shows the implementation results for the AES-GCM-based DPR system (PR-AES-GCM) along with AES-CBC and SHA-256-based DPR systems (PR-AES-SHA) for comparison. As the table shows in the case of PR-AES-GCM, the hardware resources used are fewer and the throughput is higher than for PR-AES-SHA.

4.3 Integration of a PUF into the DPR System

Since a PUF is considered a fingerprint of the device, it can be used for device authentication in a manner similar to biometrics. To perform biometric authentication, several CRPs are exchanged between the server and the device in the DPR system, in which the server knows the correct responses in advance. The actual responses are sent by the system to the server and compared to the correct responses. If the error rate is lower than a certain threshold, the system is successfully authenticated.

Note, however, that the simple use of a PUF provides neither a strict authentication scheme nor a solution to the problem of key exchange. In a secure DPR system, a (partial) bitstream of an FPGA is usually encrypted with a symmetric cipher. In light of the possibility of an SCA attack, the key should not be stored in the device in advance. Therefore, the key must be *generated* in the device in some way. Here, note that a PUF cannot provide exact reproducibility since the output of the PUF is affected by random fluctuations in the device. The sole use of a PUF cannot generate identical keys from the same challenge set, and thus error correction code (ECC) is often used in key generation. In this regard, a scheme known as a *fuzzy extractor* [46] is widely used for ECC-based key generation [31], [47], and other key generation methods have been recently reported, such as in [48] and [49].

As an ongoing project, we are developing a video playback system based on AE and PUFs. The development platform is SASEBO-GIII, and an FMC daughter board with an LSI socket for implementing ASIC PUFs is currently being developed. An off-the-shelf FMC board is used for HDMI input/output ports. Error correction as well as the computation of hashes and other parameters in the fuzzy extractor are implemented on Kintex-7 on a SASEBO-GIII. Figure 22 shows the development platform, including SASEBO-GIII and the HDMI FMC board.

It should be noted that an SCA attack against a fuzzy extractor was recently reported [50]. We believe that SASEBO-GIII is the most suitable platform for investigating the security of the proposed PUF-based video playback system since it supports simple and straightforward implementation

Table 4: Comparison of the performance of secure PR systems (14,112-byte PRM).

System	Device	Slice	Authentication	Decryption	Configuration	Overall
PR-AES-GCM	XC5VLX50T	2,687*	106.43 μs 1,067 Mbps		35.3 μs 3,200 Mbps	141.73 μs 797 Mbps
PR-AES-SHA256	XC5VLX50T	2,730*	160.97 μs 701 Mbps	97.14 μs 1,164 Mbps	35.3 μs 3,200 Mbps	196.27 μs 575 Mbps

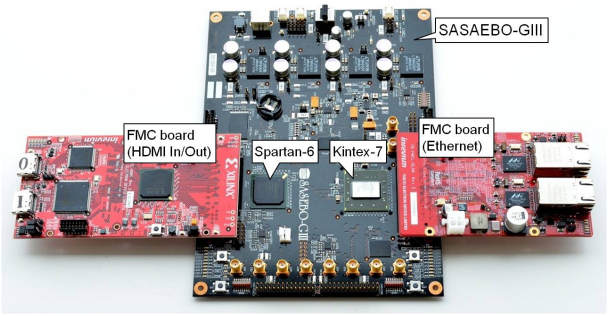


Figure 22: A PUF-based video playback system.

of SCA experiments. We are currently in the process of implementing the entire system, and the plans for future work include testing the security and feasibility of the system.

5. Conclusion

This paper introduced certain security issues associated with partial reconfiguration of FPGAs together with studies on counteracting these issues by using SASEBO, PUFs and secure DPR systems. Since FPGA bitstreams are electronic data downloaded from a host computer or the Internet, they are always susceptible to problems such as piracy, reverse engineering, tampering and hardware trojans. Authenticated encryption (AE), which guarantees both the confidentiality and the authenticity of the encrypted data, can serve as a solution to these problems, however, a recently reported type of attack referred to as SCA poses serious concern for the security of cryptographic systems. One solution to SCA attacks might be a PUF which generates device-specific IDs by using process variation of the device.

First, we introduced a family of boards named SASEBO (Side-channel Attack Standard Evaluation Board). The latest version of SASEBO, SASEBO-GIII, with the newest FPGA Kintex-7, will be made available in 2012 or 2013.

In addition, our PL-PUF is a compact and secure delay-based PUF achieving high throughput. Unlike other PUFs, PL-PUF outputs an N-bit response from an N-bit challenge, which enables fast and attack-resistant key generation. The values of both FAR and FRR of PL-PUF are rather small in the case of a short active duration, and therefore PL-PUF is suitable for device identification as well as for key generation with fuzzy extractors.

A direction of future work is to develop an entire video playback DPR system including AE, PUF, and fuzzy extractors along with the video decoders.

Acknowledgements

The parts of this work related to SASEBO, SASEBO-G, -B, -R and -GII was funded by the Ministry of Economy, Trade and Industry (METI), Japan. In addition, the part of this work related to SASEBO-W was funded by the Strategic International Research Cooperative Program (SICP), the Japan Science and Technology Agency (JST). Finally, the part of this work related to SASEBO-GIII was funded by the Core Research for Evolutional Science & Technology (CREST), JST.

References

[1] Y. Hori, H. Yokoyama, H. Sakane, and K. Toda, "A secure content delivery system based on a partially reconfigurable FPGA," *IEICE Trans. Inf.&Syst.*, vol. E91-D, no. 5, May 2008, (to be publised).

[2] C. Albrecht, J. Foag, R. Koch, and E. Maehle, "DynaCORE—a dynamically reconfigurable coprocessor architecture for network processors," in *PDP 2006*, 2006, pp. 101–108.

[3] M. Rummele-Werner, T. Perschke, L. Braun, M. Hubner, and J. Becker, "A FPGA based fast runtime reconfigurable real-time multi-object-tracker," in *ISCAS 2011*, 2011, pp. 853–856.

[4] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proc. IEEE*, vol. 95, no. 2, pp. 438–452, 2007.

[5] A. Akoglu, A. Sreeramareddy, and J. Josiah, "Fpga based distributed self healing architecture for reusable systems," *Cluster Computing*, vol. 12, no. 3, pp. 269–284, 2009.

[6] A. Mecwan and N. Gajjar, "Implementation of software defined radio on FPGA," in *NUiCONE 2011*. IEEE, 2011.

[7] A. Moradi, A. Barenghi, T. Kasper, and C. Parr, "On the vulnerability of FPGA bitstream encryption against power analysis attacks—extracting keys from Xilinx Virtex-II FPGAs," Cryptology ePrint Archive, 2011.

[8] A. Moradi, M. Kasper, and C. Paar, "On the portability of side-channel attacks –an analysis of the Xilinx Virtex 4 and Virtex 5 bitstream encryption mechanism," Cryptology ePrint Archive, 2011.

[9] D. A. McGrew and J. Viega, "The Galois/counter mode of operation (GCM)," May 2005, http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html.

[10] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, SP 800-38D ed., National Institute of Standards and Technology, Nov. 2007.

[11] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *ASIACRYPT 2000*, 2000, pp. 531–545.

[12] A. Seffrin and S. Huss, "Ensuring secure information flow in partially reconfigurable architectures by means of process algebra analysis," in *TrustCom 2011*, 2011, pp. 443–450.

[13] Y. Hori, A. Satoh, H. Sakane, and K. Toda, "Bitstream encryption and authentication using AES-GCM in dynamically reconfigurable systems," in *IWSEC 2008*, 2008, pp. 261–278.

[14] ——, "Bitstream encryption and authentication with aes-gcm in dynamically reconfigurable systems," in *FPL 2008*, 2008, pp. 23–28.
[15] L. Bossuet and G. Gogniat, "Dynamically configurable security for SRAM FPGA bitstreams," *Int. J. Embedded Systems*, vol. 2, no. 1/2, pp. 73–85, 2006.
[16] M. M. Parelkar, "Authenticated encryption in hardware," Master's thesis, George Mason University, 2005.
[17] A. S. Zeineddini and K. Gaj, "Secure partial reconfiguration of FPGAs," in *ICFPT'05*, 2005, pp. 155–162.
[18] T. Kean, "Secure configuration of field programmable gate arrays," in *Field-Programmable Logic and Applications*, 2001, pp. 142–151.
[19] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *CRYPTO'96*, 1996, pp. 104–113.
[20] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO'99*, 1999, pp. 388–397.
[21] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES 2004*, 2004, pp. 16–29.
[22] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *CHES'01*, vol. LNCS 2162, 2001, pp. 251–261.
[23] J. J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and countermeasures for smart card," in *e-Smart'01*, vol. LNCS 2140, 2001, pp. 200–210.
[24] B. Gierlichs, L. Batina, and P. Tuyls, "Mutual information analysis," in *CHES2008*, 2008.
[25] S. Akashi, K. Toshihiro, and S. Hirofumi, "Secure implementation of cryptographic modules—development of a standard evaluation environment for side channel attacks," *Synthesiology*, vol. 3, no. 1, pp. 56–65, 2010.
[26] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *CCS 2002*. ACM, 2002, pp. 148–160.
[27] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds. Springer-Verlag, 2010, ch. 1, pp. 3–37.
[28] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Trans. VLSI Syst.*, vol. 13, no. 10, pp. 1200–1205, 2005.
[29] G. E. Suh and S. Devadas, "Physical physical unclonable functions for device authentication and secret key generation," in *DAC'07*, 2007, pp. 9–14.
[30] D. Suzuki and K. Shimizu, "The glitch PUF: A new delay-PUF architecture exploiting glitch shapes," in *Proc. CHES2010*, 2010, pp. 366–382.
[31] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *CHES'07*, 2007, pp. 63–80.
[32] S. S. Kumar, J. Guajardo, R. Maesyz, G.-J. Schrijen, and P. Tuyls, "The butterfly PUF," in *HOST'08*, 2008, pp. 67–70.
[33] E. Ozturk, G. Hammouri, and B. Sunar, "Physical unclonable function with tristate buffers," in *ISCAS'08*, 2008, pp. 3194–3197.
[34] Y. Hori, H. Kang, T. Katashita, and A. Satoh, "Pseudo-LFSR PUF: A compact, efficient and reliable physical unclonable function," in *ReConFig 2011*, 2011, pp. 223–228.
[35] *Virtex-5 User Guide*, Xilinx, Inc., 2007.
[36] "Side-channel attack standard evaluation board (sasebo)," http://www.rcis.aist.go.jp/special/SASEBO/, research Center for Information Security, National Institute of Advanced Industrial Science and Technology.
[37] "Cryptographic hardware project," http://www.aoki.ecei.tohoku.ac.jp/crypto/, aoki Lab., Tohoku University.
[38] M. George and P. Alfke, "Linear feedback shift registers in Virtex devices," Xilinx apprication note XAPP210, 2007.
[39] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs," in *Proc. ReConFig2010*, 2010, pp. 298–303.
[40] U.S. Department of Commerce/National Institute of Standards and Technology, "Announcing the advanced encryption standard (AES)," FIPS PUB 197, Nov. 2001.
[41] ——, "Data encryption standard (DES)," FIPS PUB 46-3, 1999.
[42] M. Dworkin, *Recommendation for Block Cipher Modes of Operation*, SP 800-38A ed., National Institute of Standards and Technology, Dec. 2001.
[43] A. Satoh, "High-speed parallel hardware architecture for Galois counter mode," in *ISCAS'07*, 2007, pp. 1863–1866.
[44] A. Satoh, T. Sugawara, and T. Aoki, "High-speed pipelined hardware architecture for Galois counter mode," in *ISC'07*, 2007, pp. 118–129.
[45] D. A. McGrew and J. Viega, "The security and performance of the Galois/counter mode (GCM) of operation," in *INDOCRYPT 2004*, 2004, pp. 343–355.
[46] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal of Computing*, vol. 38, no. 1, pp. 97–139, 2008.
[47] C. Bosch, J. Guajardo, A.-R. Sadegh, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *CHES'08*, 2008, pp. 181–197.
[48] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.
[49] M.-d. M. Yu, R. Sowell, A. Singh, D. M'Raïhi, and S. Devadas, "Performance metrics and empirical results of a PUF cryptographic key generation ASIC," in *HOST 2012*, 2012.
[50] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of pufs and fuzzy extractors," in *Trust 2011*, 2011, pp. 33–47.

Ensuring Design Integrity through Analysis of FPGA Bitstreams and IP Cores

Jonathan P. Graf, Scott H. Harper, and Lee W. Lerner
Luna Innovations Incorporated
1 Riverside Circle, Suite 400
Roanoke, VA 24016
fpga@lunainnovations.com

Abstract - *In this paper, we introduce a novel, broad definition of field programmable gate array (FPGA) design integrity and explore its value in the domains of Trust, high-reliability design, and design anti-obsolescence for FPGA-based systems. We claim that an FPGA design with integrity must continuously provide the FPGA user with the function described by the designer and no other function. A common starting point for approaching design integrity in each of the explored domains is the FPGA bitstream. Luna's unique software that evaluates the previously inaccessible designs inside of these bitstreams and third-party intellectual property (IP) provides a firm foundation for analysis of FPGA design integrity.*

Keywords: field programmable gate array, FPGA, intellectual property, reverse engineering, security, trust

1 Introduction

Every field programmable gate array (FPGA) embodies the core concept that its function is realized by a specialized, custom design that was created separately and by different agency than the general design of the silicon of the FPGA. Thus, when considering these devices, we must address two aspects: the FPGA vendor architecture and the user design. The purpose of the vendor architecture is to create a general sea of unprogrammed logic that can be configured by a user design to realize the user's intended application-specific function. Protecting and measuring the device integrity of the vendor silicon architecture involves the same set of challenges addressed in recent projects to trust Application Specific Integrated Circuits (ASICs) and control ASIC supply chain risk [1]. In this paper, we do not treat the integrity of the vendor device itself. Rather, we focus on the integrity of the user design that is embedded in that device. We introduce a novel, broad definition of FPGA *design integrity* and demonstrate the value of this definition to the concise statement of FPGA security challenges. Finally, we describe several applied methods for guaranteeing that FPGA design integrity is maintained, each of which make use of unique software that evaluates the contents of FPGA bitstreams and third-party intellectual property (IP).

2 Definition and Attributes of FPGA Design Integrity

An FPGA design with integrity continuously provides the *User* with the function described by the *Designer* and no other function. The *User* is the party wishing to make use of the function in the FPGA, and the *Designer* is the party or parties responsible for creating the design that realizes that function in the FPGA. The above definition leads to the following three guarantees that must be provided to the *User* in order for the design to have integrity.

1) A trusted description of the function
2) The function described is realized in the design
3) The design realizes only the function described

An FPGA design that cannot guarantee all of the above attributes cannot be said to have complete integrity.[1]

Note that this definition encompasses both physical correctness of the programming stream (bitstream) and the implementation of a design within that stream. Traditionally, physical correctness is assessed using a mechanism such as a hash or checksum to test the bitstream as it is being loaded on the FPGA device to ensure it has not been changed since it was first generated by the designer [2]. This capability addresses a portion of design integrity, but it does not comprise the full range of the term's potential. Instead, using the above attributes of an FPGA design with integrity, we may explore how this simple definition elegantly expresses the commonality of the goals within many FPGA security domains. While traversing these topics, we provide examples of technology solutions

[1] A similar set of attributes may be easily developed to describe the integrity of any kind of microelectronic design.

provided by Luna Innovations that make use of our ability to directly evaluate the designs inside bitstreams and IP cores. First, however, we make brief comments on the common formats of both the user-trusted description of the function and the design itself.

3 Functional Description Formats

Before we address design integrity challenges and the technologies used address them, we must briefly consider the functional description formats used to describe the designer's intent. Luna's FPGA integrity technologies are intended to operate with a variety of functional specification formats. As new forms of functional specification are developed, our technology will adapt to accommodate those formats common in FPGA design flows. Depending on the particular case in which we are seeking to guarantee FPGA integrity, we may use Hardware Description Language (HDL) source, a simulatable behavioral model, or even a datasheet alone as the user-trusted functional description. As other functional descriptions gain industry acceptance, our FPGA integrity technologies are malleable to accommodate them.

4 Evaluating Bitstreams and Third-Party IP

Just as the user-trusted functional description may take many forms, we may similarly wish to guarantee the integrity of a design when it is contained in any of a variety of formats. For FPGAs, the common formats in which a design might be expressed include HDL, synthesized netlist, and bitstream. Luna's work has primarily focused on the challenge of trust when the design is in the synthesized netlist format or in the bitstream format.

4.1 Synthesized Netlist Design

Designs expressed in this format have been synthesized from the HDL created by the designer. Once synthesized, the design may still be represented in an HDL such as Verilog or in another common electronic design format such as the Electronic Design Interchange Format (EDIF). Whether Verilog or EDIF, however, the synthesized netlist is expressed as a connected and configured arrangement of FPGA resources necessary to realize the design. Third-party intellectual property cores are commonly distributed to the user purchasing the core as synthesized netlists targeted towards the resources provided by the FPGA of interest. It is not uncommon for IP core designers to encrypt and obfuscate the proprietary implementation details of their cores. These measures taken to hide details of the design add to the challenge of guaranteeing the design's integrity. However, Luna has developed technologies and techniques to explore the implementation of FPGA IP cores sufficiently either to guarantee or to expose problems with their integrity.

4.2 Bitstream Designs

The final design format that is configures the silicon to accomplish the user's application-specific task is the bitstream. It has long been desirable to evaluate the bitstream directly in order to verify the contents of the design in its deployed form [2]. The challenge faced by FPGA users, however, has been that the bitstream formats are not documented by FPGA vendors sufficiently to allow evaluation of the designs they contain. To address this challenge, Luna has developed software that analyzes an FPGA bitstream and describes the design it contains as a synthesized human-readable netlist (Fig. 1). This capability enables Luna to make uniquely comprehensive claims about the integrity of FPGA designs all the way down to their bitstream implementation.

With the variety of functional description and design formats described, we may now look to various FPGA security challenges and how they are viewed through the lens of our FPGA design integrity definition.

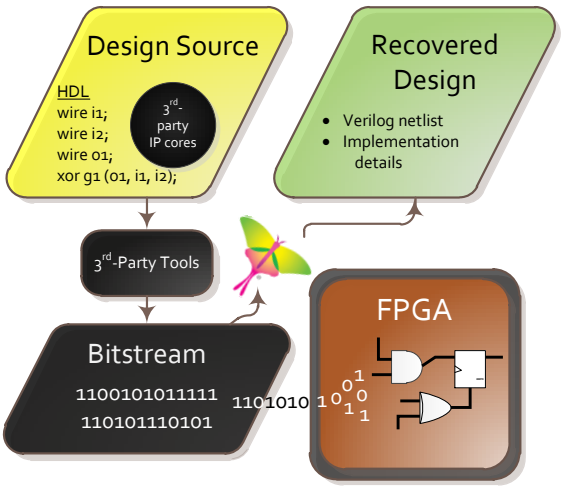


Fig. 1. Recovering design details from a bitstream

5 FPGA TRUST

The challenge of FPGA Trust is that of guaranteeing both integrity attributes (2) and (3) in reference to attribute (1). Taken on its own, attribute (2) – knowledge that the function described is realized in the design – is the traditional challenge of FPGA design verification. With (2) and (3) taken together, we have a definition of the goal of FPGA Trust. Restated, in FPGA Trust we wish to guarantee that the design provides only that function described in the user-trusted description and nothing more. Luna has done a variety of work in the FPGA Trust domain, primarily through our work on the DARPA Trust and IRIS programs [3] [4]. These programs may be summarized into three major FPGA Trust domains.

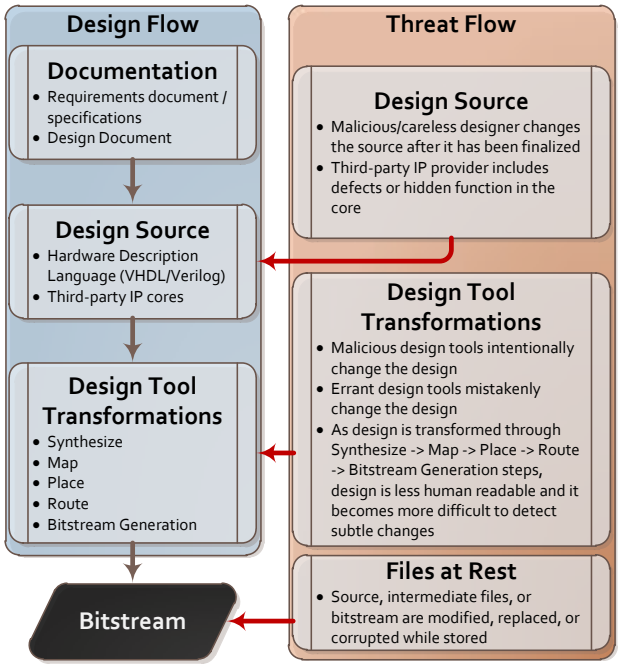


Fig. 2. Threats to FPGA design Trust

5.1 Design-to-Source Trust

In this domain, the user and the designer are both trusted. What is not trusted is the design environment in which the design – as expressed as HDL source – has been transformed into its final implementation format. There are a number of factors that may lead to the lack of trust in the design environment (Fig. 2). The most common factors are design software with unknown provenance that transforms the HDL in ways hidden from the user and the threat of an insider that may modify the design. When performing this type of integrity evaluation, we wish to treat the design in its final implementation format, the bitstream. Since we trust the designer in this scenario, we use the designer's HDL source as the trusted design description.

Luna has developed software to automate the many steps required to guarantee that a design has maintained its integrity when being transformed from HDL into its implementation bitstream. The first automated step is the back-conversion of the bitstream into a synthesized netlist format. Thereafter, our software evaluates the extracted netlist with reference to the HDL source, applying structural, simulation-based, and formal mathematical algorithms to prove integrity or expose differences. Together, these evaluation methods (described in more detail in [4]) provide a guarantee either that the design contained in the bitstream matches the intent, and only the intent, expressed by the designer in their source HDL or that it does not. In the case that it does not, each non-matching feature is exposed for further consideration. Luna has named the software that performs this kind of evaluation the Change Detection Platform (CDP) (Fig. 3).

5.2 Netlist-to-Model Trust

As described in the previous section, the case of evaluating a netlist against a model arises when purchasing a design element from an IP vendor. This is common practice in modern development as the use of pre-built components speeds design construction. Here, both the designer and user might themselves be trusted, but the use of outside material in the design process introduces an untrusted element. In this case, the vendor commonly

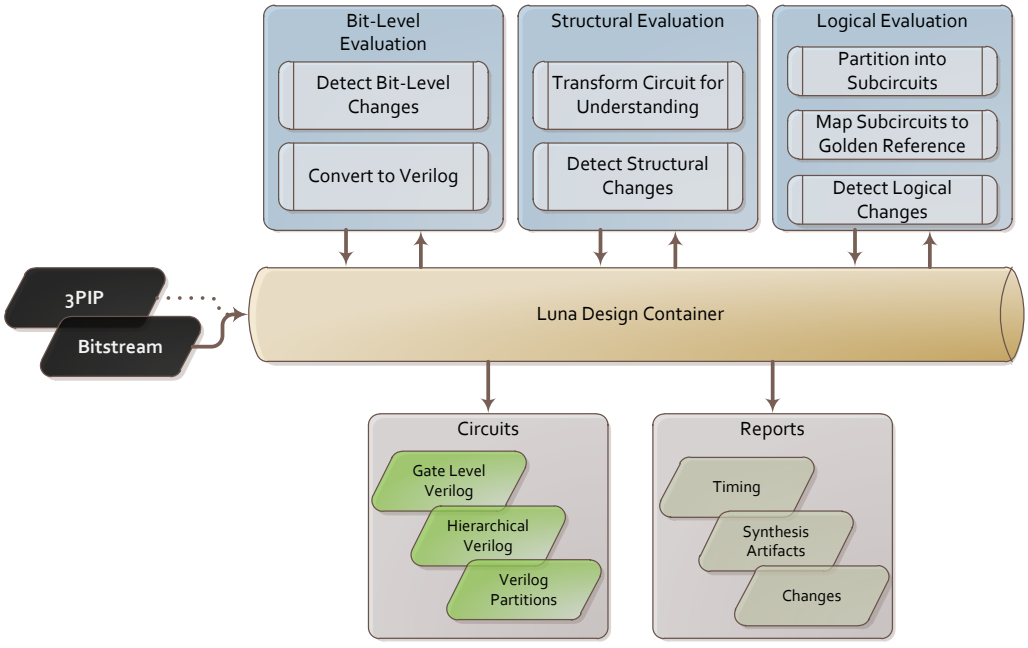


Fig. 3. Luna's Change Detection Platform (CDP)

provides a model to serve as a simulation reference. The designer relies on this model as an accurate representation of the IP when developing their application. This model may not provide the implementation details of the IP core; it may simply replicate the behavior of the core when simulated, and the core itself may be an obfuscated synthesized netlist. Luna has developed technology that can create an evaluable netlist from encrypted and obfuscated third-party IP. In this netlist-to-model instance, the design portion we wish to trust is the IP core, and the design specification against which we may evaluate the core is the model provided by the vendor. The Luna Change Detection Platform contains technologies designed to assist in the performance of this kind of evaluation.

One feature of the Change Detection Platform that is particularly useful here is its ability to create mappings. A mapping is a collection of equivalent reference points between a design under test and its trusted reference. Formal equivalence checking (FEC) tools such as Cadence Conformal [5] and OneSpin 360 EC [6] currently perform similar mappings when assessing design equivalence. Their mappings are most useful when comparing two structural and highly similar designs, such as those being evaluated in the bitstream-to-source case. They are not as useful in the the netlist-to-model case, however, due both to the obfuscation the third-party vendor may have instituted in the IP core and the fact that the reference model may be an inexact representation of the implementation details of the core that it models. Luna has created mapping technologies that move beyond FEC tools to solve this problem. While the mapping step is key to the netlist-to-model trust evaluation, each of the change detection steps mentioned in the bitstream-to-source evaluation are also used to prove whether or not differences exist between the netlist and the model.

5.3 Netlist-to-Datasheet Trust

There are many cases in which it may be desirable to trust a design for which neither source HDL nor a behavioral model may be referenced. For example, it may be that an IP provider does not provide a trustable behavioral model or that a design has been purchased as a bitstream for which there is no accompanying trusted source HDL. It may be that the only trusted reference available is a datasheet describing the function of the design. As mentioned in the previous two sections, Luna has developed technologies that can convert both third-party IP cores and bitstreams into evaluable synthesized netlists. Thus, the remaining challenge is that of comparing a netlist to the trusted datasheet.

Luna is currently developing a software platform, the Functional Derivation Platform (FDP), to address this challenge (Fig. 4). Our approach is multifaceted. From the design netlist, we derive its function using a combination of novel top-down and bottom-up reverse engineering methods. Working from the top down, we define the major functions of the designs and their boundaries and then drill into them hierarchically to further define internal functions. From the bottom up, we transform the unstructured netlist into its basic low-level functional constituents and then work up to define the function of groups of low-level functions. The approach is unified and managed in the FDP such that the top-down and bottom-up methodologies converge, completing our understanding of the netlist. We then make use of semi-automated datasheet analysis techniques to turn the datasheet into provable propositions and compare those propositions against the function derived from the netlist. We are in the process of developing

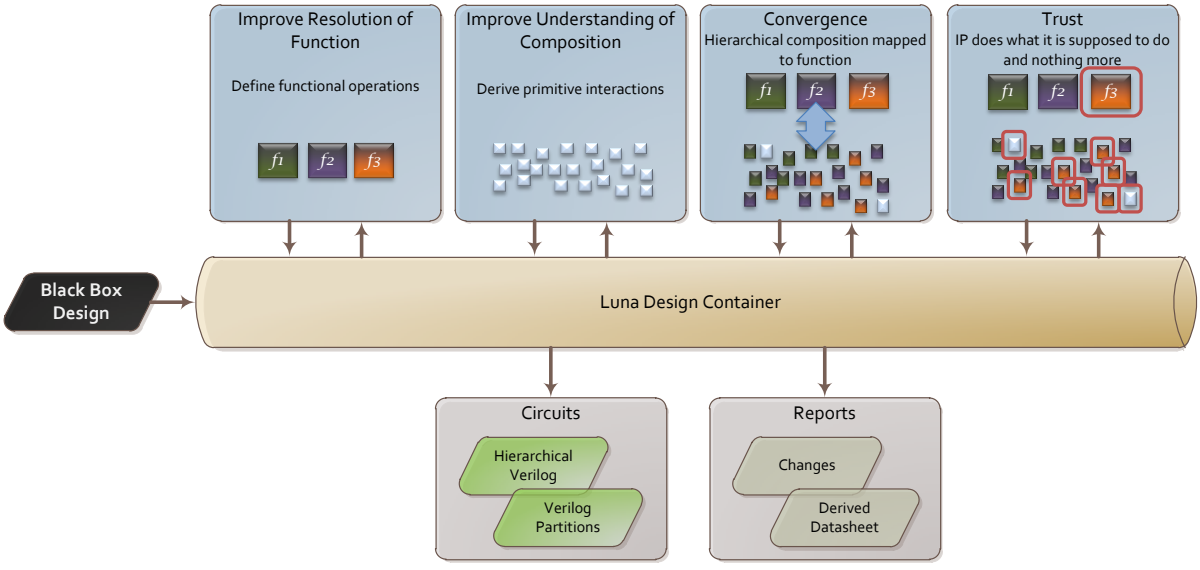


Fig. 4. Luna's Functional Derivation Platform (FDP)

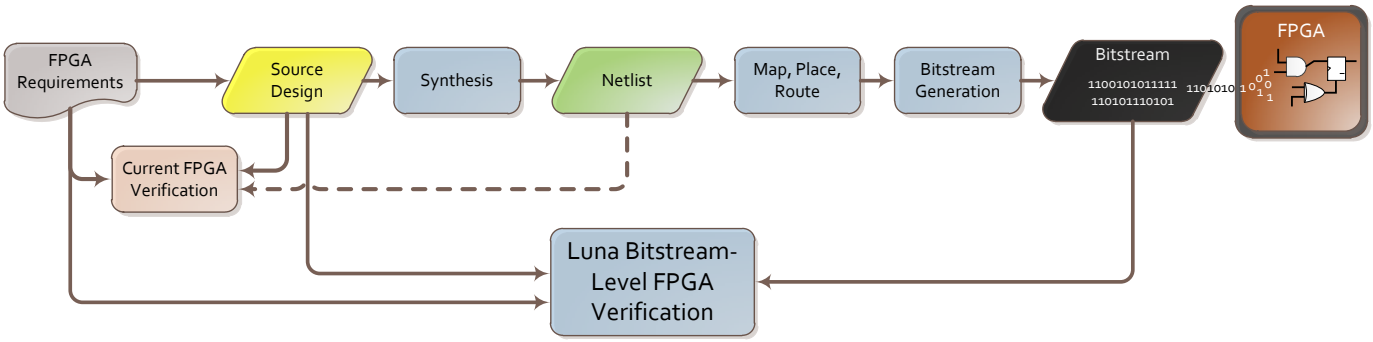


Fig. 5. Luna's FPGA high-reliability verification flow

multiple techniques and algorithms to automate the top-down, bottom-up, convergence, and analysis processes involved in our approach. Using this software, we will be able to describe the level of similarity or difference between the trusted datasheet and the design contained in the netlist.

5.4 Documentation-to-Source Trust

The issue of assessing trust in the presence of an untrusted designer is partially addressed by the datasheet-driven analysis described above. A more encompassing solution to this problem, however, requires a means of assessing source in terms of design requirements. Several point solutions have the potential of addressing aspects of this issue, including commercial tools for assertion-based verification [7] [8], application of proof-carrying code [9] [10] concepts to hardware designs, and annotation methods that communicate information throughout the design flow in a manner that allows for verification [11] [12]. These methods are at varying levels of maturity from basic research that has yet to be applied within hardware designs to end products available from commercial companies. A challenge still remains to integrate these solutions into an end-to-end evaluation flow for design integrity.

6 High-Reliability FPGA Applications

Design integrity is also a concern in high-reliability FPGA applications. FPGAs used in the aerospace industry, for example, may be subject to strict design assurance standards, such as DO-254 [13]. Designers in high-reliability areas must have the assurance that their FPGA bitstream exactly instantiates their intended design. Until now, their only means of verifying the final implementation of their design has been through board-level testing. Similar to the three FPGA Trust scenarios outlined above, high-reliability designers may be interested in performing bitstream-to-source, netlist-to-model, and netlist-to-datasheet evaluations for a slightly different purpose. The only difference is the agent of change in each domain. In FPGA Trust, the described evaluations are done to determine if a malicious party has changed the design environment, IP core, or application bitstream to add to, remove from, or modify the application. In FPGA high-reliability applications, the evaluation is done to ensure that no mistake by the designer or in the design software has led to an error in its final implementation. The Luna CDP has the ability to verify design integrity down to the bitstream level, uniquely addressing this challenge, as illustrated in Fig. 5. By leveraging the original design source verification results, end-to-end requirements traceability is established.

6.1 Radiation-Induced Upsets

A further agent of concern that can cause changes in some high-reliability aerospace applications is the environment in which the FPGA might be deployed. For example, FPGA users that deploy applications in space wish to know that their designs will maintain integrity in spite of single event upsets (SEUs). SEUs or faults in configuration memory can cause changes to the structure and function of the implemented design culminating in erroneous or undesirable behaviors. SEU effects in a design are typically evaluated at design time by analyzing the response of hardware actively running the design to ion beam irradiation or internally injected bit flips [14] [15]. Unfortunately, these hardware-in-the-loop techniques are often prohibitively expensive, time consuming, and require special test facilities.

Luna's bitstream analysis capabilities provide the foundation for a purely software-based methodology for investigating the effects of faults in FPGA configurations. This is illustrated in Fig. 6. Here, SEUs or faults are emulated by toggling bits directly in the bitstream of the design under test. Luna's mapping of design resources to configuration bits enables a drastic reduction in the number of bits that must be considered for fault injection by identifying those that are essential to configuring the design. Testing time is further reduced by injecting multiple, non-overlapping faults in parallel. In testing the effect of a fault, Luna's tools are used to recover the modified netlists describing design resource and implementation details as well as device functionality.

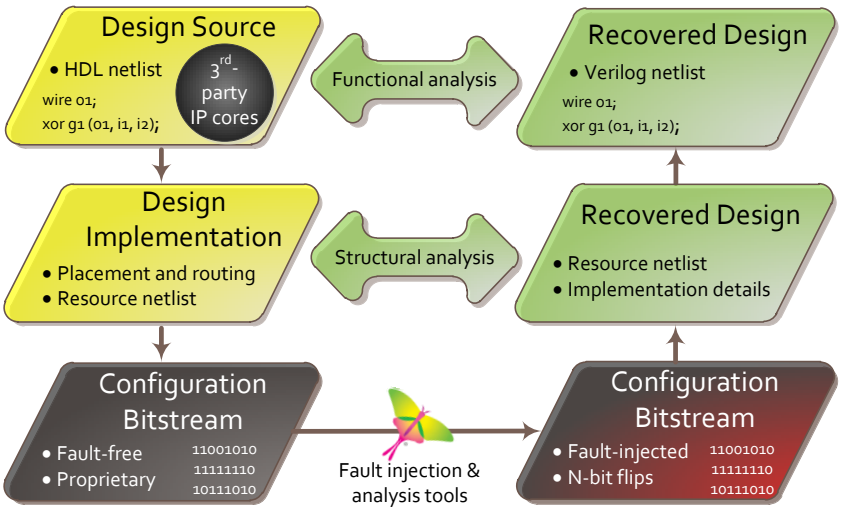


Fig. 6. Luna's FPGA configuration fault injection and analysis flow

With this netlist in hand, standard formal equivalence checkers and other tools can be used to compare the recovered design under test to the original design in order to analyze the specific structural or functional effects of a fault.

We anticipate that this work will lead to an enhanced understanding of how to implement FPGA designs such that their critical functions are less susceptible to failure in the face of SEUs or other faults affecting configuration. In this way, we improve the radiation tolerance of the design, leading to the maintenance of design integrity when deployed in space. A further effect of such a software platform may be the reduction of time spent on ion beams to settle questions of FPGA radiation tolerance. Rather, many experiments now performed on ion beams might be performed virtually using software models of the effects of bit flips on the design.

7 Anti-Obsolescence

The final FPGA design integrity challenge we treat is that of anti-obsolescence. In this instance, the user and the design are both trusted and the operating environment is not considered. Rather, we here wish to guarantee that the design maintains its integrity regardless of the FPGA on which it is instantiated. In anti-obsolescence, we are concerned with the antiquation of the FPGA silicon on which the design was originally instantiated. In many cases, the source HDL for designs on older FPGAs may be lost, but the useful life of the design has not yet expired. In these instances, our ability to convert bitstreams into netlists again demonstrates its value. We may first recover an HDL representation of the FPGA design from the bitstream. Then we can re-synthesize that design for a more modern FPGA, as illustrated in Fig. 7. In this manner, we maintain the integrity of the design independent from the FPGA silicon on which it is instantiated. Our Change Detection Platform can perform formal comparisons to ensure that the design does not change between the devices on which it is realized.

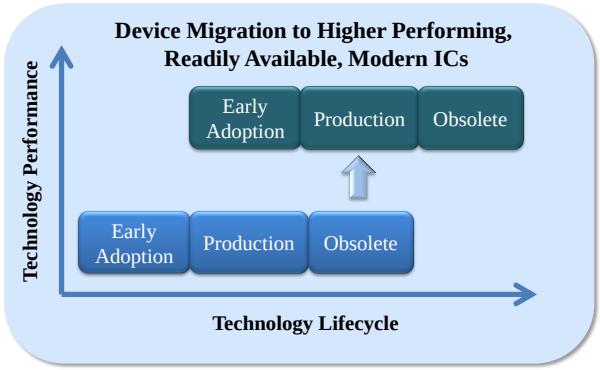


Fig. 7. Device migration

8 Conclusion and Future work

We have presented a novel definition of FPGA design integrity, and we have used it as a means of drawing a cohesive line between various disciplines of FPGA security and reliability. This perspective has proven its value in our work creating software to solve these design challenges. It has allowed us to identify common features that apply across the various aspects and craft a software foundation that is broadly applicable. Critical among these common features are Luna's bitstream and third-party IP evaluation

software. These technologies have been assessed in the course of the DARPA Trust and IRIS programs, and are currently being made available to select customers.

While this technology provides the basis for assessment and preservation of design integrity, challenges still remain to fill gaps such as establishing trust in initial HDL created from a specification as well as folding the functional and change detection methods into a unified system. The ability to understand bitstream contents truly opens up a range of new possibilities that have only begun to be addressed.

9 References

[1] D. Collins, "TRUST in ICs Program Overview," in *GOMACTech*, Las Vegas, NV, 2008.

[2] S. Trimberger, "Trusted Design in FPGAs," in *44th Design Automation Conference (DAC)*, San Diego, CA, 2007.

[3] J. Graf, J. Hallman and S. Harper, "Trust in the FPGA Supply Chain Using Physically Unclonable Functions," in *GOMACTech*, Reno, NV, 2010.

[4] J. Graf, "Change Detection Platform for FPGA Trust," in *GOMACTech*, Orlando, FL, 2011.

[5] Cadence Inc., "Encounter Conformal Equivalence Checker," [Online]. Available: http://www.cadence.com/rl/Resources/datasheets/encounter_conformal_EC_ds.pdf. [Accessed 9 May 2012].

[6] OneSpin Solutions, "OneSpin 360 EC – FPGA," [Online]. Available: http://www.onespin-solutions.com/downloads/OneSpin-360-EC-FPGA.pdf. [Accessed 9 May 2012].

[7] Zocalo Tech, "Enabling Assertion Based Verification," 2010. [Online]. Available: http://www.zocalo-tech.com/files/assertionverification_whitepaper.pdf. [Accessed 9 May 2012].

[8] Mentor Graphics, "Assertion-Based Verification," [Online]. Available: http://www.mentor.com/products/fv/methodologies/abv/. [Accessed 9 May 2012].

[9] G. C. Necula, "Proof-carrying code," in *Proceedings of the 24th ACM SIGLPAN-SIGACT symposium on principles of programming languages*, 1997.

[10] S. Drzevitzky, U. Kastens and M. Platzner, "Proof-carrying Hardware: Towards Runtime Verification of Reconfigurable Modules," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2009.

[11] A. Bruhlmann, T. Girba, O. Greevy and O. Nierstrasz, "Enriching Reverse Engineering with Annotations," in *Proceedings of the international conference on Model Driven Engineering Languages and Systems (MoDELS)*, 2008.

[12] I. Weber, G. Governatori and J. Hoffmann, "Approximate Compliance Checking for Annotated Process Models," in *International Workshop on Governance, Risk and Compliance in Information Systems (GRCIS)*, 2008.

[13] RTCA, *Design Assurance Guidance for Airborne Electronic Hardware*, http://www.rtca.org/onlinecart/product.cfm?id=194, 2000.

[14] M. Bellato, M. Ceschia, M. Menichelli, A. Papi, J. Wyss and A. Paccagnella, "Ion beam testing of SRAM-based FPGA's," in *Radiation and Its Effects on Components and Systems, 6th European Conference on*, Padova Univ., Italy, 2001.

[15] Xilinx, Inc., *LogiCORE IP Soft Error Mitigation Controller v3.1 (DS796)*, 2011.

Cryptanalysis on Reconfigurable Computers

Tim Güneysu
Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
Email: tim.gueneysu@rub.de

***Abstract*—In this work, we present five years of development and improvements on COPACOBANA, a reconfigurable cluster system dedicated specifically to the task of cryptanalysis. Latest changes on the architecture involve modifications for larger and more powerful FPGA devices with dedicated 32 MB of external RAM and point-to-point communication links for improved data throughput. We present a series of cryptanalytic applications that are currently available and benefit from the latest improvements on the architecture.**

I. Introduction

The security of symmetric and asymmetric ciphers is usually determined by the size of their security parameters, in particular the key-length. Hence, when designing a cryptosystem, these parameters need to be chosen according to the assumed computational capabilities of an attacker. Depending on the chosen security margin, many cryptosystems are potentially vulnerable to attacks when the attacker's computational power increases unexpectedly. In real life, the limiting factor of an attacker is often the financial resources. Thus, it is quite crucial from a cryptographic point of view to not only investigate the complexity of an attack, but also to study possibilities to lower the cost-performance ratio of attack hardware. For instance, a cost-performance improvement of an attack machine by a factor of 1000 effectively reduces the key lengths of a symmetric cipher by roughly 10 bit (since $1000 \approx 2^{10}$). Many cryptanalytical schemes spend their computations in independent operations, which allows for a high degree of parallelism. Such parallel functionality can be realized by individual hardware blocks that operate simultaneously, improving the running time of the overall computation by a perfect linear factor. At this point, it should be remarked that the high non-recurring engineering costs for ASICs have put most projects for building special-purpose hardware for cryptanalysis out of reach for commercial or research institutions. However, with the recent advent of low-cost programmable ICs which host vast amounts of logic resources, special-purpose cryptanalytical machines have now become a possibility outside government agencies.

In this work we present the evolution of a special-purpose hardware computer dedicated to the task of cryptanalysis which provides a cost-performance that can be significantly better than that of recent PCs (e.g., for the exhaustive key search on DES). The hardware architecture of this Cost-Optimized Parallel Code Breaker (COPACOBANA) was initially introduced in 2006 [12]. In this contribution we provide an overview on cryptanalytical applications and present improvements on the architecture to cope with requirements of further, advanced applications.

The original prototype of the COPACOBANA cluster consists of up to 120 FPGA nodes which are connected by a shared bus providing an aggregate bandwidth of 1.6 Gbps on the backplane of the machine. COPACOBANA is not equipped with dedicated memory modules, but offers a limited number of RAM blocks inside each FPGA. Furthermore, COPACOBANA is connected to a host PC with a single interface to control all operations and provide a little amount of I/O data.

In the following sections, we present cryptanalytic case studies for a large variety of attacks which all make use of the COPACOBANA cluster system. Examples for these case studies include exhaustive key search attacks on the Data Encryption Standard (DES) blockcipher and related systems and the GSM mobile phone encryption based on the A5/1 streamcipher. More advanced attacks with COPACOBANA comprise implementations for integer factorization (with the Elliptic Curve Method), computations on elliptic curve discrete logarithms and Time-Memory Tradeoffs (TMTO). We briefly review attack implementations and compile a list of improvements on hardware level that can lead to improved performance. Finally, we present a modified cluster architecture which addresses most of the determined issues and promises excellent performance results for the next generation of cryptanalytic applications.

The paper is structured as follows: we begin with a brief review of the original COPACOBANA architecture and a list of case studies on cryptanalytic attacks in Section III. Next, we present the modified cluster architecture with improvements based on our findings

in the previous section. We conclude with an outlook on future cryptanalytic implementations based on COPACOBANA.

II. Architecture of COPACOBANA

Our first prototype of an Cost-Optimized Parallel Code Breaker (COPACOBANA) was produced for less than € 10,000 (material and manufacturing costs only). It was primarily designed for applications and simple cryptanalytic attacks with high compuational complexity but minimal requirements on communications and local memory. In addition to that, it assumes that the computationally expensive operations are inherently parallelizable, i.e., single parallel instances do not need to communicate with each other. The design for limited communication bandwidth was driven by the fact that the computation phase heavily outweighs the data input and output phases. In fact, COPACOBANA was designed for applications in which processes are computing most of the time, without any input or output. Communication was assumed to be almost exclusively used for initialization and reporting of results. A central control instance for the communication can easily be accomplished by a conventional (low-cost) PC, connected to the FPGAs on the cluster by a simple interface. Furthermore, simple brute-force attacks typically demand for very little memory so that we considered the available memory on low-cost FPGAs (such as the Xilinx Spartan-3 devices) to be sufficient.

Recapitulating, COPACOBANA consists of many independent low-cost FPGAs, connected to a host-PC via a standard interface, e.g., USB or Ethernet. The benefit of such a standard interface is the easy scalability and to attach more than one COPACOBANA device to a single host-PC. Note that the initialization, control of FPGAs, and the accumulation of results is done by the host. All time-critical computations such as the cryptanalytical core tasks are performed on the FPGAs. The first prototype of COPACOBANA was equipped with up to 120 FPGAs, distributed along 20 slots - in FPGA modules which can be plugged into a single backplane. Note that the choice for 120 FPGAs was driven by the form factor of the FPGA module which was designed according to cheap and standardized DIMM interface specifications. One (single-sided) DIMM-sized FPGA module can host 6 FPGA devices in a 17×17 mm package, such as the Xilinx Spartan3-1000 FPGA (XC3S1000, speed grade -4, FT256 packaging). This device comes with 1 million system gates, 17280 equivalent logic cells, 1920 Configurable Logic Blocks (CLBs) equivalent to 7680 slices, 120 Kbit Distributed RAM (DRAM), 432 Kbit Block RAM (BRAM), and 4 digital clock managers (DCMs). The backplane of COPACOBANA connects all FPGA devices with a shared 64-bit data and 16-bit address bus. The entire cluster system is depicted in Figure 1.

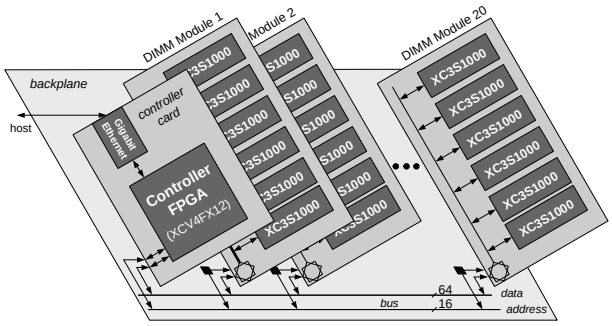


Fig. 1. Architecture of COPACOBANA

COPACOBANA was designed for single master bus arbitration for simplified control. However, in case the communication scheduling of an application is not predictable, the bus master is required to poll all FPGAs for new events and returned data. This significantly slows down the communication performance and increases latencies when reading data back from the FPGAs. Data transfer from and to the FPGAs is accomplished by a dedicated control unit. Originally, we decided to pick a small development board with an FPGA (CESYS USB2FPGA) in favor of a flexible design. The board provides an easy-pluggable 96-pin connector which we use for the connection to the backplane. In later versions of COPACOBANA, we replaced the USB controller using an TCP/IP-based unit so that COPACOBANA can be controlled remotely and can be placed externally, for example in a server room.

III. Cryptanalytic Applications for COPACOBANA

In this section, we briefly describe cryptanalytic applications which we have already implemented on our initial release of the COPACOBANA cluster system. We compiled the most important facts and key points of each application into individual case studies which eventually should help to identify shortcomings and potential enhancements of our cluster architecture.

A. Key Search Applications

In the following sections, we present a short survey about our work on exhaustive key search and guessing attacks on a variety of real-world systems. Since all these applications consist mostly out of very basic tasks that can be efficiently parallelized on completely independent

computational cores, they can be perfectly mapped onto a highly parallel cluster system such as COPACOBANA.

1) Case-Study I: Breaking DES with Exhaustive Search: Our first cryptanalytic target application was the exhaustive key search on the DES block cipher. We implemented a known-plaintext attack and used an improved version of the DES engine of the Université Catholique de Louvain's Crypto Group [18] as a core component. Inside a single FPGA, we could place four of such DES engines which allows for sharing plaintext-ciphertext input pairs and the key space. Our first implementation and successful attack was presented in [12]. Since this original publication, we were able to improve the system performance by use of additional pipelined comparators and simplified control logic. Now, we are able to operate each of the FPGAs at an increased clock rate of 136 MHz with a overall gain in performance by 36%, compared to [12]. Consequently, 2^{42} keys can be checked in $2^{40} \times 7.35\,\text{ns}$ by a single FPGA, which is approximately 135 minutes. Since COPACOBANA hosts 120 of these low-cost FPGAs, the entire system can check $4 \times 120 = 480$ keys every $7.35\,\text{ns}$, i.e., 65.28 billion keys per second. To find the correct key, COPACOBANA has to search through an average of 2^{55} different keys. Thus, it can find the right key in approximately $T = 6.4$ days on average. By increasing the number n of COPACOBANAs used for this task, we can further decrease this average runtime of the attack by a linear factor $1/n$.

2) Case-Study II: Breaking the A5/1 Streamcipher: A5/1 is a synchronous stream cipher that is used for protecting GSM communication. In the GSM protocol, the communication channel is organized in 114-bit frames that are encrypted by XORing them with 114-bit blocks of the keystream produced by the cipher as follows: A5/1 is based on three LFSRs, that are irregularly clocked. The three registers are 23, 22 and 19 bits long, representing the internal 64-bit state of the cipher. During initialization, a 64-bit key k is clocked in, followed by a 22-bit initialization vector that is derived from the publicly known frame number. After that a warm-up phase is performed where the cipher is clocked 100 times and the output is discarded. For a detailed description of A5/1 please refer to [1].

Most of previously proposed attacks against A5/1 lack from practicability and/or have never been fully implemented. In contrast to these attacks, we present in [6] a real-world attack revealing the internal state of A5/1 in about 6 hours on average (and about 12 hours in the worst-case) using COPACOBANA. The implementation is an optimization of a *guess-and-determine* attack as proposed in [11], including an improvement in runtime of about 13% compared to their original approach. Each FPGA contains 23 guessing engines running in parallel at a clock frequencey of 104 MHz each. To mount the attack, only 64 consecutive bits of a known keystream are required and we do not need any precomputed data. Note, however, that an average of 6 hours runtime still cannot be considered a real-time attack when using a single COPACOBANA. In this case, we need to record the full communication first and attack its encryption offline afterwards. Alternatively, by adding further machines the attack time will be linearly reduced, e.g., 100 machines only require 3.6 minutes for a sucessfull attack on average.

B. Advanced Cryptanalytic Applications

In the last section, we briefly surveyed simple attacks based on exhaustive key searches or guessing. All these attacks have in common that their performance is basically limited by the number of computations. In other words, the available logic of the FPGA devices on COPACOBANA directly determines the performance of the attack. By incrementing the number of COPACOBANA units we yield a speed-up in performance by a perfect linear factor. This, however, does not hold for the following, more advanced attacks.

1) Case-Study III: Time-Memory Tradeoffs: The Time-Memory Tradeoff (TMTO) method was designed as a compromise between the two well-known extreme approaches: either to perform an exhaustive search on the entire key space of the cipher or precomputing exhaustive tables representing all possible combinations of keys and ciphertexts for a given plaintext. The TMTO strategy offers a way to reasonably reduce the actual search complexity (by doing some kind of precomputation) while keeping the amount of precomputed data reasonably low, whereas "reasonably" has to be defined more precisely. Roughly speaking, it depends on the concrete attack scenario (e.g., real-time attack), the internal step function and the available resources for the precomputation and online (search) phase.

Existing TMTO methods by Hellman, Rivest and Oechslin [10], [3], [15] share the natural property that in order to achieve a significant success rate much precomputation effort is required on chained computations. A representation of start point and end point of each chain is stored in (a set of) large tables, e.g., on hard disk drives. The actual attack takes place in a second search phase (online phase) in which another chain computation is performed on the actual data and compared to the stored endpoints in the tables. In case a matching endpoint is found in the table, the sequence

Method	DU [GB]	PT (COPA) [days]	OT [ops]	TA	SR
Hellman	1897	24	$2^{40.2}$	$2^{40.2}$	0.80
Rivest	1690	95	2^{21}	$2^{39.7}$	0.80
Oechslin	1820	23	$2^{21.8}$	$2^{40.3}$	0.80

TABLE I
TMTO METHODS ACCORDING TO: EXPECTED RUNTIMES AND MEMORY REQUIREMENTS USING COPACOBANA FOR PRECOMPUTATIONS.

of keys can be reconstructed using the corresponding start point. There are few contributions attacking DES with the TMTO approach. In [20] an FPGA design for an attack on a 40-bit DES variant using Rivest's TMTO method [3] was proposed. In [14] a hardware architecture for UNIX password cracking based on Oechslin's method [15] was presented. However, to the best of our knowledge, a set of complete TMTO precomputation tables for *full* 56-bit DES was never created up to now.

In [8] we present possible configurations and parameters to use COPACOBANA for TMTO/TMDTO precomputations both to attack the DES blockcipher and the A5/1 streamcipher. Our estimates took the assumed communication bandwidth between host-PC and backplane of 24 MBit/s into account. To break DES with TMTOs on COPACOBANA, Table I presents our worst case expectations concerning success rate (SR), disk usage (DU), the duration of the precomputation phase (PT) as well as the number of table accesses (TA) and calculations (C) during the online phase (OT). Note that for this extrapolation, we have used again the implementation of our exhaustive key search on DES (cf. Section III-A1).

2) Case-Study IV: Integer Factorization: The factorization of a large composite integer n is a well-known mathematical problem which has attracted special attention since the invention of public key cryptography. RSA is known as the most popular asymmetric cryptosystem and was originally developed by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977 [17]. Since the security of RSA relies on the attacker's inability to factor large numbers, the development of a fast factorization method could allow for cryptanalysis of RSA messages and signatures. Recently, the best known method for factoring large RSA integers is the General Number-Field Sieve (GNFS). An important step in the GNFS algorithm is the factorization of mid-sized numbers for smoothness testing. For this purpose, the Elliptic Curve Method (ECM) has been proposed by Lenstra [13] which has been proved to be suitable for parallel hardware architectures in [19], [5], [4], particularly on FPGAs.

In [4] it has been shown that the utilization of DSP slices in Virtex-4 FPGAs for implementing a Montgomery multiplication can significantly improve the ECM performance. In that work, the authors used a fully parallel multiplier implementation which provides the best known performance figures for ECM phase 1 so far, however they did provide details how to realize ECM phase 2.

To accelerate integer arithmetic using a similar strategy, we designed a new slot-in module for use with a second release of COPACOBANA hosting 8 Xilinx Virtex-4 XC4VSX35 FPGAs, each providing 192 DSP slices. Due to the larger physical package of the FPGAs (FF668 package with dimension of 27x27 mm) we enlarged the modules. This included also modifications of the corresponding connectors on the backplane. For more efficient heat dissipation at high clock frequencies up to 400 MHz, an actively ventilated heat sink is attached to each FPGA. With a more powerful power supply providing 1.5 kW at 12 V, we could run a total of 128 Virtex 4 SX35 FPGAs distributed over 16 plug-in modules.

In contrast to [4], we used a multi-core ECM design per FPGA. A single ECM engine comprises of an arithmetic unit computing modular multiplication and additions, a point multiplication unit for phase 1 and ROM tables for phase 2. Only point operations on the elliptic curve are performed on FPGAs, this means that the setup of the Montgomery curve needs to be done on the host-PC and then transferred to the FPGAs. Table **??** shows the results for the block size $b = 10$, factoring integers up to 151 bit in 4.73 ms (phase 1) and 5.12 ms (phase 2). This corresponds to 5,064 computations per second for the first phase and 2,424 for phase 1+2.

A main issue of ECM is memory. Although ECM phase 1 has only very moderate memory constraints, phase 2 involves a significant amount of precomputations as well as storage for prime numbers. Since memory on FPGA devices is rather limited (192 $\times$ 18 KBit BRAM elements per device), a fast accessible, external memory could help to improve the impact of phase 2 by storing even larger tables than presented above.

3) Case-Study V: Solving Elliptic Curve Discrete Logarithms: Another popular problem used for building public-key cryptosystems is known as the Discrete Logarithm Problem (DLP) where the exponent ℓ should be determined for a given $a^{\ell} \mod n$. A popular derivative is the Elliptic Curve Discrete Logarithm Problem (ECDLP) for Elliptic Curve Cryptosystems (ECC) [9].

Aspect	Gaj et al. [5]	This work	Factor
FPGA Device	V4LX200-11	V4SX35-10	
Supported Bits	198	202	1.02
Max. ECM Cores	24	24	1.00
Max. Frequency	104 MHz	200 MHz	1.92
Cycles for Addition	41	29	0.71
Cycles for Multiplication	216	201	0.93
Cycles for Phase 1	1,666,500	1,473,596	0.88
Time for Phase 1	16 ms	7.37 ms	0.46
(# Phase 1)/s	1,448	3,240	2.24
(# Phase 1+2)/s	696	1,560	2.24

TABLE II
COMPARISON OF OUR RESULTS USING $b = 13$ (L = 202 BIT) AND $b = 9$ (L = 134 BIT) BY GAJ ET AL. FOR VIRTEX-4 FPGAS.

k	Certicom Est. [2]	XC3S1000 FPGA	COPACOBANA
79	146 d	15.3 d	3.06 h
89	12.0 y	1.62 y	4.93 d
97	197 y	30.7 y	93.4 d
109	$2.47 \cdot 10^4$ y	$2.91 \cdot 10^3$ y	24.3 y
131	$6.30 \cdot 10^7$ y	$7.40 \cdot 10^6$ y	$6.17 \cdot 10^4$ y
163	$6.30 \cdot 10^{12}$ y	$9.15 \cdot 10^{11}$ y	$7.62 \cdot 10^9$ y
191	$1.32 \cdot 10^{17}$ y	$1.89 \cdot 10^{16}$ y	$1.57 \cdot 10^{14}$ y
239	$3.84 \cdot 10^{24}$ y	$8.62 \cdot 10^{23}$ y	$7.18 \cdot 10^{21}$ y

TABLE III
EXPECTED RUNTIME ON DIFFERENT PLATFORMS AND FOR DIFFERENT CERTICOM ECC CHALLENGES

An attack on ECC relies on the same algorithmic primitives as the crypto system itself, namely point addition and point doubling. Up to now, the best known algorithm for this purpose is the Pollard's Rho (PR) algorithm for parallel implementation described in [21]. This variant of the original PR method [16] allows for a linear gain in performance with the number of available processors. This can be efficiently implemented in hardware as presented in [7].

The PR algorithm essentially determines distinguished points on the elliptic curve. These points are reported to a central host computer which awaits a collision of two points. A distinguished point is defined to be a point with a specific characteristic, e.g., its x-coordinate has a fixed number of leading zero bits. To reach such a distinguished point, PR follows a so called pseudo-random walk on the elliptic curve by subsequently adding points from a fixed, finite set of random points. Hence, with careful parametrization of the distinguished point criterion, the duration of a computation until a distinguished point is found can be adapted to the bandwidth constraints of the system. Furthermore, the PR does not need a large memory for computation so that the COPACOBANA system seems to be a suitable platform for running the algorithm. As with the ECM unit, a single PR unit is comprised of an arithmetic unit, a few kilobytes of RAM and control logic. The arithmetic unit supports modular inversion as an additional function required for uniquely determining distinguished points.

For a parallelized PR on COPACOBANA according to the method presented in [21], all instances of the algorithm can run completely independent from each other. For solving the discrete logarithm problem over curves defined over prime fields $\mathbb{F}_p$, we have to compute approximately $\sqrt{q}$ points, where q is the largest prime power of the order of the curve. Note that the transfer of data between host computer and point processing units on the FPGA can be performed independently from the computations.

Implementing the PR on Spartan-3 FPGAs for solving the ECDLP over curves with a length of 160 bits and using an affine point representation, we achieve a maximum clock frequency of approximately 40 MHz and an area usage of 6067 slices (79%) for two parallel instances. The corresponding point addition requires 846 cycles so that slightly less than 50,000 point operations can be performed per second by one unit. Consequently, a single COPACOBANA can compute about 11.3 million point operations per second. Table III compares our results for COPACOBANA with the corresponding estimates from Certicom for their challenges (based on the computing time of an (outdated) Intel Pentium 100).

IV. IMPROVING THE ARCHITECTURE

According to the issues of our original COPACOBANA architecture as shown above, we initiated a major redesign of the machine with respect to the following aspects:

- *Larger FPGA Devices:* more logical elements on an FPGA enable more complex applications or more computational cores per device.
- *Local Memory per FPGA:* a few megabytes of fast SRAM should be placed adjacent to each FPGA device to provide additional storage while solving more complex problems.
- *Dedicated Communication Links*: Only a single access to an FPGA at a time was possible in the recent communication models. Individual links for each FPGA simplify data communication and also enable high performance from simultaneous data exchange.
- *Improved Host Controller*: the main bottleneck with respect to data communication is the controller interface between backplane and host-PC. To improve

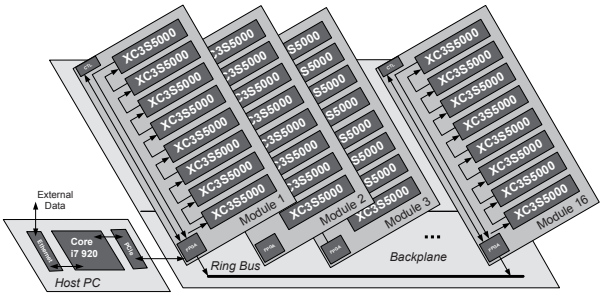


Fig. 2. Enhanced COPACOBANA Architecture based on Xilinx Spartan-3 5000 FPGAs

the situation, a host-PC could be integrated inside COPACOBANA and directly connect the host's mainboard with the backplane using a short, high-performance link.

To address the aspects mentioned above, we entirely redesigned the backplane and FPGA module of our cluster system. The new FPGA module consists of 8 FPGAs with a package size up to $27 \times 27mm$, a CPLD for system control (e.g., temperature and voltage monitoring) and DC/DC converters (depending on the FPGA type used). Most suitable FPGA devices for the new systems are the low-cost Xilinx Spartan-3 5000 with up to 74,880 logic cells (104 hardware multipliers, 104 BRAMs, FG676 packaging), the Xilinx Spartan-3A DSP 3400 with 53,712 logic cells (126 DSP48A, 126 BRAMs, FG676 packaging) and the upcoming class of Spartan-6 FPGAs. For the first prototype of the enhanced COPACOBANA, we chose 128 Spartan-3 5000 devices. Moreover, we placed 32 MB of SRAM adjacent to each FPGA that can be accessed by the FGPA within a single clock cycle at 100 MHz.

The new COPACOBANA design integrates the host-PC in the same case. Target applications, as for example Time-Memory Tradeoffs, require a high data rate between COPACOBANA and an external hard disk drive. With the PC we placed a hard disk drive physically inside COPACOBANA and attached it via SATA to the integrated host-PC. Additional components of the new system are the $1.5kW$ main power supply unit (with 125A at 12V), six high-performance fans and a 19-inch rack of three hight units for the housing.

There are fast serial point-to-point connections between every two neighbors building a chain of FPGAs. In the configuration as described below, we employed eight Xilinx Spartan-3 5000 which are arranged as a systolic one-dimensional array, i.e., in each clock cycle data is transferred from one FPGA to the next in pipeline fashion according to a global, synchronous clock. Note that transferring data using a systolic array introduces significant latencies on the data path. However, since the target applications do not have real-time requirements, this should not be an issue for our cryptanalytic applications where operations usually can be interleaved to hide any latencies.

As in the original system, the backplane connects to all FPGA cards on which individually the clock signals, data and power are (re-)generated and distributed. However, the bus is managed now cooperatively by all FPGA modules instead of a single bus master, i.e., a central controller. Each FPGA module can transfer incoming data to the next slot or it can remove the data out of the stream. In this case, an empty data frame cycles through the bus pipeline from slot to slot. Note that another card is allowed to insert new data now into this empty slot. The two counter-rotating systolic datapaths allow to minimize the worst case latency to half of the total number of slots multiplied by the clock cycle time. The enhanced bus system assigns one ascending and one descending slot to each single card. This leads to a ring of point to point connections in which the bus system can be seen as a circular, parallel shift register.

Due to the modular architecture further developments can be incorporated seamlessly. For example, alternative FPGA modules equipped with a Spartan-3A DSP 3400 or Spartan-6 FPGAs can easily plugged into the backplane. In particular, it is possible to run even a heterogenous configuration, with a mixed set of FPGAs for different tasks.

V. Improvements in Cryptanalytical Applications

At the time of writing, the cluster incorporating the presented enhancements as described in Section IV is still in production and is expected to become available in October 2009. Hence, we now provide first estimates and projections concerning the expected performance of the new cluster system. We will revise our figures as soon as the system (and adapted cryptanalytical implementations) become available. With the design modifications on the COPACOBANA architecture, we can firstly make use of more logic resources due to the larger Spartan-3 5000 FPGAs. With respect to the original Spartan-3 1000 FPGAs, the amount of logic has increased by a factor of 4.5 per FPGA device. For our exhaustive key search applications as shown in Section III-A, we achieve a linear speedup by a factor of 4. More precisely, the original DES breaking application implemented four engines per Spartan-3 1000 so that we could place 16 engines on each Spartan-3 5000. This reduces the average runtime to break DES to 1.6 days

on average. Similar linear performance speed-ups can be gained for similar applications and the A5/1 breaker (about 1.5 hours). However, note that due to the higher cost of the enhanced COPACOBANA (which grows by a factor of 4.5 as well), the cost-performance is not better than with the original machine. In general, brute-force techniques do not benefit from the new architectural improvements. In this case, the only advantage of the new design is due to the reduced power consumption.

Advanced cryptanalytical applications with additional requirements benefit significantly on communication throughput and fast, local memory. The efficient generation of TMTO tables will now become available so that we expect to finish the tables for A5/1 in less than a month. Furthermore, we are working towards an implementation of our ECM core (cf. Section III-B2) for Spartan-3A DSP 3400 FPGAs which also integrate 126 DSP slices, however at much lower costs compared to Virtex-4 devices. Finally, we plan to adapt the same implementation strategy based on DSP slices also for the Pollard-Rho ALU. This can also result in a gain in performance by an order of magnitude. Last but not least, the new COPACOBANA also could provide a suitable platform for even more complex applications, e.g., additional tasks required by the number field sieve, index calculus methods or lattice basis reduction algorithms.

VI. Conclusion

In this work, we presented a series of cryptanalytical applications for cost-efficient hardware architectures. According to our findings based on a variety of cryptanalytical implementations, we identified shortcomings in the design of our cluster. Then, we came up with an enhanced version which also provides larger and more powerful FPGAs, up to 4 GB of local memory and fast point-to-point serial communication links between all devices and the controller. These modifications enable the computation of even more complex cryptanalytic tasks but also other problems from other domains. With all enhancements, in particular for communication and local memory, the new cluster becomes useful beyond pure code-breaking, catching up with respect to other supercomputing platforms, but still at low costs.

Acknowledgments

This work has been supported by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02/2-005-WFBO-009).

References

[1] A. Biryukov, A. Shamir, and D. Wagner. Real Time Cryptanalysis of A5/1 on a PC. In *FSE*, volume 1978 of *LNCS*, pages 1–18. Springer, 2001.

[2] Certicom Corporation. Certicom ECC Challenges, 2005. http://www.certicom.com.

[3] D. Denning. Cryptography and Data Security. Addison-Wesley, 1982.

[4] G. de Meulenaer, F. Gosset, M. M. de Dormale, and J.-J. Quisqater. Integer Factorization Based on Elliptic Curve Method: Towards Better Exploitation of Reconfigurable Hardware. In *FCCM*, pages 197–206. IEEE Computer Society, 2007.

[5] K. Gaj, S. Kwon, P Baier, P. Kohlbrenner, H. Le, M. Khaleeluddin, and R. Bachimanchi. Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware. In *CHES*, volume 4249 of *LNCS*, pages 119–133. Springer, 2006.

[6] Timo Gendrullis, Martin Novotný, and Andy Rupp. A real-world attack breaking A5/1 within hours. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *LNCS*, pages 266–282. Springer, 2008.

[7] T. Güneysu, C. Paar, and J. Pelzl. Attacking Elliptic Curve Cryptosystems with Special-Purpose Hardware. In *FPGA*, pages 207–215. ACM Press, 2007.

[8] Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, and Andy Rupp. Cryptanalysis with COPACOBANA. *IEEE Trans. Comput.*, 57(11):1498–1513, November 2008.

[9] D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.

[10] M. E. Hellman. A Cryptanalytic Time-Memory Trade-Off. In *IEEE Transactions on Information Theory*, volume 26, pages 401–406, 1980.

[11] J. Keller and B. Seitz. A Hardware-Based Attack on the A5/1 Stream Cipher. Technical report, Fernuni Hagen, Germany, 2001. http://pv.fernuni-hagen.de/docs/apc2001-final.pdf.

[12] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In *CHES*, volume 4249 of *LNCS*, pages 101–118. Springer, 2006.

[13] H. Lenstra. Factoring Integers with Elliptic Curves. *Annals Math.*, 126:649–673, 1987.

[14] N. Mentens, L. Batina, B. Prenel, and I. Verbauwhede. Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking. In *ARC*, volume 3985 of *LNCS*, pages 323–334. Springer, 2006.

[15] P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *CRYPTO*, volume 2729 of *LNCS*, pages 617–630. Springer, 2003.

[16] J. M. Pollard. Monte Carlo Methods for Index Computation mod *p*. *Mathematics of Computation*, 32(143):918–924, July 1978.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[18] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES. In *FPGA*, pages 247–247. ACM, 2003.

[19] M. Šimka, J. Pelzl, T. Kleinjung, J. Franke, C. Priplata, C. Stahlke, M. Drutarovský, V. Fischer, and C. Paar. Hardware Factorization Based on Elliptic Curve Method. In *FCCM*, pages 107–116. IEEE Computer Society, 2005.

[20] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat. A Time-Memory Tradeoff using Distinguished Points: New Analysis & FPGA Results. In *CHES*, volume 2523 of *LNCS*, pages 596–611. Springer, 2002.

[21] P.C. van Oorschot and M.J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999.

A Practical FPGA Implementation of Regular Expression Matching with Look-ahead Assertion

Yoichi Wakaba, Masato Inagi, Shin'ichi Wakabayashi
Graduate School of Information Sciences, Hiroshima City University
3-4-1 Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194 Japan
email: waka27@lcs.info.hiroshima-cu.ac.jp {inagi, wakaba}@hiroshima-cu.ac.jp

***Abstract*—This paper presents a practical FPGA implementation of a hardware algorithm for regular expression matching with look-ahead assertion. Look-ahead assertion is an operation in Perl compatible regular expression, and used for conditional matching. A two-stage FPGA implementation that can directly handle look-ahead assertion has already been proposed. However, it cannot handle long texts due to the limited size of on-chip memory used for the buffer between stages. Our proposed implementation functions in cooperation with a PC and an off-chip memory with little degradation by using properly-sized buffers to communicate with the off-chip memory.**

I. Introduction

Regular expression matching is playing important roles in various fields. For example, it is used in network intrusion detection systems (NIDSs), which detect malicious traffic (*e.g.*, computer virus) on computer networks and notify a firewall of the detections, in order to describe virus patterns and so on. Because of today's high-speed networks and increase of the virus patterns, speed-up of regular expression matching is being required. Thus, hardware algorithms for regular expression matching have been being studied (*e.g.*, [2]–[9]).

How to handle a lot of virus patterns efficiently is an essential matter for regular expression matching hardware engines for virus detection. To reduce the necessary amount of hardware resources, the one proposed in [4] shortens the total length of patterns by finding common sub-patterns (*e.g.*, prefix) and merging those.

In a well-known software NIDS, Snort, virus patterns are described in extended regular expression, called Perl compatible regular expression (PCRE). One of the operations in PCRE, called look-ahead assertion, is written as "R_1 (?=R_2) R_3" or "R_1 (?!R_2) R_3", where R_1, R_2 and R_3 are arbitrary regular expressions. "R_1 (?=R_2) R_3" matches a string whose first part matches R_1 and whose second part matches both R_2 and R_3. Likewise, "R_1 (?!R_2) R_3" matches a string whose first part matches R_1 and whose second part does not match R_2 and matches R_3. By using look-ahead assertion, strings not including a given sub-string are simply and efficiently described. For example, "$.^*@(?!\ hiroshima\text{-}cu\backslash.ac\backslash.jp)$" detects e-mails from domains other than "@hiroshima-cu.ac.jp". Since most of the existing hardware matching engines cannot directly handle look-ahead assertion, look-ahead assertion patterns need to be transformed into general regular expression patterns. However, in most cases, transformed regular expression patterns are very long and require a huge amount of hardware resources.

Thus, the authors of [11] proposed a hardware algorithm for directly handling look-ahead assertion, and implemented it on an FPGA. To handle look-ahead assertion, they introduced a preprocessing circuit which conducts matching for R_2. The matching result for R_2 is used in the main circuit to determine if it starts matching for R_3 after matching for R_1 succeeds. It is performed without backtracks, and thus in a pipeline fashion. Furthermore, they introduced a special ring buffer structure for the preprocessing circuit and the main one to achieve high-throughput processing. However, the straightforward FPGA implementation of the algorithm can handle only short texts (*e.g.*, IP packet), because of the limited amount of on-chip memory (*i.e.*, block RAM) used for the buffers.

In this paper, we present a practical FPGA implementation of the algorithm that can handle long packets (e.g., TCP packet). While the entire process of matching is performed inside the FPGA in the previous implementation, our proposed implementation functions in cooperation with a PC and an off-chip memory. Overhead time to communicate with the PC and memory is hidden by properly-sized buffers, and thus it functions with little degradation.

The rest of this paper is organized as follows. In Section II, look-ahead assertion is defined, and the previous FPGA implementation of the hardware algorithm for look-ahead assertion is explained. Section III presents our proposed implementation. In Section IV, we analyze the performance of our proposed implementation. Finally, Section V gives some concluding remarks.

II. Preliminaries

A. Regular Expression

Regular expression is a method to represent a set of strings as a single string with operators. It is often used to describe a pattern compactly. A regular expression represents a set of strings by using three basic operators: union (|), concatenation ($\cdot$) and Kleene operator ($*$). These operators are defined as follows: $R_1|R_2 = \mathrm{L}_1 \cup \mathrm{L}_2$, $R_1R_2 = \mathrm{L}_1\mathrm{L}_2 = \{s_1s_2 | s_1 \in \mathrm{L}_1, s_2 \in \mathrm{L}_2\}$ and $R^* = \{\varepsilon\} \cup \mathrm{L}^1 \cup \mathrm{L}^2 \cup \cdots$, where ε is a special character which matches the string with no character (*i.e.*, ""), R, R_1 and R_2 are arbitrary regular expressions, L, L_1 and L_2 are sets of strings corresponding to R, R_1 and R_2, respectively.

Extended regular expressions can describe a pattern more compactly. In this paper, we focus on one of the extended regular expressions, Perl compatible regular expression, which is widely used in various fields. It has various operators (*e.g.*, back reference, look-ahead assertion, etc.). Due to limited space, we explain look-ahead assertion, focused on in this paper.

Look-ahead assertion can be used to compactly describe patterns that match strings not including specified substrings, and patterns that match stings including both of specified two substrings [10]. They are described as follows: one of them is called positive look-ahead and described as"R_1 $(?{=}R_2)$ R_3", and the other is called negative look-ahead and described as "R_1 $(?!R_2)$ R_3", where R_1, R_2 and R_3 are arbitrary regular expressions. Positive look-ahead represents a set of strings that match both R_1R_2 and R_1R_3. (Strings that match R_2 are substrings of strings that match R_3, or vice versa.)

Negative look-ahead represents a set of strings that match R_1R_3 and does not match R_1R_2. In this paper, a look-ahead assertion denotes a whole pattern "R_1 $(?{=}R_2)$ R_3" or "R_1 $(?!R_2)$ R_3", and a look-ahead pattern denotes only R_2.

We show examples of matching for look-ahead assertion in Fig.1. Fig.1(a) shows a normal regula expression, and Fig.1 (b) and (c) are look-ahead assertion.

B. Hardware-based regular expression matching with look-ahead assertion

In this subsection, we briefly explain an existing two-stage hardware engine for regular expression matching with look-ahead assertion [11], which our proposed implementation is based on.

1) Method for handling look-ahead assertion: Due to limited space, we here focus on how to handle look-ahead assertion using general regular expression matching engines. (In this paper, *general regular expression* refers to unextended regular expression.)

As mentioned above, the existing matching engine for look-ahead assertion has two stages, and thus consists of two general regular expression matching engines.

In the engine, matching for R_2 is performed in the first stage, and then matching for a pattern R_1R_3 is performed in the second stage by using the matching result for R_2.

Next, we explain the behavior of the hardware matching algorithm using Fig.2. In order to realize a look-ahead assertion, it first performs matching for the reversed pattern of R_2 and the reversed text to find the initial characters of the substrings that match R_2. Note that the general regular expression matching engine employed in [11] can find only the terminal characters of the substrings that match a given pattern. This is the reason why R_2 and the text are reversed. Please refer to [11] for more details about the reversed-order matching. Then, it performs matching of R_1R_3 and the text. In a positive (negative) look-ahead assertion, if the matching start position for R_3 is the same as that of the initial character of a substring that matches R_2 (does not match R_2), the second-stage matching engine performs matching for R_3.

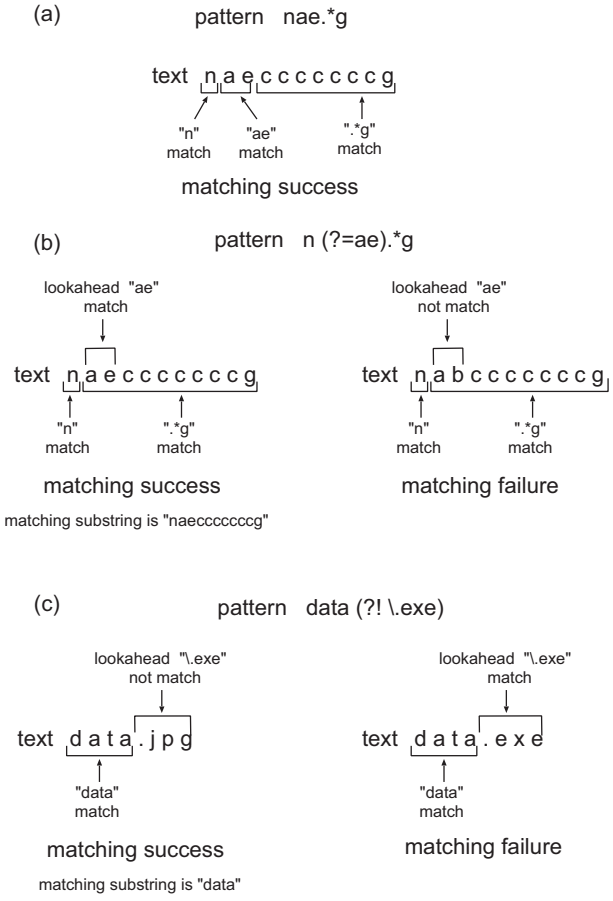


Fig. 1. Examples of matching for look-ahead assertion.

2) Architecture of the previous implementation: In this subsection, we present an architecture which realizes an existing algorithm for matching look-ahead assertion. The architecture is shown in Fig.3. The architecture consists of two matching engines M_b which performs matching for R_1R_3 and a preprocessing circuit which performs matching for R_2. The preprocessing circuit consists of a matching engine M_a and two stack memories S_a and S_b. Note that, if a given pattern does not include a look-ahead assertion, two matching engines, M_a and M_b, can be used as one matching engine by ignoring the matching result at M_a. The behavior of the proposed architecture is shown in the following enumeration.

1) The reversed look-ahead pattern ${R_2}^R$ is set to the matching engine M_a, and R_1R_3 is set to the matching engine M_b.
2) When matching starts, a text is stacked in the stack memory S_a to make a reversed text.
3) After the terminal character of a text is stacked in the stack memory S_a, the characters of the text are input to the matching engine M_a from the end of the text.
4) The matching engine M_a outputs characters with in-

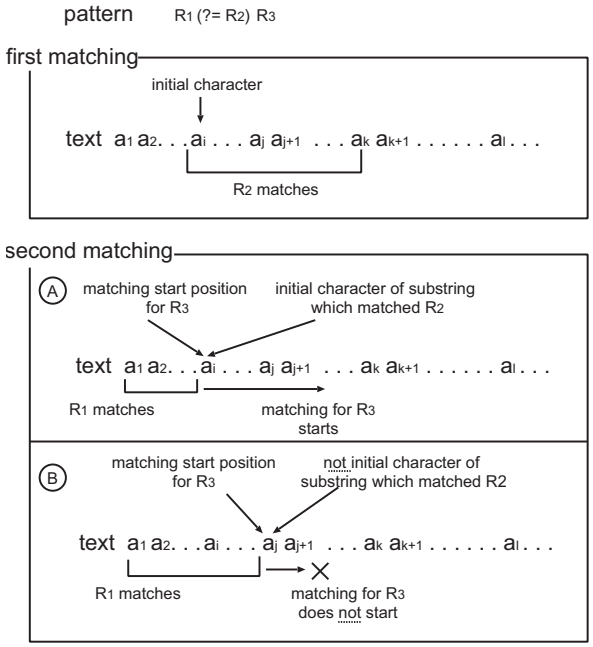


Fig. 2. The behavior of a matching method for look-ahead assertion

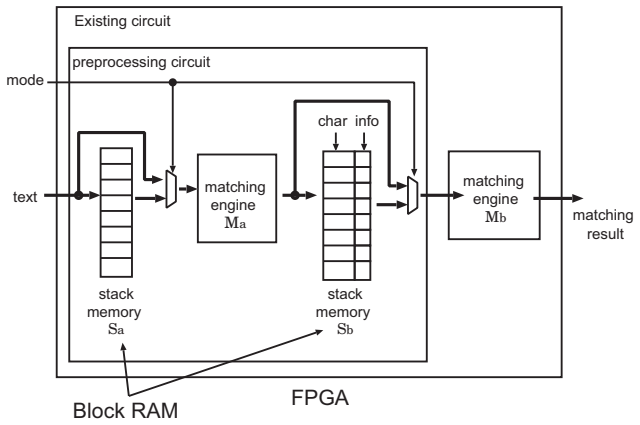


Fig. 3. The existing architecture

formation about whether the characters are the initial characters of substrings that match R_2 or not, and they are stacked in the stack memory S_b.

5) After the initial character of the text is stacked in the stack memory S_b, the characters are input to the matching engine M_b from the beginning.

6) The matching engine M_b performs matching for R_1R_3 using the information generated by the preprocessing circuit.

In the FPGA implementation in [11], the two stack memories are realized by block RAMs inside the FPGA.

3) Memories to reverse texts: In this architecture, the engine needs to make reversed packets contiguously. Thus, its stack memories have a special structure to buffer contiguously input texts and reverse the texts. Here, we explain the structure.

The size of each stack memory is $2n$ where n is the maximum length of a text. They also work as ring buffers to handle multiple variable-length texts contiguously. In the ring stack memory, texts are ring buffered, and text characters are stacked (LIFO). To simultaneously read and write texts, dual-port memories are required to realize the ring stack memories.

Let us explain the behavior of the ring stack memory briefly. (Please refer to [11] for more details.)

At first, n characters are input to the memory independently of texts' lengths. That is, multiple texts may be stored in the memory at a time. At this time, the input of the latest text possibly has not been completed yet. However, it is guaranteed that at least one text is completely stored in the buffer. Then, stored texts *set_*P_1 except a text which has not been completely stored yet begin to be output from the memory. The output order is from the latest input character to the earliest input character if *set_*P_1. While the texts *set_*P_1 are output from the memory, new texts *set_*P_2 are input to the memory. Immediately after all the texts *set_*P_1 are output from the memory, the newly stored texts *set_*P_2 begin to be output from the memory. In this way, the inputs of texts and the outputs of the reversed packets are performed simultaneously and contiguously. Fig.4 shows the behavior of the memory, where the maximum length of packets is 8.

In their FPGA implementation, the ring stack memories are realized by block RAMs in an FPGA.

III. Proposed Implementation

In this section, we present a practical FPGA implementation of regular expression matching with look-ahead assertion for NIDSs. Our implementation functions in cooperation with a PC and an off-chip memory to deal with a long text (*e.g.*, TCP packet).

In the existing FPGA implementation proposed in [11], matching engines and stack memories to reverse texts are implemented in an FPGA. The stack memories are realized by block RAMs in the FPGA. Since input to and output from a single block RAM are performed within a single clock cycle, it can work as a high-speed stack memory. However, a large memory composed of multiple block RAMs degrades the circuit speed. In our preliminary experiment, the existing FPGA implementation with 1.6Mbit stack memories, which can handle TCP packets, was half as fast as that with 36Kbit stack memories. Note that a 1.6Mbit stack memory consists of 45 block RAMs, and thus is much slower than a single block RAM. To handle longer packets, since more block RAMs are necessary, the circuit speed becomes even slower. Thus, it is not suitable to handling long packets.

Fig. 5 illustrates our proposed implementation of the hardware algorithm for look-ahead assertion. Our proposed implementation consists of a PC, an FPGA, and an off-chip memory. The FPGA and off-chip memory are implemented on a printed

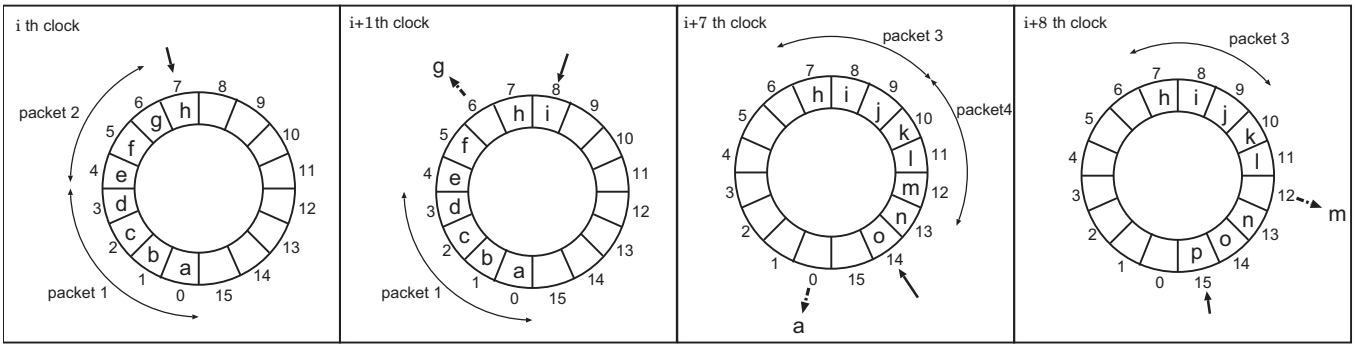


Fig. 4. The behavior of the buffer organization

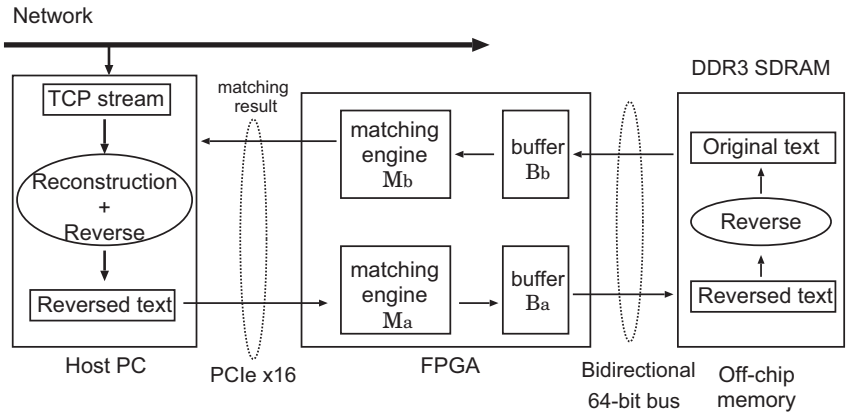


Fig. 5. The proposed implementation

circuit board, and the FPGA and the PC are connected by a PCI Express (PCIe) x16 bus. We assume the PC is equipped with an Intel Core i7 CPU and DDR3-2000 SDRAM. As off-chip memory, a DDR3-2000 SDRAM (1000MHz) is also used. In the proposed implementation, matching engines M_a and M_b are implemented on the FPGA, and stack memories S_a and S_b are implemented by using the main memory of the PC and the off-chip memory, respectively. In addition, it has dual-buffers B_a and B_b for burst data transmission between the off-chip memory and matching engine M_a and M_b, respectively. Note that since the maximum data size of burst transmission is limited, data to/from the off-chip memory is divided into data blocks for burst transmission. The flow of the matching is as follows. First, fragments of a packet on a network are stored in the main memory of the PC. At this step, the packet is reconstructed. This is to avoid overlooking fragmented viruses. After the packet reconstruction, the reversed packet P^R is constructed. P^R in the main memory is transferred to the FPGA through the PCIe bus, and the matching engine M_a performs the matching between P^R and ${R_2}^R$. After the first-stage matching, P^R and the matching result are stored in the off-chip memory by burst transmission in a pipeline fashion. Note that multiple characters and matching results are buffered in dual buffer B_a before the transmission for using burst transmission mode. Each of the buffers B_a and B_b uses double buffering to write and read data simultaneously. B_b is also used to reverse a data block from the off-chip memory, because a data block transmitted by burst transmission cannot be reversed during the transmission. Note that data blocks are transmitted in the reversed order, but each data block cannot be reversed during the burst transmission. In this way, P^R and the matching result are reversed and transferred to the matching engine M_b in the FPGA via dual-buffer B_b. That is, the original packet and the initial characters of substrings that match the pattern are input to the matching engine M_b. The matching engine M_b handles the pattern R_1R_3, considering the matching result for R_2 when determining if it starts matching for R_3. Note that all the components in our implementation work in a pipeline fashion.

Now, we discuss communication between each matching engine and the off-chip memory, considering the behavior of the memory. In the proposed implementation, the off-chip memory has to be written and read data by time-division because DDR3 SDRAM is a single-port memory. In some cases, the memory has to be read data twice per one data write. The reason is shown in the following. In the ring stack memory, after all the stored packets *set_*P_1 are output from the memory, the newly stored packets *set_*P_2 begin to be output from the memory. While *set_*P_1 are output from the memory,

read addresses are sequential. However, when *set_P*$_2$ begin to be output from the memory, read address is not sequential. Thus, it is necessary to restart burst transmission. For example, in Fig.4, at $i+7$th cycle, a (character in *set_P*$_1$) is read from address 0, at $i+8$th cycle, m (character in *set_P*$_2$) is read from address 12. Since burst transmission can read data which have sequential addresses, data consisting of *set_P*$_1$ and *set_P*$_2$ cannot be read by one burst transmission. Therefore, the off-chip memory has to be read data by burst transmission twice.

Since the data capacity of discrete memory chips can be much larger than that of block RAMs in an FPGA, using an off-chip memory as stack memory resolves the problem of the limited stack size. The remaining problem is that communication among the discrete components may cause overhead that degrades the throughput of the system.

In the next section, we analyze the performance of the proposed implementation, and derive the appropriate size of buffers between the FPGA and the off-chip memory. The main problem is that communication among the components may cause overhead that degrades the throughput of the system.

IV. Performance Analysis of the Proposed Implementation

In this section, we conduct a performance analysis on the proposed implementation. In the implementation, stack memories S_a and S_b is realized by the main memory of the PC and the off-chip memory. We analyze the communication overhead among the components. In the analysis, since the clock frequencies of the existing matching engines [2]–[9] are from 200 to 300 MHz, we assume that the matching engines in our implementation can work at 300 MHz.

First, we discuss the processing on the PC. The PC assembles the fragments of a packet and reconstructs the packet. We assume that the packet reconstruction in the PC is sufficiently fast compared with the network speed. Then, the PC makes the reversed packet of the packet and transfers it to the FPGA through the PCIe x16 bus. Since reversing a packet is a simple process, it can be done quicker than the packet reconstruction, for which the PC needs to wait for the packet fragments. Thus, it can be hidden in the packet reconstruction by using multiple processes.

Now, let us consider the data transfer between the PC and the FPGA. The bandwidth of the PCIe x16 bus between the PC and the FPGA is about 40 Gbit/s. Since the throughput of the matching engines are 2.4 Gbit/s (= 300 MHz $\times$ 8 bit), the PCIe bus is much faster than the engines, and thus the PCIe bus does not degrade the system performance. Consequently, the main memory of the PC can be used as the stack memory S_a without degradation.

Next, we discuss the communication overhead between the FPGA and the off-chip memory. We assume that the off-chip memory is a DDR3 SDRAM-2000, specified in Table I. A DDR3 SDRAM-2000 has burst transfer mode, and it can read/write consective data after waiting for the 55ns latency time. A DDR3 SDRAM-2000 reads/writes an eight-byte data per one memory clock cycle (1ns) after the latency time.

TABLE I
Specification of DDR3 SDRAM-2000

Bus width	64 bits
Bus clock frequency	1000MHz
Clock period	1ns
Latency	55ns
Burst length	8
No. of banks	8

Since the matching engine M_a works at 300 MHz, it processes one character per 3.3ns. Since each character is accompanied with a one-bit matching result, a nine-bit data is stored in the off-chip memory per 3.3ns. Each nine-bit data is stored using a two-byte memory space. The data is sent to the matching engine M_b in a similer way. Thus, the off-chip memory must read and write the two-byte data within 3.3ns (in average) to achieve the same performance as that of a block RAM.

The clock frequency of the off-chip memory is 3.3 times as fast as that of the matching engine M_a and M_b. Thus, the latency can be hidden by using burst transfer mode.

Let us discuss the sufficient data block size of a burst transmission to hide the latency. Let y be the number of 8-byte (= 64-bit) data sets because a DDR3 SDRAM has a 64-bit data bus. Then, the necessary time to transmit $y \times 8$ bytes to/from the memory is

$$1[\text{ns}] \times y + 55[\text{ns}]. \tag{1}$$

To emulate a dual-port memory by time-division, three data blocks (one for memory write, the others for memory read) need to be transmitted in a time-slice. (The reason why two data blocks are read is described in the previous section.) Thus, the necessary time to transmit the data is

$$3 \times (1[\text{ns}] \times y + 55[\text{ns}]). \tag{2}$$

Since a matching engine outputs $y \times 8$ bytes (= $(y \times 8)/2$ text characters and $(y \times 8)/2$-byte matching result data) within

$$3.3[\text{ns}] \times (y \times 8)/2, \tag{3}$$

the condition for hiding the communication overhead is represented by the inequality (4).

$$3.3[\text{ns}] \times (y \times 8)/2 \geq 3 \times (1[\text{ns}] \times y + 55[\text{ns}]). \tag{4}$$

Note that data transmission and string matching are performed in a pipeline fashion. Solving this inequality about y, we get

$$y \geq 16.2. \tag{5}$$

Thus, by transmitting 17 data sets by burst transmission using 136 (= 17 $\times$ 8) byte buffers (*i.e.*, the data block size is set to 136 bytes), the off-chip memory plays the same role as the original buffer realized by block RAMs, hiding its read/write latency. Fig. 6 shows the setting of the buffer and data block size, and transmission delays. Consequently, our proposed implementation is able to handle long packets as fast as the existing implementation for short packets.

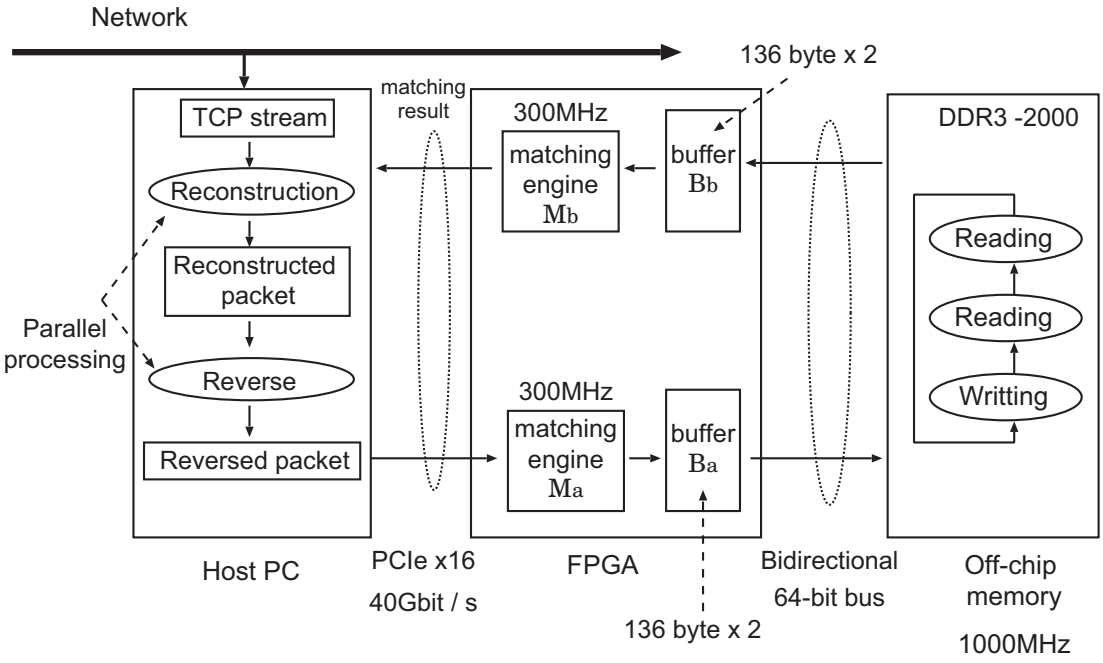


Fig. 6. Details of the proposed implementation

V. Conclusions

In this paper, we proposed a practical implementation method of a hardware algorithm for regular expression matching with look-ahead assertion, using a PC and an FPGA. In addition, we analyzed its performance considering the memory latency and communication delay to show its efficiency. Our future work includes evaluation by circuit implementation.

References

[1] J. E. Hopcroft, R. Motwani, J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation (3rd Edition)," Pearson Education, 2006.
[2] J. Bispo, I. Sourdis, J.M.P. Cardoso, S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection," Proc. 2006 IEEE ICFPT, pp.119-126, 2006.
[3] T. Ganegedara, Y.E. Yang, V.K. Prasanna, "Automation framework for large-scale regular expression matching on FPGA," Proc. 2010 IEEE ICFPL, pp.50-55, 2010.
[4] T. Trung Hieu, T. Ngoc Thinh, T. Huy Vu, S. Tomiyama, "Optimization of Regular Expression Processing Circuits for NIDS on FPGA," Proc. 2011 IEEE ICNC, pp.105-112, 2011.
[5] Y.K. Chang, C.R. Chang, C.C. Su, "The cost effective pre-processing based NFA pattern matching architecture for NIDS," Proc. 2010 IEEE ICAINA, pp.385-391, 2010.
[6] J. Divyasree, H. Rajashekar, Kuruvilla Varghese, "Dynamically reconfigurable regular expression matching architecture," Proc. 2008 ASAP, pp.120-125, 2008.
[7] Y. Kaneta, S. Yoshizawa, S. Minato, H. Arimura, Y. Miyanaga, "Dynamic reconfigurable bit-parallel architecture for large-scale regular expression matching," Proc. 2010 IEEE ICFPT, pp.21-28, 2010.
[8] C.R. Clark, D.E. Schimmel, "Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns," Proc. 2003 IEEE ICFPL, pp.956-959, 2003.
[9] Y. Wakaba, M. Inagi, S. Wakabayashi, S. Nagayama, "An efficient hardware matching engine for regular expression with nested Kleene operators," Proc. 2011 IEEE ICFPL, pp.157-161, 2011.
[10] J. Goyvaerts, "Regular-expressions.info", http://www.Regular-Expressions.info/lookaround.html
[11] Y. Wakaba, S. Nagayama, M. Inagi, S. Wakabayashi, "A Matching Method for Look-ahead Assertion on Pattern Independent Regular Expression Matching Engine," Proc. the 17th Workshop on Synthesis And System Integration of Mixed Information Technology (SASIMI2012), pp.361-366, 2012.

SESSION

RECONFIGURABLE ARCHITECTURE

Chair(s)

Dr. G. Botella
Complutense University of Madrid
Spain

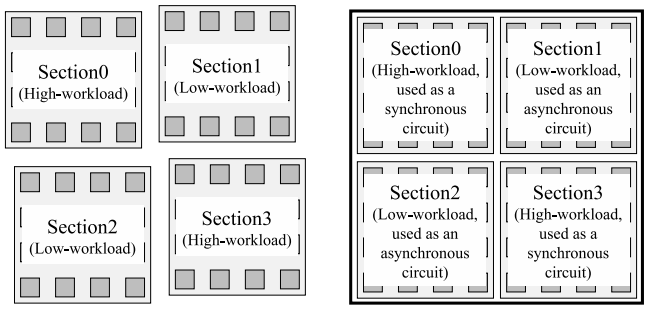
Area-Efficient Design of Asynchronous Circuits Based on Balsa Framework for Synchronous FPGAs

ERSA'12 Distinguished Paper

Yoshiya Komatsu, Masanori Hariyama, and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan

Abstract— *This paper presents an efficient asynchronous design methodology for synchronous FPGAs. The mixed synchronous/asynchronous design is the best way to minimize the power consumption of a circuit implemented on a synchronous FPGA. For asynchronous circuit synthesis, Balsa was proposed. However, the problem is that circuits synthesized from Balsa description need a lot of logic resources. To solve this problem, we propose two optimization methods for gate-level netlist. First, we introduce an area-efficient C-element suitable for FPGAs. Then, we propose optimization methods for an adder with a carry input and constant adder. The evaluation results show that the proposed method reduces the logic resource consumption by 26% to 47%.*

Keywords: FPGA, Reconfigurable LSI, Self-timed circuit, Asynchronous circuit, Balsa



(a) Workloads of sections vary from place to place.

(b) Mixed synchronous/asynchronous design.

Fig. 1: Advantage of mixed synchronous/asynchronous design.

1. Introduction

Field-Programmable Gate Arrays (FPGAs) are widely used to implement special-purpose processors. FPGAs are cost-effective for small-lot production because functions and interconnections of logic resources can be directly programmed by end users. Despite their design cost advantage, conventional FPGAs impose a large power consumption overhead compared to custom silicon alternatives [1]. This overhead increases packaging costs and limits integrations of FPGAs into portable devices.

An asynchronous circuit is power-efficient for a low-workload section including Logic Blocks (LBs) since it does not require the clock tree which consumes the power even if an LB is inactive [2], [3]. Instead of using the clock, the handshake protocol is used for synchronization. To implement the handshake protocol, the circuit becomes more complex than that of the synchronous circuit. In a low-workload condition, the power overhead for the handshake protocol is much smaller than the power reduction of the clock tree, and the power consumption is greatly reduced. However, in a high-workload condition, the power overhead for the handshake protocol is larger than the power reduction of the clock tree. On the other hand, a synchronous circuit is power-efficient for a high-workload section because of its simple hardware. However, the synchronous circuit is not power-efficient for a low-workload section, since the power of the clock tree is always consumed even if an LB is inactive. In a synchronous FPGA, the clock tree occupies a large proportion of the dynamic power because it has significantly more registers than a custom VLSI. Moreover, a gated clock technique for reducing the power of the clock tree is difficult to implement in an FPGA [4]–[6].

An SoC (System-on-a-Chip) consists of several sections, and each of the sections is used for a dedicated application. As shown in Fig. 1(a), some sections are for high-workload applications such as 3D graphics, and some are for low-workload applications such as audio playback [7]. Since the synchronous circuit is power-efficient for high-workload sections and the asynchronous circuit is power-efficient for low-workload sections, the mixed synchronous/asynchronous design is the best way to minimize the power consumption for each section as shown in Fig. 1(b).

In asynchronous circuits, CAD tools that is different from ones for synchronous circuits is necessary to implement applications. As the design methods for asynchronous circuits, some method uses the Signal Transition Graph [8] and another method employs handshake components [9] [10]. Besides, Balsa [10] is proposed as the framework for synthesizing asynchronous hardware systems that uses handshake components. Balsa is also the name of the hardware

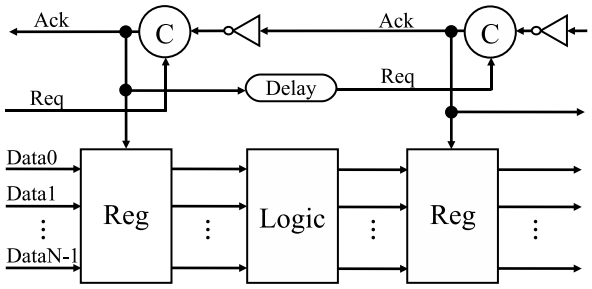


Fig. 2: Simple bundled-data pipeline.

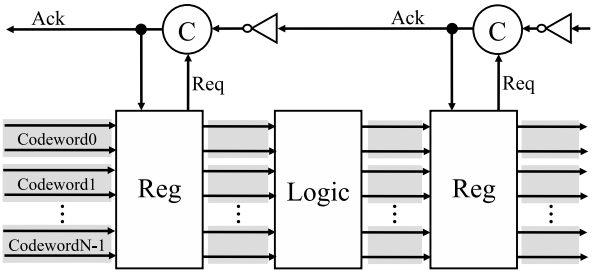


Fig. 3: Simple dual-rail pipeline.

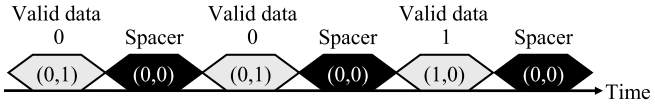


Fig. 4: Example of the FPDR encoding.

description language and it allows circuit designers not to pay attention to low-level details such as control of handshake. Thus, it is suitable for designing complex large-scale circuits such as a DMA controller [10] and a microprocessor [11]. Balsa framework can generate gate-level verilog netlist for FPGAs. Therefore, Balsa is suitable as an asynchronous circuit design tool for FPGAs. However, the problem is that circuits synthesized from Balsa description need a lot of logic resources. To solve this problem, we propose two optimization methods for gate-level netlist. First, we introduce an area-efficient C-element suitable for FPGAs. Then, we propose optimization methods for adders that have a carry input and constant adder.

2. Optimization methods for circuits synthesized from Balsa description

2.1 Asynchronous encodings

Asynchronous encoding schemes are mainly classified into

- Single-rail encoding (ex. the bundled-data encoding),
- Dual-rail encoding (ex. the Four-Phase Dual-Rail encoding).

Table 1: Code table of the FPDR encoding.

	Codeword (truth, false)
Valid data 0	(0, 1)
Valid data 1	(1, 0)
Spacer	(0, 0)

* Codeword (1, 1) is not used

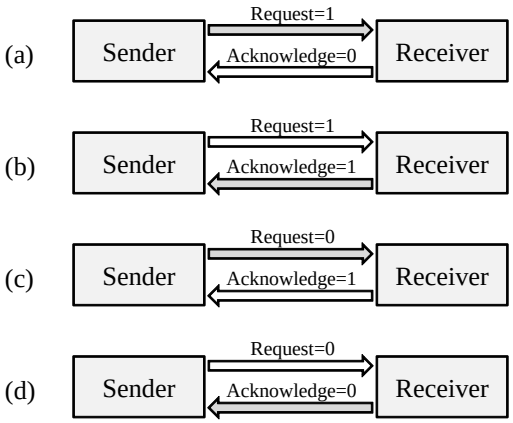


Fig. 5: A four-phase handshake sequence.

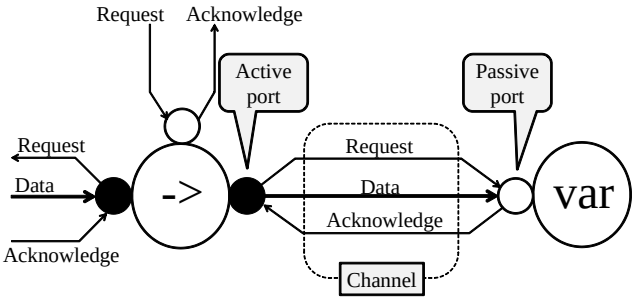


Fig. 6: Handshake components and channels.

The bundled-data encoding is the most common one in single-rail encodings. Fig. 2 shows a simple bundled-data pipeline. In this example, data signals and request signals are oriented in the same direction. In the bundled-data encoding, request and value are split into separate wires. The value is encoded as in a synchronous circuit using N wires to denote an N-bit value, and request is encoded using a dedicated request wire denoted by *Req*. The bundled-data encoding requires the explicit insertion of matching delays in *Req* to ensure that the request is never received before the bundled value is valid. The bundled-data encoding is the most frequently-used way in ASICs since its hardware overhead is relatively small. This is because the *Req* wire is shared among all the N wires. Hence, to transfer an N-bit value, only $N + 2$ wires are required. The major disadvantage of

```
import [balsa.types.basic]

procedure count16 (output count : 4 bits) is
        variable count_reg, tmp : 4 bits
begin
        loop
                tmp := (count_reg + 1 as 4 bits)
                ||
                count <- count_reg
                ;
                count_reg := tmp
        end
end
```

Fig. 7: An example of a Balsa description (4 bit counter).

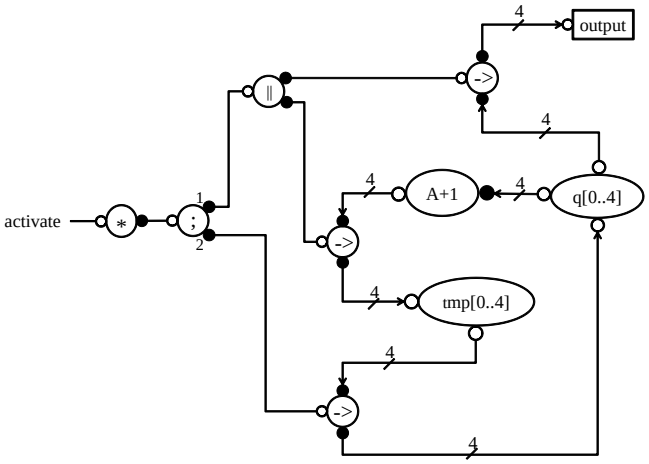


Fig. 8: A simple handshake circuit (4 bit counter).

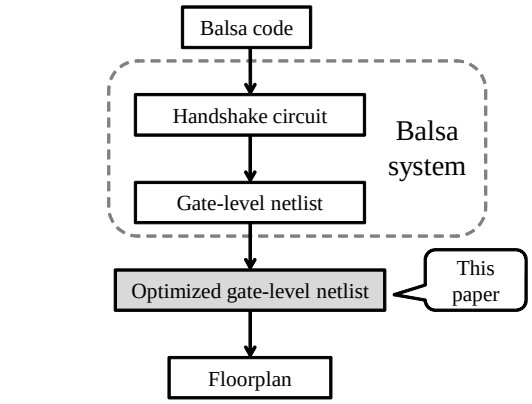


Fig. 9: Design flow for asynchronous circuits using Balsa.

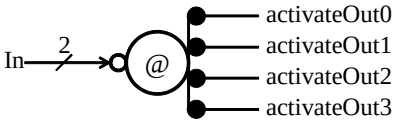


Fig. 10: Case component.

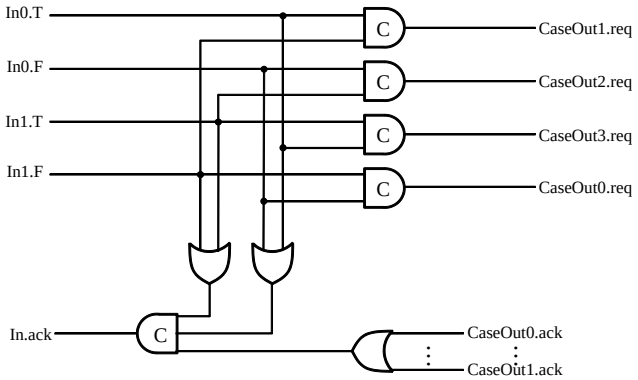
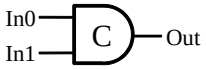


Fig. 11: Corresponding circuit of a Case component.

the bundled-data encoding is the difficulty of determining delay times of Request signals. A Request signal must reach a receiver after an arrival of a data signal. Therefore, delay times of Request signals should be determined by the result of placement and routing of the circuit. Furthermore, a circuit designer should consider the fluctuations of supply voltage, temperature and threshold voltage of transistors. As a result, it is difficult to design a circuit based on the bundled-data encoding. The dual-rail encoding encodes a data bit onto two wires. Fig. 3 shows a simple dual-rail pipeline. In the dual-rail encoding, value is made implicit in the request and no delay insertion is therefore required [12]. Hence, the dual-rail encoding is the ideal one for FPGAs. In the dual-rail encoding, to transfer an N-bit value, $2N + 1$ wires are required. The Four-Phase Dual-Rail (FPDR) encoding is the most common one in dual-rail encodings. Table 1 shows the code table of the FPDR encoding. The code word consists of *t* (truth bit) and *f* (false bit). The data value 0 is encoded as $(0, 1)$ and 1 is encoded as $(1, 0)$. Moreover, the spacer is encoded as $(0, 0)$. Fig. 4 shows the example where data values 0, 0 and 1 are transferred. The main feature is that the sender sends a spacer after a valid data. The receiver knows the arrival of a data value by detecting the change of either bit: 0 to 1. The insertion of spacers makes the encoding law simple. This results in simple hardware for the function unit. Therefore, the FPDR encoding is chosen in this paper.

2.2 Handshake-component-based asynchronous circuit design

In asynchronous circuits, the handshake protocol is used for synchronization instead of using the clock. Figure 5 shows a four-phase handshake sequence. First, the sender



In0	In1	Out
0	0	0
0	1	Out
1	0	Out
1	1	1

Fig. 12: Truth table for a C-element.

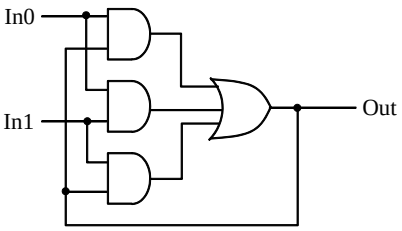
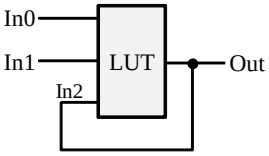


Fig. 13: Conventional implementation of a C-element.



In0	In1	In2	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig. 14: Implementation of a C-element using three-input Look-up table.

sets the request wire to 1 as shown in Fig. 5(a). Second, the receiver sets the acknowledge wire to 1 as shown in Fig. 5(b). Third, the sender sets the request wire to 0 as shown in Fig. 5(c). Finally, the receiver sets the acknowledge wire to 0 as shown in Fig. 5(d) and wire values return to initial state.

In Balsa, asynchronous functional element such as a binary operator is denoted by a handshake component. Figure 6 shows handshake components. Each handshake component has ports and is connected to another handshake component through a channel. Communication between handshake components is done by sending request signal from the active port and acknowledge signal from the passive port. Depending on the kind of handshake components, data signals are sent along with request signals or acknowledge signals. As mentioned in Sec. 2.1, FPDR encoding is employed in synthesized circuits. Therefore, either a request signal or an acknowledge signal is made implicit in a data signal. The

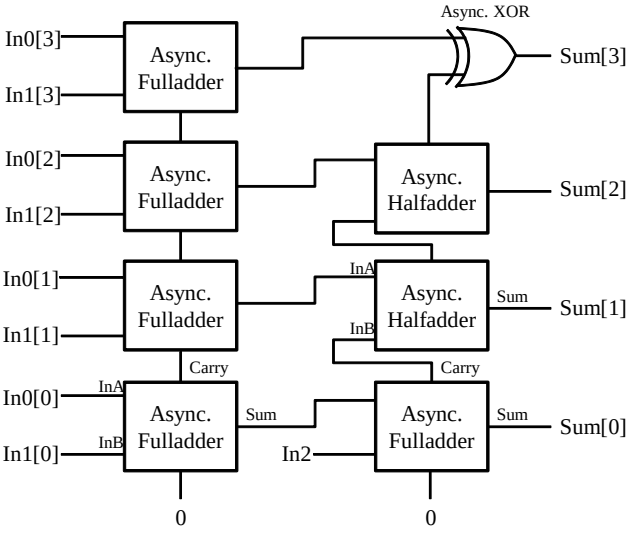


Fig. 15: 4-bit adder generated from Balsa description.

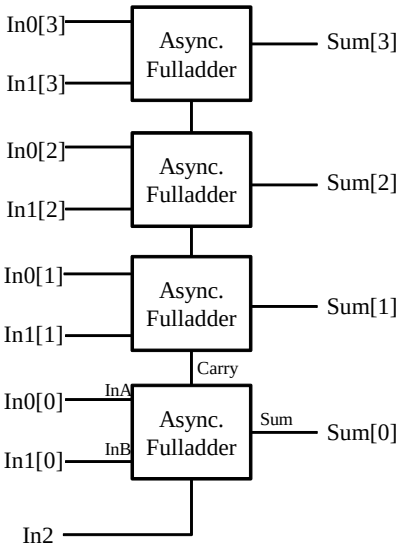


Fig. 16: Modified 4-bit adder

number of ports and the width of data signal can be varied. Each function of handshake component is simple and clear. Furthermore, handshaking that consists of request signal and acknowledge signal is symbolized as a channel. Therefore, handshake circuits are easily understandable and manageable. Handshake components constitute a handshake circuit. Figure 8 shows an example of a handshake circuit. Circuit synthesis is done by replacing each handshake component with corresponding asynchronous circuit. As a result, circuits become large. To implement an asynchronous circuit on an FPGA area-efficiently, optimizing gate-level netlist generated from a handshake circuit is required. The proposed design flow for asynchronous circuits is shown in Fig. 9.

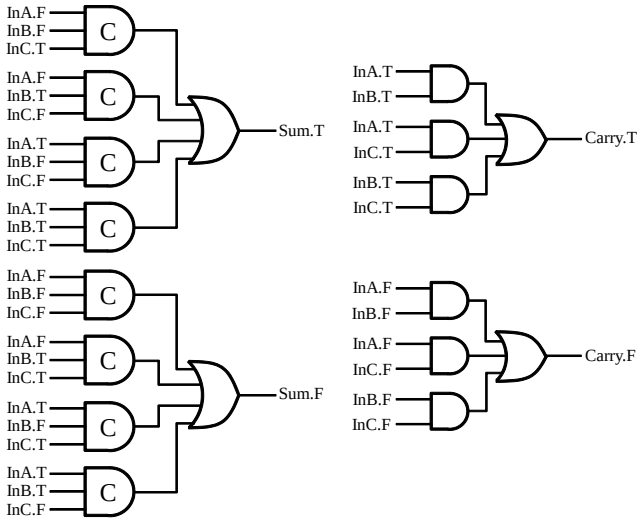


Fig. 17: Structure of an FPDR full adder.

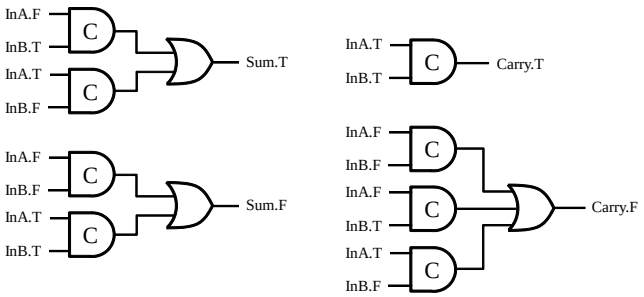


Fig. 18: Structure of an FPDR half adder.

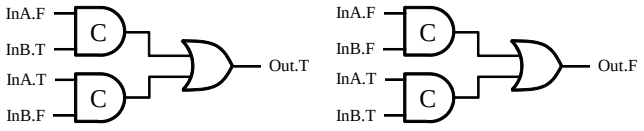


Fig. 19: Structure of an FPDR XOR gate.

2.3 Optimization methods for asynchronous circuits

2.3.1 Optimization of the Muller C-element

An asynchronous circuit synthesis using Balsa is done by following steps. First, a Balsa description is converted to a handshake circuit. Then, a gate-level netlist is obtained by replacing each handshake component with corresponding asynchronous circuit. Figures 10 and 11 show the Case component and the corresponding circuit. As these show, the Muller C-element is frequently used in asynchronous circuits. Figure 12 shows the truth table for a C-element. To implement this on a FPGA, the circuit shown in Fig. 13 is ordinarily used. However, this circuit is not area-efficient because this implementation uses four primitive gates. Therefore, we propose an alternative implementation of a C-element as shown in Fig. 14. This implementation is area-efficient because only one three-input Look-up table is required.

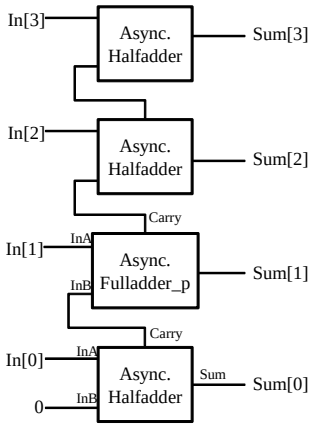


Fig. 20: Constant adder which adds two.

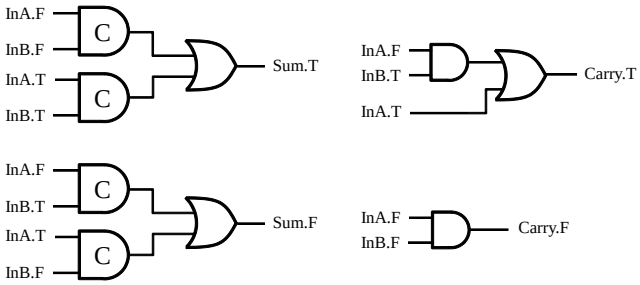


Fig. 21: Structure of a FullAdder_p.

2.3.2 Optimization of adders

Some adders execute additions with 1-bit carry signals for multi-byte additions or subtractions. However, an adder synthesized from Balsa description consists of an N-bit adder and an adder that adds N-bit data and 1-bit data as shown in Fig. 15. The latter adder can be trimmed by employing Carry input of the former adder as the one-bit input, as shown in Fig. 16. The structures of asynchronous FPDR full adder, half adder and XOR gate are shown in Fig.17, 18 and 19. As mentioned in Sec. 2.1, the FPDR encoding requires two wires to transfer a bit. This leads to larger number of inputs and circuits compared to conventional synchronous circuits. Therefore, trimming adders is effective to reduce circuit size. Also, constant adders are frequently used. Figure 20 shows an asynchronous circuit which adds two to an input value. Figure 21 shows the structure of a FullAdder_p.

FullAdder_p is a full adder whose one input is fixed to one. Despite the least significant bit doesn't change, it goes through a half adder. Therefore, used logic resources can be

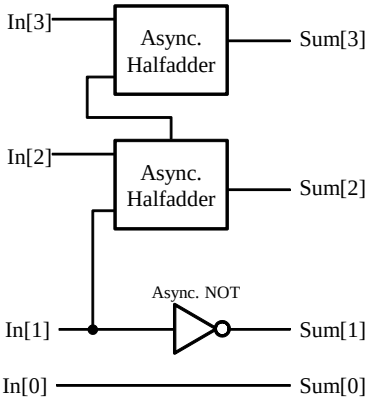


Fig. 22: Modified constant adder.

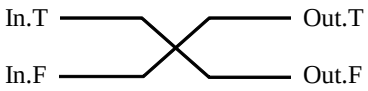


Fig. 23: Structure of an FPDR NOT gate.

reduced by connecting input wires and output wires directly. The modified constant adder is shown in Fig. 22. As shown in Fig. 23, an FPDR NOT gate requires only two wires. As a result, the number of required logic resources is reduced.

3. Evaluation

The proposed methodology is evaluated by implementing a 4-bit counter and a 8-bit multiplier on a Xilinx Spartan3E XA3S500E FPGA. Figures 24 and 25 show the implementation results of the counter and the multiplier. The proposed C-element implementation reduces the number of used slices by 21% and 43% respectively. In addition, the number of used slices is reduced by 26% and 47% thanks to efficient implementations of adders.

4. Conclusions

This paper proposed an efficient asynchronous design methodology for synchronous FPGAs. The logic resources to implement asynchronous circuit can be reduced by optimizing the structure of the C-element and adders. As a future work, developing the optimal methodology for synthesizing asynchronous circuits and synchronous circuits from a hardware description is important to implement synchronous/asynchronous hybrid circuits.

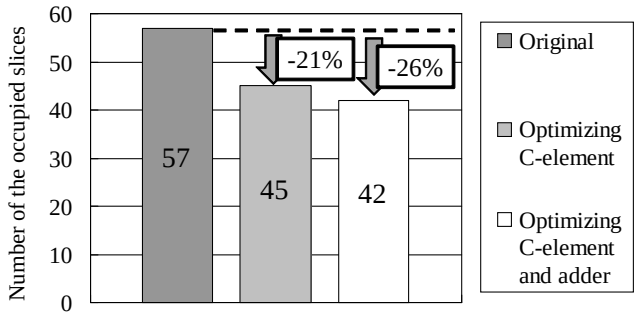


Fig. 24: Evaluation result of 4-bit counter.

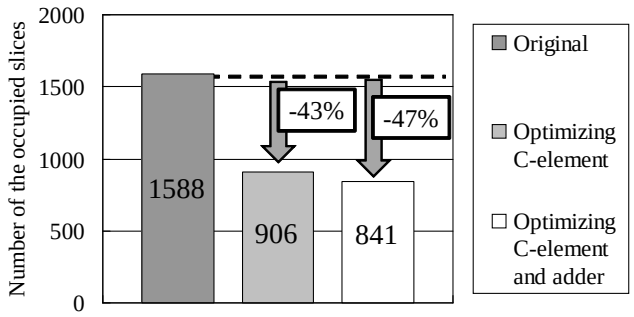


Fig. 25: Evaluation result of 8-bit multiplier

References

[1] V. George H. Zhang. and J. Rabaey, "The design of a low energy FPGA," in *Proceedings of 1999 International Symposium on Low Power Electronics and Design*, California, USA, Aug 1999, pp. 188–193.

[2] M. Hariyama, S. Ishihara, and M. Kameyama, "Evaluation of a Field-Programmable VLSI Based on an Asynchronous Bit- Serial Architecture," *IEICE Trans. Electron*, vol. E91-C, no. 9, pp. 1419–1426, 2008.

[3] S. Ishihara, Y. Komatsu, M. Hariyama and M. Kameyama, "An Asynchronous Field-Programmable VLSI Using LEDR/4-Phase-Dual-Rail Protocol Converters," in *Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas(USA), Jul 2009, pp. 145–150.

[4] Synplicity Inc., Gated Clock Conversion with Synplicity 's Synthesis Products, 2003.

[5] Xilinx Inc., Synthesis and Simulation Design Guide, 2008.

[6] Y. Zhang, J. Roivainen, and A. Mammela, " Clock-Gating in FPGAs: A Novel and Comparative Evaluation, " Proc. the EUROMICRO Conference on Digital System Design (DSD '06), pp.584—590, 2006.

[7] M. Shirasaki, Y. Miyazaki, M. Hoshaku, H. Yamamoto, S. Ogawa, T. Arimura, H. Hirai, Y. Iizuka, T. Sekibe, Y. Nishida, T. Ishioka, and J. Michiyama, " A 45nm singlechip application-and-baseband processor using an intermittent operation technique, " IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers, pp.156—158, 2009.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Logic Synthesis of Asynchronous Controllers and Interfaces, Springer, 2002.

[9] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, " The VLSI-programming language Tangram and its translation into handshake circuits, " in *Proc. EDAC*, 1991, pp. 384-389.

[10] A. Bardsley, "Implementing Balsa Handshake Circuits," Ph.D. thesis, Dept. of Computer Science, University of Manchester, 2000.

[11] Q. Zhang, G. Theodoropoulos, "Modelling SAMIPS: A Synthesisable Asynchronous MIPS Processor," *Proc. of the 37th Annual Simulation Symposium*, pp. 205-212, 2004

[12] Jens Sparsø and Steve Furber, Principles of Asynchronous Circuit Design: A Systems Perspective, Kluwer Academic Publishers, 2001

FPGA-based Implementation of Compact Compressor and Decompressor of Floating-Point Data-Stream for Bandwidth Reduction

Tomohiro Ueno[1], **Yoshiaki Kono**[1], **Kentaro Sano**[1] **and Satoru Yamamoto**[1]
[1]Graduate School of Information Sciences, Tohoku University, Sendai, Miyagi, JAPAN

Abstract— *This paper presents FPGA-based implementation and performance analysis of the hardware for lossless compression of a floating-point data stream. The implementation includes the variable-to-fixed length converter (VFCONV) and the fixed-to-variable length converter (FVCONV) that have not been designed and evaluated so far. The prototype system on ALTERA Stratix IV FPGA demonstrates that the compressor and the decompressor are very small so that they each consume only 0.5 % adaptive LUTs of the total resource on the FPGA. The compressor and the decompressor can operate at about 250 MHz and 200 MHz, respectively, while further optimization is still possible. Such small but high-speed hardware modules can reduce the bandwidth by a factor of an average compression ratio. Evaluation with the computational results of 2D fluid simulation shows that the compressor has an average compression ratio of 3.7. This means that the compressor operating at 200 MHz can supply the bandwidth of 800 MB/s for single-precision floating-point numbers with the compressed bandwidth of 216 MB/s.*

Keywords: data compression, floating-point, bandwidth reduction, FPGA

1. Introduction

Stream computation is one of the very useful and promising approaches for custom computing machines (CCMs) to accelerate scientific computations, such as computational fluid dynamics (CFD). So far we have proposed and prototyped an FPGA-based custom accelerator for the lattice Boltzmann method (LBM) [1], [2], which is one of the CFD schemes. Due to the high-throughput design based on stream computation which exploits both spatial and temporal parallelism of LBM computation, the accelerator demonstrated that the logic elements and DSP blocks on state-of-the-art FPGAs could achieve the higher arithmetic performance than that of general-purpose microprocessors [2]. However, it also showed that the achievable performance is limited by the I/O bandwidth rather than arithmetic performance of circuits even if regularity of stream computation exploits the available bandwidth efficiently. Since it is not easy to drastically increase the bandwidth of off-chip I/O pins, present and future chips are perpetually suffering from the insufficiency of I/O bandwidth for on-chip computing resources increased by semiconductor scaling.

In our research project, we have proposed enhancement of I/O or memory bandwidth by employing lossless compression of floating-point data [3], [4]. The physical bandwidth can be efficiently used with *compressed bandwidth* by excluding redundancy of bits in data streams. The lossless compression guarantees the complete reconstruction of original data, thus giving no additional error to computation. Fig.1 shows the accelerator based on stream computation with data compression. If the compressor reduces the data size to $1/r$ of the original size on average, we can substantially use the physical stream bandwidth as r times wider bandwidth. Of course, the decompressor and compressor slightly increase the delay, however, stream computation is inherently tolerant to latency. Therefore only high throughput is required for compression and decompression.

So far, we have presented algorithms for lossless compression of a floating-point data stream, and partially designed a high-throughput hardware compressor [3], [4]. We demonstrated that the algorithm achieves the compression ratio of around 3.5 for single-precision floating-point data, such as computational results of CFD. This means that the stream bandwidth can be enhanced to 3.5 times wider for CFD computation.

Since the compressor and the decompressor are auxiliary hardware for computation itself, they should be implemented with as less resources as possible, while higher operating frequency is required. In our prior work, however, only the compressor is designed and partially implemented for evaluation of an operating frequency and resource utilization on FPGA, and therefore both compression and decompression had not been tested and verified on actual implementation on FPGA. Furthermore, we did not show the design of the important modules to handle variable-length compressed data, which are referred to as *a variable-to-fixed length converter (VFCONV)* and *a fixed-to-variable length converter (FVCONV)* in the compressor and decompressor, respectively. VFCONV in the compressor packs variable-length data into words and output them discontinuously as a compressed-data stream, while FVCONV in the decompressor extracts the variable-length data from the received words of a compressed data stream. Since VFCONV and

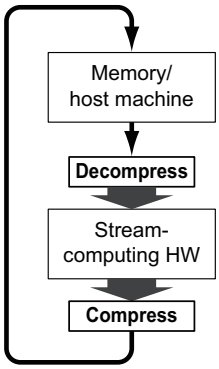


Fig. 1: Stream computation with data decompression and compression.

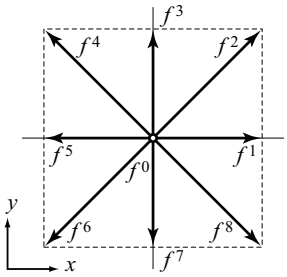


Fig. 2: Particle distribution function f_k of 2D9V LBM.

FVCONV contain feedback loops with buffer registers, we need make critical paths as short as possible for higher operating frequency.

In this paper, we present an entire design of the compressor and decompressor. Their full-implementation on FPGA demonstrates that compact hardware can compress and decompress a floating-point data stream at an average ratio of 3.7. Based on timing analysis, we show that our designed and implemented hardware can operate at 198 MHz at least, bringing high-throughput compression and decomrpession required for bandwidth enhancement. We will also estimate resource consumption for compressing and decompressing multiple data-streams in parallel.

This paper is organized as follows. Section 2 briefly describes the prediction-based algorithm and the hardware for lossless compression of a floating-point data stream. Section 3 shows the detailed design and behavior of the variable-to-fixed length converter and the fixed-to-variable length converter. Section 4 presents implementation, verification and evaluation of a prototype system with the compressor and decompressor on FPGA. Finally, Section 5 gives conclusions and future work.

2. Lossless compressor and decompressor of floating-point data stream

2.1 Requirements for data compression in stream computation

In our research, we made a choice of a compression algorithm and designed its hardware based on the following requirements for bandwidth enhancement in scientific stream computation [3], [4].

1) Lossless compression.
2) Direct compression of FP data.
3) Single-pass compression.
4) Acceptable compression-ratio for improvement of memory bandwidth.
5) High-throughput by small hardware.

To avoid unnecessary errors in scientific computation, compression should be lossless so that the original data can be completely reconstructed with the compressed data. Recently prediction-based lossless algorithms [5], [6], [7], [8], [9] have been proposed to directly compress floating-point data, which achieve better compression ratios than general-purpose compressors like BZIP2[10]. These algorithms predict the next input with the previous ones in a sequence, and then encode the difference between the prediction and the actual value.

The single-pass of requirement 3 means that the entire data set can not be traversed in advance of stream compression. The prediction-based compression algorithms are of single-pass algorithms. For requirements 4 and 5, compression using arithmetic predictors [5], [6], [7] is suitable. Based on these discussions, we adopted lossless compression algorithms with 1D arithmetic prediction to directly compress a floating-point data stream [3], [4].

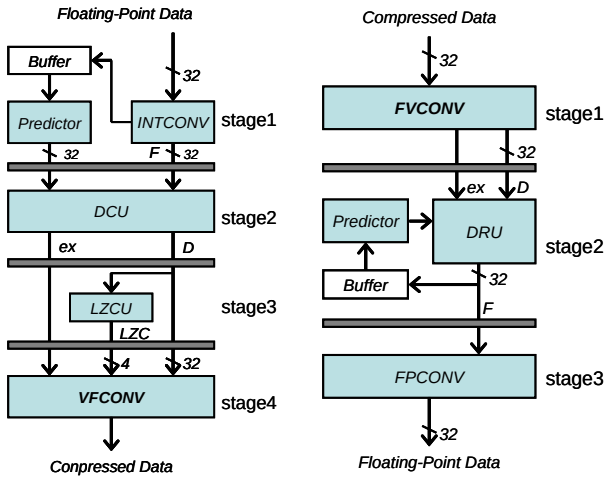
2.2 Prediction-based compression algorithm

We assume that an IEEE754 floating-point data stream, $S = \{..., f_{i-2}, f_{i-1}, f_i, ...\}$, is input to the compressor, where f_i denotes the current input. Such a stream is generated by traversing a given 2D or 3D computational grid in a certain order. For f_i, the compressor computes its prediction p_i with some of the previous inputs stored in a buffer memory. When prediction is made with good accuracy, p_i has a closer bit pattern to that of f_i, where a lot of bits from MSB become zeros in their difference. By encoding these zeros with their length, we represent the difference between the prediction and the original datum in fewer bits. The following subsections describe the detail of the predictor and the difference encoder.

2.2.1 Predictor

The computational results of scientific simulation such as CFD typically have some spatial and temporal continuity because they are the solutions of the partial differential equations governing the physical phenomena. This allows us to assume that they can be well locally-approximated by polynomial functions, similarly to the proposal of [11]. Under this assumption, we use *polynomial predictors* to obtain good prediction for data given by such computations.

Here let's consider that a numerical sequence $S = \{..., f_{i-1}, f_i\}$ is a set of regularly sampled values of 1D function $f(x)$, so that $f_i = f(x_i) = f(i\Delta x)$ for integer i. Assuming that $f(x)$ is locally approximated by the $(n-1)$-th order polynomial function $f(x) = a_0 + a_1 x + a_2 + x^2 + ... + a_{n-1}x^{n-1}$, we can predict the next value $f_i = f(i\Delta x)$ by determining all a_k and computing $f(x)$ for $x = i\Delta x$. We can determine the n parameters, a_k, by evaluating the Lagrange polynomial [11] with the n previous values, $\{f_{i-n}, ..., f_{i-1}\}$.



a. compressor b. decompressor

Fig. 3: Overview of compressor and decompressor.

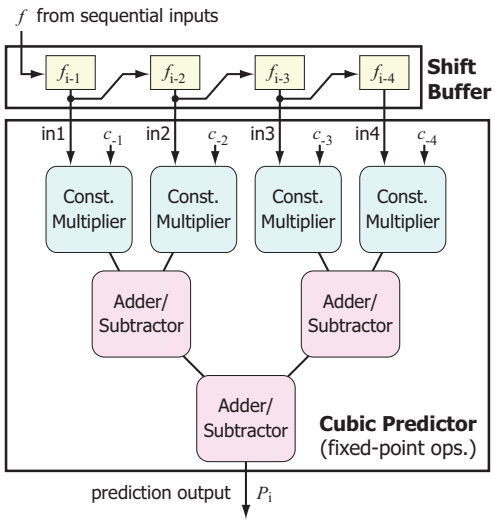


Fig. 4: Structure of the cubic predictor and the shift buffer.

Considering $\Delta x = 1$ for simplicity, we obtain polynomial predictions of p_i as follows:

$$p_i = 4f_{i-1} - 6f_{i-2} + 4f_{i-3} - f_{i-4}, \quad (1)$$

where $n = 4$. We refer to this predictor as a *1D cubic polynomial-predictor* or simply a *Cubic*.

2.2.2 Difference encoder

We encode the difference between p_i and f_i with its leading-zero count (LZC), which is the number of successive zeros from MSB, and the remaining bits. We refer to the remaining bits as *residual*, denoted by r. The integer-subtraction evaluates the difference after the FP numbers are converted to unsigned integers so that the closer numbers always give smaller difference as mapped integers. The integer conversion is performed by flipping the sign bit for positive FP numbers, or all bits for negative numbers [5]. As a result, the positive and negative FP numbers are mapped continuously to the higher and lower space of unsigned integers, respectively. After p_i and f_i are converted to their integers, P and F, we compute $D = P - F$ for $P > F$, or $D = F - P$ for $P \leq F$. We also output whether $P > F$ or not, with an exchange signal, ex, where

$$\text{ex} = \begin{cases} 1 & \text{for} \quad P > F \\ 0 & \text{for} \quad P \leq F. \end{cases} \quad (2)$$

The obtained LZC can be recorded as it is, however we use *4-bit coding* [8], [9], [3] for easier handling of a variable length of the residual. Here we assume that an FP number has 32 bits for single precision. In this case, LZC can be 0 to 31, which is represented in 5 bits. The residual has $(32 - \text{LZC})$ bits. On the other hand, the 4-bit coding represents LZC with a multiple of 4. For example, LZC=15 naively gives 17 residual-bits. In the 4-bit coding, the LZC is truncated to 12, and the residual becomes 20 bits being padded with 0s. The 4-bit coding is expected to have the advantage of 4-bit alignment that allows us to more simply output residuals with variable bit-length.

For IEEE754 single-precision floating-point data, the procedure of the integer subtraction and the 4-bit coding is summarized as follows. We convert p_i and f_i to their unsigned integer, P and F, respectively. We compute D by subtraction, and obtain $\text{ex} = 1$ or 0 for $P > F$ or not, respectively. Then we obtain the LZC of D. Instead of using 5 bits to represent the LZC, we encode $\lfloor \text{LZC}/4 \rfloor$ with 3 bits so that the LZC is a multiple of 4. We pad necessary 0s to r. The length of encoded bits is $(1 + 3 + 32 - 4\lfloor \text{LZC}/4 \rfloor)$ for $\{\text{ex}, \text{LZC}, r\}$.

2.3 Overview of hardware compressor and decompressor

Based on the compression algorithm mentioned above, we designed hardware compressor and decompressor for a single stream of single-precision floating-point numbers [4]. Figs.3a and b show the overviews of the compressor and the decompressor. Datum from a floating-point data stream, $\{f_i\}$, is input to the compressor one by one every cycle, and then it outputs a compressed bit-stream $\{(\text{ex}, \text{LZC}, r)_i\}$. The compressor is composed of *integer converter (INTCONV)*, *a predictor with a buffer*, *a difference-computing unit (DCU)*, *an LZC unit (LZCU)* and *a variable-to-fixed length converter (VFCONV)*. The input floating-point datum f is firstly converted to an unsigned integer F by flipping the sign bit for a positive number or all bits for a negative number. The converted integer is stored in the buffer. The buffer stores a necessary number of previous inputs.

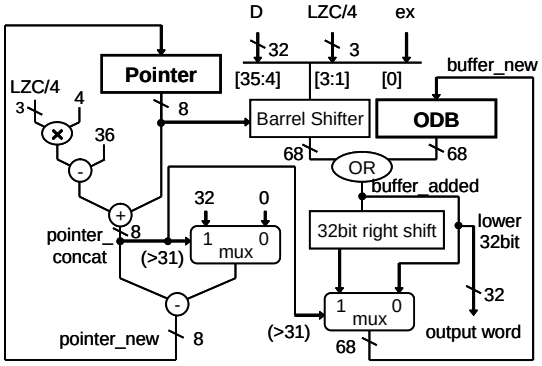


Fig. 5: Design of the variable-to-fixed length converter.

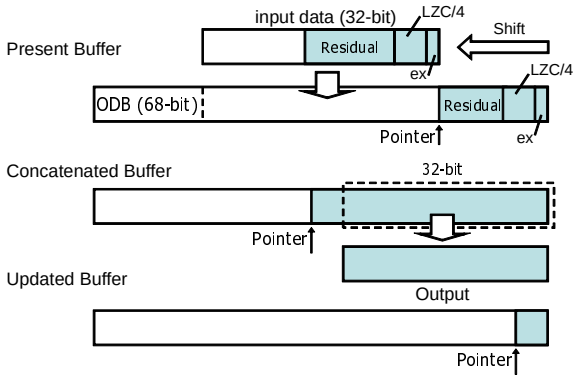


Fig. 6: Buffer behavior of the variable-to-fixed length converter.

Then the predictor computes Eq.(1) to give prediction P for the current input F. Fig.4 shows the structure of the cubic predictor, which takes the four inputs from the shift buffer in parallel. Here we use integer operations for prediction with Eq.(1) instead of floating-point operations because floating-point units have high hardware costs and longer delays than integer units. Our experiments show that the prediction in integer provides almost the same compression performance as the floating-point prediction, especially for floating-point numbers whose exponents rarely change. In the case of the fluid data computed by the lattice Boltzmann method, the integer prediction achieves a comparative compression-ratio to that by the floating-point prediction thanks to the small fluctuation of the data.

DCU computes the difference, D, between P and F by subtracting the smaller from the larger after swapping P and F if necessary. DCU also outputs the exchange signal, ex, which is asserted when P and F are swapped. LZCU computes (LZC) of D. Finally VFCONV outputs words of compressed data, which are of $\{r, \mathrm{LZC}, \mathrm{ex}\}$.

The decompressor consists of *a fixed-to-variable length converter (FVCONV), the predictor with the buffer, a data-reconstruction unit (DRU)* and *a floating-point converter (FPCONV)*. The predictor and the buffer are the same as those of the compressor. The decompressor outputs the original floating-point data-stream from the compressed data stream. FVCONV generates D with ex, LZC and r. The predictor gives prediction P with the previously decompressed numbers stored in the buffer. DRU reconstructs the integer of the original data, F, with D, P and ex by performing the inverse operation of the difference-computation. Finally FPCONV converts F to its floating-point number f.

As shown in Figs.3a and b, the compressor and the decompressor are each pipelined with the four stages and the three stages, respectively, to process one floating-point datum every cycle at high operating frequency. The details of the units in the compressor and decompressor except VFCONV and FVCONV are described in [4].

2.4 Related work

Several lossless compression algorithms for floating-point data have been proposed, which are targetting at software implementation. Lindstorm et al.[5] proposed the compression algorithm that combines prediction and entropy coding. They achieved high compression ratios for 2D and 3D data sets by using 2D or 3D prediction functions. Ratanaworabhan et al.[8] proposed compression for double-precision floating-point numbers using a hash table for context-based prediction. Their algorithm also achieved high compression ratio when using a sufficiently large table. As stated above, these software implementations are able to achieve high compression ratio. However, they have problems of area when considering hardware implementation. For example, the 2D or 3D prediction require large circuits and buffer memory. For good context-based prediction, we also need a large hash table

On the other hand, some hardware designs of lossless compression have been reported. Sukhwani et al.[12] presented compression hardware based on context-based prediction using a hash table. In their hardware, 8 bytes of input data get converted to 2, 4, 8 bytes for different predictions in parallel, and the most efficient one is selected. This hardware provides very high-throughput compression, however, it is not designed to directly compress floating-point numbers. And therefore an expected compression ratio is not so high for floating-point numbers. Tomori et al.[13] reported hardware design to compress double-precision floating-point data. Their design achieves high throughput compression by limiting entropy coding to only the upper 12 bits of the data. However, the compression ratio by this hardware is low, and only the decompressor was implemented and evaluated.

Our hardware consumes less resources, and achieves both a high compression ratio and a high throughput for floating-point numbers. We present complete implementation of the

compressor and the decompressor for evaluation with an FPGA-based experiment system.

3. Design of variable-to-fixed and fixed-to-variable length converters

In our prior research [4], the variable-to-fixed length converter (VFCONV) and the fixed-to-variable length converter (FVCONV) were not designed and evaluated. However, these modules are also important for conversion between a variable-length but every-cycle output to a fixed-length but intermittent output for compressed data. The following subsections describe the detailed design and behavior of VFCONV and FVCONV.

3.1 Variable-to-fixed length converter

Fig.5 shows the design of the variable-to-fixed length converter (VFCONV). VFCONV consists of *a 68-bit output data buffer (ODB)* and *a Pointer*. *Pointer* specifies the number of bits accumulated in *ODB* at the moment. *ODB* accumulates the inputs in *ODB* and outputs a 32-bit word when *Pointer* is greater than or equal to 32.

VFCONV is in the stage 4 of the compressor. It receives ex and *D* from DCU and LZC/4 from LZCU, and outputs words of compressed data. Fig.6 shows the behavior of the *ODB*. The input is of 8 to 36 bits with a variable length of a multiple of 4, while the output is a fixed 32-bit word.

First, VFCONV concatenates ex, LZC/4 and *D* into a block, so that they are aligned from LSB to MSB. Next, the block is shifted left so that the block exists in the next empty bits of ODB. The shift amount is specified by *Pointer*. The shifted block is inserted into the next empty positions of the present ODB by OR operation, giving *buffer_added*. The width of the block is obtained with LZC/4, then *pointer_concat* is computed by adding *Pointer* and the width of the block.

If *pointer_concat* is greater than or equal to 32, the lowest 32 bits of *buffer_added* are outputted with a valid signal. Simultaneously, *Buffer_added* is shifted 32-bit right to form *Buffer_new*, which is used to update *ODB* for the next cycle. *Pointer* is also updated with the value given by subtracting *Pointer_concat* with 32. Then the next datum is inputted to VFCONV. If *Pointer* exceeds 67, *ODB* and *Pointer* are not updated and a signal is outputted to stall the entire pipeline of the compressor. When the last datum is inputted into VFCONV, *ODB* can have remaining bits shorter than 32 bits. We flush them out at the end of compression with zeros padded to form a 32-bit word.

3.2 Fixed-to-variable length converter

Fig.7 shows the design of the fixed-to-variable length converter (FVCONV). FVCONV is in the stage 1 of the decompressor. It receives words of the compressed data containing ex, LZC/4 and *residual bits*, and outputs ex and *D* to DRU for each encoded number. FVCONV consists of *a 68-bit input data buffer (IDB)*, *a Pointer* and *a Mask Generation Unit (MGEN)*. Along with VFCONV, *Pointer* shows the number of bits accumulated in *IDB*. *IDB* receives and accumulates the next word when *Pointer* is less than 32. *MGEN* makes a 32-bit mask to clear unnecessary bits from the output of *D*.

Fig.8 shows the behavior of the *IDB*. *IDB* stores ex, LZC/4 and *residual bits* for multiple compressed numbers. By reading LZC/4, the length of the *residual bits* is computed. If the entire *residual bits* exist in *IDB*, it is outputted as *D* and *IDB* is shifted. Then the next input word is inserted into *IDB*.

The input word is shifted left so that it fits the next empty positions in *IDB*. The shift amount is given with *pointer_reduced* $= (Pointer - bit\_length)$, where *bit_length* is the length of *residual bits*, LZC/4, ex in Fig.7. If *pointer_reduced* is less than 36, the shifted bits are inserted into *IDB* by using OR operation, and *Pointer* is updated with $(pointer\_reduced + 32)$. In order to extract *D* from *IDB*, it needs to get the bit length of each compressed data. Since the bits of LZC/4, ex are always at the LSB of *IDB*, we can read and use LZC/4 to compute the bit length of *residual bits* by computing $(bit\_length - 4)$. Then *MGEN* makes a 32-bit mask with *bit_length*. The 32-bit *D* is generated by performing AND operation with the mask and the 32-bit of *IDB*[35:4].

4. FPGA-based implementation and performance evaluation

4.1 Implementation of prototype system

We have implemented the compressor and the decompressor in the FPGA-based prototype system of Fig. 9. The system is composed of the PCI-Express Gen.2 x4 controller, the two DDR2 memory controllers, and the compressor and the decompressor. Please note that the prototype system is designed just for verification and experiments, so that non-compressed data are read and written from/to the external memories. For real applications, we will use the compressor and the decompressor as shown in Fig.1 to provide a computing engine the enhanced bandwidth from the memory bandwidth by reading and writing compressed data from/to the memories. By executing a control software on the host PC, we can stream data from a DDR2 memory to the other being compressed or decompressed by the compressor or the decompressor. We used ALTERA Qsys development tool to generate the system with the PCI-Express and DDR2 controllers. We implemented the prototype system with ALTERA Stratix IV FPGA EP4SGX230, which is on the TERASIC DE4 development-board [14], while the PCI-Express controller operates at 250MHz, the compressor and the decompressor in the prototype system operate at 125MHz. We implemented the four-stage compressor and

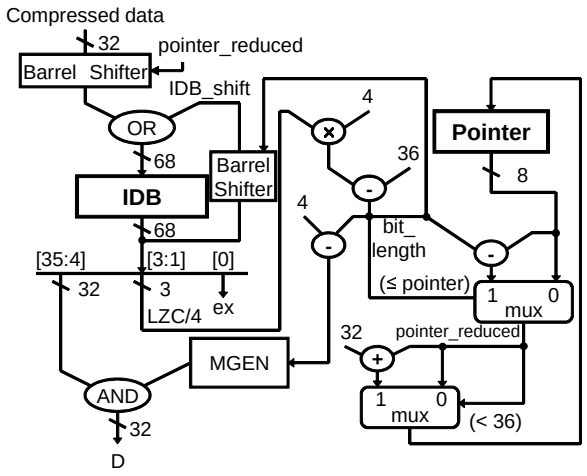


Fig. 7: Design of the fixed-to-variable length converter.

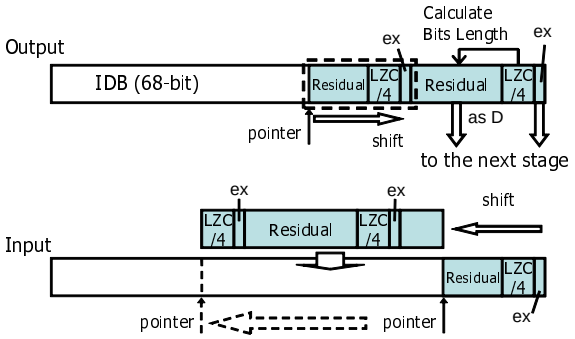


Fig. 8: Buffer behavior of the fixed-to-variable length converter.

the three-stage decompressor of Fig.3 for IEEE754 single precision floating-point numbers with the cubic predictor of Fig.4. All the logics are written in Verilog-HDL, and compiled with ALTERA Quartus II compiler ver.11.1 where "Speed" option is specified.

4.2 Resource consumption

Table 1 shows resource consumption of the compressor and the decompressor. The compressor consumes only 909 Adaptive look-up tables (ALUTs) and 612 dedicated registers, which correspond to 0.5 % and 0.34 % of the total resources on EP4SGX230 FPGA, respectively. The biggest module in the compressor is VFCONV, which consumes 313 ALUTs (0.17 %) and 110 registers (0.06 %). The decompressor is also very small, consuming only 822 ALUTs and 439 dedicated registers which are 0.45 % and 0.24 % of the total resources, respectively. FVCONV is the largest module, consuming 428 ALUTs (0.23 %) and 106 registers (0.058 %). These results show that the designed compressor and decompressor are so small that they occupy less than 1 % of the total resources of this FPGA, consuming less resources than those of [12]. Furthermore, they require no block memory and no DSP block. This is very important for auxiliary utilization of compression because such embedded hard macros should be used by major computing modules.

4.3 Operating frequency and throughput

For the compilation results, we obtained the critical paths and their delays of each pipeline stage by using the Timing-Analyzer tool of Quartus II compiler. Table 2 shows the critical-path delays and the maximum operating frequency, $F_{\max}$, calculated with the delays. All the stages of the compressor and the decompressor have $F_{\max}$ higher than 125 MHz of the operating frequency of the prototype system.

In the compressor, Stage 1 has the longest delay of 4.141 ns, because of the critical path from the shift buffer to the pipeline register through the cubic predictor. The second longest delay is of the critical-path in Stage 4 that includes VFCONV. This is due to the shift and concatenation operations in VFCONV. Consequently, the maximum frequency of the compressor is given $F_{\max} = 241$ MHz by Stage 1. We expect operation at 250 MHz for the compressor by further optimizing or pipelining the cubic predictor and VFCONV.

In the decompressor, Stage 2 has the longest delay of 5.052 ns, which gives $F_{\max} = 197.9$ MHz to the decompressor. Stage 1 of the compressor has the shorter delay than 5.052 ns irrespective of using the predictor. This is because the critical path of the decompressor's Stage 2 includes the data-reconstruction unit in the path from and to the buffer. Since the feedback loop in Stage 2 cannot be pipelined further with 1-cycle feedback maintained, we need much more effort to redesign and optimize Stage 2 for $F_{\max}$ of 250 MHz or higher. This is our future work.

4.4 Verification and compression ratio

We verified correct compression by the implemented hardware in comparison with software-based implementation. For verification, we used the computational results of the 2D9V LBM [4] shown in Figs.10. Each computational result is composed of nine streams, each of which contains 800×480 floating-point numbers in single precision. As a result, we made sure that the hardware compressor correctly outputs the compressed data, and the decompressor reconstructs the original floating-point data completely.

Then we evaluated *a compression ratio* which is defined with $R_{\text{comp}} = \frac{\text{(Size of original data)}}{\text{(Size of compressed data)}}$. We also used the same data as of Figs.10 for evaluation. The average compression-ratio of the data is $R_{\text{comp}} = 3.7$, where most of floating-point

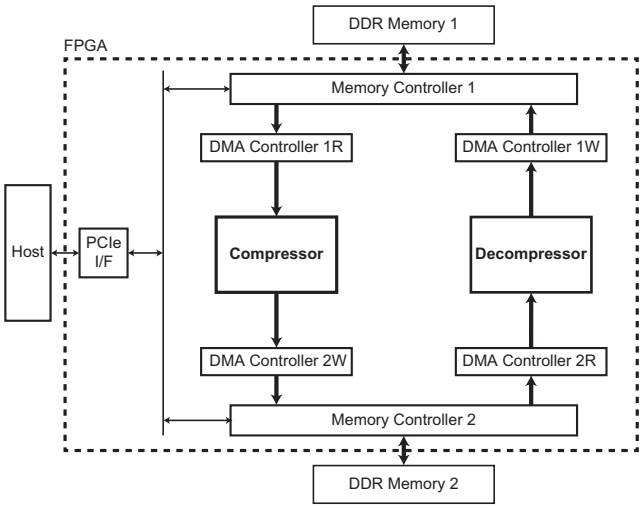


Fig. 9: Prototype implementation for experiments.

Table 1: Resource consumption of Stratix IV EP4SGX230.

Modules	ALUTs	Dedicated registers	Block memory bits
Compressor	909 (0.50%)	612 (0.34%)	0 (0.0%)
Controller	119 (0.065%)	166 (0.091%)	0 (0.0%)
Cubic predictor	195 (0.11%)	132 (0.72%)	0 (0.0%)
LZCU	4 (0.002%)	0 (0.0%)	0 (0.0%)
VFCONV	313 (0.17%)	110 (0.060%)	0 (0.0%)
Others	278 (0.15%)	204 (0.11%)	0 (0.0%)
Decompressor	822 (0.45%)	439 (0.24%)	0 (0.0%)
Controller	100 (0.055%)	166 (0.091%)	0 (0.0%)
Cubic predictor	228 (0.12%)	132 (0.072%)	0 (0.0%)
FVCONV	428 (0.23%)	106 (0.058%)	0 (0.0%)
Others	66 (0.036%)	35 (0.019%)	0 (0.0%)
Stratix IV EP4SGX230	182400	182400	14625792

numbers are compressed at the ratio of 4 while some few numbers intermittently have lower ratios from 2 to 3. Please note that the theoretically maximum compression ratio is $32/8 = 4$ for the compressor. The average compression-ratio, $R_{\text{comp}} = 3.7$, means that the compressor and decompressor can reduce the bandwidth required for the floating-point data-stream by a factor of 3.7. Compression ratios for different predictors are reported in [4].

In the prototype system running at 125 MHz, the bandwidth of the input data is $W_{\text{orig}} = 4 \times 125 = 500MB/s$. The compressor reduces W_{orig} into $W_{\text{comp}} = 500/3.7 = 135$ MB/s. In other words, we can feed the decompressor with a compressed data at a rate of 135 MB/s or less, which is enhanced to 500 MB/s at most. If we assume operation at 200 MHz, the compressor can reduce the bandwidth of 800 MB/s into 216 MB/s. However, the peak bandwidth of DDR3-1600 memory at 800MHz or PCI-Express Gen3.0 with 8 lanes are 12000 MB/s and 16000 MB/s, respectively, which are much higher than the input bandwidth of the compressor operating at 200 MHz. If we need enhance such high bandwidth further, we can use multiple compressors and decompressors to simultaneously transmit multiple data-streams in parallel. Designing such a parallel compressor is our future work.

Table 2: Critical path delay and maximum frequency (F_{max}) of each pipeline stage.

	Compressor				Decompressor		
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 1	Stage 2	Stage 3
Delay of critical path [ns]	**4.141**	3.946	3.596	4.045	4.550	**5.052**	3.439
F_{max} [MHz]	**241.5**	253.4	278.1	247.2	219.8	**197.9**	290.8

5. Conclusions

In this paper, we have presented full implementation of FPGA-based lossless compressor and decompressor for a floating-point data stream. We have verified their behavior and evaluated resource utilization, an operational frequency and compression performance of the implemented hardware. We implemented the prototype system with a PCI-Express interface and two DDR2 controllers on ALTERA Stratix IV EP4SGX230 FPGA. Compilation results show that the compressor and the decompressor are very small, each of which consumes only 0.5 % of the total ALUTs of the FPGA without using block memories and DSP blocks. Such very compact modules are suitable for auxiliary utilization to enhance the bandwidth in HPC systems. Timing analysis shows that the compressor and the decompressor can operate at 200 MHz at least, while we need to optimize or redesign the feedback loop including the predictor for higher frequency of the decompressor. The average compression ratio of 3.7 means that the compressor can reduce the bandwidth to 1/3.7 of the original one.

Since the compressor can have the input bandwidth of up to 800 MB/s assuming 200 MHz operation, which is less than the bandwidth of a DDR3 memory or a PCI-Express interface. To apply compression hardware to these wider bandwidth, we will design a parallel compressor and decompressor to handle multiple data-streams with multiple compression modules. We will develop their IP cores and apply them to our LBM accelerator.

Acknowledgments

This research was partially supported by Grant-in-Aid for Scientific Research (B) No.23300012 and Grant-in-Aid for Challenging Exploratory Research No.23650021 from the Ministry of Education, Culture, Sports, Science and Technology, Japan. We thank the support by the ALTERA university program.

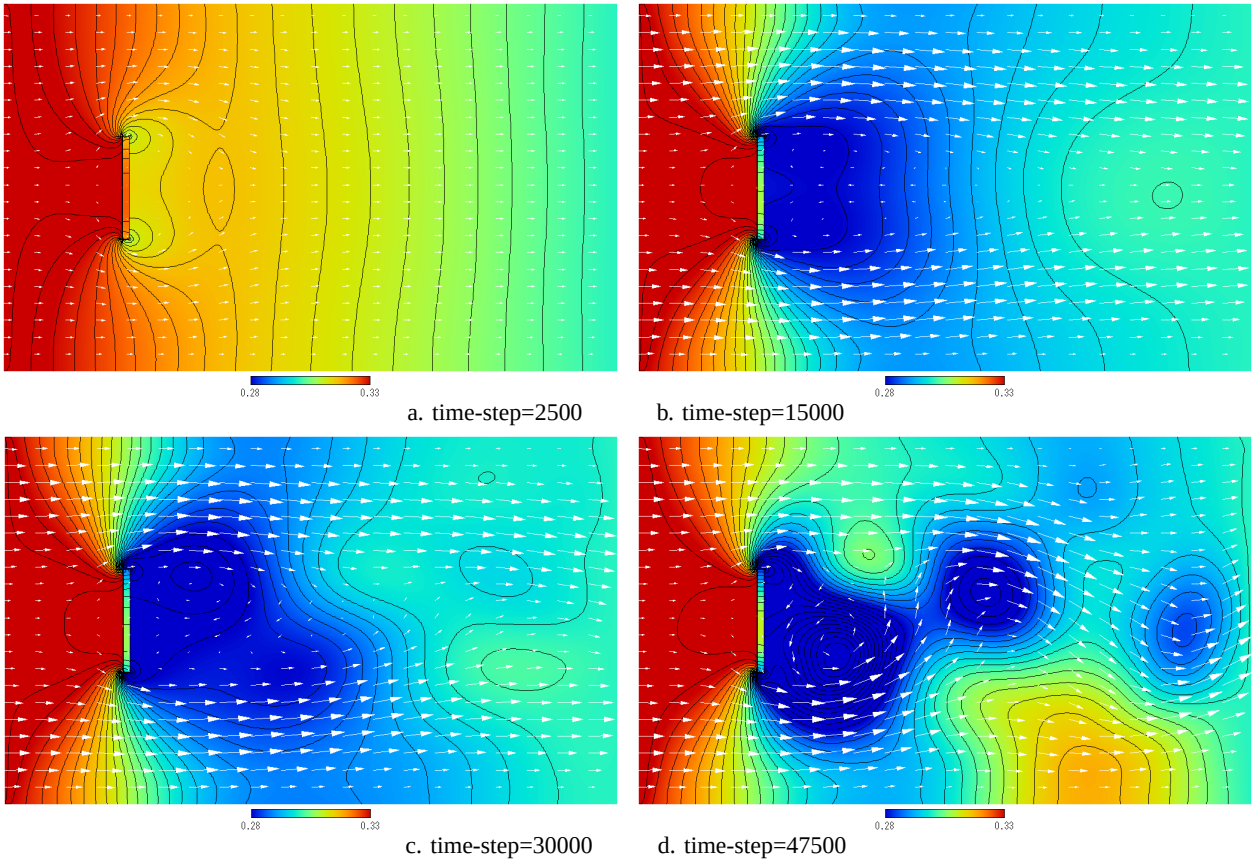


Fig. 10: Computational results of 2D time-dependent flow on a 800x480 grid by LBM. Background shows pressure/density. Vectors show velocities.

References

[1] K. Sano, O. Mencer, and W. Luk, "FPGA-based acceleration of the lattice boltzmann method," *Proceedings of the International Conference on Parallel Computational Fluid Dynamics*, May 2007, CDROM (paper-041).

[2] K. Sano, O. Pell, W. Luk, and S. Yamamoto, "FPGA-based streaming computation for lattice boltzmann method," *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pp. 233–236, December 2007.

[3] K. Sano, K. Katahira, and S. Yamamoto, "Segment-parallel predictor for FPGA-based hardware compressor and decompressor of floating-point data streams to enhance memory i/o bandwidth," *Proceedings of the Data Compression Conference (DCC)*, pp. 416–425, March 2010.

[4] K. Katahira, K. Sano, and S. Yamamoto, "FPGA-based lossless compressors of floating-point data streams to enhance memory bandwidth," *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 246–253, July 2010.

[5] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visual and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, September/October 2006.

[6] M. Isenburg, P. Lindstrom, and J. Snoeyink, "Lossless compression of predicted floating-point geometry," *Computer-Aided Design*, vol. 37, no. 8, pp. 869–877, January 2005.

[7] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," *Proceedings of Eurographics*, vol. 22, no. 3, pp. 343–348, September 2003.

[8] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," *Proceedings of Data Compression Conference*, pp. 133–142, March 2006.

[9] M. Burtscher and P. Ratanaworabhan, "FPC: a high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computer*, vol. 58, no. 1, pp. 18–31, January 2009.

[10] "bzip2," *http://www.bzip.org/*, 2010.

[11] V. Engelson, D. Fritzson, and P. Fritzson, "Lossless compression of high-volume numerical data from simulations," *Proceedings of Data Compression Conference (DCC)*, pp. 574–586, September 2000.

[12] B. Sukhwani, B. Abali, B. Brezzo, and S. Asaad, "High-throughput, lossless data compression on FPGAs," *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 113–116, May 2011.

[13] H. Tomari, M. Inaba, and K. Hiraki, "Compressing floating-point number stream for numerical applications," *2010 First International Conference on Networking and Computing*, pp. 112–119, Novenber 2010.

[14] Terasic Technologies, http://www.terasic.com.

Optical configuration acceleration on a new optically reconfigurable gate array VLSI using a negative logic implementation

Retsu Moriwaki and Minoru Watanabe
Electrical and Electronic Engineering
Shizuoka University
3-5-1 Johoku, Hamamatsu, Shizuoka 432-8561, JAPAN
Email: tmwatan@ipc.shizuoka.ac.jp

***Abstract*—Since progress in very large scale integration (VLSI) process technology, on which field programmable gate arrays (FPGAs) are based, is slowing, alternative methods to increase FPGA performance that are independent of process technology progress are being explored. Among such explored methods, high-performance FPGAs that extract optical capabilities have been developed. One of them is a holographic FPGA that can support nanosecond-order high-speed reconfiguration along with numerous reconfiguration contexts. To date, some reports have described that the performance of a programmable gate array with the same structure as an FPGA can be increased dramatically by exploiting such a high-speed reconfiguration capability along with numerous reconfiguration contexts. Therefore, although the reconfiguration speed of a holographic FPGA is of nanosecond-order, this paper presents acceleration experiments of the optical reconfiguration on a new fabricated optically reconfigurable gate array VLSI. Based on the experimental results, this paper clarifies the acceleration method's effectiveness.**

***Keywords*-High speed dynamic reconfiguration, fine-grained programmable gate arrays, negative logic implementations, optical configurations.**

I. Introduction

Recently, use of field programmable gate arrays (FPGAs) is increasing drastically, with extension to mass production from smaller production scales [1]–[4]. However, progress in very large scale integration (VLSI) process technology, on which the FPGA is based, is slowing because the gate insulator thickness of transistors is approaching atomic scale [5][6]. Therefore, alternative methods to increase FPGA performance that are independent of process-technology progress are being explored [7],[8].

Among such exploration methods, high-performance FPGAs extracting optical capabilities have been developed. Although mainstream optical FPGAs target optical communication and optical sensing [9]–[11], some studies of holographic FPGAs have been conducted as VLSI accelerators. They can realize a larger virtual gate array than a physical gate array on a VLSI and a high-speed reconfiguration [12]–[14]. A programmable gate array on a VLSI-chip inside the holographic FPGA is a fine-grained gate array as well as that of FPGAs. The basic function and construction of the programmable gate array are the same as those of currently available FPGAs.

However, by exploiting such a high-speed reconfiguration capability along with numerous reconfiguration contexts that cannot be realized on currently available FPGAs, the performance of a programmable gate array having the same structure as FPGA can be increased dramatically. For example, FPGAs are implemented onto many embedded systems currently, and demand for implementing a processor onto an FPGA is increasing. To satisfy that demand, FPGA vendors have come to provide soft-core processors for FPGAs [15]–[18]. Altera Corp. provides a NIOS processor [15][16], whereas Xilinx Inc. provides its Micro Blaze processor [17],[18]. However, the soft-core processors invariably have lower performance than hard-core processors inside FPGAs, Intel's processors, ARM processors [19][20], and so on. Therefore, a hard-core processor or an external processor chip must invariably be implemented in addition to an FPGA to produce high-performance embedded systems.

Such low performance of soft-core processors on FPGAs results from its look-up table (LUT) and switching matrix (SM) architecture. In a custom processor, dedicated logic circuits and metal wires are used, respectively, instead of such LUTs and SMs. Therefore, it is difficult for an FPGA's soft-core processor under the same static implementation to surpass the performance of custom processors. However, if FPGA programmability can be exploited, then the performance of its programmable gate array can be increased. The idea is based on high-speed dynamic reconfiguration. Such a holographic FPGA can support such high-speed dynamic reconfiguration.

Recently, almost all computer systems have come to use reduced instruction set computer (RISC) architecture [21],[22]. RISC architectures can offer benefits in terms of higher clock frequency, smaller implementation area, and lower power consumption than conventional complex instruction set computer (CISC) architectures [23],[24]. Their success is based on the fundamental principle that the simplest circuit can function with the highest clock frequency in the smallest implementation area, and with the lowest power consumption because the simplest circuit can be constructed with fewer selector passes, less load capacitance of fewer gates, and less capacitance of short metal wires. This principle is also applicable to fine-grained programmable gate arrays. Advancing that

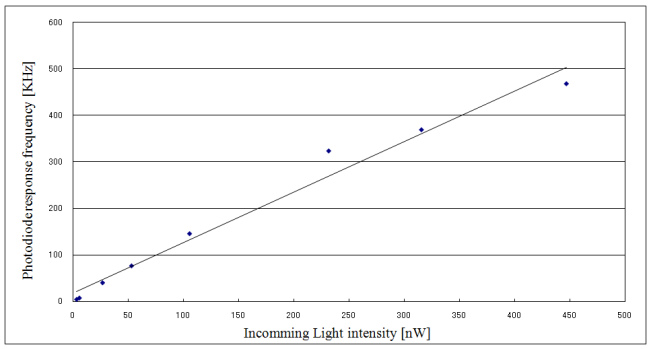


Fig. 1. Relation between photodiode response frequency and its incoming light intensity.

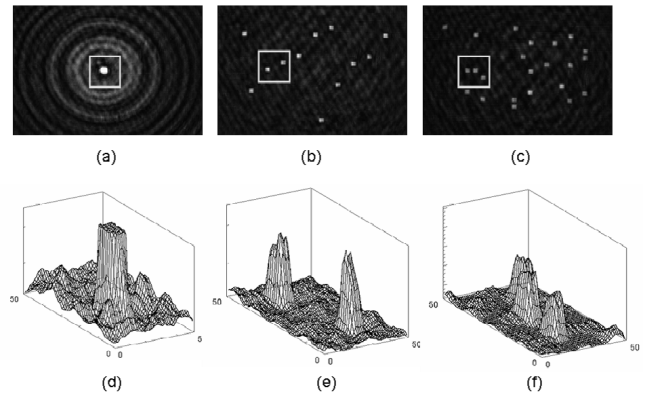


Fig. 2. Light intensity of each bit of optical configuration context diffracted from a holographic memory. Panels (a) and (d) show the light intensity of a configuration context including a single bright bit. Panels (b) and (e) show the light intensity of a configuration context including 12 bright bits. Panels (c) and (f) show the light intensity of a configuration context including 23 bright bits.

consideration, ultimately the simplest processor type reaches a mono-instruction set computer (MISC). In the MISC, only those instructions necessary for a single clock cycle can be implemented onto a clock-by-clock reconfigurable gate array. Such a processor can operate at the highest clock frequency, with the lowest power consumption, and in the smallest implementation area. Moreover, the extremely small implementation area enables large parallel computation. Consequently, its overall performance can be increased dramatically.

Therefore, the high-speed reconfiguration capability of a gate array is a key technology to accelerate the gate array's performance. In terms of such high-speed reconfigurations, a holographic FPGA is superior to current VLSI-based programmable gate arrays. However, to accelerate the reconfiguration procedures on a holographic FPGA further, laser power must be increased. However, the use of such high-power lasers increases power consumption and package size; in the worst case, their use might require a special cooling system. Therefore, this paper presents a negative logic circuit implementation method by which optical configurations can be accelerated without any increase of laser power. Based on the experimental results, this paper clarifies the acceleration method 's effectiveness.

II. Negative logic implementation

A holographic FPGA receives optical configuration contexts on photodiodes. The photodiode response frequency is typically proportional to the light intensity: high-intensity light reduces the period of optical reconfiguration, as shown in Fig. 1. Therefore, although the easiest way to reduce reconfiguration periods is to use high-power lasers, the use of such lasers increases the unit's power consumption and might necessitate the use of a cooling system, which would greatly increase the package size. For that reason, high-power lasers should be used only as a last resort.

On the other hand, in a holographic FPGA, a holographic memory generates optical configuration contexts. A holographic memory has a property by which the light intensity of each bit that is diffracted from a holographic memory is inversely proportional to the number of bright bits included in a configuration context, as shown in Fig. 2. Summarizing those points very simply, if the number of bright bits in a certain configuration context could be decreased, then the reconfiguration speed of the context can be accelerated with no increase of laser power. Such a method also obviates the use of high-power lasers or components for cooling the device. This proposed negative logic implementation is a method to reduce the number of such bright bits included in a configuration context.

A. Generation of an optical configuration vector

This strategy treats configuration data used for a look-up table. The following discussion uses bit-length N, which represents the number of bits included in a look-up table. A configuration vector $\boldsymbol{\alpha}$ is used for programming a gate array and an optical configuration vector $\boldsymbol{\gamma}$, the information of which is programmed onto a holographic memory, are signified as N-dimensional vectors, as described below.

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_N), \quad (1)$$

$$\boldsymbol{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_N), \quad (2)$$

Therein, each element of the vectors takes a binary value {0,1}. The optical configuration vector $\boldsymbol{\gamma}$ is defined as follows.

$$\boldsymbol{\gamma} = \boldsymbol{\alpha} \oplus I_s, \quad (3)$$

In that equation, $\boldsymbol{\alpha}$ is a certain configuration context and I_s is an inversion state included in an optical configuration vector $\boldsymbol{\gamma}$, and $\oplus$ signifies an exclusive OR operator. The inversion state is defined in the following equation.

$$I_s = \begin{cases} 1 & : \sum_{i=1}^{N} \alpha_i \geq [\frac{N}{2} + 1], \\ 0 & : \text{otherwise.} \end{cases} \quad (4)$$

In that equation, $[N/2+1]$ denotes that $N/2+1$ is rounded down to the nearest whole number. When $I_s = 1$, an inverter function must be added onto an optically reconfigurable logic block in addition to the look-up table implementation explained above. Consequently, a calculated optical configuration

vector $\boldsymbol{\gamma}$ and an inverter, if it is needed, are programmed onto a holographic memory.

1) Actual implementation: When the inversion state is high, the implementation is a negative logic circuit. For example, for an OR circuit implementation, the same logic circuit can be realized using a NOR circuit, with an inverter connected to the NOR circuit output. Therefore, the inversion bit signifies the implementation of an inverter connected to the output of a negative logic circuit. Using a negative logic circuit and an inverter, the number of bright bits can be decreased with no accompanying increase of hardware.

2) Estimation equation: The reduction efficiency of the number of bright bits in a configuration context of conventional ORGAs without negative logic implementation is estimated first. The average number of 1s corresponding to laser irradiation is calculated by counting bit 1 of all possible vectors and dividing it by $N2^N$ of the summation of bits of all possible vectors, as in the following equation.

$$\kappa_{ORGA} = \frac{\sum_{r=1}^{N} r \cdot {}_N C_r}{2^N N} = \frac{1}{2}, \tag{5}$$

In that equation, ${}_N C_r$ represents a combination. Here, assuming that configuration contexts are given continuously for an ORGA-VLSI and assuming that they include all possible patterns uniformly, the average number of 1s can be estimated as 0.5.

Similarly, the reduction of the number of bright bits included in a configuration context of the negative logic implementation method can be estimated. The average number of writing '1' corresponding to laser irradiation is calculated by counting the '1' bits of all possible vectors and then dividing that number by $N2^N$ of the summation of bits of all possible vectors, as presented in the following equation.

$$\kappa_{new} = \frac{\sum_{r=1}^{[\frac{N}{2}]} r \cdot {}_N C_r}{2^N N} + \frac{\sum_{r=[\frac{N}{2}+1]}^{N} (N - r + 1) \cdot {}_N C_r}{2^N N} \tag{6}$$

The first term in the right side of the upper Eq. (6) is identical to Eq. (5). In this case, an inversion state I_s is equal to 0. In addition, the second term in the right side of the upper Eq. (6) represents the case in which the inversion state I_s is equal to 1.

3) Reduction effect for a look-up table: Here, the reduction effect for a look-up table is estimated. When a two-input one-output circuit is implemented to a look-up table, the number N of states on a look-up table is 4. In this case, the reduction effect is calculable as 21.9%. Furthermore, when a three-input one-output circuit is implemented to a look-up table, the number N of states on a look-up table is 8. At that time, the reduction effect is calculable as 18.3%. In addition, when a four-input one-output circuit is implemented to a look-up table, the number N of states on a look-up table is 16. In this case, the reduction effect is calculable as 14.6%. Finally, in the case of four-input one-output look-up table, the reduction effect of 14.6% - 21.9% can be expected. Consequently, the reconfiguration frequency can be increased by the reduction.

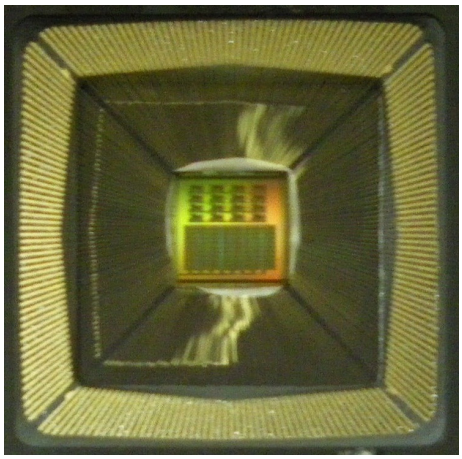


Fig. 3. Photograph of a new 0.18 μm CMOS process ORGA-VLSI.

III. New 0.18 μm CMOS process optically reconfigurable gate array VLSI

Figure 3 portrays a photograph of a new 0.18 μm CMOS process optically reconfigurable gate array (ORGA) -VLSI chip that was fabricated using 0.18 μm standard CMOS process technology. The gate array of the ORGA-VLSI occupies half of a 5.0 mm by 5.0 mm chip. The ORGA-VLSI was designed mainly using a standard cell-based design except for some custom designed cells, photodiode cells, transmission gate cells, and so on. The voltages of the core and I/O cells of the ORGA-VLSI were designed, respectively, as 1.8 V and 3.3 V. The photodiode's acceptance surface size is 4.54 μm $\times$ 4.40 μm. The photodiodes were constructed between N+ diffusion and the P-substrate. Photodiode cells are arranged at 30.24 μm horizontal intervals and at 30.10 μm vertical intervals. This design incorporates 10,304 photodiodes. The gate array of the ORGA-VLSI uses an island style, as FPGAs do. In all, 80 optically reconfigurable logic blocks (ORLBs), including two 4 input – 1 output look-up tables (LUTs), 90 optically reconfigurable switching matrices (ORSMs), and 8 optically reconfigurable I/O blocks (ORIOBs), which include 4 programmable I/O bits, were implemented in the gate array. In addition, the ORGA-VLSI is the first VLSI capable of supporting the negative logic implementation. For that reason, the output of each LUT can be inverted depending on its configuration context. The direct output or inverted output can be selected only irradiating a single bit. Therefore, the ORGA-VLSI can support the acceleration method of the negative logic implementation perfectly. Although a selector and a photodoide to program the selector must be added for each logic block, the additional implementation area is less than 0.5 % of the entire gate array. The gate array's density is nearly equal to that of conventional ones.

IV. Experimental system

A. Hologram calculation method

Here, a hologram for ORGAs is assumed as a thin holographic medium. A laser aperture plane, a holographic plane,

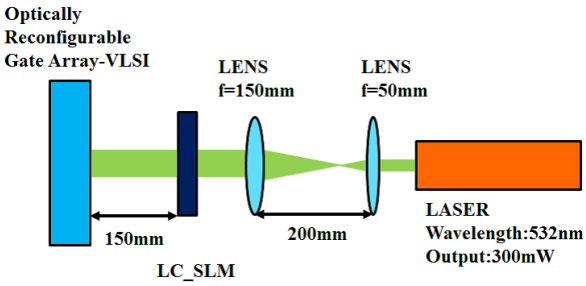


Fig. 4. Block diagram of the experimental system.

and an ORGA-VLSI plane are parallelized. The laser beam is expanded. It is assumed that the aperture is sufficiently wide for the holographic medium. Consequently, the laser beam can be considered as a plane wave. The reference wave from the laser propagates into the holographic plane. The holographic medium comprises rectangular pixels on the $x_1 - y_1$ holographic plane. The pixels are assumed as analog values. The input object comprises rectangular pixels on the $x_2 - y_2$ object plane. The pixels can be modulated to be either on or off. The intensity distribution of a holographic medium is calculable using the following equation.

$$H(x_1,y_1) \propto \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} O(x_2,y_2)\sin(kr)dx_2dy_2, \quad r = \sqrt{Z_L^2+(x_1-x_2)^2+(y_1-y_2)^2}. \tag{7}$$

In that equation, $O(x_2,y_2)$ is a binary value of a reconfiguration context, k is the wave number, and Z_L represents the distance between the holographic plane and the object plane. The value $H(x_1,y_1)$ is normalized as 0–1 for minimum intensity H_{min} and maximum intensity H_{max}, as shown below.

$$H'(x_1,y_1) = \frac{H(x_1,y_1) - H_{min}}{H_{max} - H_{min}}. \tag{8}$$

Finally, the normalized image H' is used for implementing the holographic memory. Other areas on the holographic plane are opaque to the illumination.

B. Experimental System

Figure 4 presents a block diagram of an ORGA holographic reconfiguration system. Figure 5 portrays a photograph of the experimental system. The ORGA holographic reconfiguration system was constructed using a 532 nm - 300 mW - laser (torus 532; Laser Quantum), a liquid crystal spatial light modulator (LC-SLM) as a holographic memory, and a 0.18 um CMOS process ORGA-VLSI. The beam from the laser source, the diameter of which is 1.7 mm, is expanded ten times to 5.1 mm using two lenses of 50 mm focal length and 150 mm focal length. The expanded beam is incident to a holographic memory on an LC-SLM. The LC-SLM is a projection TV panel (L3D07U-81G00; Seiko Epson Corp.). It is a 90° twisted nematic device with a thin film transistor. The panel consists of 1,920 × 1,080 pixels, each having size of 8.5 × 8.5 μm^2. The

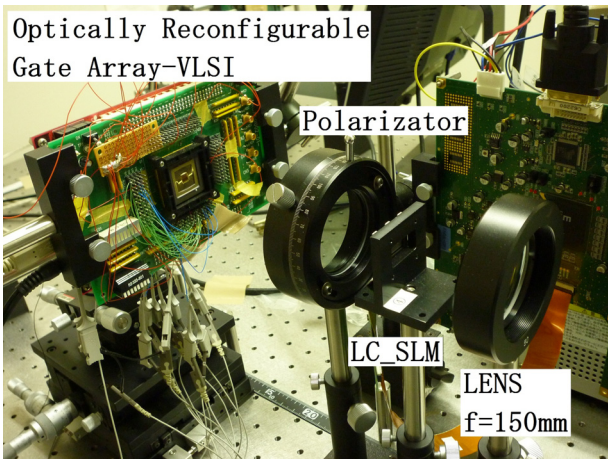
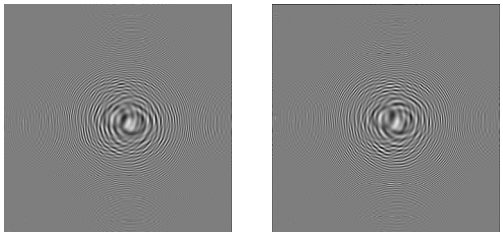
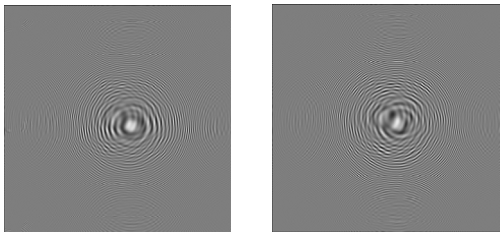


Fig. 5. Experimental system.



(a) 2-input OR (b) 2-input NOR+INV

Fig. 6. Holographic patterns of combinational logic circuits with two inputs.



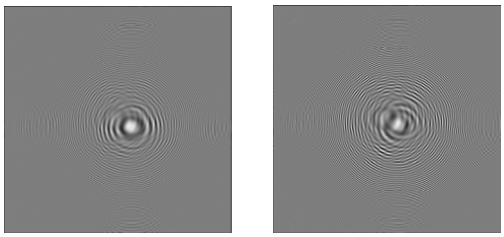
(a) 3-input OR (b) 3-input NOR+INV

Fig. 7. Holographic patterns of combinational logic circuits with three inputs.

LC-SLM is connected to an evaluation board (L3B07-E60A; Seiko Epson Corp.). The board's video input is connected to the external display terminal of a personal computer. Programming for the LC-SLM is executed by displaying a holographic memory pattern with 256 gradation levels on the personal computer display. Each holographic memory pattern was designed as 700 × 700 pixels, as shown in Figs. 6, 7, and 8.

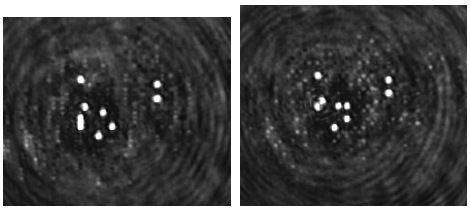
V. Experimental Results

To estimate the configuration acceleration method using negative logic implementation, 3 combinational circuits were implemented onto the system described above. All of their holographic memory patterns are shown in Figs. 6, 7, and 8. Additionally, all CCD-captured configuration context patterns are shown in Figs. 9, 10, and 11. High-contrast configuration context patterns were confirmed. The reconfiguration periods



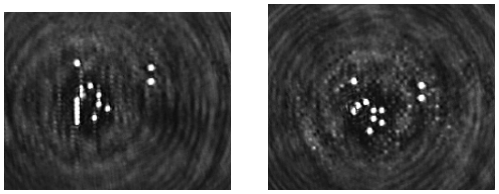
(a) 4-input OR (b) 4-input NOR+INV

Fig. 8. Holographic patterns of combinational logic circuits with four inputs.



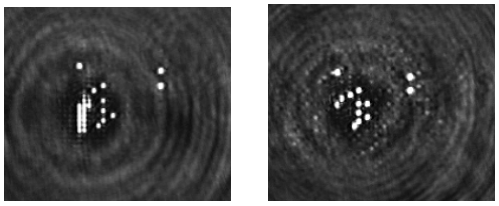
(a) 2-input OR (b) 2-input NOR+INV

Fig. 9. Context patterns of combinational logic circuits with two inputs.



(a) 3-input OR (b) 3-inputNOR+INV

Fig. 10. Context patterns of combinational logic circuits with three inputs.



(a) 4-input OR (b) 4-inputNOR+INV

Fig. 11. Context patterns of combinational logic circuits with four inputs.

were measured using the configuration contexts. The number of bright bits and configuration speed are shown in Tables 1 and 2. Results confirmed that the average reconfiguration time of a conventional ORGA was 106 ns. In contrast, the average configuration time of this negative circuit implementation was improved to 36 ns. Using this negative logic implementation, as an average, 74 percent bright bits can be removed so that reconfiguration speeds were improved to three-times faster than those of normal configurations.

VI. Conclusion

This paper has presented acceleration experiments of optical reconfigurations on a new fabricated optically reconfigurable gate array VLSI that can perfectly support a negative logic implementation. The reconfiguration frequency of the proposed method was confirmed experimentally as 3 times higher than that of a normal ORGA architecture. The result was achieved without any increase of laser power. Although a selector and a phtodoide to program the selector must be added for each logic block, the additional implementation area is very small. Consequently, the gate array's density is nearly equal to that of conventional gate arrays. Based on the experimental results, this study has clarified the acceleration method's effectiveness. Results show that we have succeeded in acceleration of a high-speed reconfigurable holographic FPGA that can execute a nanosecond-order reconfiguration.

Acknowledgments

This research was supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B), No. 24300017. The VLSI chip in this study was fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Rohm Co. Ltd. and Toppan Printing Co. Ltd.

References

[1] Altera Corporation, "Altera Unveils 28-nm Stratix V FPGA Family," http://www. altera.com, 2010.
[2] S. Chen et al., "Xilinx Next Generation 28 nm FPGA Technology Overview," http://www. xilinx.com, 2010.
[3] Xilinx Inc., "Xilinx Product Data Sheets," http://www. xilinx.com.
[4] Altera Corporation, "Actel Product Catalog," http://www.actel.com/, 2010.
[5] T. Ghani et al., "A 90nm high volume manufacturing logic technology featuring novel 45nm gate length strained silicon CMOS transistors," IEEE International Electron Devices Meeting, pp. 11.6.1-11.6.3, 2003.
[6] V. Barral et al., "Strained FDSOI CMOS technology scalability down to 2.5nm film thickness and 18nm gate length with a TiN/HfO2 gate stack," IEEE International Electron Devices Meeting, pp. 61-64, 2007.
[7] E. Culurciello, A.G. Andreou, "Capacitive coupling of data and power for 3D silicon-on-insulator VLSI," IEEE International Symposium on Circuits and Systems, Vol. 4, pp. 4142 - 4145, 2005.
[8] R. Hentschke, G. Flach, F. Pinto, R. Reis, "3D-Vias Aware Quadratic Placement for 3D VLSI Circuits," IEEE Computer Society Annual Symposium on VLSI, pp. 67-72, 2007.
[9] Prosenjit Mal, Prerna D. Patel, and Fred R. Beyette, "Design and Demonstration of a Fully Integrated Multi-Technology FPGA: A Reconfigurable Architecture for Photonic and Other Multi-Technology Applications," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS, VOL. 56, NO. 6, pp. 1182-1191, 2009.
[10] J. V. Campenhout, H. V. Marck, J. Depreitere, J. Dambre, "Optoelectronic FPGA's," IEEE JOURNAL OF SELECTED TOPICS IN QUANTUM ELECTRONICS, VOL. 5, NO. 2, 1999
[11] F. Breyer, S. C. J. Lee, D. Cardenas, S. Randel, N. Hanik, "Real-time gigabit ethernet transmission over up to 25 m Step-Index Polymer Optical Fibre using LEDs and FPGA-based signal processing, " European Conference on Optical Communication, pp. 1-2, 2009.
[12] J. Mumbru, G. Panotopoulos, D. Psaltis, X. An, F. Mok, S. Ay, S. Barna, E. Fossum, "Optically Programmable Gate Array," SPIE of Optics in Computing 2000, Vol. 4089, pp. 763–771, 2000.
[13] J. Mumbru, G. Zhou, X. An, W. Liu, G. Panotopoulos, F. Mok, and D. Psaltis, "Optical memory for computing and information processing," SPIE on Algorithms, Devices, and Systems for Optical Information Processing III, Vol. 3804, pp. 14–24, 1999.
[14] J. Mumbru, G. Zhou, S. Ay, X. An, G. Panotopoulos, F. Mok, and D. Psaltis, "Optically Reconfigurable Processors," SPIE Critical Review 1999 Euro-American Workshop on Optoelectronic Information Processing, Vol. 74, pp. 265-288, 1999.
[15] J. Perez Acle, M. S. Reorda, M. Violante, "Implementing a safe embedded computing system in SRAM-based FPGAs using IP cores: A case study based on the Altera NIOS-II soft processor," IEEE Second Latin American Symposium on Circuits and Systems, pp. 1-5, 2011.

TABLE I
BRIGHT BITS OF IMPLEMENTED CIRCUITS.

non-inversion		inversion	
Circuit Name	Bright Bits	Circuit Name	Bright Bits
2-input OR	10	2-input NOR+INV	9
3input OR	16	3-input NOR+INV	11
4-inputOR	26	4-input NOR+INV	13

TABLE II
EXPERIMENTAL RESULTS.

non-inversion		inversion		
Circuit No.	Configuration Time (ns)	Circuit No.	Configuration Time(ns)	Reduction ratio (%)
2-input OR	35	2-input NOR+INV	24	31
3inputOR	94	3-input NOR+INV	41	56
4-input OR	188	4-input NOR+INV	42	78
Average	106	Average	36	34

[16] O. A. Al Rayahi, M. A. S. Khalid, "UWindsor Nios II: A soft-core processor for design space exploration," IEEE International Conference on Electro/Information Technology, pp. 451 - 457, 2009.
[17] R. H. Klenke, "Experiences Using the Xilinx Microblaze Softcore Processor and uCLinux in Computer Engineering Capstone Senior Design Projects," IEEE International Conference on Microelectronic Systems Education, pp. 123 - 124, 2007.
[18] A. Klimm, O. Sander, J. Becker, "A MicroBlaze specific co-processor for real-time hyperelliptic curve cryptography on Xilinx FPGAs," IEEE International Symposium on Parallel & Distributed Processing, pp. 1 - 8, 2009.
[19] G. R. Allen, G. M. Swift, G. Miller, "Upset Characterization and Test Methodology of the PowerPC405 Hard-Core Processor Embedded in Xilinx Field Programmable Gate Arrays," IEEE Radiation Effects Data Workshop, Vol. 0, pp. 167 - 171, 2007.
[20] R. Ali, R. Radhakrishnan, G. Kochhar, J. Hsieh, O. Celebioglu, K. Chadalavada, R. Rajagopalan, "Evaluating performance of BLAST on Intel Xeon and Itanium2 processors," IEEE International Workshop on Workload Characterization, pp. 81-88, 2004.
[21] Y. Pizhou, L. Chaodong, "A RISC CPU IP core," International Conference on Anti-counterfeiting, Security and Identification, pp. 356 - 359, 2008.
[22] J. Goodacre, A. N. Sloss, "Parallelism and the ARM instruction set architecture," Computer, Vol. 38, Issue 7, pp. 42 - 50, 2005.
[23] T. Jamil, "RISC versus CISC," IEEE Potentials, Vol. 14, Issue 3, pp. 13-16, 1995.
[24] D. B. Tolley, "Analysis of CISC versus RISC microprocessors for FDDI network interfaces," Conference on Local Computer Networks, pp. 485 - 493, 1991.

Architecture of an Asynchronous FPGA for Handshake-Component-Based Design

ERSA'12 Regular Paper

Yoshiya Komatsu, Masanori Hariyama, and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan

Abstract— *This paper presents a novel architecture of an asynchronous FPGA for handshake-component-based design. The handshake-component-based design is suitable for large-scale, complex asynchronous circuit because of its understandability. This paper proposes an area-efficient architecture of an FPGA that is suitable for handshake-component-based asynchronous circuit. Moreover, the Four-Phase Dual-Rail encoding is employed to construct circuits robust to delay variation because the data paths are programmable in FPGA. The FPGA based on the proposed architecture is implemented in a 65nm process. Its evaluation results show that the proposed FPGA can implement handshake components efficiently.*

Keywords: FPGA, Reconfigurable LSI, Self-timed circuit, Asynchronous circuit

1. Introduction

Field-programmable gate arrays (FPGAs) are widely used to implement special-purpose processors. FPGAs are cost-effective for small-lot production because functions and interconnections of logic resources can be directly programmed by end users. Despite their design cost advantage, FPGAs impose large power consumption overhead compared to custom silicon alternatives [1]. The overhead increases packaging costs and limits integrations of FPGAs into portable devices. In FPGAs, the power consumption of clock distribution is a serious problem because it has an enormously large number of registers than custom VLSIs. To cut the clock distribution power, some asynchronous FPGAs has been proposed [2], [3], [4], [5], [6].

In asynchronous FPGAs, CAD tools that is different from ones for synchronous FPGAs is necessary to implement applications. Although, few CAD tools or design flow for asynchronous FPGAs have been introduced. As the design methods for asynchronous circuits, some method uses the Signal Transition Graph[8] and another method employs handshake components[9][10]. Handshake-component-based design is easy to understand and easy to construct datapath. Besides, Balsa[10] is proposed as a design methodology that uses handshake components. Balsa is a hardware description language and it allows circuit designers not to pay attention to low-level details such as control of handshake. Thus, it is suitable for designing complex large-scale circuits such as a DMA controller[10] and a microprocessor[11]. In Balsa, 46 handshake components are defined and complex asynchronous circuits are synthesized by combining them. Moreover, there are synthesis tools that generate a handshake circuit that consists of handshake components and a netlist consists of standard cells. Therefore, Balsa is desirable as a inputs of CAD tools for asynchronous FPGAs.

This paper proposes an area-efficient architecture of an FPGA that is suitable for handshake-component-based asynchronous circuit. The proposed architecture implements handshake components that is defined in Balsa efficiently. Small frequently-used handshake components are implemented on a logic block (LB), and other handshake components are implemented using more than one LB. As handshake components can be mapped directly on the proposed architecture, circuit designers can utilize existing CAD tools that generate a netlist of handshake components. Therefore, a design method for the proposed FPGA is established.

2. Architecture

2.1 Handshake-component-based design methodology

In asynchronous circuits, the handshake protocol is used for synchronization instead of using the clock. Figure 1 shows a four-phase handshake sequence. First, the sender sets the request wire to 1 as shown in Fig. 1(a). Second, the receiver sets the acknowledge wire to 1 as shown in Fig. 1(b). Third, the sender sets the request wire to 0 as shown in Fig. 1(c). Finally, the receiver sets the acknowledge wire to 0 as shown in Fig. 1(d) and wire values return to initial state.

A asynchronous functional element such as a binary operator is denoted by a handshake component. Figure 2 shows handshake components. Each handshake component has ports and is connected to another handshake component through a channel. Communication between handshake components is done by sending request signal from the active port and acknowledge signal from the passive port. Depending on the kind of handshake components, data

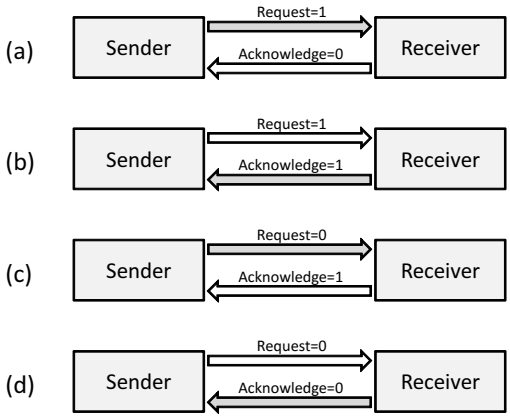


Fig. 1: A four-phase handshake sequence.

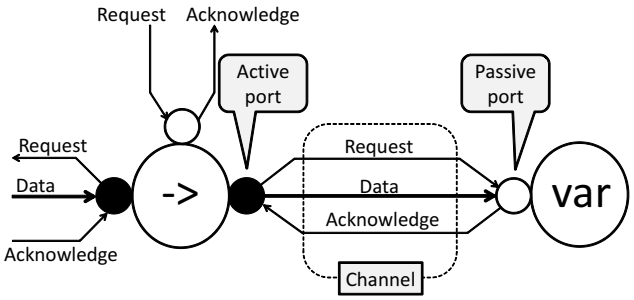


Fig. 2: Handshake components and channels.

signals are sent along with request signals or acknowledge signals. The number of ports and the width of data signal can be varied. Each function of handshake component is simple and clear. Furthermore, handshaking that consists of request signal and acknowledge signal is symbolized as a channel. Therefore, handshake circuits are easily understandable and manageable.

Handshake components constitute a handshake circuit. Figure 3 shows an example of a handshake circuit. Circuit synthesis is done by replacing each handshake component with corresponding asynchronous circuit.

2.2 FPGA architecture for Handshake-component-based design

As mentioned in preceding section, circuit synthesis is done by replacing each handshake component with corresponding asynchronous circuit. Thus, asynchronous circuits can be implemented on a conventional FPGA by replacing each handshake component with a combination of LBs. However, because it is difficult to implement the C-element that is frequently used in asynchronous circuit area-efficiently, hardware cost of a handshake component becomes large. In the proposed architecture, each LB includes dedicated circuits for implementing handshake components. Therefore, the proposed architecture can implement

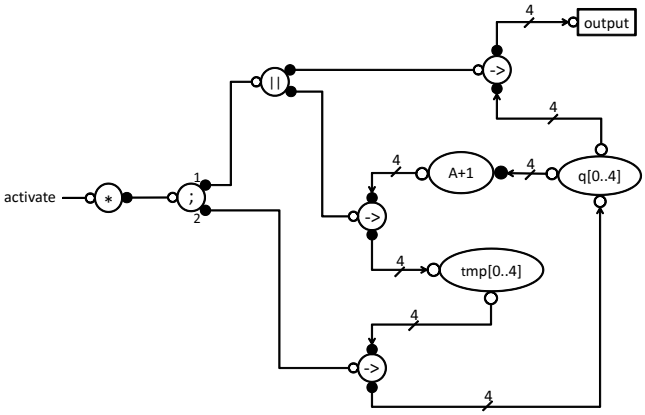


Fig. 3: A simple handshake circuit (4 bit counter).

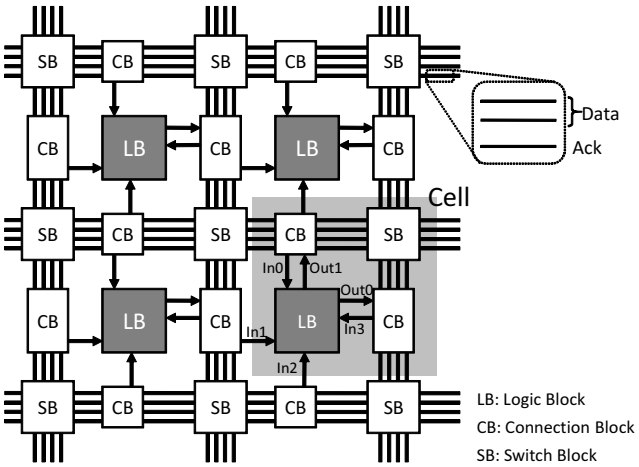


Fig. 4: Overall architecture.

handshake circuits efficiently. The proposed architecture can implement 37 out of 46 handshake components defined in Balsa. Handshake components that have multiple ports or wide datapath can be implemented using several LBs.

2.3 Overall architecture

Figure 4 shows the overall architecture of the proposed FPGA. The FPGA consists of a mesh-connected cellular array likes conventional FPGAs. In the proposed FPGA architecture, the Four-Phase Dual-Rail (FPDR) encoding is employed for asynchronous data encoding. The FPDR encoding encodes a bit and a request signal onto two wires. Table 1 shows the code table of the FPDR encoding. The main feature is that the sender sends a spacer and a valid data alternately as shown in Fig. 5. FPDR circuits are robust to the delay variation. Hence, the FPDR encoding is the ideal one for FPGAs in which the data path is programmable. Because the FPDR encoding is employed, three wires are required for a data bit. Two wires are used for the data encoded in FPDR encoding, and one wire for the acknowledge signal.

Table 1: Code table of the FPDR encoding.

	Code word (T, F)
Data 0	(0,1)
Data 1	(1,0)
Spacer	(0,0)

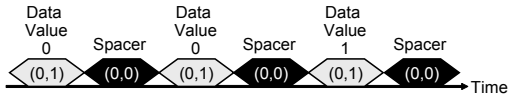


Fig. 5: Example of the FPDR encoding.

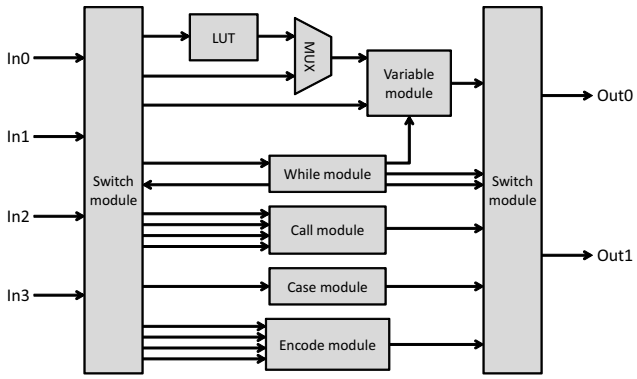


Fig. 6: Structure of an LB.

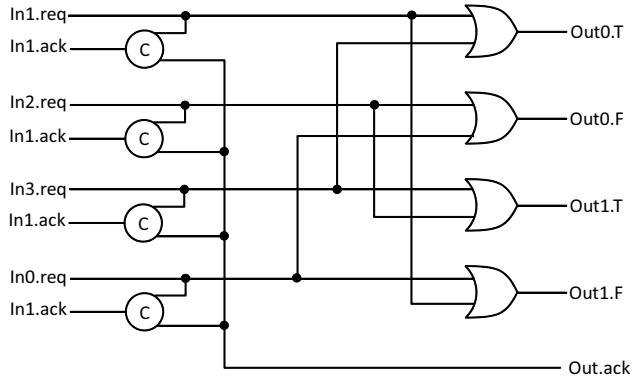


Fig. 7: Structure of an Encode module.

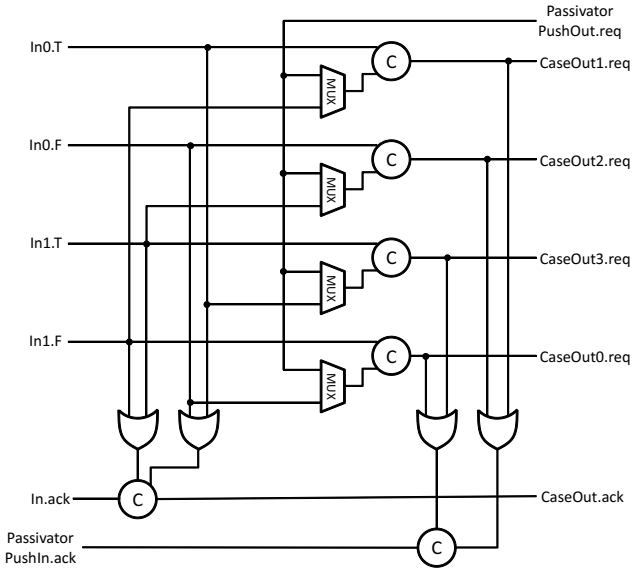


Fig. 8: Structure of a Case module.

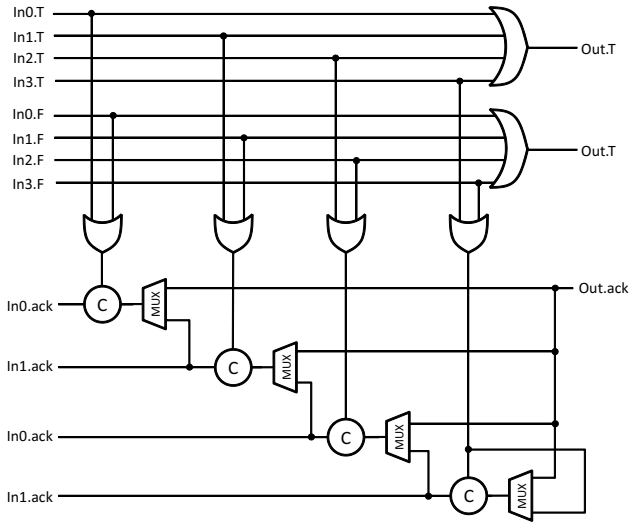


Fig. 9: Structure of a Call module.

2.4 Logic block structure

Figure 6 shows a LB of the proposed architecture. The proposed FPGA architecture can implement 37 handshake components. The LB consists of an LUT, a Variable module, a While module, a Call module, a Case module, an Encode module, multiplexers and a demultiplexer. The detailed circuits of modules are shown in Fig. 7, 8, 9 and 10. As shown in Table 2, each module implements several handshake components. In addition, several handshake components are implemented by employing programmable interconnection resources or combining two modules as shown in Table 3. The number of the transistors of the proposed FPGA is small because of resource sharing.

3. Evaluation

The proposed FPGA is implemented in a 65nm CMOS process. Table 4 shows the comparison result of the cells of the proposed architecture and the conventional architecture. The number of transistors of the proposed architecture is 2.5 times larger than the conventional one. Table 5 shows the implementation result of the Case handshake component that has four output ports. Compared to conventional architecture, the number of the transistors is reduced by 37%. This is because the proposed architecture requires one cell for implementing the four-output Case handshake component while the conventional architecture requires four cells.

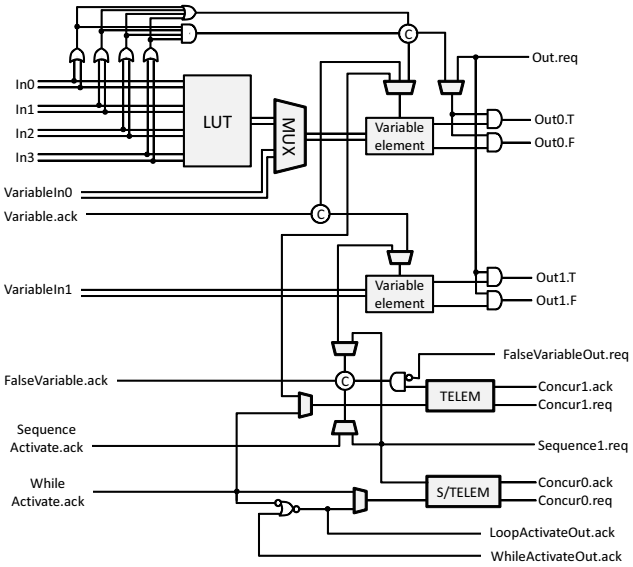


Fig. 10: Structure of an LUT, Variable module and While module.

Table 2: Handshake components and its corresponding modules.

Module	Handshake component
Variable	Variable, FalseVariable, ActiveEagerFalseVariable
While	While, Loop, Sequence
Call	Call, CallMux, ContinuePush
Case	Case, CaseFetch, PassivatorPush, SynchPush, CallDemux, DecisionWait
Encode	Encode

Table 3: Other handshake components and its corresponding resources.

Module	Handshake component
LUT, Variable	BinaryFunc, BinaryFuncConstR, UnaryFunc
Variable, While	Concur
Programmable interconnect resources	Adapt, Combine, CombineEqual, Constant, Continue, Fetch, Fork, ForkPush, Halt, HaltPush, Slice, Split, SplitEqual, Synch, SynchPull, WireFork

Table 4: Comparison of cells of the conventional architecture and the proposed architecture.

	Conventional architecture	Proposed architecture
Number of transistors	1344	3372

Table 5: Comparison of cells that implement Case handshake components.

	Conventional architecture	Proposed architecture
Number of transistors	5376	3372

4. Conclusions

This paper presented an architecture of an asynchronous FPGA for handshake-component-based design. The proposed FPGA architecture implements handshake components efficiently. Therefore, the proposed architecture is suitable for the synthesis tools that generate netlists consist of handshake components, such as Balsa. As a future work, we are evaluating the proposed FPGA architecture on some practical benchmarks.

Acknowledgment

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with STARC, e-Shuttle, Inc., Fujitsu Ltd., Cadence Design Systems Inc. and Synopsys Inc.

References

[1] V. George H. Zhang. and J. Rabaey, "The design of a low energy FPGA," in *Proceedings of 1999 International Symposium on Low Power Electronics and Design*, California, USA, Aug 1999, pp. 188–193.

[2] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1376–1392, 2004.

[3] R. Manohar, "Reconfigurable Asynchronous Logic," in *Proceedings of IEEE Custom Integrated Circuits Conference*, Sep. 2006, pp. 13–20.

[4] M. Hariyama, S. Ishihara, and M. Kameyama, "Evaluation of a Field-Programmable VLSI Based on an Asynchronous Bit- Serial Architecture," *IEICE Trans. Electron*, vol. E91-C, no. 9, pp. 1419–1426, 2008.

[5] M. Hariyama, S. Ishihara, , and M. Kameyama, "A Low-Power Field-Programmable VLSI Based on a Fine-Grained Power-Gating Scheme," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Knoxville(USA), Aug 2008, pp. 430–433.

[6] S. Ishihara, Y. Komatsu, M. Hariyama and M. Kameyama, "An Asynchronous Field-Programmable VLSI Using LEDR/4-Phase-Dual-Rail Protocol Converters," in *Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas(USA), Jul 2009, pp. 145–150.

[7] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Logic Synthesis of Asynchronous Controllers and Interfaces, Springer, 2002.

[9] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, " The VLSI-programming language Tangram and its translation into handshake circuits, " in *Proc. EDAC*, 1991, pp. 384—389.

[10] A. Bardsley, "Implementing Balsa Handshake Circuits," Ph.D. thesis, Dept. of Computer Science, University of Manchester, 2000.

[11] Q. Zhang, G. Theodoropoulos, "Modelling SAMIPS: A Synthesisable Asynchronous MIPS Processor," *Proc. of the 37th Annual Simulation Symposium*, pp. 205-212, 2004

SESSION

SHORT PAPERS and POSTERS

Chair(s)

Dr. Toomas P. Plaks
UK

An Asynchronous FPGA Based on Dual/Single-Rail Hybrid Architecture

ERSA'12 Short Paper

Zhengfan Xia, Shota Ishihara, Masanori Hariyama, and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {xiazhengfan, ishihara, hariyama, kameyama}ecei.tohoku.ac.jp

Abstract— *This paper presents a novel asynchronous FPGA architecture that dual-rail encoding and single-rail encoding are respectively configured in the critical path and non-critical paths. A dual/single-rail convertible LB (Logic Block) is designed to realize such configuration. 4-phase dual-rail protocol is applied to realize asynchronous handshake in only the critical path, which saves handshake overhead. The non-critical paths use single-rail logic which has small logic overhead. Compared to conventional dual-rail design, the proposed FPGA shows its advantages of low power and high logic density when the percentage of on-critical-path logic cells in the whole mapping circuit are less than 80%.*

Keywords: FPGA, dual-rail encoding, single-rail encoding, logic density

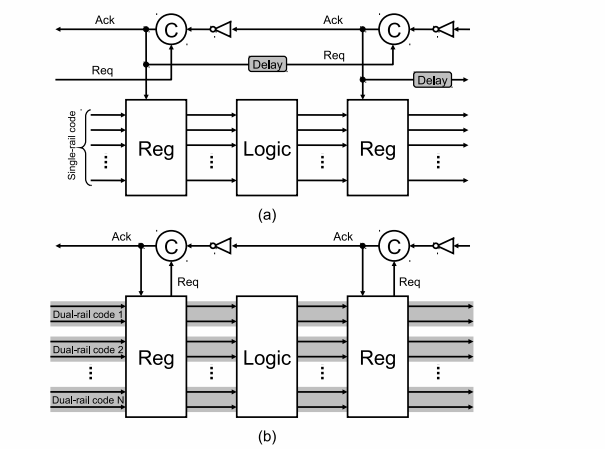


Fig. 1: Asynchronous pipelines. (a) A diagram of bundled-data pipeline. (b) A diagram of dual-rail pipeline.

1. Introduction

Field-programmable gate arrays (FPGAs) are wildly used to implement special-purpose processors. FPGAs are cost-effective for small-lot production because functions and interconnections of logic resources can be directly programmed by the user. Despite their design cost advantages, FPGAs impose large power consumption and silicon area overheads compared to the custom-desinged alternatives [1]. These imposed overheads increase packaging costs and limit the usage of FPGAs in portable devices. Compared to the custom VLSIs, FPGA has an enormous number of registers which need to be drived by a complex clock distribution network. The high power consumption of the clock distribution network is a serious problem. To cut this clock power, some asynchronous FPGAs have been proposed [2], [3], [4], [5]. Asynchronous FPGAs are mainly seperated into two types.

- Bundled-data, or single-rail, asynchronous FPGAs [2], [3]
- Dual-rail asynchronous FPGAs [4], [5]

Bundled-data asynchronous FPGAs have been based largely on programmable clocked circuits. They are limited to low-throughput because their asynchronous pipeline stages use bundled-data pipelines (Fig.1(a)) that rely on interconnects controlled by delay lines (e.g. [2]). For example, a fabricated asynchronous FPGA chip using bundled-data pipelines operated at a maximum of 20 MHz in a 0.35μm CMOS process [3]. On the other hand, dual-rail asynchronous FPGAs use fully dual-rail delay-insensitive interconnects, and are based on high-speed asynchronous pipelined circuits (Fig.1(b)). They can realize much higher throughput. For example, a dual-rail asynchronous FPGA architecture [4] is reported that operate at up to 400MHz in a TSMC 0.25μm CMOS process. However, in spite of the high-throughput of dual-rail asynchronous FPGAs, the delay-insensitive design is realized at the expense of dual-rail logic overhead which deteriorates the power consumption and logic density of FPGA.

This paper proposes an asynchronous FPGA based on a dual/single-rail hybrid architecture. A dual/single-rail convertible LB is designed, which can be configured to implement a dual-rail logic or two single-rail logics. When a circuit is mapped to the proposed FPGA, the convertible LBs would be selectively configured into dual-rail logic or single-rail logic by deciding whether the LBs are located on the critical path or on the non-critical paths. The dual-

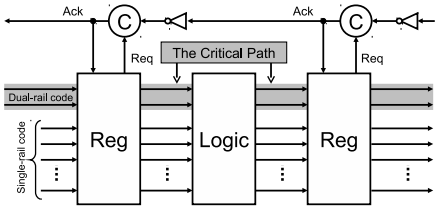


Fig. 2: A diagram of dual/single-rail hybrid asynchronous pipeline.

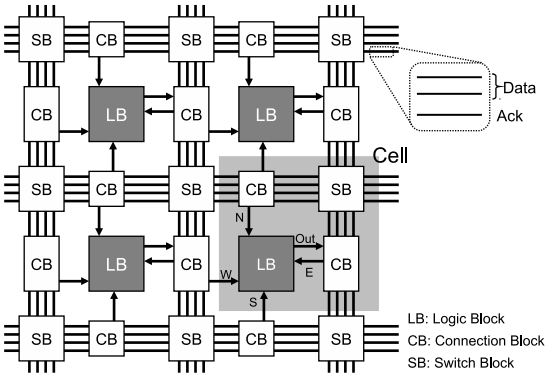


Fig. 3: Overall architecture.

rail logic on the critical path is used to transfer dual-rail data and handshake signal. Because the handshake circuit only works in the critical path, the handshake overhead is greatly reduced. Moreover, when the convertible LB is configured into two single-rail logics in non-critical paths, it has higher logic density. In addition, the problem of critical path mapping can be solved by using the currently existing routing resources through a mapping algorithm.

2. Architecture

2.1 Overall architecture

Fig.3 shows the overall architecture of the proposed asynchronous FPGA. The FPGA consists of a mesh-connected cellular array which is same as the conventional FPGAs. In order to maintain the identical of each cell in the dual/single-rail hybrid architecture, a dual/single-rail convertible LB is designed. Fig.4 (a) shows the concept of the convertible LB. A convertible LB consists of two dual/single-rail convertible LUTs. According to $Mode\_sel$ signal, the convertible LB can works in dual/single-rail hybrid mode or single-rail mode. Dual/Single-rail hybrid mode LB merges the two LUTs to output a dual-rail data. It is named as dual/single-rail hybrid mode because it has a mixture of a dual-rail input and many single-rail inputs. On the other hand, single-rail mode LB seperates the two LUTs to be configured into two independent single-rail logics. Fig.4 (b) shows the mapping concept of 3-input convertible LBs. The LBs on the critical path are configured into hybrid mode LBs. Each hybrid mode

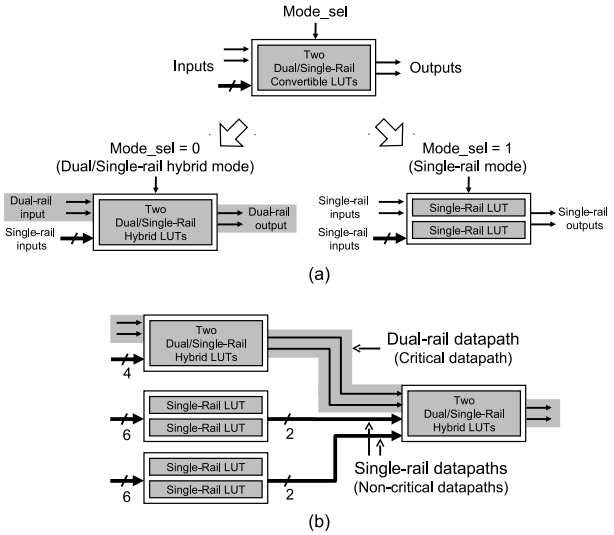


Fig. 4: Design concepts. (a) Concept of dual/single-rail convertible LB. (b) Mapping concept of 3-input convertible LBs.

Table 1: Code table of the 4-phase dual-rail encoding

	Codeword (w_t, w_f)
Data 0	(0, 1)
Data 1	(1, 0)
Spacer	(0, 0)
Not used	(1, 1)

LB produces a dual-rail data and encoded handshake signal. On the other hand, the LBs on the non-critical paths are configured into single-rail mode LBs. Each signel-rail LB prodcues two signel-rail data. As a result, the whole mapping has small handshake overheand and high logic density.

2.2 Asynchronous protocols

As we mentioned in introduction, there are mainly two types of asynchronous FPGAs. These asynchronous FPGAs are respectively based on bundled-data, or single-ral, protocols or dual-rail protocols. Furthermore, these protocols are detailly seperated into 4-phase protocol and 2-phase protocol [6]. Because our proposed design is based on 4-phase protocol, we just focus on 4-phase dual-rail protocol in this paper.

Table1 shows code table of the 4-phase dual-rail encoding. 4-phase dual-rail encoding encodes a bit onto two wires, (w_t, w_f). The data value 0 is encoded as (0,1) and 1 is encoded as (1,0); the spacer is encoded as (0,0). Fig.5 shows an example of the 4-phase dual-rail encoding. In the data transfer, every data is separated by a spacer. The timing information is encoded in dual-rail data.

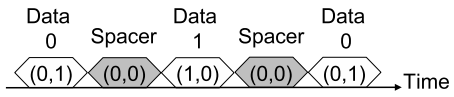


Fig. 5: An example of the 4-phase dual-rail encoding.

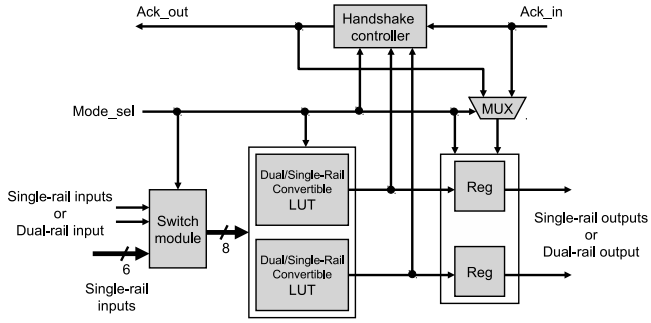


Fig. 6: Structure of dual/single-rail convertible logic block.

2.3 Structure of logic block

Fig.6 shows the structure of dual/single-rail convertible logic block. There are a swtich module, two dual/single-rail convertible LUTs, two registers and a handshake controller. Depending on the working modes, the switch module controls the connetion between inputs and LUTs to output a dual-rail data or two single-rail data. The handshake controller would be disabled when the convertible LB works in single-rail mode to save power.

In the convertible LB design, LUT design is the most important part. Conventional dual-rail asynchronous FPGAs prefer using dynamic LUT circuit because of its advantages such as high speed, low power and hazard-free. In spite of these advantages, dyamaic LUT circuit has a timing constraint that all inputs should become valid before circuit enter evaluation phase. This timing constraint can be easily satisfied with the timing information encoded in dual-rail data. However, the non-critical data paths in our design use single-rail encoding which losses the timing information. It would cause serious problems with using dynamic LUT circuit. Therefore, a static LUT circuit is chosen to be used in our design.

Fig.7 shows a 3-input dual/single-rail convertible LUT. According to $mode\_sel$ signal, the convertible LUT works in dual/single-rail hybrid mode or single-rail mode. When the convertible LUT works in single-rail mode ($mode\_sel = 1$), the inputs are $in0$, $in1$ and $in2\_t$. The convertible LUT works same as the conventional LUT in synchronous FPGA design. When it works in dual/single-rail hybrid mode ($mode\_sel = 0$), the inputs are $in0$, $in1$ and ($in2\_f, in2\_t$). Two convertible LUTs are respectively configured to true and complement outputs. In addition, the hybrid mode LUT has a timing constraint that the dual-rail data should be the last to arrive at the inputs. Otherwise, hazards would occur.

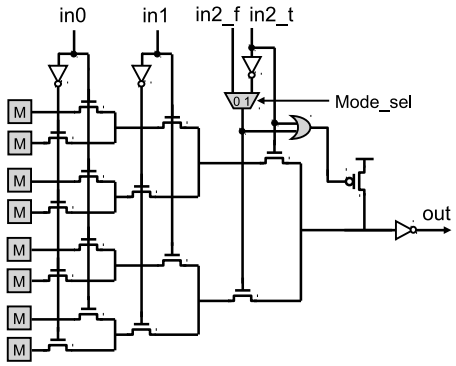


Fig. 7: A 3-input dual/single-rail convertible LUT.

2.4 Critical path mapping

The mapping of a robust critical path is important for the proposed architecture. A robust critical path guarantees not only the satisfaction of timing constraint in the convertible LUT but also the correctness of data flow in pipeline. Fortunately, a robust critical path can be easily mapped by using a placement and routing algorithm. For conventional FPGAs, many placement and routing algorithms have been proposed to minimize the delay time of the critical path, such as [7]. These algorithms can also be adversely used to enhance the critical path by enlarging the delay time of the critical path. In addition, the mapping problems of placement, routing and partitioning in asynchronous FPGAs are simplified without the requirement of meeting the global clock constraint [8]. It offers us more freedom to choose a appropriate mapping algorithm to get a robust critical path.

In FPGA mapping, a path delay can be estimated by using the information of LB delay, CB (Connection Block) delay, SB (Swicth Block) delay and their process variations. Our mapping algorithm is simply explained as follows:

1) Analyze a circuit to get the critical path before it is mapped to the FPGA.
2) Map the circuit to FPGA with the lowest priority of the analyzed critical path.
3) Calculate N_{LB}, N_{CB} and N_{SB} in every data paths. N is the numbers of LB, CB or SB.
4) Estimate delay time of each path by using the equation of $(D_{LB}+V_{LB})N_{LB}+(D_{CB}+V_{CB})N_{CB}+(D_{SB}+V_{SB})N_{SB}$. D is the delay time and V is process variations.
5) Remap the critical path longer by increasing N_{LB}, N_{CB} or N_{SB}, if the critical path has not the largest delay time.
6) Repeat step 3), 4) and 5) until geting a reliable critical path.

Table 2: Comparison of FPGA cells of dual-rail design and the hybrid design

Asynchronous FPGA Cell		Proposed Design		Dual-rail Design
		Dual/Single-rail hybrid mode	Single-rail mode	
Delay (ps)		385	375	405
Energy (fJ)	Per-cell	253.5	172	238
	Per-logic	253.5 (295%)	86 (100%)	238 (277%)
Transistor Count	Per-cell	1470		1304
	Per-logic	1470 (200%)	735 (100%)	1304 (177%)

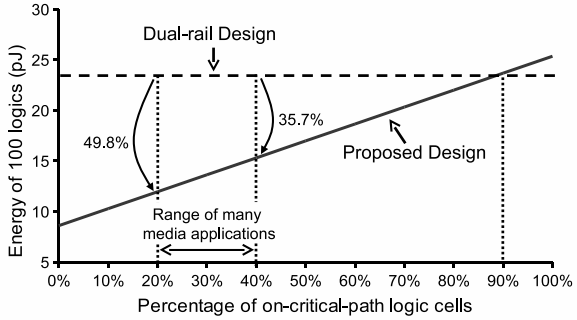


Fig. 8: The relationship between energy consumption and percentage of on-critical-path logic cells in 100 logics.

3. Evaluation

Table 2 shows comparison of FPGA cells of dual-rail design [4] and the proposed hybrid design. The results are simulated by HSPICE in a 65nm CMOS process. A FPGA cell consists of one LB, five CBs and one SB, which is shown in Fig.3. The results show that single-rail mode of a proposed cell saves lots of power and transistor count to realize a logic function. Fig.8 shows the relationship between energy consumption and pcercentage of on-critical-path logic cells in 100 logics. Fig.9 shows the relationship between transistor count and pcercentage of on-critical-path logic cells in 100 logics. Multiple supply voltages researchs [9], [10] show that the percentage of on-critical-path logic cells in many media applications are between 20% to 40%. If 20% to 40% of 100 logics use hybrid mode logic, Fig.8 shows that the hybrid design saves 35.7% to 49.8% energy compared to conventional dual-rail design. At the same time, Fig.9 shows that the hybrid design saves 21.1% to 32.4% transistors.

4. Conclusion

This paper introduced an asynchronous FPGA based on dual/single-rail hybrid architecture. When a circuit is mapped to the proposed FPGA, dual/single-rail hybrid logic is configured in critical path to transfer dual-rail data and realize handshake process. On the other hand, single-rail logic is configured in non-critical paths, which has small logic overhead. As a result, the proposed asynchronus FPGA has small handshake overhead, low power consumption and high logic density. A mapping algrothm for the critical path is also simply introduced, which will be further developed in our future work.

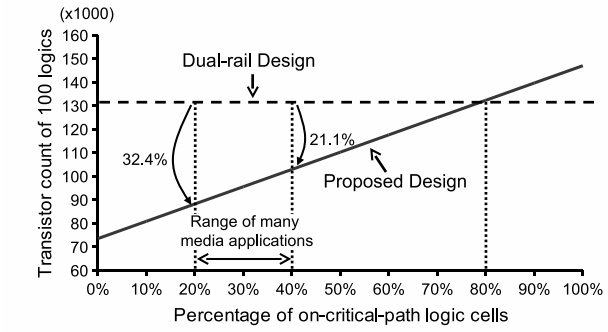


Fig. 9: The relationship between transistor count and percentage of on-critical-path logic cells in 100 logics.

Acknowledgment

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with STARC, Fujitsu Limited, Matsushita Electric Industrial Company Limited, NEC Electronics Corporation, Renesas Technology Corporation, Toshiba Corporation, Cadence Design Systems Inc. and Synopsys Inc.

References

[1] H. Z. V. George and J. Rabaey, "The Design of a Low Energy FPGA," in Proceedings of 1999 International Symposium on Low Power Electronics and Design, August 1999, pp. 188-193.

[2] R. Payne, "Asynchronous FPGA Architectures," IEE Proceedings, Computers and Digital Techniques, Volume 143, Issue 5, 1996, pp. 282-286.

[3] R. Konishi, H. Ito, H. Nakada, A. Nagoya, K. Oguri, N. Imlig, T. Shiozawa, M. Inamori, and K. Nagami, "PCA-1: A Fully Asynchronous Self-Reconfigurable LSI," Proc. Int'l Symp. Asynchronous Circuits and Systems, 2001.

[4] J. Teifel and R. Manohar, "An asynchronous dataflow FGPA architecture," IEEE Transactions on Computers, Vol. 53, No. 11, 2004, pp. 1376-1392.

[5] M. Hariyama, S. Ishihara, and M. Kameyama, "A Low-Power Field-Programmable VLSI Based on an Asynchronous Bit-Serial Architecture," IEICE Transactions on Electronics, Vol. E91-C, No. 9, 2008, pp. 1419-1426.

[6] J. Spars and S. Furber, "Principles of Asynchronous Circuit Design: A Systems Perspective," Kluwer Academic Publishers, 2001.

[7] Alexander Marquart, Vaughn Betz, and Jonathan Rose, "Timing Driven Placement for FPGAs," Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2000.

[8] R. Payne, "Self-timed FPGA systems," 5th International workshop on Field programmable logic and applications, LNCS 975, 1995, pp. 21-35.

[9] T. Kuroda and M. Hamada, "Low-Power CMOS Digital Design with Dual Embedded Adaptive Power Supplies," IEEE Journal of Solid-State Circuits, Vol. 35, No. 4, 2000, pp. 652-655.

[10] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanzawa, M. Ichida and K. Nogami, "Automated Low-Power Technique Exploiting Multiple Supply Voltages Applied to a Media Processor," IEEE Journal of Solid-State Circuits, Vol. 33, Issue 3, 1998, pp.463-472.

IPP Watermarking-Based Protection of Embedded Cores on FPL Devices

L. Parrilla, E. Castillo, A. García
Dept. Electronics and Computer Technology
University of Granada
Granada, Spain
lparrilla@ditec.ugr.es

G. Botella
Dept. of Computer Architecture and Automation
Complutense University of Madrid.
Madrid, Spain

Abstract—**This paper proposes a system to extract digital signatures from watermark-protected cores that are embedded into complex systems, without direct access to the in/out pins. Two subsystems are developed: the first one detects the signature extraction sequence using the reset line; and the second one—the signature extractor subsystem—directs the recovery of the digital signature by measuring variations in the power consumption of the overall system. Experimental results show the viability of the system proposed, with very low impact on the area, while providing high robustness and security.**

Keywords-embedded systems, IP cores, intellectual property protection, watermarking, FPGA

I. INTRODUCTION

The intellectual property protection (IPP) of cores continues to exist as an important subject in the field of modern digital design techniques [1]. Reuse-based design requires an easy, robust, and secure protection framework for claiming authorship rights inside the fast-emerging market of IP cores. For this reason, multiple solutions have been proposed [1-7], those based on watermarking techniques being especially affordable and adequate for the IPP of reusable modules [2-4]. Most of these modules (or cores) are used as constructive elements for the implementation of complex systems. In this situation, the cores are embedded into the system under the design, making access difficult to the protection mechanisms implemented for rights verification purposes. Therefore, the protection of cores embedded into complex systems represents a challenge, due to this lack of control over the I/O pins of the core.

In order to approach the protection of embedded cores, a well-checked and robust procedure for IP core protection must be adapted to support the hard restrictions access to a core that is under protection. The IPP@HDL [7] procedure has been selected because of the following characteristics: (1) It can be applied to any digital system; (2) The protection is performed at the high-level design stages; and (3) the method for spreading the signature into the combinational logic makes removal of the watermark very difficult. Otherwise, IPP@HDL presumes, in order to activate the signature extraction process and the signature recovery process, that the I/O pins of the core under protection are very accessible. Therefore, the techniques used in IPP@HDL to activate the signature extraction, and the extraction process itself, must be significantly modified to achieve the protection of an isolated core embedded in a complex system.

In embedded cores, the only pin accessible a priori, is the reset line. Since it is an input, it is the candidate to be used for receiving the signature extraction sequence (SES) and to activate the extraction process. There are no output pins accessible, so the extraction of the signature must be performed introducing modifications on the overall parameters of the system. One parameter that can be affected for the embedded core is the power consumption of the system, which is also a parameter that is easy to measure.

In this sense, the present paper outlines a new technique for introducing the SES through the reset line, which activates the extraction process. For the extraction of the signature, a procedure based on the variation of power consumption is proposed. The resulting embedded core protection procedure is named e-IPP@HDL and is described alongside the article.

II. PROTECTION OF IP CORES: IPP@HDL FRAMEWORK

Among the various procedures for IP core protection, those based on watermarking techniques are the most secure and robust. The underlying idea is to introduce a digital signature in the core under protection, which remains hidden for the user, but contains author rights information [2]. The watermark must be difficult to remove and cannot interfere with the normal functioning of the system.

Several watermarking techniques for the IPP of cores have been proposed in the literature, at different design levels and using various methods to hide the digital signature [4-7]. The most complete and general is IPP@HDL [7], which spreads the signature over the combinational logic, becoming part of the design. With this technique, the removal of the signature makes the system fail, which makes this technique very tamper-resistant. IPP@HDL also provides a procedure to extract the signature, based on the introduction of a Signature Extraction Sequence (SES), which puts the system in "extraction mode," performing the extraction through the output pins. The SES must be safe enough to cause the system to enter "extraction mode" accidentally or by means of an attack.

Once the core has been watermarked, the digital signature can be recovered introducing the SES and observing the data output pins. If the core is embedded and the output pins are not

directly accessible, it will be necessary to use other alternatives. In this paper, we propose to produce detectable modifications on the power consumption of the overall system to perform the signature extraction.

III. ACTIVATION OF SIGNATURE EXTRACTION ON EMBEDDED CORES

We assume that a core has been protected spreading a digital signature by means of the IPP@HDL techniques. Thus, the signature is hosted in the combinational logic of the core in the appropriate signature locations (SLs), and a secure SES has been selected. The next step is to create the additional logic for the signature extraction. If the protected core will be used as part of a complex system, the data input pins are not practicable, as assumed in [7], and the only option is to use the reset line to detect the SES and activate the extraction process.

In this scenario, the SES must be introduced as serial bit stream through the reset line. As we have only one line, this will have to be used to synchronize the transmitter and the receiver and to transmit the bit stream corresponding to the SES. Also, in order for the SES detector to operate, a clock signal is required, and this internal clock must be synchronized with the external source providing the SES. With these considerations, the block diagram of the SES detector operating over the reset line is presented in Figure 1.

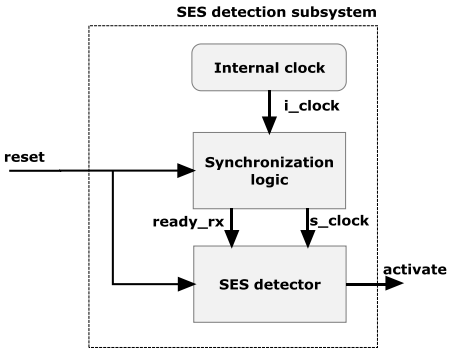


Figure 1. Block diagram for the SES detection subsystem

The developed system contains three blocks, the reset line input, and the *activate* output that signals the extraction subsystem to initiate the extraction process. Below, the function of each one of these blocks is described.

A. Internal clock

The SES detection subsystem requires a clock signal to operate. The general clock signal of the core cannot be accessible; therefore, it is necessary to internally generate a signal clock. This operation is performed using ring buffers, which presents some difficulties because of the inherent instability of the design and simulation software tools. The time interval the clock generates depends on the specific device, so it is not difficult to make experimental characterizations of the devices when the cores are to be implemented on FPGAs. This block only possesses an output, named *i_clock,* which drives the internal auto-generated signal clock to the other blocks.

B. Synchronization logic

This block analyzes the reset line, which waits until an "initiating transmitting" pattern is detected. When this pattern arrives, the logic extracts the period of the clock signal which uses the external source in order to synthesize an external clock imitation to synchronize the reception of the SES bits. When the clock has been synthesized, the *ready_rx* signal indicates that all is ready to receive the SES bit stream. The block has the *reset* and *i_clock* signals as inputs and generates the *s_clock* (carrying the synthesized clock signal) and *ready_rx* in order to synchronize the reception of the SES stream.

C. SES detector

Using the clock signal synthesized by the "synchronization logic" block, the SES detector can read the SES bits transmitted by the external source. The SES received is compared with the SES expected, in order to assign the *activate* output to the '1' value and initiate the extraction process. IPP@HDL proposes an alternative to using LFSRs to generate safe SESs, and this option is preferred in e-IPP@HDL in order to simplify the SES detector. Figure 2 shows the block diagram of the SES detector.

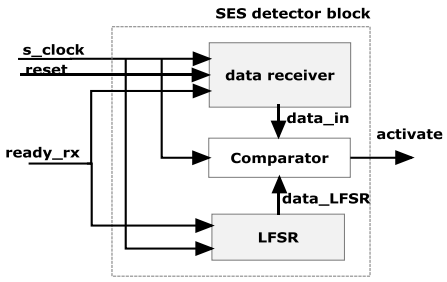


Figure 2. Diagram of the SES detector block

The elements integrating the SES detection subsystem, the timing definitions, and the requirements for performing the SES detection are described below.

D. Timing definitions and requeriments for detecting the SES

Let's define $t_{internal}$ as the time interval of the internal signal clock and $t_{external}$ as the time interval of the external source clock. If the external clock must be synthesized from the internal one, then:

$$t_{external} = \mathrm{n}\ t_{internal} \quad (1),$$

with n being an integer. To minimize round errors, synchronization problems, and thus guarantee that it is possible to synthesize the external clock,

$$t_{external} >> t_{internal} \quad (2),$$

thus, slow clock signals will be used to transmit SES.

Otherwise, to signal that a SES is going to be transmitted, a large pulse will be sent using the reset line. This pulse must be longer than the overall SES to avoid confusion (e.g. a SES "111111…" could be interpreted like an initiating transmitting pulse). If a maximum of a 512 bit-length is assumed for the SES (and it is secure enough to avoid accidental activations and attacks):

$$t_{pulse} > 512\ t_{external} \quad (3),$$

The synchronization logic (SYNL, Figure 3) will expect a pulse t_{init} long, being:

$$t_{init} > t_{pulse} \quad (4),$$

which guarantees that SYNL detects the pulse, providing a safety margin defined as:

$$t_{safe} = t_{pulse} - t_{init} = n_{safe}\, t_{internal} \quad (5),$$

A practical value for t_{init} can be 500ms, and 550 ms for t_{pulse}, avoiding any problem related with the SES pulses ($t_{external}$ could be ≈ 1ms), and forcing the synchronization logic to detect a 500ms pulse in the reset line. Later, the SYNL will wait for the next pulse, which provides the external clock time interval, $t_{external}$. With this information, the SYNL can reconstruct the external clock in order to synchronize the reception, and to signal (using *ready_rx*) the SES detector to receive the SES.

Table I resumes the values for the timing parameters analyzed, assuming $t_{internal}$ ≈ 30ns (an estimate from the datasheet and simulation), and Figure 5 shows the data that needs to be put on the reset line in order to introduce the SES into an e-IPP@HDL protected core.

TABLE I. TIMING REQUERIMENTS FOR SES SYNCRONIZATION

Timing Parameters recommended			
Parameter	***Min***	***Max.***	***e-IPP@HDL***
t_pulse	200ms	2s	550ms
t_init	190ms	1.5s	500ms
t_safe	10ms	500ms	40ms
n_safe	300	3e6	8e5
t_external	100us	100ms	1ms

On the other hand, the SES detector module will wait for the 1.1 s pulse, will signal the SYNL module to reconstruct the external signal clock, and then the SYNL will trigger the LFSR and the comparator block synchronized with the *s_clock* to verify the SES. Figure 3 shows the corresponding timing diagram. The first row shows the *s_clock*; the second row shows the *ready_rx*, which triggers the LFSR and the comparator; and the third row shows the expected SES, as generated by the LFSR.

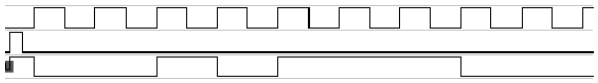


Figure 3. Timing for generating the expected SES for comparison purposes

E. Implementation issues

The implementation of the proposed system needs to solve some questions. First, it's necessary to make an estimation of the internal clock period in order to detect the 500ms pulse marking the beginning of the transmitting SES. This period depends on the physical device used for implementation, but it is not critical while meeting conditions (2), (3), and (4) and whether or not the t_safe is wide enough (Table I). With $t_{internal}$ ≈30ns and t_{safe} = 40ms, we have n_{safe} = 8e5. Depending on the characteristics of the devices, t_{safe} can be increased as needed.

Moreover, the different order of magnitude of the times defined, with the instabilities introduced by the oscillation structures used in the clock generator, makes it difficult to simulate. The design must be carefully partitioned to be simulated and checked prior to its physical implementation.

IV. DESIGN EXAMPLE FOR THE SES DETECTOR

With the aim of test being the functioning of the proposed structure for the SES detector for embedded cores, a design example has been carried out. The SES detector for e-IPP@HDL has been implemented in a SPARTAN 3A evaluation kit by AVNET [8], which has a Xilinx XC3S400A-4FTG256C Spartan-3A FPGA [9]. The tool used was ISE 10.1.

Table II shows the implementation results for the internal clock generator. The area occupied is negligible, and Figure 4 shows the experimental waveform taken with an Agilent 16901 logic analysis system.

TABLE II. INTERNAL CLOCK GENERATOR IMPLEMENTATION

Device utilization summary			
Logic utilization	***Used***	***Available***	***Utilization***
Number of Slice Flips Flops	4	7168	<1%
Numer of Occupied Slices	4	3584	<1%
Total Number of 4 input LUTS	5	7168	<1%

The estimation from gate delays in post-place and route simulation for the clock time interval is ≈ 30ns and is measured with the logic analyzer $t_{internal}$=24ns.

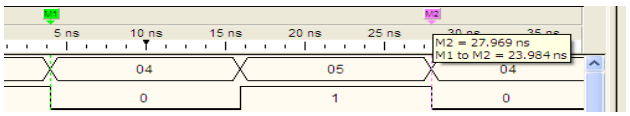


Figure 4. Waveform for the generated internal clock

Table III shows the implementation data for the entire SYNL module. A row with the number of slice registers is included because ISE has used them for counter synthesis. The overall occupation (the total number of the 4 input LUTS) rounds the 1%. Figure 5 shows, in the third row, the 1.1s pulse to signal the beginning of the transmitted SES, followed by the 1ms synchronization pulse ($t_{external}$ =1ms has been used). The fifth row in Figure 5 shows the synthesized clock by SYNL, which also has a 1ms time interval and is clearly synchronized with the external source, allowing synchronization with the SES transmitting.

TABLE III. SYNL IMPLEMENTATION RESULTS

Device utilization summary			
Logic utilization	***Used***	***Available***	***Utilization***
Number of Slice registers	65	7168	1%
Number of occupied Slices	67	3586	1%
Total Number of 4 input LUTS	102	7168	1%

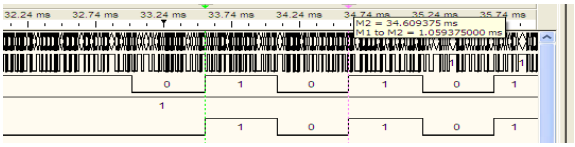


Figure 5. Waveform for the synthesized clock

V. EXTRACTION OF SIGNATURES IN EMBEDDED SYSTEMS

The SES detector developed in Sections III and IV can detect the introduction of a valid SES through the reset line and send a signal to the extraction logic in order to perform the digital signature extraction. If the core under protection is embedded, there are no output pins accessible, and the signature must then be recovered, changing other parameters affecting the entire system.

In [10], a procedure for extracting digital signatures by producing variations on the power consumption is proposed. The method presented has some drawbacks, the main one being the complex signal processing needed to ensure the correct recovery of the signature. The aliasing problems have their origin in the use of the main clock signal of the system that feeds the ring oscillators, which constitute the base of the structures for the increase in power-consumption. In e-IPP@HDL, buffer oscillator structures are proposed, without using any clock signal, avoiding aliasing problems. The idea is to construct a signal generator operating with a low-frequency clock obtained from the internal clock developed in Section IV. As shown in Figure 6, the hosted digital signature is recovered and the high consumption module (HCM) is driven, producing appreciable variations in the power-consumption of the entire system.

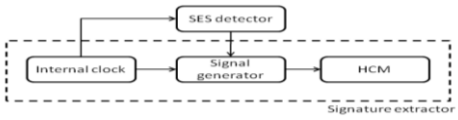


Figure 6. Block diagram of the signature extractor

The main block of the signature extractor is the HCM. It must have a low area and high power-consumption, when required. In order to test its viability it was put into a medium-size core with high activity, like a clone of a Z80 µP [11], a T80 [12] operating at 40Mhz, implemented in a Virtex 5 xc5vlx30-1ff676 Xilinx device [13]. For power consumption measures, a Virtex-5 LX prototype platform [14] with an Agilent N6705A DC power analyzer [15] was used. While testing the HCM, the µP was forced to high activity.

Table IV presents the magnitude of current pulses when the HCM went into high power mode (IH) compared with normal operation mode (IL). The time interval used was 10ms.

It can be concluded that while the minimum size HCM produced a significant increase in the power line current, using a 4-block HCM raised the difference to 5mA, producing more reliable output signals. Bigger HCMs do not affect the size of current pulses and consume more area. Thus, 4 block-size HCMs are committed for implementing the signature extractor, making the current variations in the power line detectable.

TABLE IV. CURRENT CONSUMPTION FOR DIFFERENT SIZES OF HCM

#n blocks	Current consumption variations		
	IH (mA)	*IL (mA)*	*Diff.(mA)*
2	252,4	249,6	2,8
4	250,7	246,1	4,6
8	254	249,2	4,8

VI. CONCLUSIONS

A framework for the protection of cores embedded in complex systems has been shown. A digital signature containing the author rights is hosted into the core to be protected using the IPP@HDL general procedure, and extending the procedure to aim in the extraction of the signature without direct access to the IN/OUT pins of the core under protection. This extension, e-IPP@HDL, activates the extraction of the digital signature using only the reset line and recovers the contents of this signature by means of the DC consumption analysis of the overall system. The results shown have a low area of impact with high performance and reliability.

ACKNOWLEDGMENT

This work was partially funded Spanish Project TEC2007-68074-C02-01/MIC. CAD tools and supporting materials were provided by Xilinx, Inc. under University Program.

REFERENCES

[1] M. Keating and P. Bricaud, Reuse Methodology Manual for System-on-a-Chip Designs. Kluwer Academic Publishers, 1998.

[2] I. Cox, M. Miller, and J. Bloom, Digital Watermarking: Principles & Practice. Morgan Kaufmann, 2001.

[3] E. Charbon and I. Torunoglu, "Watermarking techniques for electronic circuit designs," LNCS, vol. 2613, pp. 147–169, 2003.

[4] B. Kahng, J. Lach,W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in Proc. of the Design Automation Conference, 1998, pp. 776–781.

[5] F. Houshanfar, I. Hong, and M. Potkonjak, "Behavioral synthesis techniques for intellectual property protection," ACM Trans.on Design Automat. of Electronic Systems, vol. 10, no. 3, pp. 523–545, July 2005.

[6] E. Castillo, U. Meyer-Baese, A. García, L. Parrilla, and A. Lloris, "IPP@HDL: Efficient intellectual property protection scheme for IP cores," IEEE Trans. VLSI Syst., vol. 15, no. 5, pp. 578–591, May 2007.

[7] E. Castillo, L. Parrilla, A. García, U. Meyer-Baese, A. Lloris, G. Botella, "Automated signature insertion in combinational logic patterns," Proc. of 4th Southern Conference on Programmable Logic, 2008, pp.183–186.

[8] Avnet Inc., Spartan-3A evaluation kit. Available: http://www.xilinx.com/products/devkits/aes_sp3a_eval400_avnet.htm

[9] Xilinx, Inc., Spartan 3 family User Guide. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[10] Ziener, D, Teich, J., "Power Signature Watermarking of IP Cores for FPGAs" Journal of Signal Processing Systemas, 51-1, pp 123-136, 2008.

[11] Gaonkar, Ramesh S: *The Z80 microprocessor: architecture, interfacing, programming, and design*, 3rd ed., Prentice-Hall, 2001.

[12] Warner, D. T80 cpu. http://www.opencores.org/?do=project&who=t80

[13] Xilinx, Inc., Virtex-5 User Guide. Available: http://www.xilinx.com/support/documentation/userguides/ug190.pdf

[14] Xilinx, Inc., Virtex-5 LX FPGA Prototype Platform. Available: http://www.xilinx.com/support/documentation/boardsandkits/ug222.pdf

[15] Agilent Technologies, N6705A DC Power Analyzer User's Guide. Available: http://cp.literature.agilent.com/litweb/pdf/N6705-90001.pdf

Low-Power Heterogeneous Platform for High Performance Computing and Its Application to 2D-FDTD Computation

ERSA'12 Short Paper

Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {hasitha, takei, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract— *Heterogeneous processing with CPUs and low-power accelerators attract many attentions since they can achieve power-efficient computing. However, the potential of using low-power accelerators such as FPGAs in high-performance computing to reduce the power consumption is rarely exploited. In this paper, we propose a CPU/FPGA heterogeneous platform to implement the finite-difference time-domain (FDTD) computation. According to the experimental results, we found that 95% of the computation can be done in FPGA using 32bit fixed point arithmetic without suffering a major precision loss. According to our estimation, we can achieve the same performance of CPU/GPU computing with 10 times less power consumption by using the proposed low-power heterogeneous platform.*

Keywords: Heterogeneous processing, high-performance computing, supercomputing, FDTD

1. Introduction

Applications used in low-power embedded processing to high performance computing have different tasks such as data-intensive tasks and control-intensive tasks. Heterogeneous processing is proposed to execute such different applications power-efficiently. In heterogeneous processing, different processors such as CPUs and accelerators are used as shown in Fig.1. If the tasks of an application are correctly allocated to the most suitable processors, all the processors work together to increase the overall performance. An example of a heterogeneous high-performance computing platform (HPC) is "Tianhe-1A" [1] which has Intel X5670 CPUs and NVDIA GPUs (graphic processing unit).

The main problem in commercially available HPC platforms such as [1] is the large power consumption. High-end CPUs and GPUs have a power consumption of more than 100W and 300W respectively. A very promising solution to reduce the power consumption of HPC platforms is to use FPGAs. Recent FPGAs have over 300,000 logic cells, over 5MBits of memory and high-speed data transfer interfaces. Therefore, a single FPGA is large enough to hold hundreds of processing elements (PEs). Although FPGAs have a lot of resources and consume a very small power, they are rarely used in the HPC platforms. One main reason for not using FPGAs is their high programming complexity. The other main reason is the weak floating-point performance in FPGAs compared to that in GPUs.

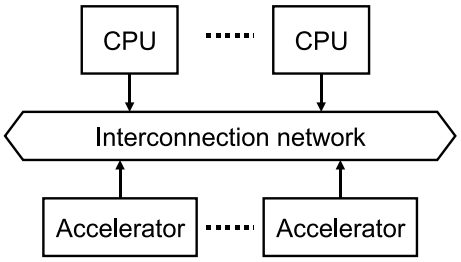


Fig. 1: Heterogeneous processing

In this paper, we propose a low-power heterogeneous platform with CPUs and FPGA accelerators. We propose a CUDA-like (Compute Unified Device Architecture) SIMD-2D (Single Instruction, Multiple Data-2 dimensional PE array) architecture to reduce the programming complexity. The basic idea of this architecture and its programming environment are already proposed in [2]. In this paper, we develop the basic SIMD-2D architecture proposed in [2] to be applicable for HPC applications. We use the finite-difference time-domain (FDTD) algorithm for electromagnetic field computation as an example. According to the experimental results with FDTD computation, we found that 95% of the computation can be done by fixed-point arithmetic without a major precision loss. However, for the rest of the calculations, we must use double-precision floating-point arithmetic. Therefore, we can perform floating-point arithmetic in CPUs and fixed-point arithmetic in FPGAs. The proposed heterogeneous environment produces almost the same performance of GPU-based processing at a 10 times smaller power consumption.

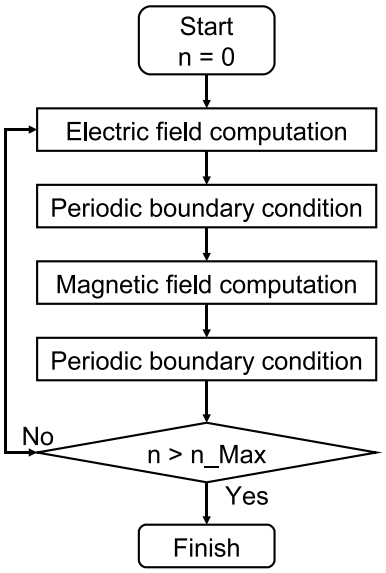


Fig. 2: Flowchart of the FDTD computation

Table 1: Precision vs. computation error

	Method	Maximum absolute error	
		Electric field	Magnetic field
1	Double-precision (DP) floating point	-	-
2	Single-precision (SP) floating point	1.16×10^{-5}	9.20×10^{-6}
3	5% DP floating point 95% 32bit fixed point	1.50×10^{-7}	1.92×10^{-7}
4	25% DP floating point 75% 32bit fixed point	6.80×10^{-8}	6.39×10^{-8}

Table 2: Specifications of the heterogeneous platform

Processor	Number of cores	Maximum power
CPU: Intel core i7 3960X	6	130W
GPU: GeFoce GTX5900	1024	360W
FPGA: DE4 board [6] Stratix IV GX EP4SGX530	191 PEs	20W

2. Heterogeneous Platform for 2D-FDTD computation

2.1 FDTD-2D computation

FDTD [3] algorithm is one of the most popular method of computational electromagnetic simulation due to its simplicity and very high computational efficiency. It has been already implemented successfully in multicore CPUs. There are many recent works such as [4] and [5] that use GPUs to accelerate the FDTD algorithm.

Figure 2 shows the main tasks of the FDTD algorithm. It starts with the initial values of the electric and magnetic fields. Then the initial data are processed to obtain the electric field information for the first time-step. After that, the periodic boundary conditions are applied. Then the magnetic field information are obtained and the periodic boundary conditions for the magnetic field are applied. This process continues for a given number of time-steps. Equation (1) shows the electric field computation and Eqs. (2) and (3) show the magnetic field computation. Electric and magnetic fields in x, y, z directions are denoted by E and H respectively. The time-step is denoted by n and the coordinates of the 2D fields are denoted by i and j. Note that, the boundaries of the electric and magnetic fields are calculated differently. A detailed description of the FDTD algorithm is given in [3]

$$\begin{aligned} &E_z^{n+1}(i,j) = E_z^n(i,j) \\ &-P_y(i,j)\left\{H_x^{n+\frac{1}{2}}(i,j+1/2) - H_x^{n+\frac{1}{2}}(i,j-1/2)\right\} \\ &+P_x(i,j)\left\{H_y^{n+\frac{1}{2}}(i+1/2,j) - H_y^{n+\frac{1}{2}}(i-1/2,j)\right\} \end{aligned} \quad (1)$$

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,j+1/2) &= H_x^{n-\frac{1}{2}}(i,j+1/2) \\ &-Q_y(i,j)\left\{E_z^n(i,j+1) - E_z^n(i,j)\right\} \end{aligned} \quad (2)$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i+1/2,j) &= H_y^{n-\frac{1}{2}}(i+1/2,j) \\ &-Q_x(i,j)\left\{E_z^n(i+1,j) - E_z^n(i,j)\right\} \end{aligned} \quad (3)$$

Table 1 shows the precision of the FDTD computation when using floating-point and fixed-point arithmetic. We assume that the method 1 that uses double-precision floating-point is the desired result. According to the data, the precision loss of methods 3 and 4 that use fixed-point arithmetic is very small. Moreover, the precisions of the methods 3 and 4 are much better than that of method 2 which uses single-precision floating-point arithmetic. In the FDTD algorithm, the dynamic range of the data is very wide close to the electric and magnetic field boundaries. Therefore, double-precision floating point arithmetic are required in this area. Compared to that, the data in the middle of the fields have a very narrow dynamic range. Therefore, 32bit fixed-point arithmetic is sufficient. According to this results, we can use FPGA with 32bit fixed-point ALUs to do 95% of the total computation at a very low power consumption of less than 20W.

2.2 Low-power heterogeneous platform

We use a heterogeneous environment with a multicore CPU, a GPU and an FPGA. Figure 3 shows a computational node of this heterogeneous platform. FPGA and GPU are connected through the PCI express bus. The specifications of the platform is shown in Table 2.

One main problem of the CPU/FPGA low-power heterogeneous platform is the programming complexity of the

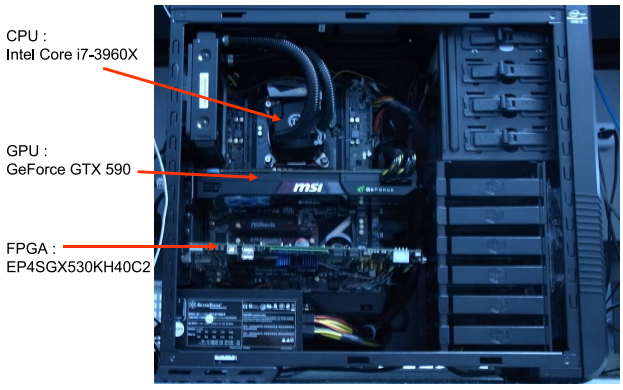


Fig. 3: Computational node of the heterogeneous platform

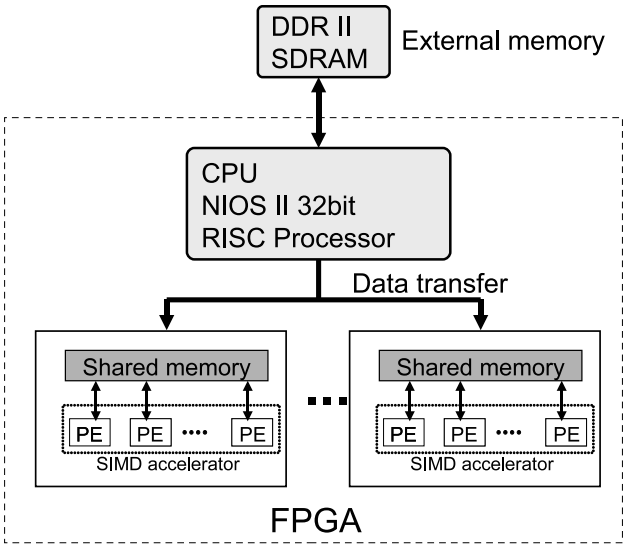


Fig. 4: Overall architecture

FPGA. To reduce the programming complexity, we propose a CUDA-like architecture for the FPGA. Figure 4 shows the proposed FPGA architecture. It consists of a Nios II CPU core, on-chip memory and SIMD accelerator cores. An external DDR2 SDRAM is connected to the CPU core through the FPGA board. The Nios II CPU core is mainly used as the control unit of the accelerators. The proposed SIMD accelerator is designed similar to the GPU accelerator so that we can use the same CUDA code. The basic idea of the SIMD accelerator and its programming environment are discussed in the previous work [2]. The major difference of the architecture in [2] and this paper is the structure of the PE.

Figure 5 shows the architecture of a 32bit fixed-point PE. It consists of 5 adders and 3 multipliers. The data path is fully pipelined so that an output is produced in every clock cycle after the pipeline is filled. We can have 191 such PEs in a single "Stratix IV GX EP4SGX530" FPGA. It is possible to increase the number of PEs by using a much powerful

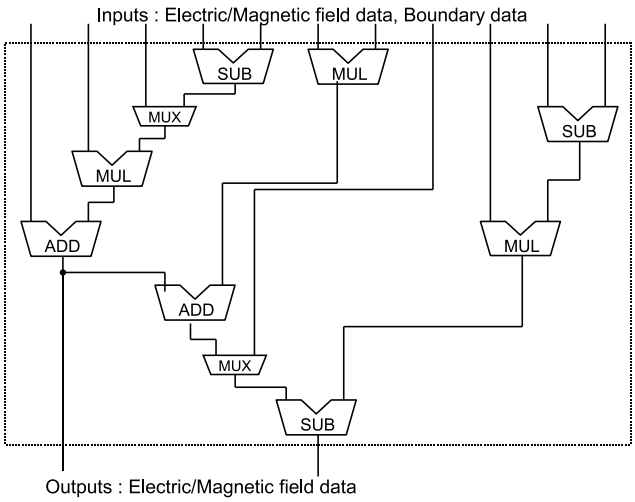


Fig. 5: Architecture of a PE

Table 3: Resource usage estimation of FPGA implementation

Resources (for 191 PEs)	394503 Logic blocks (92.8%) 417140 Registers (98.1 %) 1020 DSP units (99.6%) 20Mbits (98.87%)
Frequency	50 MHz
Power (estimated)	Less than 20W

FPGA.

In the proposed platform, the data close to the boundaries of the electric and magnetic fields are processed in the CPU (Intel core i7-3960X) and the rest of the data are processed in the FPGA. After the boundary data are processed, they are transferred to the FPGA. We can hide this CPU processing overhead by overlapping the boundary data processing in the CPU with the electric and magnetic field computation in the FPGA.

3. Evaluation

We estimated the resource usage, power and frequency of the proposed architecture for Stratix IV GX EP4SGX530 FPGA in the Altera DE4 [6] board. According to the estimation shown in Table 3, we can implement up to 191 PEs in the FPGA at 50MHz frequency. The power consumption is less than 20W.

Table 4 shows the comparison with the CPU/GPU heterogeneous platform. The processing time of the CPU/GPU implementation is measured under the visual studio 2008 environment using CUDA timer. The time for the CPU/FPGA environment is estimated by calculating the number of clock cycles at 50MHz frequency. We consider the method 3 in Table 1 for the CPU/FPGA implementation. According to the results, CPU/FPGA implementation is slightly slower than the CPU/GPU implementation. However, the power consumption is reduced by more than 10 times. Note that,

Table 4: Processing time

Method	Processing time (s)
CPU + GPU	10.08 (measured)
CPU + FPGA	11.26 (estimated)

over 90% of the processing time in CPU/FPGA implementation is due to the boundary data transfers from the FPGA global memory to the local memories. Therefore, reducing the data amount by applying techniques such as data compression is necessary.

4. Conclusion

We have proposed a low-power heterogeneous HPC platform with a multicore CPU and an FPGA. We have shown that 95% of the FDTD computation can be done using fixed-point computation without a major precision loss. The low power consumption is achieved by using FPGA accelerators instead of GPU accelerators for most of the computation. Using the proposed low-power heterogeneous platform, we achieved almost the same performance of the CPU/GPU computing at a 10 times lower power consumption.

Acknowledgment

We would like to thank Altera university program for donating us the DE4 FPGA boards for this research.

References

[1] http://www.top500.org/system/10587
[2] H. M. Waidyasooriya, M. Hariyama and M. Kameyama, "Architecture of an FPGA-Oriented Heterogeneous Multi-core Processor with SIMD-Accelerator Cores", International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pp.179-186, 2010.
[3] H. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media", IEEE Transactions on Antennas and Propagation, Vol.14, No.3, pp.302-307, 1966.
[4] Z. Bo, X. Zheng-hui, R. Wu, L. Wei-ming, S. Xin-qing, "Accelerating FDTD algorithm using GPU computing", International Conference on Microwave Technology & Computational Electromagnetics (ICMTCE), pp.410-413, 2011.
[5] T. Nagaoka and S. Watanabe, "A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis", International Conference on Engineering in Medicine and Biology Society (EMBC), pp.327-330, 2010.
[6] http://www.altera.com/education/univ/materials/boards/de4/unv-de4-board.html