Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

A survey on reducing reconfiguration cost: reconfigurable PID control as a special case ⋆

Rikus R. le Roux * **George van Schoor** **
Pieter A. van Vuuren *

* *School of Electrical, Electronic and Computer Engineering, North-West University, Potchefstroom, South-Africa (e-mail: rikus.leroux,pieter.vanvuuren@nwu.ac.za)*
** *Unit for Energy Systems, North-West University, Potchefstroom, South-Africa (e-mail: george.vanschoor@nwu.ac.za)*

Abstract: Reconfigurable computing is a paradigm in computing architecture that refers to the practice of using interchangeable hardware modules to enhance the performance of conventional Von-Neumann style computing. Despite the numerous advantages of reconfiguration, it is only suitable for quasi-static applications with slowly changing reconfiguration criteria. In general, it is only advantageous to reconfigure if the execution time exceeds reconfiguration time. Since the execution time of real-time systems are quite limited, the control of dynamic non-linear systems are typically not reconfigured. Instead, adaptivity is mostly gained from reading coefficients or gains from memory. Where different controller architectures are required, the route of parallel implementation could be taken, switching between architectures. The drawback of this approach is an increase in area required to implement the controllers. Reconfiguration on the other hand could allow the different control architectures to be swapped on the fly without interrupting operation of the controller, while minimizing the area required. Unfortunately, this process is limited by the overhead introduced by the reconfiguration. Even though various survey papers exist on the topic of reconfiguration, none really focus on methods to reduce the cost of reconfiguration. This survey summarizes different means of reducing configuration overhead in an attempt to allow reconfiguration of applications with limited execution time. A block RAM-based (BRAM) architecture is proposed as the optimal architecture for reconfiguring dynamic applications. As an example, this architecture is used to discuss the methodology used to design a reconfigurable PID controller.

Keywords: FPGA, survey, reconfiguration, bitstream, PID

1. INTRODUCTION

Reconfigurable computing allows improving system performance by utilizing customizable hardware. Initially, this was done by a modular design where a hardware module can be substituted with another to perform a specialized function. Field-programmable gate arrays (FPGAs) allow their hardware to be changed and are thus a viable option to be used in reconfigurable computing systems. In fact, some vendors incorporate a feature called dynamic partial reconfiguration that allows a section of the FPGA to be reconfigured while the rest of the device remains operational. Most of Xilinx®'s FPGAs from the Virtex™-II series incorporate this feature with the addition of the internal configuration access port (ICAP) that allows the developer direct access to the configuration memory.

Reconfigurable computing is not only used to improve system performance, but also reduces power consumption (Kusse and Rabaey (1998)) and component count (Todman et al. (2005); Stitt et al. (2004)). The improvement in system performance is due to the circuit being tailored for the specific application, which improves the functional density. Functional density is a metric to measure the composite benefits obtained by a specialization technique (Wirthlin and Hutchings (1998)). It is defined as the cost of implementing the computation in hardware and measures the computational throughput of the hardware resources in operations per second:

$$D = \frac{1}{C} = \frac{1}{AT}, \tag{1}$$

with D the functional density, C the cost of a computation, T the total execution time and A the total area required to implement the computation in hardware. The total execution time includes the execution time of the hardware (T_{hw}), the time required to generate the new hardware (T_{gen}) and the time to configure the FPGA (T_{conf}). The equation can thus be expanded to:

$$D = \frac{1}{C} = \frac{1}{AT} = \frac{1}{A(T_{hw} + T_{gen} + T_{conf})}. \tag{2}$$

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

Despite the various advantages of reconfigurable computing, it has several drawbacks and limitations. One of the most significant drawbacks is the high overhead introduced by the reconfiguration process. The two primary contributors of this overhead are the placement and routing (PAR) methods used by conventional design tools. Due to this, only quasi-static applications gain an advantage from reconfiguration. Various attempts are made in an attempt to mitigate the configuration overhead to allow reconfiguration of applications with more dynamic behaviour, such as adaptive control.

This paper supplements other survey papers, such as Todman et al. (2005), Bondalapati and Prasanna (2002), Compton and Hauck (2002), Schaumont et al. (2001), Hartenstein (2001) and Papadimitriou et al. (2011), by listing various research techniques aimed towards reducing the configuration overhead. It starts off in section 2 by discussing instances where reconfiguration has been applied to control theory. Section 3 discusses the methods used to reduce reconfiguration overhead. The term "hardware controlled reconfiguration" of section 4 is used in this paper to describe the reconfiguration using block RAM (BRAM)-based architectures. The design of a gain scheduled reconfigurable PID controller is discussed in section 5.

2. RECONFIGURABLE COMPUTING IN CONTROL

As mentioned in the previous section, reconfiguration is only suitable for quasi-static applications. Typical examples include: key specific data encryption standard (DES) (Leonard and Mangione-Smith (1997)), sub-graph isomorphism (Ichikawa and Yamamoto (2002)), Boolean satisfiability (SAT) (Zhong et al. (1998)), adaptive filters (Bruneel et al. (2007)), reconfigurable artificial neural networks (Eldredge and Hutchings (1994)) and FSK modulation (Giovagnini and Marzo (2012)).

Eldredge and Hutchings (1994) and [1996] showed that run-time reconfiguration can be used to enhance the functional density of an artificial neural network. This network was dubbed the Run-time Reconfigurable Artificial Neural Network (RRANN) and reconfiguration was used to adapt each stage of the backpropagation algorithm to suit specific requirements. The reconfiguration process is controlled using an external processor that adds between 14 and 21 ms to the execution time and the configuration data are stored on a host computer. It was shown that reconfiguration expands the number of neurons that can be implemented, which in turn increases the functional density. It was stated that an FPGA can be used to further improve the reconfiguration time by storing multiple configurations and switching between configurations using external pins. However, this approach will further reduce the functional density due to the additional overhead of the implementation.

Zhao et al. (2005), Chan et al. (2004) and Chan et al. (2007) each derived a static PID controller for implementation on an FPGA. By far the most popular implementation of PID control is distributed arithmetic (Sen et al. (2007a); Zhou and Shi (2011a,a)). This is due to the fact that multipliers are seen as an expensive resource on FPGAs and since FPGAs are memory rich, distributed arithmetic allows for optimal usage of the resources. In fact, any multiply-accumulate (MAC) instruction on an FPGA can by implemented using distributed arithmetic (other examples can be found in Sen et al. (2007b) and Zhou and Shi (2011b)). However, most of these controllers are static in nature.

In order to allow for more intelligent control, fuzzy logic-based controllers can be used. Examples of such an approach can be found in Lago et al. (1998), Sánchez-Solano et al. (2002) and Vuong et al. (2006). A different approach is to use fuzzy logic in conjunction with a PID controller to refine the parameters (Zhao et al. (1993); Li and Hu (1996); Tipsuwanporn et al. (2004)). This allows for an adaptive PID controller, but in most cases only the constants are adjusted while the rest of the controller remains fixed.

Kim (2000) shows an example of a fuzzy logic controller being segmented so that it can be implemented on an FGPA using run-time reconfiguration. This is especially handy if the controller is too large for the resources available on the FPGA.

An attempt to reconfigure a PID controller for control purposes can be found in Economakos and Economakos (2007). Again, fuzzy logic was used to refine the PID parameters, but contrary to Tipsuwanporn's method, a micro-controller was used to reconfigure the parameters using configuration data stored in the on-chip block RAM (BRAM). The configuration data have to be transferred to the configuration memory via the internal configuration access port, or ICAP. The smallest configuration segment that can be transferred through the ICAP is defined as a frame, which consists of 1312 bits. By placing a set of PID parameters inside a frame, Economakos showed that the reconfiguration time for each parameter change is 0.41 μs, while the fuzzy controller takes 0.75 μs to refine the parameters. The drawback of this implementation is the large number of resources required to implement the softcore processor. A softcore processor, such as the MicroBlaze™ from Xilinx®, requires FPGA resources to implement. A hardcore processor on the other hand has dedicated hardware embedded into the die. This implies that this particular implementation is not scalable to multiple PID controllers. Another drawback is that only the gains of the controller can be reconfigured.

3. REDUCING RECONFIGURATION COST

As can be seen from (2), any form of reconfiguration adds additional cost (C) to a system, either in terms of area (A) or time. In reconfiguration terms, the time required to both generate new hardware (T_{hw}) and to configure the FPGA (T_{conf}) is referred to as the specialization time (Bruneel et al. (2007)). The time required to generate new hardware is represented by T_{gen}. The fuzzy specializer used by Economakos and Economakos (2007) (mentioned in section 2) contributes to T_{gen}, thus increasing functional density even more.

As already mentioned, the two primary factors contributing to T_{gen} is the placement and routing (PAR) required to generate instance-specific configurations. Traditionally, negotiation-based algorithms are used to determine the

optimal placement and routing, which adds a significant overhead to the cost of dynamic reconfiguration. As shown in Table 1 and Table 2 extensive research aim to minimize the specialization cost using advanced PAR techniques.

Of particular interest for this paper is the methods used to improve the throughput of the system after PAR. Claus et al. (2007) states that after placement and routing, there are three methods to reduce configuration cost:

- Reducing the bitstream size
- Optimizing the way the bitstreams are written to the configuration memory
- Optimizing the transfer of the bitstream from the memory to the ICAP

These methods aim to improve the throughput of the system to rival that of the ICAP. This allows the ICAP to process new data every clock cycle. As will be shown in section 3.1, the first two methods both reduce the size of the bitstream. The general idea is that a smaller bitstream requires less time to be transferred and by changing the ay the configuration data is written to memory, T_{conf} can be greatly reduced.

The transfer of the configuration data to the ICAP is governed by the reconfiguration architecture used. By making changes to the architecture, it is possible to reduce T_{conf}. The different adaptations to the architecture are discussed in section 3.2.

3.1 Bitstream generation

The bitstream contains the configuration data of the FPGA and can be generated on-line and off-line (Bruneel and Stroobandt (2008b)). Off-line generation implies that the bitstreams are generated independently from the FPGA, usually with conventional design tools. These bitstreams can contain the information required to configure the FPGA with an initial configuration, or partial configuration data used during dynamic partial reconfiguration. For a limited set of configurations, these bitstreams can be stored in on-board memory from where the FPGA can be reconfigured. Applications where this technique have been successful include DNA sequencing (Davidson et al. (2012)), neural networks (Wirthlin and Hutchings (1998)) and automatic target recognition (Villasenor et al. (1996)).

On-line bitstream generation refers to generating a bitstream dynamically while the FPGA is running and refers to specializing the initial bitstream for a specific application. It is possible to use conventional tools, but this induces a significant amount of configuration manager (CM) overhead. This adds to T_{gen} in (2), which reduces the functional density advantage due to the additional time spent generating the new hardware. The improvement obtained in the circuit should always justify the configuration time (Leonard and Mangione-Smith (1997); Singh et al. (1996)). For this reason, only applications with quasi-static behaviour can benefit from this method, due to their pseudo-static dynamics and small changes in the circuit during reconfiguration.

Table 3 summarizes some of the changes made in the bitstream to improve the reconfiguration throughput by minimizing the CM overhead.

3.2 Reconfiguration throughput

Reconfiguration throughput refers to the maintainable speed of the system between the memory housing the partial bitstream and the internal configuration access port (ICAP). As the name suggests, the ICAP is an internal port allowing access to the configuration registers (Xilinx (2010)). Assuming that the ICAP is capable of processing data every clock cycle, the maximum theoretical throughput (MTT) is defined by Claus et al. (2007):

$$MTT = \frac{IDIW}{Clock\ period}. \tag{3}$$

IDIW is the ICAP data input width, which is 8 and 32-bit for the Virtex®-II and 5 respectively. The maximum recommended clock frequency for the ICAP is 100 MHz. Substituting these values into (3) gives an ICAP *MTT* of 800 Mbps for the Virtex®-II and 3.2 Gbps for the Virtex®-5 to 7.

Unfortunately, the throughput of the system is lower than that of the ICAP, due to the bus-architectures most commonly used, making it impossible for the ICAP to process new data every clock cycle. Table 4 compares various architectures from literature used to improve the throughput. Each architecture is comparable to the Xilinx® default ICAP controllers listed at the top of the table. The *buffer* is used to store the bitstream locally after being fetched from external *memory*. The *processor* can be used to initiate and control the reconfiguration process. *Direct memory access* (DMA) allows the ICAP controller to directly access the bitstream located in the external memory, without processor intervention. Coupling DMA with *streaming* (or burst-modes) allows the throughput of the system architecture to rival that of the ICAP. The architectures listed without a processor are equipped with hardware reconfiguration controllers used to control the reconfiguration process and might result in a pure hardware reconfiguration solution. Even though not prominent in the table, some architectures utilize bitstream *compression* to reduce the size of the bitstream. *Readback* refers to the capability of the reconfiguration controller to read the current configuration of the device.

The work listed in the table mostly refer to improving the general architecture of the system in order to improve throughput. However, various attempts is made to improve the MTT of the ICAP. This is mostly done using overclocking techniques. This implies clocking the ICAP at a frequency higher than the recommended 100 MHz (Claus et al. (2010); Hansen et al. (2011); Hoffman and Pattichis (2011); Shelburne et al. (2010)). However, the reason for the 100 MHz recommendation is that this is the maximum frequency at which Xilinx® can guarantee stable reconfiguration. At higher frequencies, errors can occur due to voltage fluctuations in the die. Error checking in the form of CRC validation should thus be included in the design.

4. HARDWARE CONTROLLED RECONFIGURATION (HCR)

Bus-based architectures are most commonly used for reconfiguration to connect the various different components

Table 1. Placement research

Placement technique	Description
Quadratic (Xu and Khalid (2005))	Quadratic placement tries to minimize total squared length by solving linear equations.
Min-cut (Shi (2009); Lee and Raahemifar (2008))	The min-cut optimization technique uses recursive partitioning to divide a net-list of circuits into increasingly smaller sub-circuits and maps these smaller circuits onto the FPGA. This leaves the highly connected blocks in one partition thus decreasing placement cost.
Parallel placement (Shi (2009))	The rapid development of multi-core CPUs make parallelization an appealing solution for providing fast placements. Multiple logic blocks are routed in parallel.
Hybrid algorithms (Lee and Raahemifar (2008); Shi (2009))	Hybrid algorithms are usually multi-stage placement algorithms that combine multiple placement techniques, one of which is usually simulated annealing.
Simulated annealing (Kirkpatrick et al. (1983))	Simulated annealing is the most widely used algorithm for placement on an FPGA and forms the basis of most placement algorithms. Simulated annealing placement mimics the annealing process used to gradually cool molten metal. The most optimal placement is obtained by initially placing random logic blocks and swapping the blocks to reduce the cost.
Versatile place and route (VPR) (Betz and Rose (1997); Lam and Delosme (1988))	VPR is a time-driven simulated annealing placement and routing technique that is based on PathFinder and includes enhancements that improve run-time and quality. Its annealing schedule is based on calculated parameters, rather than fixed start and end temperatures.
Ultra fast placement (UFP) (Sankar and Rose (1999))	UFP aims to improve on VPR by combining VPR with a multi-level clustering strategy. This improves the scalability of the placer at the cost of an increase in wirelength.
Analytical placers (Chan and Schlag (2003); Xu (2009); Xu et al. (2011))	Analytical placers aim to improve scaling issues without a reduction in quality by creating a smooth placement function that approximates routed wirelength. Analytic placers tackle the placement problem from the top-down and considers global connectivity, rather than iteratively evaluating small-scale modifications.
Hardware-assisted simulated annealing (Wrighton and DeHon (2003))	Each space where a lookup-table (LUT) could reside is assigned its own processing element. The processing element is responsible for keeping track of which LUT it contains, as well as the connectivity to its neighbours. The processing element is also aware of its position and an estimate is kept of the connected LUTs.

Table 2. Routing strategies

Routing technique	Description
Rip-up and reroute (Dees Jr and Smith II (1981); Brown et al. (1992))	Rip-up and reroute was proposed to remedy the unrouted nets of other techniques. The success of the routes are dependent on the order in which they are routed. Additional cost functions can be added to ensure critical paths are routed first.
PathFinder (McMurchie and Ebeling (1995))	Pathfinder is a router that aims to find a balance between performance and routability. An iterative algorithm is used to negotiate which signal needs a resource the most. Delay is minimized by allowing critical signals a higher preference in resource-allocation.
Versatile place and route (VPR) (Betz and Rose (1997); Lam and Delosme (1988))	As already discussed, VPR is an optimization and extension to PathFinder and is arguably the most popular placement and routing technique.
Stochastically (Lin et al. (2010); Lin and Gamal (2008))	The idea is to use stochastic methods to locate near-optimal placement solutions without exhaustively enumerating all design points.
Parallel placement (Chan et al. (2000); Fatima and Rao (2008))	Parallel placement aims to increase performance by implementing standard negotiation-based routing algorithms in parallel without a reduction in the quality of the results.
Hardware assisted (DeHon et al. (2002))	Adding hardware to the routing network, assists the routing network in finding free routes to be used in the routing process.

of the system. In fact, even the configuration controllers provided by Xilinx® (HWICAP) are bus-based, as illustrated in Fig. 1. The major drawback of these architectures is that the bus adds additional overhead to the configuration process, which increases configuration time. It was shown by Bruneel (2011) that almost 20% of the specialization overhead is spent in the Xilinx® driver function. Another drawback of these architectures is that multitasking is limited by the use of the bus and while the bus is in use for reconfiguration, operation in the other modules attached to the bus is suspended (Hoffman and Pattichis (2011)).

As seen in the preceding section, various additions to this architecture, such as DMA and burst modes, either aim to mitigate the overhead induced by the bus, or to remove the

Table 3. Research in bitstream development

Bitstream generation	Description
Lean versions (Lysecky et al. (2006, 2004))	A just-in-time approach is taken to dynamically convert software binary instructions onto FPGAs. These compilers require lean versions of the conventional mapping, placement and routing algorithms to improve mapping, placement and routing times respectively.
Reducing quality (Sankar and Rose (1999))	Reducing the quality of the placement and routing allows quicker convergence of the tools that results in quicker bitstream generation. In this context, reduction in quality is defined as an increase in the wiring area of the circuit, a reduction in the operating speed of the circuit, greater wirelength of the mapped circuit, and an unnecessary increase in resource-utilization.
Partial evaluation (Hauck and DeHon (2008))	Partial evaluation is a process that automates specialization in software and hardware and aims to produce a circuit that performs faster than the original.
Generic netlists (Leonard and Mangione-Smith (1997); Singh et al. (1996); Steiner et al. (2011))	A generic netlist is a netlist not physically mapped to a device. This research aims to improve mapping of generic FPGA netlists to physical netlists for real architectures.
Reusing place and route (McKay et al. (1998); Bruneel et al. (2007))	Placement and routing take a significant amount of time. By reusing the place and route netlists saves configuration time.
Constant multiplication (Wirthlin (2004))	Constant multiplication is a technique used to reduce FPGA resource requirements by exploiting constant-specific optimizations.
Tunable lookup tables (TLUTs) (Bruneel et al. (2009b); Bruneel and Stroobandt (2008b,a, 2010); Bruneel et al. (2007, 2009a))	The configuration bits of the lookup tables are expressed as a Boolean function of the parameter inputs. All the other configuration bits are static. These Boolean functions are evaluated at run time to produce a new configuration. This leads to fast reconfiguration, but is not the most compact implementation. A tunable mapper is used to map a gate-level circuit into these tunable lookup tables.
Tunable connections (Bruneel and Stroobandt (2010, 2008b); Bruneel (2011))	These aim to expand on TLUTs by also expressing the routing configuration bits of an FPGA as a Boolean function. This allows for faster rerouting.
Combitgen (Claus et al. (2006, 2007))	Combitgen is a technique that combines the advantages of existing Xilinx partial dynamic reconfiguration flows. It also utilizes redundancy to reduce the number of frames in the bitstream without an increase in quantity.
Compression (Bayar and Yurdakul (2008); Liu et al. (2010))	This aims to improve reconfiguration time by reducing the bitstream size either by eliminating the redundant frames in the bitstream or by traditional compression techniques.
Shift register lookup table (SRL) (Bruneel (2011); Heyse et al. (2012))	The SRL capability of the Xilinx Virtex-series FPGAs are used to reconfigure the functionality of the LUTs. SRLs are LUTs whose elements are organised as a shift register. By shifting the data into the SRL, the functionality is changed.

use of the bus completely. Even though these alterations enable throughputs rivalling that of the ICAP, most of these architectures suffer from configuration latency due to the multiple clock cycles required to transfer the initial configuration frames to the local memory where it can be used by the ICAP.

Liu et al. (2010) aimed to minimize the configuration overhead by proposing an architecture incorporating streaming, compression and DMA into an intelligent ICAP controller. The proposed architecture is shown in Fig. 2. It utilizes a system bus to connect the independent ICAP state machine to the external memory, but are equipped with units capable of accessing the memory directly, thus eliminating the need for a bus controller. The ICAP state machine issues the reconfiguration command. It is then the responsibility of the DMA to fetch the partial bitstream from the external memory and load it into the localized FIFO buffer, from where it it used by the ICAP controller to reconfigure the device via the ICAP port. These architectures aim to minimize the large overhead associated with bus-transfers.

Despite the fact that the architecture proposed nearly saturates the ICAP, the DMA and compression adds configuration overhead of 17 and 6 clock cycles respectively. The BRAM-based architectures illustrated in Fig. 3 aim to address this issue by using dedicated BRAM to store the configuration data. Evidently, the BRAM should be large enough to hold the data. Unfortunately, the BRAM available on the FPGA is extremely limited. For configuration data too large to fit inside the BRAM, the data has to be transferred from external memory to local BRAM using the bus.

The architectures proposed by Liu et al. (2010) can be classified as:

- DMA-based architectures (Fig. 2)
- Localized architectures (Fig. 3)

Since these two architectures have no bus-overhead and no configuration latency, T_{conf} is minimized. Using these architectures, it is possible to implement a reconfigurable PID controller with the least reconfiguration overhead.

Table 4. Architectures to improve reconfiguration throughput

Method	Buffer	Bus type	Stream	Memory	DMA	Processor	Read	Compress
OPB_HWICAP	DP_RAM	OPB	N	DDR SDRAM	N	MB	Y	Possible
XPS_HWICAP	FIFO	PLB	Y	DDR SDRAM	N	PPC	Y	Possible
OPB_HWICAP	BRAM	OPB	N	DDR SDRAM	N	PPC	Y	Possible
XPS_HWICAP	FIFO	PLB	Y	DDR SDRAM	N	MB	Y	Possible
Cuoccio et al. (2008)	FIFO	Any	N	Slice BRAM	Y	PPC/MB	Y	N
Claus et al. (2007)	FIFO	PLB	N	SDRAM	Y	PPC	N	Combitgen
Claus et al. (2008)	BRAM	PLB	N	DDR	Y	PPC	Y	N
Claus et al. (2010)	FIFO	PLB	Y	DDR SDRAM	Y	PPC	N	N
Sedcole et al. (2006)	FIFO	OPB	N	DDR	N	PPC/MB	Y	N
Van der Bok et al. (2007)	FIFO BRAM	N	N	Y	Y	None	N	N
Liu et al. (2010)	FIFO	Any	Y	SRAM	Y	None	N	Proprietary
Liu et al. (2009)	FIFO FIFO None	PLB PLB None	N Y N	DDR DDR BRAM	Y Y N	PPC/MB PPC/MB None	Y Y Y	N N N
Lamonnier et al. (2012)	FIFO BRAM	None	N	Compact Flash	Y	None	Y	N
Papadimitriou et al. (2007)	BRAM	OPB	N	Compact Flash	N	PPC	Y	N
Papadimitriou et al. (2010)	BRAM	OPB	N	PPC memory	N	PPC	Y	N
Delahaye et al. (2007)	FIFO BRAM	OPB	N	SRAM	N	MB	Y	N

5. DESIGN OF A RECONFIGURABLE PID CONTROLLER

Due to the dynamic nature of their applications, PID controllers are typically not reconfigured. When adaptive PID is required for an application, their gains are rather adjusted using a micro-controller, which reads the gains from a memory space. However, as this survey showed, it is possible to reduce the configuration cost to such an extend that reconfiguration of dynamic applications should be possible. The advantage of reconfiguration above its adaptive counterpart is that the architecture of the controller can be adapted. This is of particular interest, since poles can be added to or removed from the control loop.

This section discusses the methodology for designing a reconfigurable PID controller using the above mentioned BRAM-based architectures. This reconfigurable PID is shown in Fig. 4. Even though this figure only shows the gains of the controller being adjusted using reconfiguration, this methodology is also applicable to architectural reconfiguration. Even though reconfiguration is capable of completely changing the architecture of the PID controller, the gain scheduled PID controller is sufficient for the discussion.

The configuration controller is responsible for transferring the configuration data from the BRAM to the configuration memory via the ICAP. This new configuration data contain a set of new PID parameters to be used in the controller.

5.1 BRAM initialization

The configuration data to be stored in the BRAM have to be generated beforehand using conventional tools. A configuration is required for each set of PID-parameters. As already mentioned, the BRAM is extremely limited and as a result, only a subset of PID-parameters can be reconfigured. External memory can be used to store a larger set of configurations, however, in most cases this requires a bus interface.

Initialization of the BRAM is done by using a *.coe*-file. A *.coe*-file is a text based file containing a header and initialization data for the BRAM. Since the bitstream consists of binary data, it has to be converted into an ASCII format before it can be loaded into the BRAM. This is done by using the *"-b"*-switch when generating the

bitstream using BitGen™. An added benefit of this format is that the data are grouped into 32-bit words, simplifying analysis and command extraction. This ASCII-formatted bitstream can easily be loaded into the BRAM as a *.coe*-file or during synthesis.

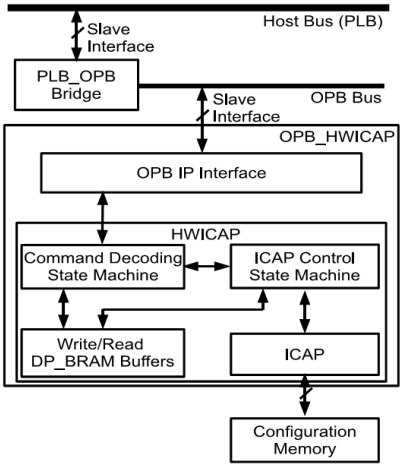
5.2 Configuration controller

The configuration controller is responsible for:

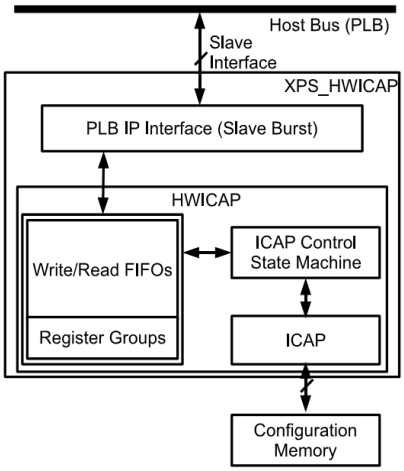
- Reading the configuration data from the memory
- Transferring this data to the configuration memory
- Driving the ICAP pins
- Controlling the ICAP timing

This functionality is achieved using a state machine based on a Xilinx® feature called Multiboot. MultiBoot 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) Xilinx (2008, 2010). This state machine is illustrated in Fig. 6.

As summarized in Table 5, each ICAP pin serves a specific function during the reconfiguration process. The reconfiguration controller is directly connected to these pins, as shown in Fig. 7, and is responsible for driving these pins according to the timing diagram shown in Fig. 5. The *IPROG* command is an external pin that prepares the device for configuration without resetting the configuration logic and is not connected to the ICAP. In fact, this pin is not used when reconfiguring via the ICAP. For the purpose of this discussion, this pin can be seen as the trigger-event for the reconfiguration.



(a) OPB_HWICAP



(b) XPS_HWICAP

Fig. 1. Xilinx® proprietary ICAP controllers

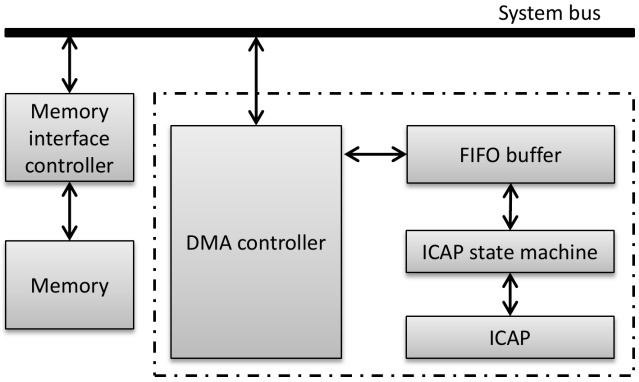


Fig. 2. Reconfigurable architecture with DMA (Le Roux et al. 2012)

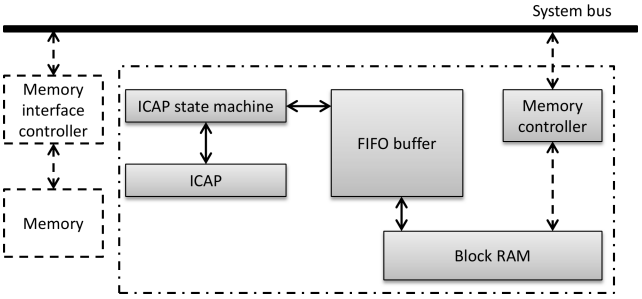


Fig. 3. Reconfigurable architecture with BRAM (Le Roux et al. 2012)

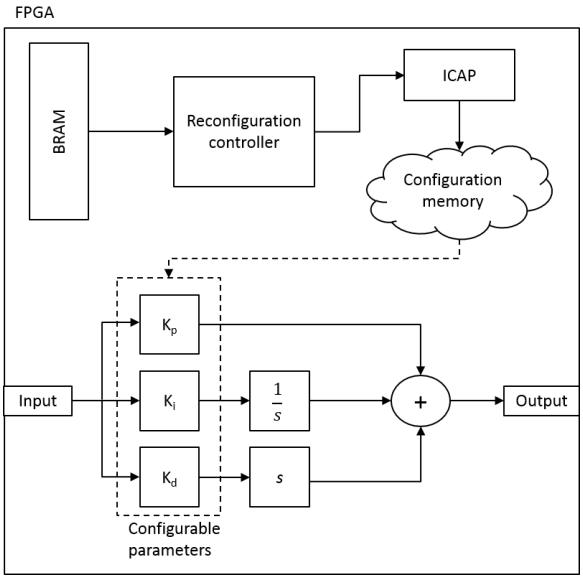


Fig. 4. Illustration of a reconfigurable PID controller

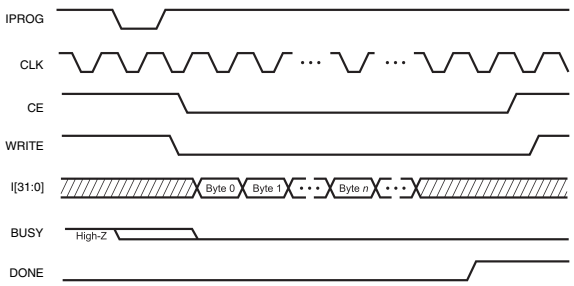


Fig. 5. Timing diagram for ICAP data loading

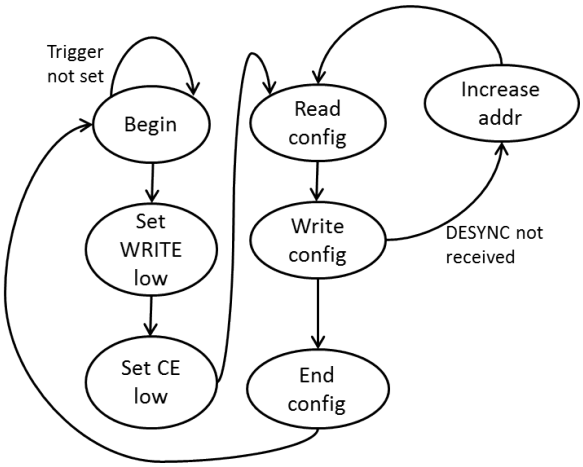


Fig. 6. Hardware reconfiguration state machine flow diagram (Le Roux et al. 2012)

6. CONCLUDING REMARKS

Despite the numerous advantages reconfigurable computing adds to a system, it is mostly limited by the long reconfiguration time. This makes reconfigurable computing only suitable for quasi-static applications. A general rule of thumb is that reconfigurable computing is only viable where execution time of the application exceeds reconfiguration time. This also implies that in most cases, it is not possible to generate reconfiguration bitstreams on-line.

In order to migrate reconfigurable computing to more dynamic applications, various researchers aim to minimize the cost of reconfiguration. One approach is to change the way the bitstream is generated. The aim is not only to enable bitstreams to be generated on-line, but also to reduce the amount of configuration data inside the bitstream. Less information in the bitstream allows for faster reconfiguration, since less data has to be transferred to the ICAP. Placement and routing add a significant amount of overhead to the configuration process and extensive research aim to improve these techniques.

A factor contributing to specialization time is the bus-based architectures most commonly used for reconfiguration. These architectures utilize a bus to connect the various components in the system. As a result, this adds additional overhead to the reconfiguration process. Various attempts are made to mitigate this overhead. This paper summarized the four primary research fields aiming to achieve this:

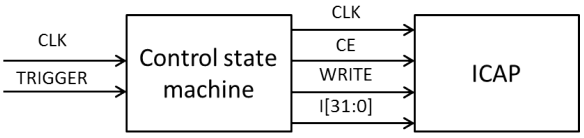


Fig. 7. Control state machine interface to the ICAP (Le Roux et al. 2012)

Table 5. ICAP pin description table

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

(1) Placement research
(2) Routing strategies
(3) Research in bitstream development
(4) Improving reconfiguration throughput

The most promising way to improve the reconfiguration throughput is to use BRAM-based architectures. These architectures mitigate configuration overhead by allowing the reconfiguration controller direct access to the configuration data and memory.

Using these architectures, the design of a reconfigurable PID controller was discussed. Issues such as loading the BRAM with the configuration data and the functionality of the configuration controller were discussed. Even though an example of gain scheduled PID was used to illustrate the design of the reconfigurable PID controller, this approach can easily be adopted to reconfigure the entire structure. The only limitation however, is the control cycle of the application using the controller especially when it is used in a real-time application. A real-time system is defined as "one in which the correctness of a result not only depends on the logical correctness of the calculation but also upon the time at which the result is made available" Gambier (2004). This implies that the reconfiguration time has to fit within one control cycle.

Take the Xilinx® Virtex-5™ for example, assuming a configuration file occupying the entire BRAM of 18 kB. If the ICAP is clocked at the Xilinx® recommended 100 MHz, this results in a maximum throughput of 3.2 Gbps with a word width of 32-bits. This implies that the entire contents of the BRAM can be transferred to the configuration memory within 45 μs. This time can be further reduced by clocking the ICAP at a higher frequency as shown by Hoffman and Pattichis (2011), Claus et al. (2010) and Hansen et al. (2011).

A reconfigurable PID controller based on the above mentioned architectures is currently being investigated.

REFERENCES

Bayar, S. and Yurdakul, A. (2008). Self-reconfiguration on Spartan-III FPGAs with compressed partial bitstreams via a parallel configuration access port (cPCAP) core.

In *Research in Microelectronics and Electronics, 2008. PRIME 2008. Ph.D.*, 137–140.

Betz, V. and Rose, J. (1997). VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field Programmable Logic and Applications*, 213–222.

Bondalapati, K. and Prasanna, V.K. (2002). Reconfigurable computing systems. In *Proceedings of the IEEE*, volume 90, 1201–1217.

Brown, S., Rose, J., and Vranesic, Z.G. (1992). A detailed router for field-programmable gate arrays. *IEEE transactions on computer-aided design of integrated ciruits and systems*, 11(5), 620–628.

Bruneel, K., Bertels, P., and Stroobandt, D. (2007). A method for fast hardware specialization at run-time. In *International conference on Field Programmable Logic and Applications, 2007. FPL 2007*, 35–40.

Bruneel, K. and Stroobandt, D. (2008a). Automatic generation of run-time parameterizable configurations. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 361–366.

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

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

Bruneel, K., Abouelella, F.M.M.A., and Stroobandt, D. (2009b). TMAP: A reconfigurability-aware technology mapper. *Design, Automation and Test Europe (DATE)*.

Bruneel, K. and Stroobandt, D. (2008b). Reconfigurability-aware structural mapping for LUT-based FPGAs. *Proceedings of the 2008 international conference on reconfigurable computing and FPGAs*, 223–228.

Bruneel, K. and Stroobandt, D. (2010). TROUTE: a reconfigurability-aware FPGA router. *Lecture notes in computer science*, 5992, 207–218.

Chan, P.K. and Schlag, M.D. (2003). Parallel placement for field-programmable gate arrays. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on field programmable gate arrays*, 43–50.

Chan, P., Schlag, M., Ebeling, C., and McMurchie, L. (2000). Distributed-memory parallel routing for field-programmable gate arrays. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(8), 850–862.

Chan, Y., Moallem, M., and Wang, W. (2004). Efficient implementation of PID control algorithm using FPGA technology. In *43rd IEEE conference on Decision and Control, 2004. CDC.*, volume 5, 4885–4890.

Chan, Y.F., Moallem, M., and Wang, W. (2007). Design and implementation of modular FPGA-based PID controllers. *IEEE transactions on Industrial Electronics*, 54(4), 1898–1906.

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

Claus, C., Muller, F., Zeppenfeld, J., and Stechele, W. (2007). A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 1–7.

Claus, C., Zhang, B., Stechele, W., Braun, L., Hubner, M., and Becker, J. (2008). A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 535–538.

Claus, C., Altenried, F., and Stechele, W. (2010). 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*, 70, 18–23.

Compton, K. and Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. *ACM computing surveys*, 34(2), 171–210.

Cuoccio, A., Grassi, P., Rana, V., Santambrogio, M., and Sciuto, D. (2008). A generation flow for self-reconfiguration controllers customization. In *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, 279–284.

Davidson, T., Abouelella, F., Bruneel, K., and Stroobandt, D. (2012). Dynamic circuit specialisation for key-based encryption algorithms and dna alignment. *International Journal of Reconfigurable Computing*, 2012, 5.

Dees Jr, W. and Smith II, R. (1981). Performance of interconnection rip-up and reroute strategies. In *18th Conference on design automation, 1981*, 382–390.

DeHon, A., Huang, R., and Wawrzynek, J. (2002). Hardware-assisted fast routing. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 205–215.

Delahaye, J.P., Palicot, J., Moy, C., and Leray, P. (2007). Partial reconfiguration of fpgas for dynamical reconfiguration of a software radio platform. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, 1–5.

Economakos, G. and Economakos, C. (2007). A run-time reconfigurable fuzzy PID controller based on modern FPGA devices. In *Proceedings of the 2007 Mediterranean Conference on Control and Automation*, 1–6.

Eldredge, J. and Hutchings, B. (1994). RRANN: the run-time reconfiguration artificial neural network. In *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, 77–80.

Eldredge, J. and Hutchings, B. (1996). Run-time reconfiguration: A method for enhancing the functional density of SRAM-based FPGAs. In *Journal of VLSI signal processing*, 67–86.

Fatima, K. and Rao, R. (2008). FPGA implementation of a new parallel routing algorithm. In *TENCON 2008 - 2008 IEEE Region 10 Conference*, 1–6.

Gambier, A. (2004). Real-time control systems: a tutorial. In *Control Conference, 2004. 5th Asian*, volume 2, 1024–1031.

Giovagnini, F. and Marzo, A.D. (2012). The ability to reconfigure a portion of a Xilinx FPGA on the fly allowed a european research team to create a more dependable system. *XCell journal*, 80, 39–43.

Hansen, S., Koch, D., and Torresen, J. (2011). High speed partial run-time reconfiguration using enhanced ICAP hard macro. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE*

International Symposium on, 174–180.

Hartenstein, R. (2001). A decade of reconfigurable computing: A visionary retrospective. *Proceedings of the conference on design, automation and test in Europe*, 642–649.

Hauck, S. and DeHon, A. (2008). *Reconfigurable computing: The theory and practice of FPGAs-based computing*. Morgan Kaufmann.

Heyse, K., Farisi, B., Bruneel, K., and Stroobandt, D. (2012). Automating reconfiguration chain generation for srl-based run-time reconfiguration. In O. Choy, R. Cheung, P. Athanas, and K. Sano (eds.), *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7199 of *Lecture Notes in Computer Science*, 1–12. Springer Berlin Heidelberg.

Hoffman, J.C. and Pattichis, M.S. (2011). 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*, 2011, 10.

Ichikawa, S. and Yamamoto, S. (2002). Data dependent circuit for subgraph isomorphism problem. *Proceedings of the international conference on field programmable logic and applications (FPL)*, 1068–1071.

Kim, D. (2000). An implementation of fuzzy logic controller on the reconfigurable fpga system. *Industrial Electronics, IEEE Transactions on*, 47(3), 703–715.

Kirkpatrick, S., Gelatt Jr, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.

Kusse, E. and Rabaey, J.M. (1998). Low-energy embedded FPGA structures. In *In Proceedings of the 1998 international symposium on low power electronics and design (ISLPED'98)*, 155–160.

Lago, E., Jiménez, C.J., Lopez, D.R., Sánchez-Solano, S., and Barriga, A. (1998). Xfvhdl: A tool for the synthesis of fuzzy logic controllers. In *Proceedings of the conference on Design, automation and test in Europe*, 102–107. IEEE Computer Society.

Lam, J. and Delosme, J. (1988). Performance of a new annealing schedule. In *Proceedings of the 25th ACM/IEEE design automation conference*, DAC '88, 306–311.

Lamonnier, S., Thoris, M., and Ambielle, M. (2012). Accelerate partial reconfiguration with a 100 *XCell journal*, 79, 44–49.

le Roux, R., van Schoor, G., and van Vuuren, P. (2012). Block RAM implementation of a reconfigurable real-time PID controller. In *The 9th IEEE international conference on embedded software and systems (ICESS)*, 1383–1390.

Lee, S.J. and Raahemifar, K. (2008). FPGA placement optimization methodology survey. In *Canadian conference on Electrical and Computer Engineering, 2008. CCECE*, 1981–1986.

Leonard, J. and Mangione-Smith, W. (1997). A case study of partially evaluated hardware circuits: key specific DES. *Proceedings of the international workshop on field programmable logic and applications (FPL)*, 151–160.

Li, J. and Hu, B.S. (1996). The architecture of fuzzy pid gain conditioner and its fpga prototype implementation. In *ASIC, 1996., 2nd International Conference on*, 61–65. IEEE.

Lin, M. and Gamal, A.E. (2008). TORCH: A design tool for routing channel segmentation in FPGAs. In *Proceedings of the ACM/SIGDA international symposium on field-programmable gate arrays*, 131–138.

Lin, M., Wawrzynek, J., and EL Gamal, A. (2010). Exploring FPGA routing architecture stochastically. *IEEE transactions on computer-aided design of integrated circuits and systems*, 29(10), 1509–1522.

Liu, M., Kuehn, W., Lu, Z., and Jantsch, A. (2009). Runtime partial reconfiguration speed investigation and architectural design space exploration. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2, 498–502.

Liu, S., Pittman, R.N., and Forin, A. (2010). 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, 292–292. ACM, New York, NY, USA.

Lysecky, R., Stitt, G., and Vahid, F. (2006). Warp processors. *Transactions on design automation of electronic systems*, 11 nr 3, 186.

Lysecky, R., Vahid, F., and Tan, S.X.D. (2004). Dynamic FPGA routing for just-in-time FPGA compilation. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, 954–959. ACM, New York, NY, USA.

McKay, N., Melham, T., and Susanto, K.W. (1998). Dynamic specialisation of XC6200 FPGAs by partial evaluation. In *IEEE symposium on FGPAs for custom computing machines*, 308–309.

McMurchie, L. and Ebeling, C. (1995). PathFinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the third international ACM symposium on Field-programmable gate arrays, 1995. FPGA '95*, 111–117.

Papadimitriou, K., Anyfantis, A., and Dollas, A. (2007). Methodology and experimental setup for the determination of system-level dynamic reconfiguration overhead. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, 335–336.

Papadimitriou, K., Anyfantis, A., and Dollas, A. (2010). An effective framework to evaluate dynamic partial reconfiguration in FPGA systems. *Instrumentation and Measurement, IEEE Transactions on*, 59(6), 1642–1651.

Papadimitriou, K., Dollas, A., and Hauck, S. (2011). Performance of partial reconfiguration in fpga systems: A survey and a cost model. *ACM Trans. Reconfigurable Technol. Syst.*, 4(4), 36:1–36:24.

Sánchez-Solano, S., Senhadji, R., Cabrera, A., Baturone, I., Jimenez, C.J., and Barriga, A. (2002). Prototyping of fuzzy logic-based controllers using standard fpga development boards. In *Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on*, 25–32. IEEE.

Sankar, Y. and Rose, J. (1999). Trading quality for compile time: ultra-fast placement for FPGAs. *Proceedings of the 1999 ACM/SIGDA seventh international symposium on field programmable gate arrays*, 157–166.

Schaumont, P., Verbauwhede, I., and Keutzer, K. (2001). A quick safari through the reconfiguration jungle. In *In design automation conference*, 172–177.

Sedcole, P., Blodget, B., Becker, T., Anderson, J., and Lysaght, P. (2006). Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proceeding of Computers and Digital Techniques*, 153(3), 157–164.

Sen, W., Bin, T., and Jim, Z. (2007a). Distributed arithmetic for fir filter design on fpga. In *Communications, Circuits and Systems, 2007. ICCCAS 2007. International Conference on*, 620–623. IEEE.

Sen, W., Bin, T., and Jim, Z. (2007b). Distributed arithmetic for fir filter design on fpga. In *Communications, Circuits and Systems, 2007. ICCCAS 2007. International Conference on*, 620–623.

Shelburne, M., Patterson, C., Athanas, P., Jones, M., Martin, B., and Fong, R. (2010). Metawire: Using fpga configuration circuitry to emulate a network-on-chip. *Computers Digital Techniques, IET*, 4(3), 159–169.

Shi, X. (2009). FPGA placement methodologies: A survey. Paper written for: "Teaching and Research methods" class.

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

Steiner, N., Wood, A., Shojaei, H., Couch, J., Athanas, P., and French, M. (2011). TORC: towards an open-source tool flow. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, 41–44. ACM, New York, NY, USA.

Stitt, G., Vahid, F., and Nematbakhsh, S. (2004). Energy savings and speedups from partitioning critical software loops to hardware in embedded systems. *ACM transactions on embedded computer systems*, 3(1), 218–232.

Tipsuwanporn, V., Runghimmawan, T., Intajag, S., and Krongratana, V. (2004). Fuzzy logic PID controller based on FPGA for process control. In *Industrial Electronics, 2004 IEEE International Symposium on*, volume 2, 1495–1500.

Todman, T.J., Constantinides, G.A., Wilton, S.J.E., Mencer, O., and Luk, W. (2005). Reconfigurable computing: architectures and design methods. In *IEE proceedings of Computers and Digital techniques*, volume 152, 193–207.

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

Villasenor, J., Schoner, B., Chia, K.N., and Zapata, C. (1996). Configurable computing solutions for automatic target recognition. *Proceedings of the IEEE symposium on FPGAs for custom computing machines (FCCM)*.

Vuong, P.T., Madni, A.M., and Vuong, J.B. (2006). Vhdl implementation for a fuzzy logic controller. In *Automation Congress, 2006. WAC'06. World*, 1–8. IEEE.

Wirthlin, M.J. (2004). Constant coefficient multiplication using look-up tables. *Journal of VLSI signal processing systems*, 36, 7–15.

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

Wrighton, M.G. and DeHon, A.M. (2003). Hardware-assisted simulated annealing with application for fast FPGA placement. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, FPGA '03, 33–42. ACM, New York, NY, USA.

Xilinx (2008). Multiboot with Virtex-5 FPGAs and Platform Flash XL. Application note. XAPP1100.

Xilinx (2010). Virtex-5 FPGA configuration user guide . UG191 (v3.9.1).

Xu, M., Grewal, G., and Areibi, S. (2011). Starplace: A new analytic method for FPGA placement. *Integration, the VLSI journal*, In Press, Corrected Proof, –.

Xu, M. (2009). *Analytic methods for the FPGA placement problem*. Ph.D. thesis, Canada: University of Guelph.

Xu, Y. and Khalid, M. (2005). QPF: efficient quadratic placement for FPGAs. In *Field Programmable Logic and Applications, 2005. International Conference on*, 555–558.

Zhao, W., Kim, B.H., Larson, A., and Voyles, R. (2005). FPGA implementation of closed-loop control system for small-scale robot. In *Proceedings of the 12th international conference on advanced robotics, 2005. ICAR '05*, 70–77.

Zhao, Z.Y., Tomizuka, M., and Isaka, S. (1993). Fuzzy gain scheduling of pid controllers. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(5), 1392–1398.

Zhong, P., Martonosi, M., Ashar, P., and Malik, S. (1998). Accelerating boolean satisfiability with configurable hardware. *Proceedings of the IEEE symposium on FPGAs for custom computing machines (FCCM)*, 186–195.

Zhou, Y. and Shi, P. (2011a). Distributed arithmetic for fir filter implementation on fpga. In *Multimedia Technology (ICMT), 2011 International Conference on*, 294–297. IEEE.

Zhou, Y. and Shi, P. (2011b). Distributed arithmetic for fir filter implementation on fpga. In *Multimedia Technology (ICMT), 2011 International Conference on*, 294–297.