High-level Design Environments for FPGA-based Content Processing

Kevin C. S. Cheng, Martin Fleury
University of Essex, CES Department, Colchester CO4 3SQ, United Kingdom
{kcsche, fleum}@essex.ac.uk

Abstract— Content processing of network packets requires real-time rates and high throughput, which a platform Field Programmable Gate Array (FPGA) can provide. It also requires a high-level design approach that allows time-to-market deadlines to be met. The paper introduces a two-step development process that allows the enhanced memory and I/O facilities of a Content Processor to be utilized but also avail of the more complete simulation and debug facilities on a compatible development board. This can be achieved by providing a complimentary or shadow design environment. High-level design is available for FPGAs in terms of hardware compilation and the provision of an on-chip runtime environment (RTE). The Tarari Content Processor supports both of these but investigation has shown that the RC200 development board assists in reducing design turn-around time.

I. Introduction

The Tarari[1] Content Processor Platform (CPP) based on dual FPGAs[2] is targeted at accelerating the processing of network packet content. Unlike deep-packet inspection using FPGAs [1], content processing normally transforms the packet data contents. The CPP is intended for application processing at layer 7 of the ISO network protocol model. In security, these applications include regular expression processing for anti-virus, anti-spam, and intrusion prevention/detection systems, as well as cryptography and cryptographic hashing.

The main contribution of this paper is to show how a high-level design environment can effectively be applied to the Tarari content processor. We show that a two-phase development process, whereby designs are prototyped on a development board using hardware compilation before transfer to the Tarari CPP can aid the rapid development of security applications. Seaway Networks SW5000 Network Content Processor[3] also is supplied with the Virtio hardware simulator to aid development and debug. However, to the best of our knowledge little attention by others has been given to design environments for content processors, despite their importance in forthcoming consumer electronic products.

Because network security threats are constantly changing, it is important to be able to develop an embedded response quickly. In a commercial setting, there is also a need to reduce the time to market by reducing product development time. High-level design environments with RTE and I/O support can help to meet these deadlines. To a certain extent, the Tarari CPP is an embodiment of pioneering hardware-software development environments such as in [2][3][4], the distinguishing feature of which is that a run-time environment (RTE) with task-support is provided on-chip.

Because an FPGA lies between hardware and software, it can take on the features of each. Hardware compilation allows the FPGA to be treated as software and consequently providing an RTE, becomes possible. The CPP's RTE eases the development process through already-developed agents that reside on an FPGA and perform processing tasks. Only the task logic needs to be programmed as a generic processing infrastructure already exists. The agent-based tasks are also transferable through software synthesis onto ASIC technology such as in Tarari's T9000, T1000 and T10 families.

Application code development for the FPGA may be through Celoxica's[4] Handel-C language [5], which allows hardware compiled programs to be written in a C-like language. Host-to-platform communication is through Celoxica's Data Streaming Manager (DSM), which is activated through a Handel-C Application Programming Interface (API). Therefore, the DSM acts as the I/O system of the RTE.

Unlike hardware description languages (HDLs) such as VHDL, Handel-C's hardware compiler places its emphasis on representing language constructs in hardware. This is the obverse of an HDL which represents hardware, not software, within a programming language. Programs can be developed from the DK Integrated Development Environment (IDE), which has the look-and-feel of Visual Studio, emphasizing the software-like approach of the CPP.

Celoxica's RC200 development board also applies Handel-C and the DK IDE to a Virtex-II FPGA [6], but without the RTE of the Tarari. The RC200's support for simulation and debug mean that it can be used in the development process, with a design prototyped on the RC200 before transfer to the Tarari. Conveniently, we have made it possible to provide a similar RTE structure

[1] Former Tarari Inc. was acquired by LSI Corp. in Sept. 2007.
[2] LSI Tarari also supplies the Tx000 series and the T10 multicore network content processors, which also exist in FPGA form though with reduced throughput. These processors allow external FPGA functionality to be added but have specialist application cores. The architecture is able to migrate to PCI Express. Power consumption is about 8-10 W for the FPGA and 4-6 W for the T10. Manufacturer's performance figures are available:
`http://www.lsi.com/networking_home/networking_products/tarari_content_processors/index.html`

[3] The SW5000 has a proprietary, multi-component architecture.

[4] From early 2008, the Celoxica Electronic System Level (ESL) business was transferred to Catalytic Inc., with full support.

on the RC200, allowing transfer of designs from one board to the other. Thus, designs prototyped on the RC200 can be transferred to the Tarari with its extra FPGAs and its enhanced memory and I/O facilities. In fact, the environment that we have provided on the RC200 can go beyond that because it supports multiple clock domains in a flexible manner, separating content processing from I/O (Tarari can support clock domains but on a per agent basis). The paper shows the effectiveness of using a shadow environment to facilitate designs.

II. DESIGN ENVIRONMENTS

A. *Tarari ContentProcessor*

Though the Tarari can act as a hardware accelerator, it is also suitable for an embedded or resource–constrained system [7], and in security applications reduces the risk of tampering inherent in software. The addition of hardware acceleration to Microsoft's Silverlight multimedia browser plug-in will provide video frame level compression. Titanic IC Systems have also developed the RXP content processor for regular expressions based on Altera rather than Xilinx FPGAs. Though the FPGA architecture developed at Washington University [8] processes content at line rates, it does not transform the payload content.

The Tarari CPP Development Kit (CP-DK) [9] is a combination of hardware (CPP) and software (drivers, libraries and documentation) that creates a reconfigurable hardware platform designed to accelerate a variety of compute-intensive algorithms and applications. The CPP has an architecture that is similar to that discussed in [2] in that an RTE provides a variety of memory access types. The CPP on a Linux host internally interfaces to a double-width 66 MHz Peripheral Component Interconnect (PCI) bus and externally to fast (100 MHz) Ethernet. Figure 1 shows a functional block diagram of the CPP. The Content Processor Controller (CPC) is a logic component that acts as a bridge between a PCI bus, agents and the Synchronous DRAM. It also contains of a PCI DMA controller, which is responsible for data transfer between the CPP and host system memory.

A Content Processor Engine (CPE) is actually a Xilinx Virtex-II XC2V1000-5 platform FPGA. There are two CPEs on the CPP [9]. Each can be dynamically configured at system startup through a script or at run time (without rebooting). After configuration, each CPE is separated into sub-components called agents. Each CPE can run one or two agents, depending on the available resources. Each agent is responsible for a specific task. The agent can be created using supplied agent creation software.

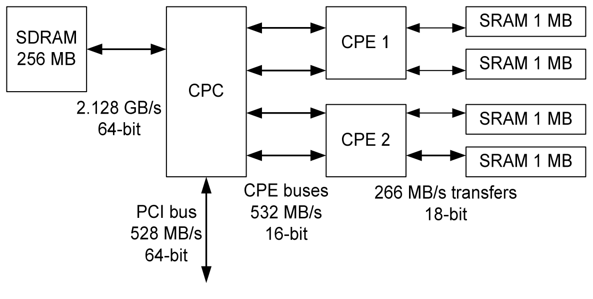

Fig. 1. Block diagram of the CPP

B. *System Support*

CPP system support is conceptualized in layered fashion, Fig. 2, in an approach modeled on operating system organization. Applications communicate via Application Device Drivers (ADD) to a Base Device Driver (BDD). A BDD is CPP resident while ADD can be loaded at runtime. The BDD acts as a mediator to the hardware-based agents, for example informing an ADD of the presence (or otherwise) of one of its agents. Communication from agent to BDD and BDD to ADD is via interrupts. Each agent using a predefined protocol requests allocation of a variably-sized segment of DRAM. Access from an ADD to this segment must also go through the BDD thus enforcing memory boundaries, as alluded to in Section II.A.

An important part of the system support is the DSM API, which creates virtual copies of the underlying hardware to provide a user-defined number of independent, unidirectional data streams between hardware and software. The API contains macros for setting up communication streams, reading and writing data, and querying the amount of data buffered within a stream.

Handel-C outputs either a netlist compatible with FPGA place-and-route tools or a standard EDIF file or Register-Transfer-Level (RTL) VHDL. Critical parts of the design may also be specified in an HDL such as VHDL. Software approaches to hardware [10] using a hardware compiler allow software to be readily ported to hardware or software to be synthesized from existing hardware designs. Compatible with ANSI-C, Handel-C adds extra features required for hardware development. These include flexible data widths, parallel processing and communication by channels between parallel threads using rendezvous.

Agent design in Handel-C is similar to the creation of other applications except that it is necessary to specify an agent id. and the memory usage. The agent code is converted to a form suitable to the CPC and the matching bitstream is downloaded by the agent manager.

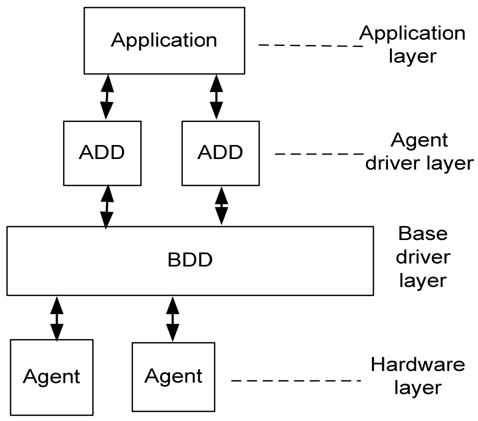

Fig. 2. Layered system support on the CPP.

C. *CP-DK Job Sequence*

CPP software communicates with hardware by passing data from host memory to DDR SDRAM on the CPP. The CPP contains a Direct Memory Access (DMA) Controller (DMAC) (also an FPGA) to move data from the PCI Bus to the DDR SDRAM at high rates with PCI burst mode. Data flow from the DDR SDRAM to one of the agents, loaded in to a CPE, through the CPC and the agent bus. The DMAC moves data back to host memory by means of the PCI bus. In detail:

1. Software builds a DMAC descriptor list containing the input data file and the input header.
2. Software initiates a DMAC operation.
3. The DMAC copies the input file and header from host memory to DDR SDRAM.
4. The DMAC notifies the agent.
5. The agent retrieves the Input Header and file from the DDR SDRAM.
6. When data is available from the agent, the agent copies the data and the output header to the DDR SDRAM.
7. Software reads the output header from DDR SDRAM. (The software is notified via of the arrival of data via an interrupt.)
8. Software builds a DMAC descriptor list to copy the output file to host memory.
9. Software initiates a DMAC operation.
10. The DMAC copies the output file from DDR SDRAM to host memory.

D. *RC200 Board*

The RC200 development board, also from Celoxica Ltd., acts as a convenient means of prototyping tasks written in Handel-C. The main features on the RC200 are: Xilinx XC2V1000-4 Virtex-II platform FPGA; an Ethernet MAC/PHY with 10/100 base-T socket; parallel port or JTAG port for bit-file download; and RS-232-variant serial port.

This is an ideal prototype environment for the Tarari because like the Tarari it has a Xilinx Virtex-II XC2V1000 FPGA, though with a speed grade of four, rather than the higher speed grade of five on the Tarari. Therefore, Handel-C with associated DK IDE can be applied to the RC200 to develop algorithms, which can subsequently be transferred to the Tarari. The version of the DK available to us for the Tarari was significantly slower in compilation turnaround time than the later versions (up to DK4) available for the RC200. DK4 is not backwards compatible with the Tarari libraries. Moreover, it was possible to simulate all functions from within the RC200 DK, as well as to debug, whereas this was not the case for the Tarari as it was available to us.

E. *Development on the RC200*

The RC200 is suitable for stand-alone network applications, though it is configured via a parallel port or JTAG port. It interfaces to Ethernet through the Platform Abstraction Layer (PAL) API for Handel-C, also part of the Nexus toolkit of which DSM, as mentioned in respect to the Tarari, is another component. The main weakness of the RC200 for network applications is that it is directly connected to the Ethernet and not through a PCI bus. The Ethernet on the RC200 is the 10 Mbps version, whereas a PCI interface gives the possibility of processing at the least at fast Ethernet rates.

We have designed an environment for the RC200 that has a similar structure to that of the Tarari. To cope with the needs of packet header scanning a number of Handel-C threads operate through a shared buffer. In Fig. 3, a received packet comes in from the left. The Ethernet Interface is implemented by means of the PAL library. Packets are read from the network and passed to the 'Writer', which then writes its Internet Protocol (IP) header into the shared buffer of width 256 bits. The shared buffer is implemented through Virtex-II dual-port block RAMs. On the other side, the 'Reader' reads the IP header from the shared buffer and passes it to the 'Process Manager'. A Handel-C channel is used together with the buffer to implement a safe FIFO queue. Since the 'Reader' needs only one clock cycle to copy the data from the buffer, there will be not any mutual exclusion problem in the design. (i.e. the 'Writer' will not write into the location the "Reader" is currently reading.) By using a channel, exclusion problems are essentially packaged in the channel construct.

Another feature of the Tarari is dual clock domains, since each agent on the FPGA can be run at a different clock speed. In Handel-C, within a single clock domain execution of instructions is clock synchronous. However, by means of multiple 'main' blocks with associated clock statements, Handel-C supports the multiple clock domains available in the Xilinx Virtex series. Though the Handel-C model is clock-synchronous, the channel primitive allows synchronized communication between parallel processes by means of a rendezvous. This channel construct also allows the designer to neglect detailed timing issues when first preparing a design.

The clock domains should be separated into different tasks if applied to the Tarari. However, due to the structure of the design, the clock domains in Fig. 4 are separated in a different manner. The reason for having three clock domains is because the network interface circuitry restricts the clock speed of the network interface. However, processes do not have such restriction since they are 'pure logic'. By splitting the FPGA into three clock domains, throughput is increased. In fact, up to eight clock domains are possible and assuming dual Ethernet interfaces (not actually present on the RC200) it is possible to have separate clock domains for reading, writing, and a variety of different tasks.

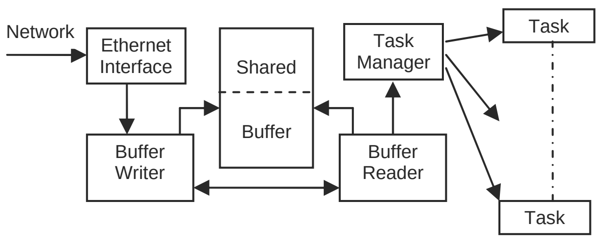

Fig. 3. Basic structure of RC200 environment

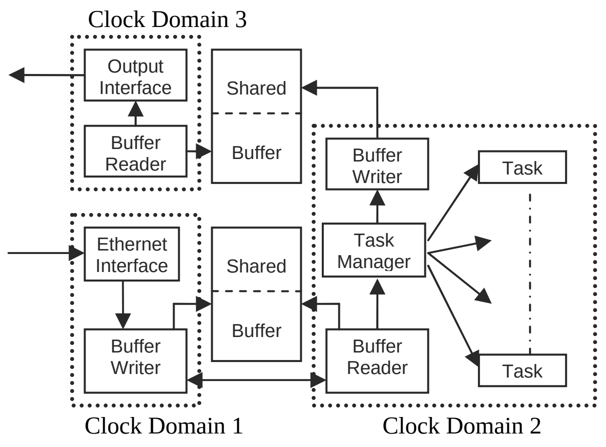

Fig. 4. Processing structure divided into three clock domains

III. TARARI DESIGN AND IMPLEMENTATION ISSUES

A. Application Programmable Interface

The Tarari CP-DK provides three development interfaces, which provide varying levels of performance and complexity:

1. The DSM interface is the simplest of the three interfaces. The Tarari DSM interface implements a subset of the full Celoxica DSM API. The standard Celoxica DSM simulation can simulate the Tarari DSM subset. The DSM Direct Port interface is useful for situations in which sequential access to input/output data is insufficient for the target application. This works in conjunction with the DSM interface.
2. The Standard Agent Interface (SAI) provides the underlying framework for the DSM implementation. The SAI exposes lower-level interfaces that are more complex than the DSM, but potentially have lower overhead.
3. The CPP Raw Interface (CRI) is the lowest-level interface that has the greatest flexibility and complexity. It directly accesses the CPP's device driver and hardware. It also includes the Layered Driver Interface (LDI) used for communication between the CPP device driver layers. These interfaces provide full customization of device drivers for a particular application.

In our experiments, only the DSM interface has been tried and tested due to its ease of hardware and software implementation. The development of drivers is out of the scope of this paper.

B. Simulation and Debug

A very important tool in designing and implementing digital designs is simulation. By having simulation rather than a real hardware implementation, development time can be largely reduced. The DK development environment does support simulation with standard DSM. However, DK only supports basic Tarari specific elements of DSM. Therefore comparing basic DSM to lower level APIs as a point of performance comparison is not possible.

Another useful tool is debugging. For example, the ChipScope Pro program from Xilinx is a real time debugging tools for its FPGA series. In order to use ChipScope Pro, the JTAG interface is needed, while this is not available on the Tarari, it is available on the RC200.

The powerful and sophisticated features of the Tarari are useful. However, they would be ideal if such a system readily supported simulation and debugging.

C. Data throughput on the PCI bus

Tarari supplies a diagnostic agent and program which returns information including release contents, software requirement, basic and detailed diagnostic result. We are interested in the DMA performance; some of the results are shown in Table 1, with "block size" being the size requested for transfer over the PCI bus, and "processes" being the number of processes or threads communicating with the Tarari.

The bitrate shown indicates the sustained throughput for simultaneous reads and writes. For example in the last row of the Table, the CPP performed 1003.30 Mb of read and writes per second, for a total throughput of 2006.60 Mbps. The conclusion can be made that the CPP performs better with multiple processes and larger `BlockSize` set.

Although the throughput seems slow at first glance, the Tarari CP-DK actually employs a relatively old standard of 66 MHz 64-bit PCI. Newer bus technology such as the PCI-X, PCI-Express 1.1 and HyperTransport interconnects can all support faster transfer rates. The latest PCI-Express 2.0 can transfer data at up to 16 GBps. Therefore, the connection interface is unlikely to become a concern in terms of throughput of a future RTE platform and, indeed, recent Tarari content processors support a variety of higher specification interconnects.

Processes	Block Size	Mbps
1	512	94.51
1	4096	471.25
2	512	171.37
2	4096	804.94
4	512	187.17
4	4096	1007.87
8	512	195.64
8	4096	1003.30

Table 1. Diagnostic results

D. Agent-to-Agent communications

'Agent-to-agent communication' would be a very powerful feature, but unfortunately the DSM API does not officially support this kind of operation. When using the DSM Direct Port, an agent can read/write to the SDRAM using addresses. According to the user manual, agents can access each other's address space but it is possible for data corruption to result. Experiments by us to try to share data between agents through the SDRAM showed that it is not apparently possible on this system, since no information is given about how to calculate memory addresses. Therefore, all communication is from agent to host via the DSM. In this paradigm, agents have independent access either to SDRAM or to SRAM, as in a normal Virtex-II system, but, no co-operative processing is possible.

Agent chaining, the ability to pass partially processed data from one agent to another, is a feature that allows pipelined processing. Pipelined processing is applicable to those security applications which are essentially serial in processing such as cryptographic hashing. The restriction of the processing paradigm does have the advantage that it guides the novice developer.

Later versions of the Tarari architecture have implemented agent chaining in the sense that it is not necessary to cross the I/O bus in order to transfer data between agents. This is obviously a welcome feature.

IV PRACTICAL DEMONSTRATION

A. *Single agent application*

MD5 [11] is the latest of the MD series of cryptographic hash functions. (Cryptographic hash functions are used extensively as message integrity checks and can be combined with a key as a form of secure authentication.) To trial the design environments a pipelined architecture was implemented in Handel-C on both the RC200 and the Tarari CPP. As mentioned in Section II.A, because agent chaining is not possible in the Tarari CPP available to us, pipelined applications are best implemented on a single agent.

The generated EDIF was translated, mapped, placed & routed by Xilinx ISE 9.2.02i. An older version of the Design Suite DK2 was used for the Tarari due to a library compatibility problem. The newer version DK4 was used for the RC200.

The RC200 was used as a prototype since it supports better simulation options. Both designs are virtually the same except the RS232 port was the communication link between the RC200 and the host computer. The summary of the RC200 implementation is shown in Table 2. The equivalent gate figure is a Xilinx metric, supplied with the intention of comparison with an equivalent ASIC design. However, caution should be exercised in regard to the assumptions of this metric.

Clock (MHz)	Occupied Slices	Used Block-Rams	Equivalent Gate
25.175	3439 (67%)	16 (40%)	1,096,995

Table 2. RC 200 resource usage summary

Clock (MHz)	Occupied Slices	Used Block-Rams	Equivalent Gate
33	4411 (86%)	23 (57%)	1,602,943

Table 3. Tarari resource usage summary

The Tarari CPP is connected to a Linux server through the 66 MHz 64-bit PCI bus. The BDD driver is *cpp_2.4.20*. The ADD driver is *cpp_dsm_2.4.20*. Both are provided by Tarari. A summary of the Tarari implementations is reported in Table 3. The results are different for several reasons. The newer DK technology is faster and better in terms of hardware compilation. Therefore, the logic usage for the Tarari is higher. However, due to its higher speed grade, the maximum achievable clock speed for the Tarari is higher than the one on the RC200.

In theory, the throughput when the pipeline is fully filled can be computed as:

*Throughput = (Block Size) * (Clock Frequency) / (Longest number of clock cycles in one stage)*

The block size for MD5 is 512 bit, the clock frequency is 33 MHz and the longest stage takes 69 clock cycles. Therefore, the throughput is 244.89 Mbps. The Tarari has two FPGAs on it so the throughput can be doubled to 489.78 Mbps if we were to take advantage of both simultaneously. The interested reader is referred to [12] for a discussion of some published FPGA-based hash algorithm implementations.

Xilinx has now released its 65 nm Virtex-5 [13] FPGA with maximums of 51,840 slices, 550 MHz clocking, as well as speed grade improvement and up to 11.6 Mb of integrated block memory. If the FPGA is larger and the speed grade of the Virtex is higher, more pipeline stages can be implemented with fewer clock cycles for each stage, resulting in greater throughput

The advantage of using the Handel-C approach is the ease of developing and modifying the design, as most hashing code can be found in C. Other hashing algorithms can easily be implemented into the pipeline by modifying the existing stages, though there may be a need for additional stages. For example, SHA-0 and SHA-1 require one more stage than MD-5. It is even possible to dynamically partially reconfigure [14] the FPGA at runtime to change to the new hashing algorithm. (The Virtex series supports 1-D (strip) partial reconfiguration, though arguably could be enhanced with 2-D reconfiguration.)

B. *Multi-agent application*

Small tasks such as virus signature checking and packet scanning are well suited to an architecture such as that of the Tarari with centralized task distribution, as it can be used to increase throughput. An example of this kind of operation is tasks carrying out fragmentation checks. IP version 6 (IPv6) supports IP fragmentation; and in [15] the authors discovered that around 0.5% of the total traffic are fragmented packets, which in absolute terms is considerable. Unfortunately, There are various kinds of IP fragmentation exploits; for a list, refer to [16]. Unlike virus signature checking there is no one method of prevention and moreover stateful processing may be needed as the threat may extend over more than one packet. A bank of tasks is required though the individual task complexity is low. These characteristics make this security application suitable to multi-agent content processing.

On the RC200, a single process tackling the IP fragmentation threat was implemented. It checks for certain patterns inside the header and counts the number of occurrences of fragmented packets in a period of time. The resource usage for a single clock domain implementation is given in Table 4. It runs at approximately 50 MHz, as buffering and calling the PAL code reduces the speed. The majority of the resource usage is taken up by the RTE but clearly many more tasks could be supported than in Table 4. In an agent-based implementation these are grouped into separate agents, because of the restriction on the number of agents.

However, the implementation is also restricted by the slowest clock speed, which is that of the network interface tasks within the environment. By means of intermediate buffering, the processing task's speed can be increased to approaching 100 MHz as shown in Table 5, while the I/O remains at 50 MHz. By means of DK's support for retiming (automatic introduction of registers to reduce the clock width) a further improvement in clock speed is possible at a cost in slice usage for the 10 task example.

Number of Virtex-II slices from 5120 total		
RTE only	1 Task	10 Tasks
501	564	581

Table 4. Resource usage of a one clock domain implementation

Retiming	Virtex –II slices	Min. Clock width (domain 2 only)
No	698	89.0 MHz
Yes	739	91.4 MHz

Table 5. Resource usage of a three clock domain implementation

V. CONCLUSIONS

This paper has demonstrated that a shadow development environment can be an effective way of speeding up design of content processing security applications. The Tarari content processor already makes use of a pioneering concept in high-level design provision of an RTE within hardware, answering the question of how to improve hardware turnaround time, when application development requires this. The concept has been transferred to recent embodiments of the architecture.

For this approach to succeed requires several inputs: pre-written agent code, library calls and drivers for I/O and memory access, and comprehensive simulation and debug facilities. When some of these facilities are not available a shadow high-level design environment is possible and it is this that the paper demonstrates. We were able to implement the environment by means of a compatible board and the final combination results in an RTE-based application with the desired throughput when transferred to the Tarari.

The ability to implement more flexible multiple clock domains (not just agent based) is a feature that can be usefully added to an FPGA-based RTE. Together with dynamic partial reconfiguration, if it were to be supported by the Tarari, this would give the ability to change the security algorithm according to the threat. By mixed language compilation, combining VHDL with Handel-C, it is possible to accelerate time critical parts of the hardware design, while retaining ease of development through the high-level approach of Handel-C.

REFERENCES

[1] S. Dharmapurikar, P. Krishnamruthy, T. S. Sproull, and J. W. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters", IEEE Micro, 24(1):52-61, 2004

[2] A. Koch, "A Comprehensive Platform for HardwareSoftware CoDesign", *Int. .Workshop on Rapid Systems Prototyping*, Paris, 2000.

[3] H. Lange and A. Koch, "Memory Access Schemes for Configurable Processors", pp. 615-625, *Field Programmable Logic and Applications*, 2000.

[4] R. P. Self, M. Fleury, and A. C. Downton, "A Design Methodology for Construction of Asynchronous Pipelines with Handel-C", IEE Proceedings Software 2003, 150(1), 39-47, 2003

[5] M. Bowen, 'Handel-C Language Reference Manual', Celoxica Ltd, Abingdon, UK, 1998.

[6] Virtex-II Platform FPGA Data Sheet, Xilinx, Inc. 2001.

[7] T. Stapko, *Practical Embedded Security: Building Secure Resource-Constrained Systems*, Newnes, Amsterdam, 2008.

[8] D. V. Schuehler, J. Moscola, and J. W. Lockwood, 'Architecture for a Hardware-based, TCP/IP Content-Processing System', IEEE Micro, 24(1):62-69, 2004

[9] Tarari Inc., Tarari Content Processor Development Kit: Developer Guide, San Diego CA, 2003.

[10] M. Fleury, R. P. Self, and A. C. Downton., "Hardware Compilation for Software Engineers: an ATM Example", *IEE Proceedings Software*, 148(1): 31-42, 2001.

[11] R. L. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.

[12] K. Jarvinen, M. Tommiska and J. Skytta, 'Comparative survey of high performance cryptographic algorithm implementations on FPGAs', Information Security, *IEE Proceedings*, vol. 152, no. 1, pp. 3 – 12, Oct. 1995.

[13] Virtex-5 Multi-Platform FPGA, Xilinx, 2007, refer to `http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/`

[14] C. Bobda, A. Ahmadinia, K. Rajesham, M. Majer, A. Niyonkuru, "Partial Configuration Design and Implementation Challenges on Xilinx Virtex FPGAs", ARCS Workshop, pp. 61-66, 2005.

[15] C. Shannon, D. Moore, K. C. Claffy, "Beyond Folklore: Observations on Fragmented Traffic", IEEE/ACM Transactions on Networks, 10(6): 709-720, 2002.

[16] J. Anderson, 'An Analysis of Fragmentation Attacks', March 2001, http://www.ouah.org/fragma.html