Hardware Implementations of Evolvable Systems
A critical analysis of autonomic systems with self-properties on reconfigurable architectures

Imara Speek
Aimee Ferouge
Delft University of Technology
November 6, 2013

Abstract

Reconfigurable hardware is becoming increasingly more important in SoC design as it allows efficient hardware acceleration and virtually unlimited adaptability. With the increase of complex, heterogeneous, multi-core and many-core processors system complexity is skyrocketing. The exponential increase of interactions among systems, the environments in which the systems are required to work and the aggregation of an increasing number of peer elements introduces the need for autonomous systems that are capable of self-adapting and self-optimization. Self-aware adaptive computing systems are capable of adapting their behavior in dynamic environments at run-time to accomplish given goals. Combining these capabilities with reconfigurable hardware allows for self-adapting and self-optimization at run-time. This will eventually lead to a high QoS, reliable systems and a continuity in the services provided.

In this paper we will propose a co-designed hardware implementation of evolvable systems that blends the power of flexible software techniques such as monitoring, decision making and self-adaptation with the speed of hardware.

Keywords: self-aware , computing systems, reconfigurable, autonomic, evolvable

1 Introduction

In the past few years, the term *evolvable hardware (EHW)* has arised. A lot of research has been done in this new field of electronics, which utilizes *evolutionary algorithms (EA)* and *self-adaptive techniques* in order to eliminate manual engineering. Reconfigurable hardware, artificial intelligence, fault tolerance and autonomous systems are brought together, enabling the system to change its architecture and behaviour dynamically and autonomously based on its internal state and its environment. By defining a specific goal or task, the system is programmed to evolve in real-time towards an implementation. It continues research on self-aware systems, machine learning and is often linked to bio-inspired hardware systems.

This paper will introduce several proposed architectures for hardware implementation of evolvable systems in Section 3. Section 4 will then present the techniques that are used in order to accomplish these implementations. Section 5 proposes the system design that is in our opinion the best hardware implementation possible given the current techniques. This design will be critically analyzed using the proposed designs presented in the discussed papers in Section 6. Finally, our literature study will be wrapped up by a conclusion in Section 7.

2 Fundamental knowledge

In this section an introduction to the fundamental terms used in this paper is given. These terms concern prior knowledge required when working with autonomic or evolvable systems and their implementations in hardware.

Computing system containing *self-properties* are capable of adapting their behavior and resources thousands of times based on changing environmental conditions and demands [5]. This allows them to automatically accomplish their goals in the best way possible. This behavior is achieved through *self-monitoring* which recognizes changes in its internal state that may require a modification, called *self-adjusting. Self-healing* concerns effective recovery under fault condition, *self-optimization* invokes optimizing operation in proactive and reactive manners and *self-configuring* concerns automatically installing, configuring and integrating new components seamlessly into the system to meet stated goals [8].

Evolvable systems exploit self-adaptive, self-configuring and self-optimizing techniques and are capable of changing their operations to meet the given performance goals by modifying either the underlying heterogeneous architecture, the operating system and the self-adaptive applications. [4]

Autonomic systems are inpired by the human body's nervous system and contains all self-properties: *self-managing, self-protecting, self-healing* and *self-optimizing* [8].

Autonomic and evolvable systems have the need for heterogeneous underlying architectures, which can be provided by using Field Programmable Gate Arrays (FPGAs).These can be configured to fullfil a desired functionality by using one or more bitstreams. These bitstreams are binary files in which FPGA specific configuration information is stored and these are to be copied along with the proper commands on to the configuration SRAM cells, or the configuration memory [9].

3 Discussion of the Different Papers

Considerable research has already been done in order to efficiently accelerate hardware while still maintaining virtually unlimited adaptability. Software techniques in autonomic computing systems such as hot-swapping and data clustering are discussed in [1]. [5] proposes a self-aware system able to adapt its behavior of FPGA-based systems. Other approaches start with a FPGA-based architecture with a reconfigurable core [2], added programming schemes and new cell structures [3], [6] or even bio-inspired hardware using the POE-model [7]. More recent papers put effort in a combined approach by either implementing autonomic systems on reconfigurable architectures [9] or creating evolvable systems via self-aware applications [4]. In this section software based designs, hardware based designs are co-designs are discussed.

3.1 Software based flexible approaches

[5] presents an implementation of a FPGA-based self-aware adaptive computing system which blends several software based techniques such as *monitoring, decision making* and *self-adaptation.* This system is built on top of a heterogeneous architecture enabling and utilizing the effectiveness of these adaptive computing systems. The operating system on top enables performance monitoring and can take actions involving software and hardware adaptation as seen in the *observe-decide-act (ODA) loop.* Given certain performance goals these loops *observe* the current state, decide upon an action and *acts* accordingly as will be discussed in Section 4.1, 4.2 and 4.3. [5] presents the *application Heartbeats* to asses and monitor performance and progress and the *Implementation Switch Service* to switch between different implementations of the same functionality using a heuristic based decision mechanism.

The Heartbeats application is a monitoring application which makes it possible to assert performance goals as heart-rate windows delimited by a minimum and maximum performance, or *heart-rate* [4]. The Heartbeats API is made of small straightforward functions and allows declaring performance goals. Software components first have to register, specifying parameters such as minimum and maxmim heart-reate, size of the windows of observation and history buffers [5]. The application then updates the progress of the execution calling the function that signifies a heartbeat [4].

The implementation switch service presented in [5] inspired by the *hot-swap mechanism* is fundamental for the system to switch between different implementations of the same functionality at run-time. Since threads can access data structures concurrently and data structures can differ for different implementations, [5] proposes a framework to account for these shortcomings in the hot-swap mechanism. This framework is divided in three sub-phases: a prior phase representing the common work scenario, a transfer phase that blocks new requests in order to reach a quiescent state, translated to fit another data structure and a post phase in which the blocked and new requests are allowed to proceed.

3.2 Hardware based fast approaches

FPGAs are commonly used device for implementing the reconfigurable architecture of evolvable systems, especially Virtex FPGAs produced by Xilinx. The *Erlangen Slot Machine (ESM)* from [6] tackles the three major limitations of the Virtex-II FPGA. A pipelined data flow architecture has been used, replacing the finite state machine by a MicroBlaze microcontroller and employing a data crossbar between plug-ins. Providing a new architecture to avoid the current physical problems of reconfigurable FPGAs, a new inter-module communication concept, as well as an intelligent module reconfiguration management has made the ESM an alternative for the Virtex FPGAs.

In [3] and [2] a Virtex-4 FPGA implementation is introduced for evolvable systems. Other then earlier versions of Virtex (e.g. the Virtex-II Pro), Virtex4 devices enable two-dimensional dynamic reconfiguration, a feature which considerably reduces the reconfiguration time and thus the evolution time ([3]). A big limitations of Virtex FPGAs is an almost unknown and undocumented bitstream data format and an unsafe configurations schema. By using both VRCs (Virtual Reconfigurable Circuits) and direct bitstream manipulation, this architecture eliminates this limitation. This Virtex-4 based device, which takes advantage of 2D reconfiguration capabilities and direct manipulation of the bitstreams is the first one of its kind and will be discussed in Section 4.3.2.

A second Virtex-4 based architecture introduced in [2] also applies a 2D reconfiguration core. Rather than direct bitstream manipulation, an optimized Dynamically and Partial Reconfiguration (DPR) control engine has been integrated. As a result, the processing elements (PEs) of the reconfigurable core are structured as a 2D systolic array, known for its high performance and restrained use of resources. Also, the reconfiguration engine has been optimized by applying three enhancements that will be discussed in 4.3.3.

3.3 Co-design based systems

[9] proposes the harvesting of the full potential of dynamic reconfiguration by carefull evaluating the overhead of reconfiguration in hardware-software interfacing. To overcome the limits introduced by increasing complexity and the workloads to main complex infrastructues they propose to adopt a codesigned self-adaptive and autonomic computing system.

Partial dynamic reconfiguration (PRD) is a key feature that makes FPGAs unique. PDR addresses the lack of resources to implement an application and its adaptability needs. Reconfigurable hardware taking advantage of partial dynamic reconfiguration is the perfect trade-off between the speed of HW and the flexibility of SW. [9] present an evaluation system, implemented on the Xilinx Virtex-II Pro VP30 on an architecture of two reconfigurable cores. Running an extended version of Linux, the cores can be partial dynamically reconfigured. They compare the performance of three different impementations of a popular encryption algorithm (advanced encryption standard): a reconfigurable IP-core that is configured at runtime (FPGA-RHW), a cached hardware IP-core that is ready to use (FPGA-CHW) and a software implementation (SW) as can be seen in figure 1. Figure 1 displays the overhead introduced by reconfiguration. Even though this data was captured in a static analysis, it displays that reconfiguration may dominate the execution. This shows the need for a efficient and fast decision engine, as reconfiguration seems to be only valid in systems of larger than 900 processed blocks.

[4] proposes an evolvable system exploiting self-adaptive techniques by running a customized version of GNU/Linux and a set of adaptive applications on top of a heterogeneous system featuring a multi-core Intel Core i7 processor and a reconfigurable devide, a Xilinx Virtex 5 FPGA. The underlying hardware architecture is made up of static area, containing

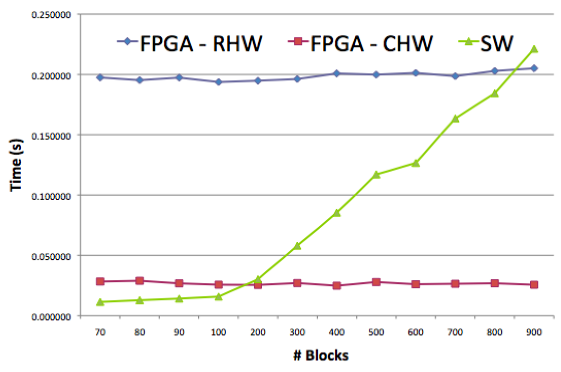

Figure 1: Execution time of the implementations of the AES algorithm as seen in [9]

the general purpose processor and a dynamic area which contains the FPGA.

The customized operating system provides common functionalities through standard libraries, but is also able to choose at runtime the best implementations for the required functionalities thanks to a set of self-adaptive libraries. Depending on the linked library, applications can either be standard or self-adaptive. They come up with an agile decisioning framework driven by a heuristic policy based on an empiric model. This policy should avoid osciliating behavior as to not interfere with the Heartbeats application that is used as a monitoring technique. This 3 tier system can be seen in figure 2.

The online reconfiguration mechanism in [4] they propose is inspired by the modern object oriented operating system called K42, developed by IBM. For the hot-swap mechanism they define a *switchable unit* with a clearly defined interface, *state quiescence* for each access is mutually exclusive and states are maintained per-thread and the *state translation* problem is solved by converting user data back-and-forth from a canonical data structure to the specific data structure used by the active implementation library.

Conludingly they present that their implementation of evolvable self-adaptive methods on a heterogeneous architecture by means observe-decide-act loops is capable of meeting the given performance goals.

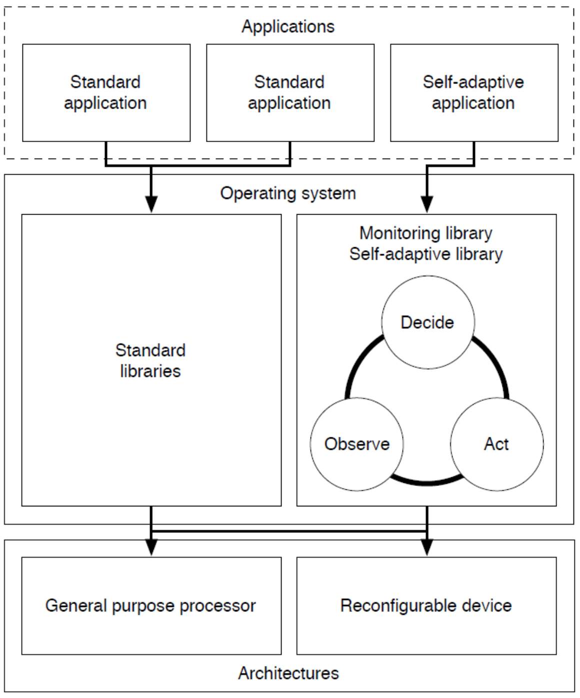

Figure 2: Overview of the evolvable system as seen in [4], providing self-adaptive libraries

4 Analysis of techniques

Reconfiguration capabilities and hardware-software codesign techniques are elements of a complex scenario. In this section we will discuss and analyze these capabilities and techniques seperately. To create such a self-adaptive autonomic system, the system has to be self-aware, able to make decisions, self-adapting and the hardware has to support a certain amount of reconfiguration. In section 5 the overall best combination will be presented, utlizing the flexibility of software and the speed of hardware.

4.1 Monitoring techniques

Most current architecture include at least basic hardware to assist in system monitoring often in the form of counter registers. These techniques suffer often from shortcomings such as not enough counters, a sampling delay and a lack of address profiling [9], which make them unreliable in a dynamic enivironment that should be able to deal with unexpected inputs. These techniques only enable collecting statistics using sampling and lack the possibility to react, therefor it can not be a part of the observe-decide-act loop. Current systems tried to address these shortcomings by introducing microarchitectural event data that can be delivering to the operating system through an exception which would imply the use of frequent interrupts [9]. Interrupts would however introduce an overhead for they need state saves and translations.

The heartbeats application as discussed in Section 3.1 is a technique used for monitoring. Using a framework like this is particularly convenient because it allows to automatically update all information concerning the global heart-rate while concurrently updating its external observers, such as decision making engines. The heartbeats application enables the operating system to directly react and receive information compared to the performance goals. Although [5] presents an average overhead of 3,52% due to system calls in order to initialize its data structures and update the global heart rate. This overhead is moderate concidering it is the fundamental part for an autonomic system and the percentage of overhead cause by the initialization of the data structures wil decrease when working with larger complex systems.

4.2 Decision making techniques

Since it is impossible to pre-configure all possible scenarios in a dynamic system, learning and decision engines are introduced in self-adaptive systems. Results from the monitoring framework are fed to these engines that decide upon an action based on the input, the predefined constraints and the expected *Quality-of-Service (QoS)* [4].

A desicioning framework can be divided into 2 categories: analytical models and empirical models [4]. The analytical models are good when working in the same environment as they are manually generated and are very precise because of this prior knowledge. However when considering evolvable systems in dynamic environments, empirical models are favourable as they exploit information collected at run-time, although they are in turn less accurate. Current trends in supporting decision making are using heuristic policies, machine learning, control theory and competitive algorithms.

Heuristics is the application of experience-derived knowledge to a problem. Heuristic software can be easily developed, applied and reused when working within a known environment including static condition. However within the context of a dynamic live application with changing environments, it is very easy to miss unforeseen problems using software that looks for known problems. Competitive analysis involves comparing the online algorithm's performance with an offline algorithm's performance. Comparing an offline solution with an online algorithm does not apply as a valid comparison as the dynamic application might account unexpected results that were not accounted, not even in the extreme inputs during the static analysis, which is the main reason there is a need for autonomic systems.

4.3 Self-adapting techniques

A self-aware adaptive computing system is an active system where the hardware, the applications and the operating system have to be seen as an unique entity that can autonomously adapt itself to achieve the best performance. [5] presents a general overview of the hardware and software architecture components of a self-aware computing system as can be seen in figure 3. Cognitive hardware mechanisms available in the underlying hardware *observe* and *affect* the exectution. A limited amount of scenarios can be pre-configured, but the learning and decision engines are needed to determine the appropriate actions based on the observations.

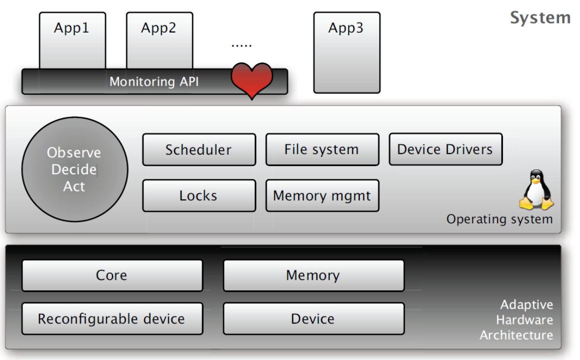

Figure 3: Overview of the proposed self-aware adaptive FPGA-based computing system presented in [5]

A fundamental part of a self-aware adaptive computing system is the ability to switch between implementations of the same functionality while the system is running. A control system has to be developed as an actuator in the oberserve-decide-act loop. The *hot-swap mechanism* is a popular tool to inplement self-configuration and self-optimization. It provides the ability of switching among different implementations of the same functionality in a transparent fashion. However state quiescence and state translation are two big issues when using this mechanism and need a framework solution to be reliable. [?] presents a three phase hot-swap that includes a transfer phase in which new requests are blocked to reach a quiescent state to solve this. This approach is solely based on data structures to decouple the data from the implementations.

Other works present *Partial Dynamic Reconfiguration (PDR)* [9] to reduce the amount of overhead introduced by reconfiguration. Hardware portions can adapt over time to cope with new requirements creating an adaptive system. If the application can be partitioned into different phases, PDR can configure the modules one after another to keep area requirements lower than having all functionalities loaded at the same time. PDR can be seen as a trade-off between the speed of hardware and the flexibility of software.

4.3.1 Virtex FPGA with a Two-Dimensional Reconfiguration Core

On Virtex-4 FPGAs, two-dimensional dynamic reconfiguration like figure 4 is supported. With 2D reconfiguration it becomes possible to reconfigure device portions whose height is not constrained to be the device height. This architecture of the reconfiguration core (or RC) speeds up the reconfiguration time and thus the evolution time. The RC can also deploy more candidate solutions as discussed in [3], which are arrays of bidimensional cell (see figure 4.3.2). An alternative for candidate solutions is a mesh-type systolic array of parallel processing elements (PEs) from [2], also following a 2D architecture for the RC. A major feature is the possibility to change the functionality of the PEs by means of Dynamic and Partial Reconfiguration (DPR). This gives the system the capability to adapt. The outputs of the PEs (east and south side) are connected to the close neighbour's input (west and north side), such that only the lowest and right-most PE has to be read for data output. This systolic approach of communication reduces the reconfiguration time and makes the architecture easy to extend.

A drawback of using Virtex FPGAs are the feed-through signales, mentioned in [6]. Each module must be implemented with all possible feed-through channels needed by other modules. Because we only know at run-time which module needs to feed through a signal, many channels reserved for a possible feed-through become redundant. Also, modules accessing external pins are no longer relocatable, because they are complied for fixed locations where a direct signal line to these pins is established.

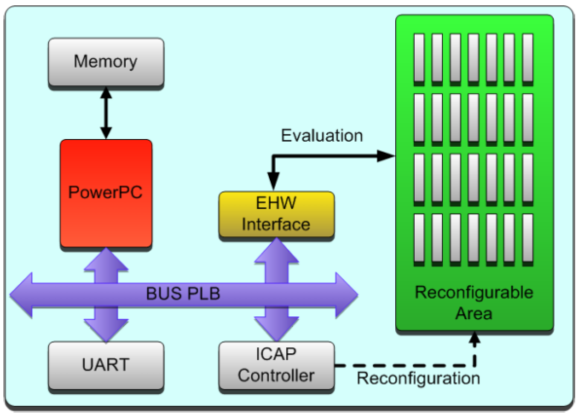

Figure 4: High-level structure of a Two-Dimensional Architecture depicted in [3]

4.3.2 Candidate Solutions and Direct Bitstream Manipulation

With a 2D architecture for the RC, [3] uses safe manipulation of the bitstream to overcome the unknown and undocumented bitstream mentioned in **??**. Candidate solutions are used to fill the RC with cells, which have an internal flip-flops allowing the evolution of synchronous circuits. This is a common structure for evolvable hardware (EHW) systems making use of direct bitstream manipulation [3]. For the cell structure of the candidate solutions the use of LUTs and a MUX is proposed, in order to provide direct bitstream manipulation. Combined with the 2D-reconfiguration mechanism, the new architecture causes a speed up of 16x factor. For this system, only Virtex-4 or Virtex-5 FPGAs can be used since Virtex-II does not support 2D reconfiguation ([3].

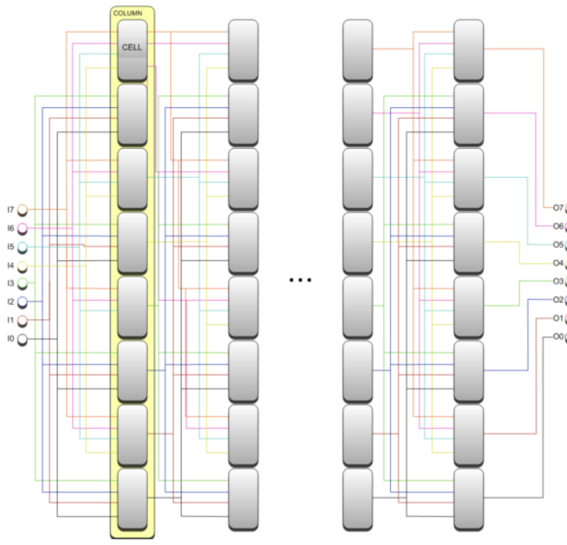

Figure 5: Internal structure of a Candidate Solution proposed in [3]

4.3.3 Systolic array of PEs and Optimized DPR

Other than [3], [2] uses a systolic array of PEs. In this approach, each PE is a basic computation unit able to perform a single operation on the data take from their close neighbors. The architecture can be easily extended to any other processing purposes, since new PEs can be added to the library. In addition, PEs included in this library can be reused among applications. As can be seen in fig. 6, the size of the implemented structure is 4x4, but it can be completely and easily scaled. This DPR with elements relocation is carried out using a special HW block, the reconfiguration engine (RE).

[2] describes an implementation of this RE. By storing only the body of the bitstream (cutting of the header and the tail), overclocking the FPGA by 2,5x and including internal memory (to avoid pasting the same configuration module in different positions of the architecture) the DPR is optimized. Due to this the reconfiguration time is greatly reduced. Adding the header and the tail of the bitstream at runtime has two additional advantages: it is allowed having a unique bitstream for each PE that can be configured in any position of the array. Also, bitstream reduction reduces the data transference time from the external memory.

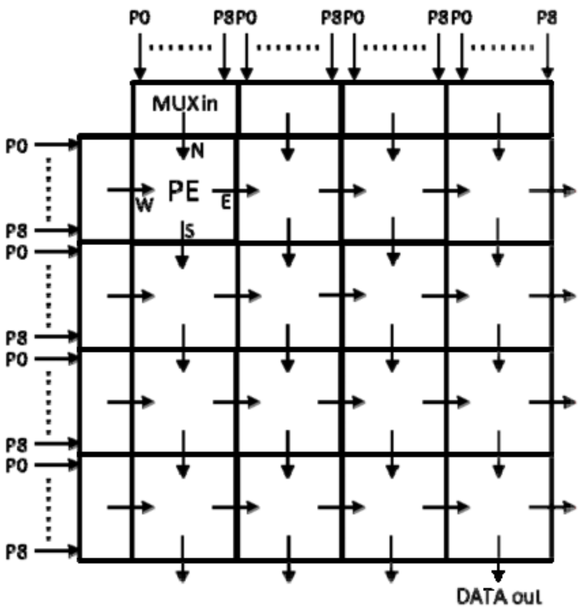

Figure 6: Systolic array of PEs introduced by [2]

4.3.4 Erlangen Slot Machine

The architecture of [6] the Erlangen Slot Machine (ESM) deals with the three drawbacks of FPGAs mentioned in 4.3.1, being fixed pins which are spread around the device (I/O dilemma0, the inter-module dilemma and the local memory dilemma.

First, the I/O dilemma caused by fixed pins spread around the device is solved by connecting all bottom pins from the FPGA to an interface controller realizing a crossbar, as can be seen in figure 7. It connects FPGA pins to peripherals automatically based on the slot position of a placed module. This I/O rerouting principle is done without reconfiguration of the crossbar FPGA.

Second, the memory dilemma has been solved. In normal Virtex-II FPGAs, a module can only occupy the memory inside its physical slot boundary. Storing data in off-chip memories is therefore the only solution. In the ESM, six SRAM banks are connected to the FPGA. Since these banks are placed at the opposite side as the crossbar, a module will connect to peripherals from one side, while the other side will be used for temporally storing computational data. In order to use a SRAM bank (called a slot), the module must have at least a width of three micro-slots, in which the total device is divided (see 7). This organization simplifies relocation, enabling a partially reconfigurable computing system. Also, equal resources will be available for each module.

Finally, the inter-module communication dilemma is dealt with. Dynamically routing signal lines on the hardware is a very difficult task. The ESM uses a combination of bus-macros, shared memory, RMB (Reconfigurable Multiple Bus) and a crossbar to take away the limiting factor for the wide use of partial dynamic reconfiguration ([6]).

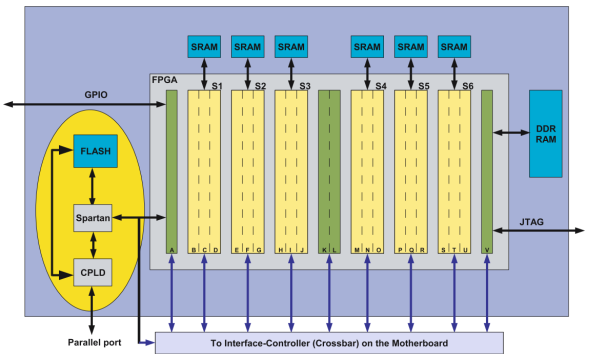

Figure 7: Architecture of the ESM board. Three consecutive micro-slots define a macro-slot, which can access one full external SRAM bank.

5 Proposed adaptive system design

* set up of a proposed system *

6 Related works

In [3], [2] and [6], the software used by the system to evolve certain application is called the Evolutionary Algorithm (EA): a generic population-based metaheuristic optimization algorithm. Candidate solutions play the role of individuals in the population. At every iteration, a new generation of candidate solution is created. For each generation, the genetic algorithm uses the current population to create the children that make up the next generation. The algorithm selects a group of individuals in the current population, called parents, who contribute their genes (the entries of their vectors) to their children. The algorithm usually selects individuals that have better fitness values as parents.

For performance experiments, [3] uses a standard testbench to prove the EHW systems effectiveness. The system is being requested to evolve a parity generator in order to report the results of the EA. A parity generator checks data to be transmitted for logic '1's.If the number is even, the parity bit is set to '1'. If the number is odd, parity bit is '0'. For executing this task, the average duration of the EA runs is 7,9 seconds for a 5-bit input number (and 76,8 seconds for an 8-bit number). A second experiment concerning the evolution of communication channels causes trouble, since the fine-grained evolution by evolving candidate solutions at low level (using direct bitstream manipulation) is not able to succeed most of the times. A third experiment involves the evolution of a counter. For a 3-bit counter, using low level evolution and six columns of cells, the average evolution time of 2,7 seconds is needed. For a 4-bit counter, however, the system failed almost every time. [add a qualitative value to the numbers of generation]

Although the ESM introduced in [6] sounds promising, the machine is a customized design that has been manufactured only in very small amounts at the University of Erlangen-Nuremberg. In order for this system to become wide-spread, all parts have to become easy to assemble for mass production. Their case study about video and audio streaming started with a data block-oriented reconfiguration manager. Unfortunately, the maximum upload speed of a bitstream to the FPGA was slowed down by factor two, due to the bottleneck of scratch pad-oriented data flow combined with the sequential execution of the instructions. When applying a pipelined data flow architecture as mentioned in 3.2, however, additional plug-ins can be added such that the bitstream can be uploaded into the FPGA at the speed of the flash interface. The article concludes with a proposal to add another decompression plug-in in order to further increase the data rate of the bitstreams from 10 MB/s to 50 MB/s, but has no further numbers to quantitatively beat the existing FGPA-based platforms. Maybe this system will have break through in a couple of years.

As for [2], evolution of image noise filters is selected as the proof of concept application. The average evolution time needed to achieve a correct result is 128 seconds, which at first seems hard to compare to the 2,7 seconds of the 3-bit counter mentioned in [3]. However, given the fact that the input image is a 256 x 256 image (with each pixel existing of 8 bits), this result is quite impressive.

7 Conclusion

Conclusion

References

[1] M. Khan A. Khalid, M. Haye and S. Shamail. Survey of frameworks, architectures and techniques in autonomic computing. In *Fifth International Conference on Autonomic and Autonomous Systems (ICAS).*

[2] J. Mora A. Oter, R. Salvador and L. Sekanina. A fast reconfigurable 2d hw core architecture on fpgas for evolvable self-adaptive systems. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS).*

[3] M. Santambrogio F. Cancare and D. Sciuto. A direct bitstream manipulation approach for virtex4-based evolvable systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS).*

[4] H. Hoffman F. Sironi, A. Cuoccio and M.D. Santambrogio M. Maggio. Evolvable systems on reconfigurable architecture via self-aware adaptive applications. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2011).*

[5] H. Hoffmann M. Maggio M.D. Santambrogio F. Sironi, M. Triverio. Self-aware adaptation in fpga-based systems. In *2010 International Conference on Field Programmable Logic and Applications.*

[6] A. Ahmadinia M. Majer, J. Teich and C. Bobda. The erlangen slot machine: A dynamically reconfigurable fpga-based computer. *Journal of VSLI Signal Processing.*

[7] D. Mange et al. M. Sipper, E. Sanchez. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, August 2002.

[8] D R. Sterritt and Bustard. Towards an autonomic computing environment.

[9] M.D. Santambrogio. From reconfigurable architectures to self-adaptive autonomic systems. In *Int. J. Embedded Systems.*