# Block RAM-based architecture for real-time reconfiguration using Xilinx® FPGAs

Rikus le Roux*, George van Schoor†, Pieter van Vuuren*

*School of Electrical, Electronic and Computer Engineering, North-West University, Potchefstroom, South-Africa
†Unit for Engineering Research, North-West University, Potchefstroom, South-Africa

## ABSTRACT

Despite the advantages dynamic reconfiguration adds to a system, it only improves system performance if the execution time exceeds the configuration time. As a result, dynamic reconfiguration is only capable of improving the performance of quasi-static applications. In order to improve the performance of dynamic applications, researchers focus on improving the reconfiguration throughput. These approaches are mostly limited by the bus commonly used to connect the configuration controller to the memory, which contributes to the configuration time. A method proposed to ameliorate this overhead is an architecture utilizing localised block RAM (BRAM) connected to the configuration controller to store the configuration bitstream [1, 2]. The aim of this paper is to illustrate the advantages of the proposed architecture, especially for reconfiguring real-time applications. This is done by validating the throughput of the architecture and comparing this to the maximum theoretical throughput of the internal configuration access port (ICAP). It was found that the proposed architecture is capable of reconfiguring an application within a time-frame suitable for real-time reconfiguration. The drawback of this method is that the BRAM is extremely limited and only a discrete set of configurations can be stored. This paper also proposes a method on how this can be mitigated without affecting the throughput.

KEYWORDS: FPGA, reconfiguration, architecture, real-time, BRAM

CATEGORIES:
C.3 [**Special-purpose and application-based systems**]
C.1.3 [**Processor architectures**]: Other architecture styles
B.5.2 [**Register-transfer-level implementation**]: Design aids

ARTICLE HISTORY

Received 29 April 2014
Accepted 4 June 2015

## 1 INTRODUCTION

Reconfigurable computing refers to the utilisation of application specific hardware in conjunction with general purpose software to improve system performance [3]. Initially, this was done using a modular design where a hardware module can be substituted with another to perform a specialised function [4]. A feature of Xilinx® field-programmable gate arrays (FPGAs), called dynamic reconfiguration, allows the device to change a section of its hardware while the rest remains operational [5]. Most of Xilinx®'s FPGAs from the Virtex-II® onward incorporate this feature, with the addition of the internal configuration access port (ICAP) that provides access to the configuration registers of the FPGA. Reconfigurable computing improves system performance by specializing the system towards a specific application. Additional advantages include a reduction in power consumption and component count [5, 6, 7]. Despite the numerous advantages, dynamic reconfiguration has one major disadvantage. Reconfiguring an application will only improve the system performance if the execution time exceeds the configuration time [8, 9]. This implies that dynamic reconfiguration will only improve the system performance of quasi-static applications. Typical reconfiguration times achieved are in the order of milliseconds and despite on-going research, this still holds true for most applications.

The reason why most reconfigurable architectures are unsuitable for real-time applications is due to their long reconfiguration time or the delay induced by the reconfiguration process. In order to mitigate these shortcomings and migrate reconfigurable computing

**Email:** Rikus le Roux `rikuslr@gmail.com`, George van Schoor `george.vanschoor@nwu.ac.za`, Pieter van Vuuren `pieter.vanvuuren@nwu.ac.za`

to dynamic applications, various attempts have been made to improve the throughput of the system to rival that of the ICAP controller. The maximum theoretical throughput of the ICAP is 800 Mbps and 3.2 Gbps for the Virtex-II® and 5 respectively. However, the throughput of the systems are significantly lower than that of the ICAP, due to the bus-based architectures used. In fact, it is estimated that about 40% of the overhead is contributed by the Xilinx® ICAP driver function [10]. Attempts to improve the throughput of the system include:

- reducing bitstream size,
- optimizing the way the bitstreams are written to the memory, and
- optimizing the transfer of the bitstream to the ICAP [11].

Improving the throughput of the system allows the ICAP to process new data every clock cycle, which optimizes reconfiguration throughput. This reduction in reconfiguration time will allow dynamic applications such as adaptive control or gain scheduling to utilize dynamic reconfiguration to not only change their parameters, but also to completely change their architectures. Reconfiguration could also improve the area utilisation. Bruneel et al. [12] showed that implementing an adaptive filter using reconfiguration requires 40% less lookup tables than its static counterpart.

The only architecture capable of maximizing throughput without any delay is the block RAM (BRAM)-based architecture proposed in [1, 2]. This architecture bypasses the system bus and is capable of reconfiguration at the maximum theoretical throughput of the ICAP. The architecture also allows the ICAP to be overclocked, further increasing the throughput.

The aim of this paper is to illustrate the advantages of the proposed BRAM-based architecture for reconfiguring real-time applications and to verify the throughput claimed in the literature. It also proposes a design methodology for the most important aspects of the architecture and proposes a method to overcome the size limitation imposed by the limited amount of BRAM. The paper starts off with Section 2 by discussing dynamic reconfiguration for quasi-static applications and its limitations for reconfiguring dynamic applications. The architectures proposed in the literature to improve the reconfiguration throughput are also discussed. From these architectures, the BRAM-based architecture was identified as the most promising for real-time reconfiguration. Section 3 discusses the design methodology for implementing this architecture along with possible issues and how they can be resolved. Section 4 discusses the experimental setup used to validate the reconfiguration throughput of the architecture and the results given in Section 5. Section 6 then concludes by proposing a method to ameliorate the size limitation imposed by the BRAM. The overall conclusion is given in Section 7.

## 2 RELATED WORK

Most research in reconfigurable computing is validated using quasi-static applications such as key specific data encryption standard (DES) [8], sub-graph isomorphism [13], Boolean satisfiability (SAT) [14] and adaptive filters [15].

Eldredge and Hutchings [16, 17] used run-time reconfiguration to enhance the functional density of an artificial neural network, dubbed the Run-Time Reconfigured Artificial Neural Network (RRANN). Functional density is a measure of the computational throughput of the system and is a function of the area and execution time [18]. The RRANN architecture divides the backpropagation algorithm into three sequential stages. Dynamic reconfiguration is then used to adapt one of the stages to suit the requirements. The reconfiguration process is controlled using an external processor of a host personal computer (PC) (which stores all the configuration information for the neural network) and adds between 14 and 21 ms to the execution time.

Economakos [19] presented an embedded run-time reconfigurable proportional-integral-derivative (PID) controller. A microcontroller was used to reconfigure the PID parameters via the ICAP using configuration data stored in the on-chip bus-connected block RAM (BRAM). Only the gain parameters are reconfigured, which are tuned using a fuzzy logic module implemented on the embedded processor. The smallest partial bitstream that can be transferred through the ICAP is 41 32-bit words, which equals 1312 bits. This implies that changes smaller than 41 words can be performed at an extremely high speed. As already mentioned, the ICAP reconfigures at a rate of 400 MBps. By placing a set of PID parameters inside a frame, Economakos showed that, considering frame length and reconfiguration rate, the reconfiguration time for each parameter change is 0.41 $\mu$s. Even though this methodology is capable of fast reconfiguration, a bus-based architecture was again used, adding additional overhead, and the results specified are assumed to be per parameter.

The drawback of most reconfigurable architectures is that buses are used to connect the various components of the architecture. In fact, as illustrated by Fig. 1 and 2, even the configuration controller intellectual property (IP) cores provided by Xilinx® are bus-based, which adds additional overhead to the configuration process. Consequently, many researchers have adapted their system architectures to mitigate the overhead incurred. This is done by adding functionality such as direct memory access (DMA) [11, 20], burst modes [21, 22] and dedicated BRAM [1, 2].

Fig. 3 illustrates a reconfigurable architecture with DMA capability. DMA functionality allows the configuration controller's hardware subsystem to access the system memory directly. This improves efficiency since the embedded processor is relieved from the configuration process. However, since the processor bus is still used to connect the DMA controller to the external memory, this type of architecture still induces

reconfiguration overhead. The addition of a multi-port memory controller (MPMC) can allow the DMA controller to access the external memory directly without the need for a system bus. The result is an average reconfiguration speed almost three times faster than that of the DMA-architecture [1].

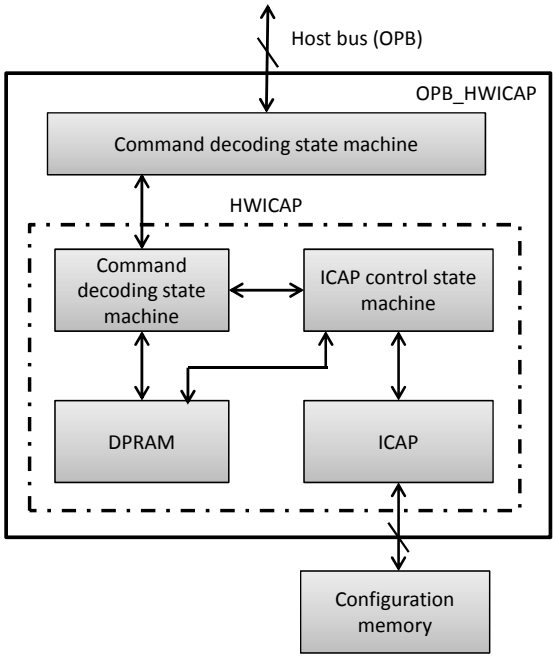


Figure 1: Xilinx® proprietary on-chip peripheral bus (OPB) ICAP controller [23]

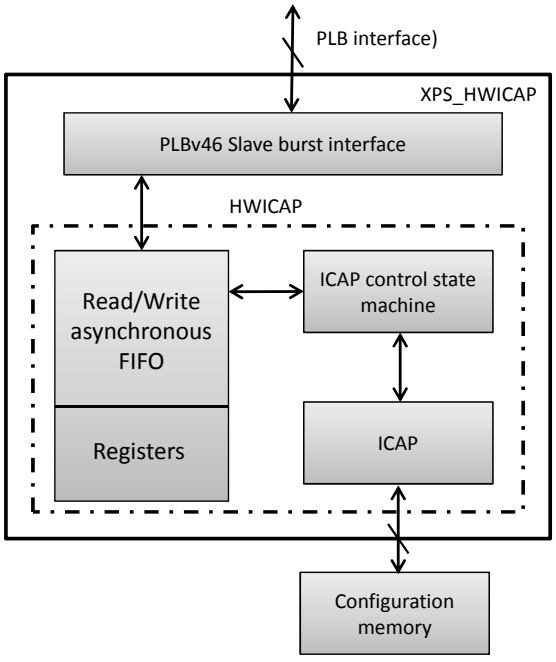


Figure 2: Xilinx® proprietary processor local bus (PLB) ICAP controller [24]

Streaming modes are also used in conjunction with DMA to improve the throughput [22]. In these designs, the bitstream can be loaded continuously as needed.

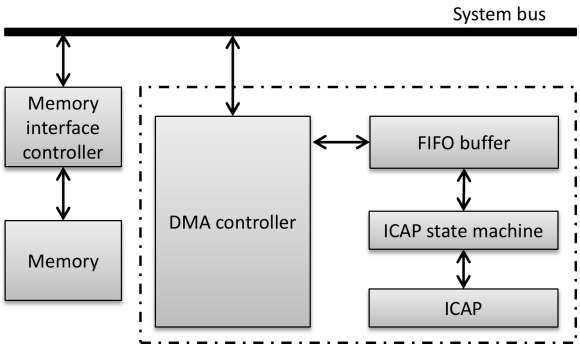


Figure 3: Reconfigurable architecture with DMA

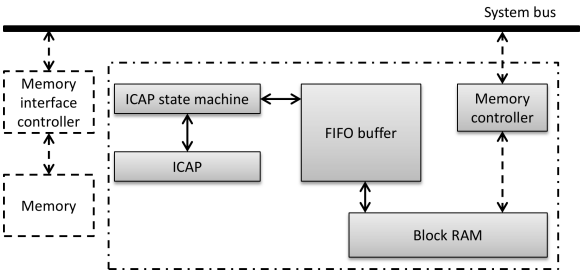


Figure 4: Reconfigurable architecture with BRAM

This ensures that the local buffer, normally a first-in, first-out (FIFO) that feeds the ICAP with configuration data, is always full. The result is a continuous source of configuration data to the ICAP, compared to the fetch-and-configure model of the traditional reconfiguration process.

Even though these improved bus-based systems are capable of reconfiguration throughputs rivalling that of the ICAP, they are limited by one major drawback. All these architectures suffer from configuration latency. Multiple clock cycles are required to transfer the initial configuration frames from external memory to the localised memory from where it can be used by the ICAP. Liu et al. [22] aimed to minimize the configuration overhead by incorporating streaming, compression and DMA into an intelligent ICAP controller. Despite their experimental results showing their implementation nearly saturates the throughput of the ICAP, the DMA and compression add configuration overheads of 17 and 6 clock cycles respectively.

The dedicated BRAM architectures shown in Fig. 4 aim to mitigate all configuration overhead by using a dedicated BRAM directly connected to the FPGA fabric to store the configuration data. The FIFO buffers shown in Figures 1 to 4 are used to store sections of the configuration data moved from external memory, whereas the BRAM is used to store the entire bitstream. Evidently, the drawback is that the BRAM should be significantly large. For bitstreams too large to fit in the BRAM, partial bitstreams can be loaded into the BRAM using the processor bus.

Alternatively, the bitstreams can also be compressed to fit into the BRAM. This could also have the added benefit of reducing the reconfiguration time, since a smaller amount of data need to be transferred to the configuration memory. In general, the reconfiguration time can be calculated by dividing the size of

the bitstream (in bits) by the throughput of the ICAP. Reducing the size of the bitstream will thus also reduce the reconfiguration time. Even though compression techniques such as Lempel-Ziv-Welch (LZW), Lempel-Ziv (LZ7) or custom algorithms [1, 25] are capable of reducing the bitstream significantly [26], the bitstream has to be decompressed before being sent to the configuration memory. Depending on the decompression algorithm used, this could contribute significantly to the reconfiguration time. The more complex the algorithms, the bigger the impact on reconfiguration time will be.

The BRAM-based architecture is therefore regarded as the most suitable, if real-time reconfiguration is required. To verify this, two simple applications were implemented and reconfigured using the proposed architecture. The next section discusses the design flow used for designing and implementing these applications, and highlights some of the pitfalls encountered.

## 3 DESIGN FLOW

The Xilinx® partial reconfiguration (PR) design flow was used to design the application using the ISE® Design Suite [27]. Even though a newer partial reconfiguration design flow is available for Xilinx®'s newer Virtex®-7, Kintex®-7 and Artix®-7 FPGA families using Vivado®, this flow is not supported on older families. However, the PR flow implemented in Xilinx®'s Integrated Synthesis Environment (ISE®) can also be applied to newer families.

Fig. 5 shows the basic premise of the PR flow. In the figure, the function implemented in *Reconfigurable block 'A'* is modified by switching between several configurations, *A1.bit*, *A2.bit*, *A3.bit* and *A4.bit*, while keeping the rest of the logic intact.

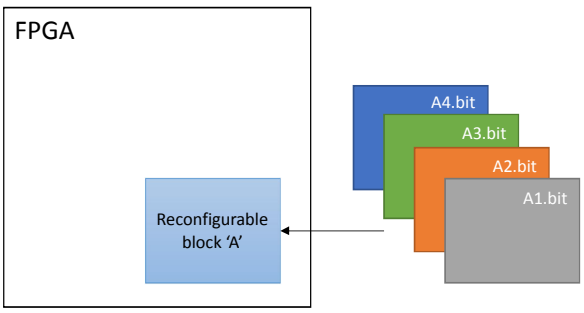


Figure 5: Basic premise of partial reconfiguration illustrating configurations being swapped to and from the device [27]

Using the PR design flow, certain issues were encountered while implementing the test applications. The following sections are dedicated to addressing these, along with important design aspects of the architectures. The first issue encountered, and an important design aspect, was initialising the BRAM with the configuration data.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | 09 | 0F | F0 | 0F | F0 | 0F | F0 | 0F | F0 | 00 | 00 | 01 | 61 | 00 | 26 | 63 | 6F | 6E |
| 13 | 66 | 69 | 67 | 5F | 32 | 5F | 72 | 6F | 75 | 74 | 65 | 64 | 2E | 6E | 63 | 64 | 3B | 55 | 73 |
| 26 | 65 | 72 | 49 | 44 | 3D | 30 | 78 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 00 | 62 | 00 | 0E |
| 39 | 35 | 76 | 66 | 78 | 37 | 30 | 74 | 66 | 66 | 31 | 31 | 33 | 36 | 00 | 63 | 00 | 0B | 32 | 30 |
| 4C | 31 | 31 | 2F | 30 | 37 | 2F | 32 | 38 | 00 | 64 | 00 | 09 | 31 | 38 | 3A | 32 | 37 | 3A | 35 |
| 5F | 31 | 00 | 65 | 00 | 33 | 8C | 00 | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| 72 | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| 85 | FF | 00 | 00 | 00 | BB | 11 | 22 | 00 | 44 | FF | FF | FF | FF | FF | FF | FF | FF | AA | 99 |
| 98 | 55 | 66 | 20 | 00 | 00 | 00 | 30 | 02 | 00 | 01 | 00 | 00 | 00 | 00 | 30 | 00 | 80 | 01 | 00 |
| AB | 00 | 00 | 00 | 20 | 00 | 00 | 00 | 30 | 00 | 80 | 01 | 00 | 00 | 00 | 07 | 20 | 00 | 00 | 00 |
| BE | 20 | 00 | 00 | 00 | 30 | 02 | 20 | 01 | 00 | 00 | 00 | 00 | 30 | 02 | 60 | 01 | 00 | 00 | 00 |
| D1 | 00 | 30 | 01 | 20 | 01 | 00 | 00 | 3F | E5 | 30 | 01 | C0 | 01 | 00 | 00 | 00 | 00 | 30 | 01 |
| E4 | 80 | 01 | 03 | 2C | 60 | 93 | 30 | 00 | 80 | 01 | 00 | 00 | 00 | 09 | 20 | 00 | 00 | 00 | 30 |
| F7 | 00 | C0 | 01 | 00 | 40 | 00 | 00 | 30 | 00 | A0 | 01 | 00 | 40 | 00 | 00 | 30 | 00 | C0 | 01 |
| 10A | 00 | 00 | 00 | 00 | 30 | 03 | 00 | 01 | 00 | 00 | 00 | 00 | 20 | 00 | 00 | 00 | 20 | 00 | 00 |

Figure 6: Sectional view of the bitstream contents

### 3.1 BRAM initialisation

The BRAM should be initialised with the reconfiguration data, also known as the bitstream. However, this poses some issues since the bitstream cannot be loaded directly into the BRAM using Xilinx®'s CORE Generator™, which only supports *.coe*-files. A *.coe*-file is a text-based file containing a header and initialisation data for the BRAM, whereas the bitstream contains binary data representing the configuration bits of the FPGA. An example of an unformatted bitstream is shown in Fig. 6 in hexadecimal-format. As can be seen, the data are not grouped which complicates the data loading process.

Using BitGen™ the bitstream can be converted into American standard code for information interchange (ASCII), shown in Fig. 7. As can be seen, the data are grouped into 32-bit sets each representing a configuration command, some of which are also listed in the figure. This ASCII-file can easily be loaded into the BRAM as a *.coe*-file. Alternatively, it can also be loaded into BRAM on synthesis using the VHDL construct shown in Listing 1. This construct is capable of reading a text-based file containing the configuration data and initializing the BRAM.

A central component in the proposed reconfiguration architecture is the hardware required to facilitate the reconfiguration process. This is discussed in the next section.

### 3.2 Hardware controlled reconfiguration

Hardware controlled reconfiguration (HCR) refers to the use of hardware implemented on the FPGA to control the reconfiguration process, compared to conventional methods that require a processor bus, such as the processor local bus (PLB) or Xilinx® Platform Studio (XPS) bus. Using hardware to control the reconfiguration process involves using a state machine. This state machine is based on the state machine used for MultiBoot, which is a feature included in Xilinx® FPGAs. It allows an active application to fall back to a previous good configuration (known as the 'golden image') in the event of a configuration failure, operational failure or single event upset (SEU). It also allows for warm boot reconfiguration, a sub-category of the fall-back reconfiguration, which allows only a section of the device to be reconfigured without affecting the remainder of the device [28, 29].

Listing 1: VHDL construct to load bitstream into BRAM

```
type <romtype> is array(0 to <rom_width>)
                  of bit_vector(<rom_addr_bits> downto 0);

impure function <rom_function_name> (<rom_file_name> : in string)
    return <romtype> is
    FILE <rom_file>           : text is in <rom_file_name>;
    variable <line_name>      : line;
    variable <rom_name>       : <romtype>;

 begin
    for I in <romtype>'range loop
        readline (rom_file>, <line_name>);
        read (<line_name>, <rom_name>(I));
    end loop;
    return <rom_name>;
 end function;

 signal <rom_name> : <romtype> := <rom_function_name>("<file_name>");
```

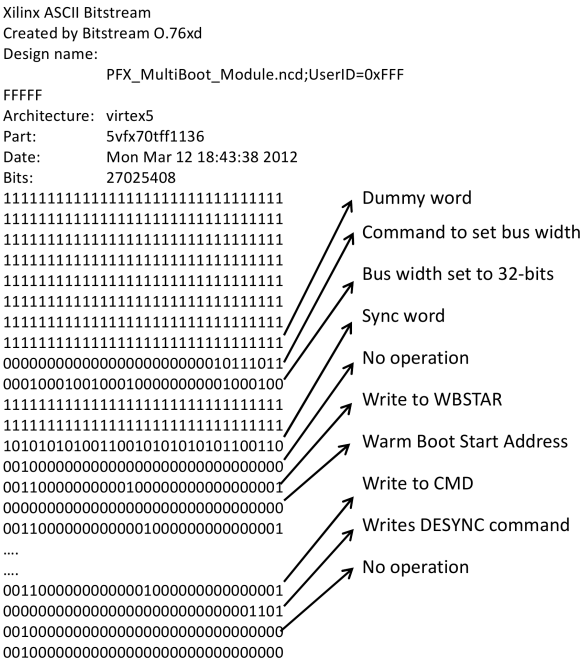


Figure 7: Sectional view of the ASCII converted bitstream contents

The state machine controlling the reconfiguration process directly drives the pins of the ICAP, as shown in Fig. 8. For the MultiBoot implementation, the configuration commands are simply supplied by the state machine. However, the state machine controller for the proposed architecture requires an interface to the BRAM from where the configuration data are read. The reconfiguration process is triggered by means of

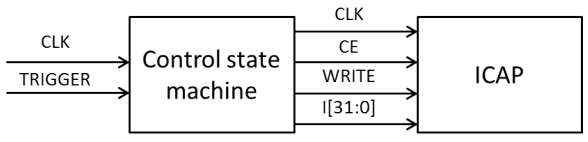


Figure 8: Control state machine interface to the ICAP

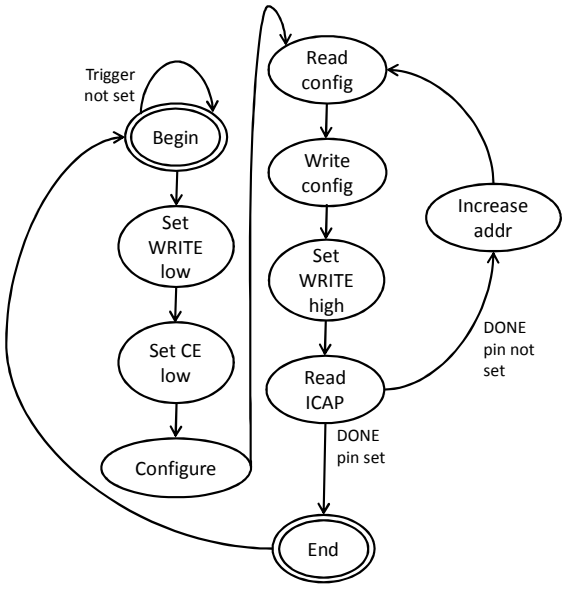


Figure 9: Hardware reconfiguration state machine flow diagram

an external trigger supplied by the user logic. This is followed by pulling the ***CE*** and ***WRITE*** pins low to enable the ICAP and write operations to the ICAP. The configuration process can then commence by reading the first configuration word from the BRAM, which is sent to the ICAP via the input port, ***I***, on each edge of ***CLK***. This process will continually monitor the bitstream to detect the ***DESYNC*** command string, which indicates a complete reconfiguration and releases the configuration logic. Alternatively, the ICAP output, ***O***, can also be used to detect the ***DESYNC*** command string. If the value on the output changes from *0xDF* to *0x9F*, the ***DESYNC*** command was received and the device is desynchronised [30]. If the ***DESYNC*** command is not received, the address pointer is increased to read the subsequent configu-

Table 1: ICAP pin description

| Pin Name | Type | Description |
|---|---|---|
| CLK | Input | ICAP interface clock |
| CE | Input | Active-low ICAP interface select |
| WRITE | Input | Selects read or write operation |
| I[31:0] | Input | ICAP write data bus |
| O[31:0] | Output | ICAP read data bus |
| BUSY | Output | Active-high busy status |

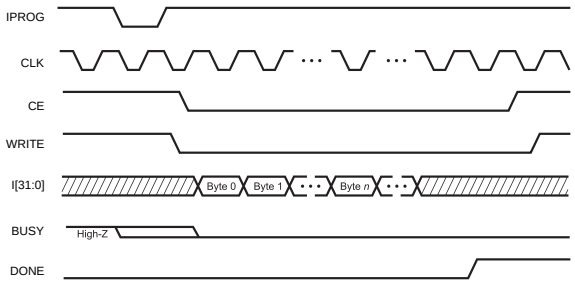


Figure 10: Timing diagram for ICAP data loading

ration word from the memory. This whole process is illustrated by the flow diagram of the state machine in Fig. 9, and the functionality of the ICAP pins is summarised in Table 1.

## 3.3 Reconfiguration timing

The ICAP port is closely related to the SelectMAP configuration interface. SelectMAP is an 8, 16 or 32-bit bidirectional external data bus interface to the configuration logic and can be used for both configuration and readback. The ICAP, as the name suggests, is an internal port with similar ports and timing as the SelectMAP interface. The timing for the ICAP is illustrated in Fig. 10. The ***IPROG*** signal prepares the device for configuration without resetting the configuration logic. If the chip is enabled and set for writing the configuration data, the data are written to the configuration memory one byte at a time. The ***DONE*** signal is not used during the configuration process and is set to a high impedance. After the configuration is done, the ***DONE*** flag is set. The hardware controlled reconfiguration process should adhere to this timing for proper reconfiguration.

# 4 EXPERIMENTAL SETUP

To evaluate the throughput of the BRAM-based architecture, two simple applications were created on a Xilinx® ML507 development board. The first application switches between two configurations by means of dynamic reconfiguration. Each configuration encapsulates a set of three parameters used to modify the frequency of a pulse width modulator connected

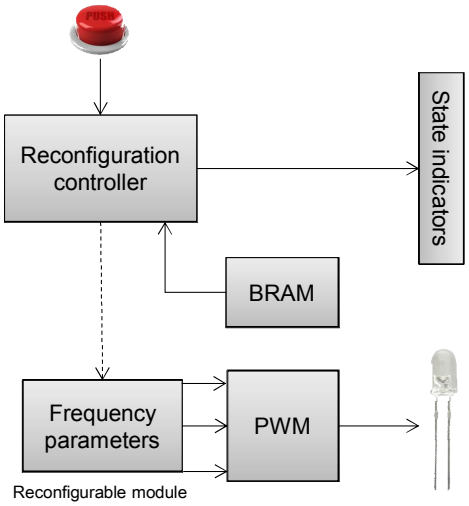


Figure 11: Block diagram of the first experimental design

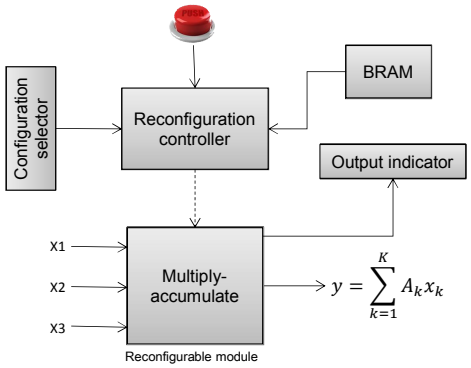


Figure 12: Block diagram of the second experimental design

to an external blinking light emitting diode (LED). The reconfiguration is triggered using an external push button and since the reconfiguration process is handled by a state machine, it is possible to verify each state of the configuration process using external LEDs.

Fig. 11 shows a block diagram depicting the interconnectivity between the components. The ***reconfiguration controller*** contains the MultiBoot state machine shown in Fig. 9 and is responsible for reconfiguring the ***frequency parameters*** controlling the ***pulse width modulation (PWM)***. The ***top level*** is only used to instantiate the components. The reconfiguration time is measured from the moment the external trigger goes high to when the active low LED, indicating the "DONE"-state of the configuration process, illuminates.

The second application further illustrates and evaluates the proposed architecture by storing nine different configurations of a multiply-accumulate (MAC) in the BRAM. Since more configurations have to be stored, it was decided to follow the difference-based reconfiguration design flow [31], instead of PR as in the first design. Difference-based reconfiguration compares two designs and only the differences are encapsulated in the configuration file. The benefit is significantly smaller configurations that result in faster reconfigu-

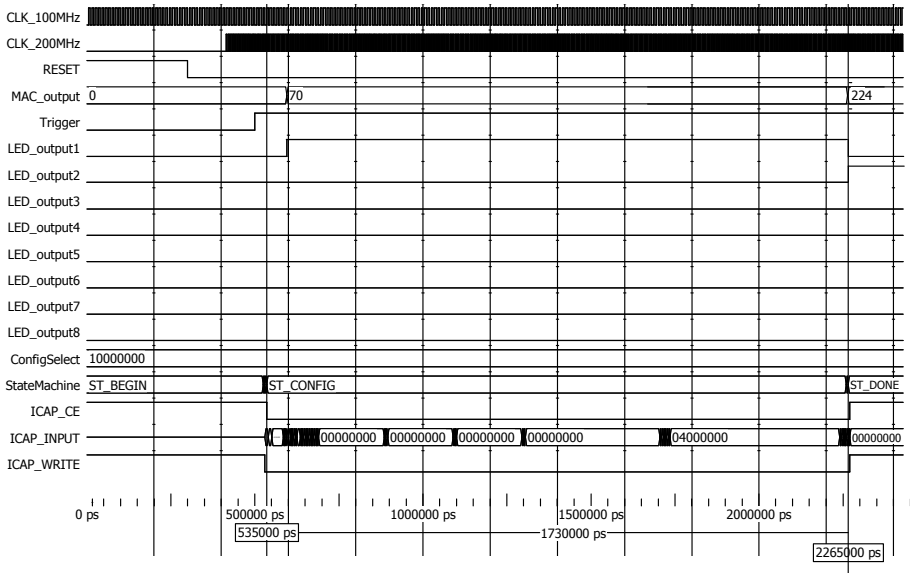


Figure 13: Timing diagram of the second experimental design

ration. However, difference-based reconfiguration has a couple of drawbacks and is not recommended for making large circuit changes, which is why Xilinx® recommends the PR design flow. Despite this, this is an ideal design to illustrate and verify the hardware controlled reconfiguration.

As seen in Fig. 12, the required configuration is selected using a dual in-line package (DIP) switch (***configuration selector***) before triggering the reconfiguration with a push button. The ***output indicator*** is connected to external LEDs and indicates the selected configuration, based on the output of the ***multiply-accumulate (MAC)***.

## 5 EXPERIMENTAL RESULTS

For the first application, the partial configuration data consists of 1658 32-bit words. Considering that the Virtex®-5 ICAP has a maximum theoretical throughput of 400 MBps, it is possible to transfer the entire bitstream through the ICAP within 16.58 $\mu$s.

It was shown in the literature that it is possible to overclock the ICAP above the Xilinx®-recommended 100 MHz [32, 33]. To investigate the maximum clock frequency of the ICAP without specifically designing for optimal clock propagation, the frequency of the ICAP was gradually increased. At twice the recommended clock frequency, the reconfiguration time of the first application can be reduced to 8.29 µs as shown in Fig. 14. At three times the recommended ICAP frequency the reconfiguration process failed.

Further investigation showed that the cause of this failure was not due to a limitation in the maximum clock frequency of the BRAM. As specified in the Xilinx® documentation [34], the maximum clock frequency of BRAM (v6.2) is 450 MHz. Hansen [32] further confirmed that it is theoretically possible to clock the ICAP at a maximum frequency of 580 MHz if a set-up time of 0.49 ns is assumed for configurable logic blocks' (CLB) flip-flops. In fact, numerous researchers have shown it is possible to overclock the ICAP above 300 MHz [32, 33]. It is thus evident that the 300 MHz limitation was due to sub-optimal design.

The bitstream of the second application contains 346 32-bit words. Keeping the ICAP clock at the maximum frequency for this design—200 MHz— it is possible to transfer the entire bitstream to the configuration memory in 1.730 $\mu$s. Fig. 13 shows the results of this application. Marker 1 is placed when the reconfiguration state starts and marker 2 is placed when the ***Done***-state is entered. The time-lapse between these two markers is measured to be 1.730 $\mu$s, which matches the calculated reconfiguration time. Also seen in the figure are the ICAP control signals, the configuration data sent to the ICAP, clock signals, configuration selected, MAC output and the LEDs indicating the current configuration. It is easy to see that the MAC output changes when the reconfiguration completes and the LED changes correspondingly.

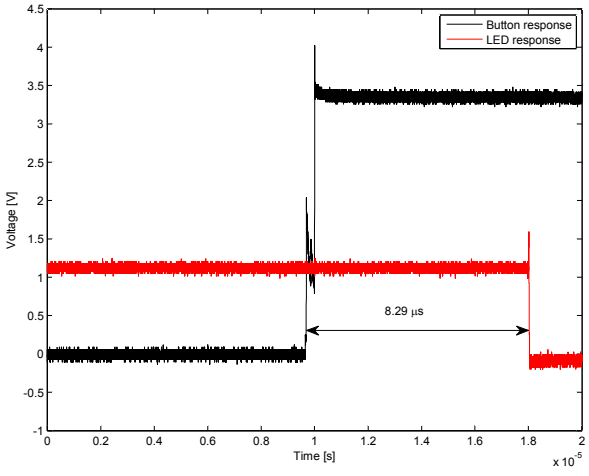


Figure 14: Reconfiguration response of the hardware controlled experimental set-up

Using difference-based reconfiguration yields a sig-

nificantly smaller configuration. For this particular application, the resulting configuration only contains 346 32-bit words. Since the reconfiguration time is directly related to the number of words in the configuration, a reconfiguration time of 1.730 µs can be measured using the experimental setup.

For the purpose of this discussion, consider an application with a control cycle of 50 µs. If a real-time system can be defined as one which "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time" [35], this implies that the reconfiguration process needs to fit within the remaining control cycle time after all other processing completes. Considering the worst reconfiguration time of the experimental setup, 16.58 µs, this leaves 33.42 µs for all other processing. Should this be insufficient, the buffer in the control cycle can be increased by overclocking the ICAP—reducing the reconfiguration time even further. If only small changes need to be made to the design, difference-based reconfiguration can be used to reduce the configuration size and obtain the best possible reconfiguration time.

## 6 OVERCOMING THE LIMITATION OF THE BRAM

As already mentioned, the primary drawback of the BRAM-based architecture is the size limitation. For example, the Virtex®-5 family of FPGAs from Xilinx® has between 936 and 16,416 Kb if all the BRAM blocks are used. This implies that only a subset of configurations can be stored.

Dynamic circuit specialisation (DCS) is a technique used to dynamically specialize a circuit implemented on an FPGA according to certain parameters. The idea is that each time a parameter changes, the device is reconfigured to fit this parameter [10]. The most commonly known DCS method is configuration swapping, which allows a section of an FPGA to be reconfigured by swapping the current configuration on the device with a new configuration. The individual configurations are generated by running the toolflow for each possible parameter value and storing the result off-line. This approach works fine for a small number of configurations, but the storage required grows exponentially with the number of parameters and the number of bits needed to represent each parameter. Representing three PID parameters using 16 bits each results in 48 configuration bits, giving a possible $2^{48}$ configurations - each requiring off-line storage space. Hence, generating configurations for each possible parameter and storing them off-line quickly becomes infeasible for real-time applications.

It is worth noting that the assumption is made that every single configuration is required by the application, whereas in practice this might not be the case. Only a subset of the configurations might be sufficient for nominal operation. In this case, it is possible to retain only the needed subset thus mitigating the storage-space limitation.

The solution would seem to generate all the configurations on-line and in real-time. This would result in a highly specialised configuration, because the configuration can be specialised for all possible parameters. However, running a conventional FPGA toolflow in real-time is computationally very expensive. A traditional toolflow typically takes minutes to hours to complete. This makes this approach only feasible for applications with slowly changing parameters.

As a result, Bruneel [10] proposed a method called parameterisable configuration that allows a bitstream to be specialised on-line according to a set of criteria. The fundamental concepts underlying this methodology are constant propagation and parameterisable configurations. The bits representing the FPGA configuration can be expressed as a Boolean function of a set of parameters (called 'tuning functions'). The result is a parameterised bitstream (PBS). Using a PBS specializer, any specific set of parameters can be evaluated to transform the PBS into a regular configuration, which can be evaluated at run-time. This process is illustrated in Fig. 15.

The toolset used by Bruneel enables automatic implementation of DCS by means of two methods [36]. The first expresses only the bits of the lookup tables in the configuration file as Boolean functions. These lookup tables are dubbed tunable lookup tables (TLUTs). All other configuration bits are static. The second method expands on this methodology and adds tunable routing bits. This new method is dubbed the tunable connections, or TCON, method. Even though TCON will lead to more compact solutions, TLUT will result in faster reconfiguration. Using these methods, Bruneel built several parameterisable systems. These include FIR filters, ternary content-addressable memory (TCAM), key-based encryption and DNA alignment systems that run on commercial FPGAs [15, 10, 37]. The specialisation was done using the embedded PowerPC and ICAP of the Virtex-II Pro®. It was determined that a coefficient change of the FIR filter can be reconfigured in 1.74 ms, whereas the content of the TCAM can be rewritten in 1.72 ms.

Unfortunately, Bruneel's toolset has an important limitation to consider when designing a real-time system. It was found that the specialisation process can take a significant amount of time and since this has to be done for each parameter change, this process will add significant overhead to the configuration process.

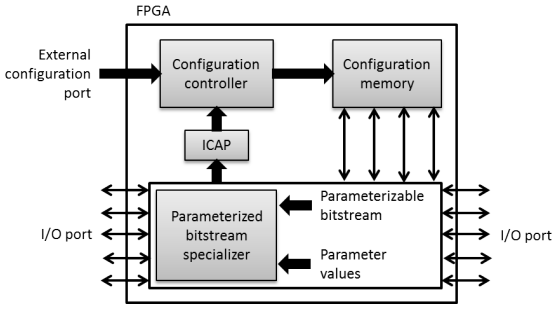


Figure 15: Parameterisable configuration specialisation architecture

The result is that parameterizing a module might not yield the expected benefits of reconfiguration. Despite these limitations for real-time applications, this concept of specializing a bitstream can be incorporated into the BRAM-based hardware-controlled reconfiguration architecture. This will allow the bitstream stored in the BRAM to be adapted according to specific conditions, overcoming the size limitation. The proposed architecture is shown in Fig. 16.

Even though parameterisation is currently unsuitable for real-time reconfiguration, methods that can be used in real-time are being investigated.

## 7 CONCLUSION

Despite the extensive research on improving the throughput of reconfigurable applications, the reconfiguration speed is limited by the processor bus connecting the individual components in the system. This severely limits the usage of reconfigurable computing in dynamic applications.

This paper focussed on the BRAM-based architecture proposed in the literature as a means to improve the throughput of reconfigurable applications, thus reducing reconfiguration time. Verifying the throughput was done by implementing and reconfiguring two simple designs using the proposed architecture. In the first application, three parameters similar to gains in PID control are reconfigured. These three parameters control the duty cycle of a blinking LED. In the second design, a multiply-accumulate (MAC) was implemented and the constants reconfigured. It was shown that a bitstream consisting of 1658 32-bit words can be transferred to the configuration memory within 8.29 µs, which is sufficient for real-time reconfiguration. This time can be reduced even further by proper constraints or by adding custom hardware, allowing the ICAP to be clocked above the recommended 100 MHz. Another alternative for reducing the reconfiguration time even further is to use the difference-based reconfiguration design flow, which reduces the size of the bitstream. However, this method can only be used when making small changes to a design.

The primary drawback of the proposed reconfiguration architecture is the limited amount of BRAM available to store configuration data. By proposing the use of a concept called parameterisable configuration, the hypothesis is that only a single parameterisable bitstream has to be stored in the BRAM, whereafter it is specialised for any required hardware set. Unfortunately, the current methods for specializing a bitstream could not yield any benefits due to the overhead from the specialisation process. Consequently it is unsuitable for real-time reconfiguration. Further research is under way to determine a suitable real-time specialisation method.

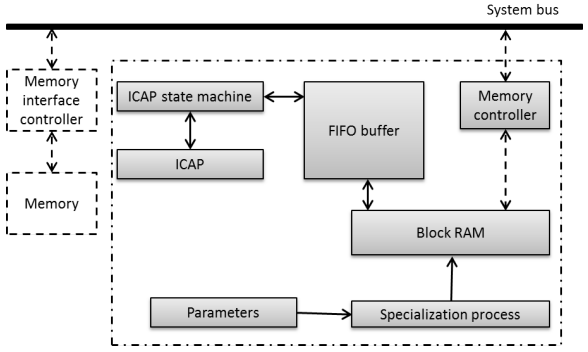


Figure 16: Block diagram of the proposed architecture

Due to the complexity of the FPGA routing, it is estimated that only lookup table (LUT) contents will be specialisable. Since LUTs are the primary components of an FPGA, this implies that most applications implemented on an FPGA would be able to benefit from this specialisation. Additionally, by using distributed arithmetic most multiply-accumulate (MAC) instructions can also be reconfigured, which will be greatly beneficial for real-time applications. This is due to MACs being the foundation of many digital implementations, including filters and PID control.

## ACKNOWLEDGEMENTS

This research was done under the Technology and Human Resources for Industry Programme (THRIP) and Oppenheimer Memorial Trust Grant (Ref. 19328/01).

## REFERENCES

[1] M. Liu, W. Kuehn, Z. Lu and A. Jantsch. "Run-time partial reconfiguration speed investigation and architectural design space exploration". In *International conference on field programmable logic and applications, 2009.*, 2, pp. 498 –502. Sept 2009. ISSN 1946-1488. DOI `http://dx.doi.org/10.1109/FPL.2009.5272463`.

[2] K. Van der Bok, R. Chaves, G. Kuzmanov, L. Sousa and A. V. Genderen. "Dynamic FPGA reconfigurations with run-time region delimitation". In *Proceedings of the 18th annual workshop on circuits, systems and signal processing (ProRISC)*, pp. 201–207. 2007.

[3] K. Compton and S. Hauck. "Reconfigurable computing: A survey of systems and software". *ACM computing surveys*, vol. 34, no. 2, pp. 171–210, June 2002. DOI `http://dx.doi.org/10.1145/508352.508353`.

[4] G. Estrin. "Parallel processing in a restructurable computer system". *IEEE transactions on electronic computers*, vol. 12(5), pp. 747–755, 1963. DOI `http://dx.doi.org/10.1109/PGEC.1963.263558`.

[5] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer and W. Luk. "Reconfigurable computing: Architectures and design methods". In *IEE proceedings–Computers and digital techniques*, vol. 152, pp. 193–207. 2005. DOI `http://dx.doi.org/10.1049/ip-cdt:20045086`.

[6] E. Kusse and J. M. Rabaey. "Low-energy embedded FPGA structures". In *Proceedings of the 1998 international symposium on low power electronics and design (ISLPED'98)*, pp. 155–160. 1998. DOI `http://dx.doi.org/10.1145/280756.280873`.

[7] G. Stitt, F. Vahid and S. Nematbakhsh. "Energy savings and speedups from partitioning critical software loops to hardware in embedded systems". *ACM*

*transactions on embedded computer systems*, vol. 3, no. 1, pp. 218–232, February 2004. DOI http://dx.doi.org/10.1145/972627.972637.

[8] J. Leonard and W. Mangione-Smith. "A case study of partially evaluated hardware circuits: Key specific DES". *Proceedings of the international workshop on field programmable logic and applications (FPL)*, pp. 151–160, 1997. DOI http://dx.doi.org/10.1007/3-540-63465-7_220.

[9] S. Singh, J. Hogg and D. McAuley. "Expressing dynamic reconfiguration by partial evaluation". *Proceedings of the IEEE symposium on FPGAs for custom computing machines (FCCM)*, 1996.

[10] K. Bruneel. *Efficient circuit specialization for dynamic reconfiguration of FPGAs*. Ph.D. thesis, Faculty of Engineering Sciences and Architectures, Ghent University, Belgium, 2011.

[11] C. Claus, F. Muller, J. Zeppenfeld and W. Stechele. "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration". In *Parallel and distributed processing symposium, 2007. IPDPS 2007.*, pp. 1–7. March 2007. DOI http://dx.doi.org/10.1109/IPDPS.2007.370362.

[12] K. Bruneel, F. M. A. Abouelella and D. Stroobandt. "Automatically mapping applications to a self-reconfiguring platform". *Proceedings of design, automation, and test Europe*, pp. 964–969, 2009.

[13] S. Ichikawa and S. Yamamoto. "Data dependent circuit for subgraph isomorphism problem". *Proceedings of the international conference on field programmable logic and applications (FPL)*, pp. 1068–1071, 2002. DOI http://dx.doi.org/10.1007/3-540-46117-5_109.

[14] P. Zhong, M. Martonosi, P. Ashar and S. Malik. "Accelerating Boolean satisfiability with configurable hardware". *Proceedings of the IEEE symposium on FPGAs for custom computing machines (FCCM)*, pp. 186–195, 1998. DOI http://dx.doi.org/10.1109/FPGA.1998.707896.

[15] K. Bruneel, P. Bertels and D. Stroobandt. "A method for fast hardware specialization at run-time". In *International conference on field programmable logic and applications, 2007. FPL 2007*, pp. 35–40. Aug 2007. DOI http://dx.doi.org/10.1109/fpl.2007.4380622.

[16] J. Eldredge and B. Hutchings. "RRANN: The run-time reconfiguration artificial neural network". In *Proceedings of the IEEE 1994 custom integrated circuits conference*, pp. 77 –80. May 1994. DOI http://dx.doi.org/10.1109/CICC.1994.379763.

[17] J. Eldredge and B. Hutchings. "Run-time reconfiguration: A method for enhancing the functional density of SRAM-based FPGAs". In *Journal of VLSI signal processing*, pp. 67–86. 1996. DOI http://dx.doi.org/10.1007/bf00936947.

[18] M. Wirthlin and B. Hutchings. "Improving functional density using run-time circuit reconfiguration [FPGAs]". *IEEE transactions on VLSI systems*, vol. 6, pp. 247–256, 1998.

[19] G. Economakos and C. Economakos. "A run-time reconfigurable fuzzy PID controller based on modern FPGA devices". In *Proceedings of the 2007 Mediterranean conference on control and automation*, pp. 1–6. Jun 2007. DOI http://dx.doi.org/10.1109/MED.2007.4433812.

[20] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner and J. Becker. "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput". In *International conference on field programmable logic and applications, 2008*, pp. 535–538. Sept. 2008. DOI http://dx.doi.org/10.1109/FPL.2008.4630002.

[21] C. Claus, F. Altenried and W. Stechele. "Dynamic partial reconfiguration of Xilinx FPGAs lets systems adapt on the fly: A video-based driver assistance application demonstrates effective use of situation-adaptive hardware". *Xcell journal*, vol. 70, pp. 18–23, First Quarter 2010.

[22] S. Liu, R. N. Pittman and A. Forin. "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller". In *Proceedings of the 18th annual ACM/SIGDA international symposium on field programmable gate arrays*, FPGA '10, pp. 292–292. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-911-4. DOI http://dx.doi.org/10.1145/1723112.1723190.

[23] Xilinx, Inc. "OPB HWICAP (v1.00.b) Product Specification". Tech. Rep. DS280, Xilinx, Inc., July 2006.

[24] Xilinx, Inc. "LogiCORE IP XPS HWICAP (v5.00.a) product specification". Tech. Rep. DS586, Xilinx, Inc., July 2010.

[25] S. Bayar and A. Yurdakul. "Dynamic partial self-reconfiguration on Spartan-III FPGAs via a parallel configuration access port (PCAP)". In *2nd HiPEAC workshop on reconfigurable computing*, vol. 8, pp. 10–20. 2008.

[26] C. Claus, F. Muller and W. Stechele. "Combitgen: A new approach for creating partial bitstreams in Virtex-II Pro devices". In *Workshop on reconfigurable computing proceedings (ARCS 06)*, pp. 122 – 131. March 2006.

[27] Xilinx Inc. "Partial reconfiguration user guide". User Guide 702, Xilinx Inc., Apr 2013. UG702.

[28] Xilinx. "Multiboot with Virtex-5 FPGAs and Platform Flash XL". Application note, November 2008. XAPP1100.

[29] Xilinx. "Virtex-5 FPGA configuration user guide ", Aug 2010. UG191 (v3.9.1).

[30] S. Lamonnier, M. Thoris and M. Ambielle. "Accelerate partial reconfiguration with a 100% hardware solution". *Xcell journal*, vol. 79, pp. 44–49, 2012.

[31] E. Eto. "Difference-based partial reconfiguration". Application note XAPP290, Xilinx, December 2007.

[32] S. Hansen, D. Koch and J. Torresen. "High speed partial run-rime reconfiguration using enhanced ICAP hard macro". In *2011 IEEE international symposium on parallel and distributed processing workshops and PhD forum (IPDPSW)*, pp. 174 –180. may 2011. ISSN 1530-2075. DOI http://dx.doi.org/10.1109/IPDPS.2011.139.

[33] J. C. Hoffman and M. S. Pattichis. "A high-speed dynamic partial reconfiguration controller using direct memory access through a multiport memory controller and overclocking with active feedback". *International journal of reconfigurable computing*, vol. 2011, p. 10, 2011. DOI http://dx.doi.org/10.1155/2011/439072.

[34] Xilinx, Inc. "LogiCORE IP block memory generator v6.2". Data Sheet DS512, Xilinx, Inc., June 2011.

[35] J. Martin. *Programming real-time computer systems*. Prentice-Hall, 1965.

[36] K. Bruneel and D. Stroobandt. "TROUTE: A reconfigurability-aware FPGA router". *Lecture notes in computer science*, vol. 5992, pp. 207–218, 2010. DOI `http://dx.doi.org/10.1007/978-3-642-12133-3_20`.

[37] T. Davidson, F. Abouelella, K. Bruneel and D. Stroobandt. "Dynamic circuit specialisation for key-based encryption algorithms and DNA alignment". *International journal of reconfigurable computing*, vol. 2012, Article ID 716984, p. 13, 2012.