Performance Evaluation of Dynamic Circuit Specialization on Xilinx FPGAs

Amit Kulkarni, Karel Heyse, Tom Davidson, and Dirk Stroobandt
ELIS department, Computer Systems Lab, Ghent University
Sint-Pietersnieuwstraat 41, Ghent B-9000, Belgium
{ Amit.Kulkarni, Karel.Heyse, Tom.Davidson, Dirk.Stroobandt }@UGent.be

ABSTRACT

Dynamic Circuit Specialization (DCS) is a technique used to optimize FPGA applications when some of the inputs, called parameters, are infrequently changing compared to other inputs. For every change of parameter input values, a specialized FPGA configuration is generated during run time and the FPGA is reconfigured with a specialized bitstream. We examine how the performance of the DCS technique evolves with the advent of newer Xilinx FPGA architectures. The performance of the DCS technique is evaluated on three different Xilinx FPGA architectures: Virtex-II Pro, Virtex-5 and Zynq SoC. We have used a 16-tap, 8-bit FIR filter as a parameterized design, with the filter coefficients as the parameters of the FIR design.

Categories and Subject Descriptors

B.5 [**EMBEDDED PLATFORM ARCHITECTURES**]: Reconfigurable architectures

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Partial Reconfiguration, Micro-reconfiguration, TLUT, TCON, PPC memory size, Reconfiguration time, Evaluation time

1. INTRODUCTION

Partial Reconfiguration is the ability to dynamically modify a part of the FPGA while the rest remains active. The modification process can be achieved by downloading partial bitstreams. The partial bitstreams represent the Partial Reconfigurable Modules (PRM), which are swapped in and out of their corresponding regions during run time [3]. The configuration manager loads the appropriate PRM into the Partially Reconfigurable Regions (PRR) during run time, depending on the set of conditions at hand. The configuration manager can be realized on an embedded processor which can be configured in FPGAs. Modern FPGAs contain hard-core processors such as the PowerPC in case of the Virtex-II Pro and the Virtex-5, while the Zynq SoC contains an advanced RISC processor: dual-core ARM Cortex-A9. The Partial Reconfiguration (PR) technique provides more flexibility to dynamically modify the logic without reconfiguring the entire FPGA. However, this flexibility comes at the cost of reconfiguration time and of the memory size needed to store partial bitstreams.

Dynamic Circuit Specialization (DCS) somewhat resembles Partial Reconfiguration and can be applied to those applications that have infrequently changing inputs and are parameterized. For every set of parameter values, the circuit can be specialized for these specific values and each parameter value change results in a reconfiguration to a new specialized circuit. There is a significant reduction in the resources utilized by the specialized circuit at the cost of reconfiguration overhead. In this technique there is no need to synthesize or store the PRM offline, the circuit is specialized at run time according to the parameterized inputs.

We implemented a 16-tap, 8-bit FIR filter using DCS with the filter coefficients as the parameter inputs. In the DCS approach the filter coefficient values (parameters) are implemented as constants and the design is optimized for these constants. When the coefficient values change, the design is re-optimized for the new constant values by run-time reconfiguring the FPGA.

In this article we evaluate the performance of DCS on three different Xilinx FPGA architectures: Virtex-II Pro, Virtex-5 and Zynq SoC. Each of these architectures has their own pros and cons. Our attempt is to see how DCS will perform under the new Xilinx FPGA architectures. We evaluate the reconfiguration time, the specialization time and the size of the memory occupied by the design while using DCS on the three Xilinx FPGA architectures and compare them accordingly.

In Section 2, we present an overview of the DCS technique. Section 3 provides a brief explanation of how DCS is being implemented on commercial Xilinx FPGAs. In Section 4, we provide the architecture details of three different Xilinx FPGAs that we used for our experiments. Section 5 describes the parameterized FIR filter design which is used as a benchmark to evaluate DCS. The results of the evaluation are tabulated in Section 6, followed by discussion, interpretation and comparison of results and finally we conclude in Section 7.

FPGAWorld '14, September 9-11, Stockholm and Copenhagen.

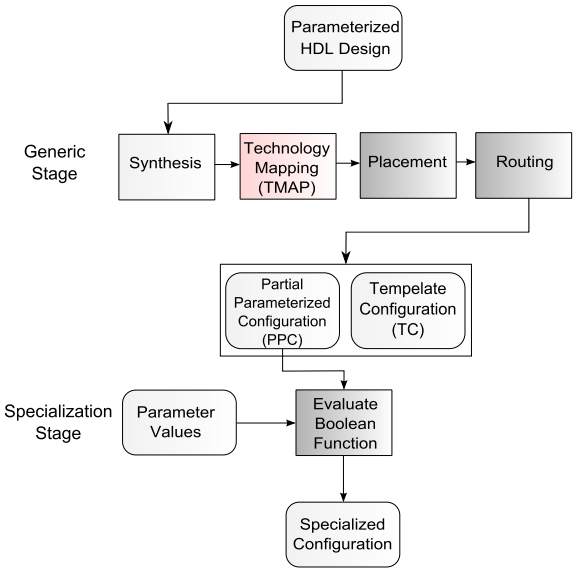

Figure 1: Tool flow for parameterized Configuration

2. DYNAMIC CIRCUIT SPECIALIZATION

In most cases, an application needs to undergo data manipulations in their subsequent stages of execution. These data manipulations often differ only by a small set of parameter values, which are inputs to the application forming a parameterized design [2]. During run time, for every change in a parameter value, a specialized circuit can be generated for this parameter value and the design can be reconfigured with the specialized circuit. This technique is called Dynamic Circuit Specialization. The specialized circuits are faster and smaller than their generic counterparts. Authors in [2] have shown that DCS significantly reduces the number of LUTs used and increases the speed of the design for various examples as compared to the conventional implementation. The process of specialization and reconfiguration results in a large overhead when conventional tools are used and these overheads nullify the advantages of using specialized circuits.

In [2], the authors propose a novel approach to accomplish DCS. The tool flow consists of two stages: a generic stage and a specialization stage as depicted in Figure 1.

In the generic stage, a HDL design with parameterized inputs (annotated by –PARAM in VHDL) is further processed to yield a partial parameterized configuration. A Partial Parameterized Configuration (PPC) is a part of a FPGA configuration that contains bitstreams expressed as boolean functions of parameters. In the specialization stage, these boolean functions are evaluated for specific parameter values to generate a specialized configuration. The following tool flow steps explains the generic stage and are adapted from the conventional tool flow.

2.1 Synthesis

In this step, the HDL design is converted into a network of logic gates. The parameter inputs described in the HDL are annotated parameter inputs and this annotation makes the difference between parameter inputs and regular inputs. The parameter inputs are also a part of the boolean network of logic gates produced after synthesis.

2.2 Technology Mapping

During the mapping stage, the synthesized boolean network is mapped onto the available resources of the target FPGA architecture such as LookUp Tables (LUTs), DSP blocks and BRAMs while optimization of circuit area and speed are being taken into consideration. The conventional mapping tool would map to the static LUTs and hence it would result in the conventional bitstreams after place and route. To generate a parameterized bitstream, authors in [2] change the conventional mapping tool to a tunable version, TMAP, so that the boolean functions of parameter inputs are mapped to Tunable LookUp Tables (TLUTs). These are virtual LUTs that differ from conventional LUTs in the fact that their lookup entries are defined as the boolean functions of the parameter inputs instead of static ones and zeros. The truth table entries will be reconfigured upon every change in parameter values.

Presently, the parameterization of BRAM and DSP blocks is not yet possible but parameterization of the routing switches called TCONs is established at the virtual FPGA level. However, the practical implementation in commercial FPGAs is yet to be done [6]. The TMAP mapping algorithm is described in [2] and can be integrated with the conventional Xilinx tool flow which is explained in [1]. We will limit the evaluation in this paper to the use of parameterization in TLUTs.

2.3 Placement and Routing

In the placement step, the mapped resources are placed or associated to specific blocks of the target FPGA architecture. Extensive optimization is considered so that interconnect wire length and interconnect delay is minimized. The router configures the physical switch blocks to achieve the required interconnect according to the circuit. Since the placement and routing does not depend on parameter inputs or TLUTs, a conventional placer and router can be used.

The final output of the generic stage is the Template Configuration (TC) and Partial Parameterized Configuration (PPC). TC is a static bitstream which contains static ones and zeros, which are used for configuring during the start of the FPGA. The PPC contains sets of multi-output boolean functions of the parameter inputs. The PPC needs to undergo the specialization stage, along with parameter values to produce an efficient specialized configuration.

The specialization stage consists of a Specialized Configuration Generator (SCG). The SCG takes the PPC and the parameter values as inputs and evaluates the boolean functions of parameter inputs for given parameter values to produce a specialized configuration and all the TLUTs are reconfigured by downloading the specialized configuration during run time and thus accomplishing run time reconfiguration. The SCG can be implemented on a hard-core embedded processor in the FPGA such as a PowerPC, on a soft-core processor such as a MicroBlaze or on a custom processor (CP) which is more specifically designed to evaluate boolean functions only. Different types of SCG implementations and its details are described in [4].

3. IMPLEMENTATION OF DCS ON XILINX FPGAS

In this section, the implementation of the DCS tool flow on commercial FPGAs such as Xilinx Virtex-II Pro, is briefly

described. We used Xilinx Platform Studio (XPS) to target Xilinx FPGAs. The TMAP algorithm is integrated with Xilinx tools and we build a self reconfigurable platform [1]. During the TMAP stage, the parameterization of TLUTs is performed and a unique name is assigned to individual TLUTs so that they can be identified and reconfigured with proper specialized truth table values. After the Place and Route step a PPC is generated which contains boolean functions of the parameter inputs, forming a parameterized bitstream. For every change in parameter input values the SCG generates a specialized configuration by evaluating the boolean functions. A new specialized truth table for each TLUT is generated. We make use of the Internal Configuration Access Port (ICAP) of the FPGA device to reconfigure TLUTs. The reconfiguration using ICAP is performed by modifying the truth table values of a TLUT, whose bits are located at certain locations within a frame. The complete truth table entries of a single TLUT are spread in multiple frames. A frame of an FPGA is a smallest addressable element in a bitstream. Hence it is required to modify appropriate locations of multiple frames during the reconfiguration. Each frame has a frame address which is used for identification of a particular frame. This is done as follows.

3.1 "*XHwIcap_SetClbBits*" function:

This is a HWICAP driver [1], used to reconfigure the TLUTs. The procedure takes the truth table values (specialized bits) and the location co-ordinates of a TLUT as inputs. It first generates the frame address from a given TLUT location, which is required to target a CLB that contains specific TLUTs to be reconfigured. The reconfiguration occurs in 3 steps:

1. Read frames: Using the frame address, the function reads the current truth table entries of a specific TLUT by fetching multiple frames from the configuration memory.

2. Modify frames: Once the current truth table values are read, they are overwritten by the specialized bits. Now the frames contain specialized bitstreams.

3. Write frames: Using the same frame address, the modified truth table entries are written back to the TLUT by swapping in multiple frames into FPGA configuration memory thus accomplishing *micro-reconfiguration*. *Micro-reconfiguration* is a fine-grain form of reconfiguration used for DCS.

3.2 Cost of micro-reconfiguration:

The *micro-reconfiguration* comes at the cost of:

- PPC memory size: memory space required to store the PPC functions.

- Evaluation time (Specialization time): time taken by SCG to evaluate the boolean functions.

- Reconfiguration time: time taken to update the configuration of all TLUTs

Figure 2 shows the principle of a DCS implementation on a commercial FPGA. The XPS Hardware ICAP (HWICAP) is used as configuration interface and the SCG is implemented on a hard coded processor which is a part of

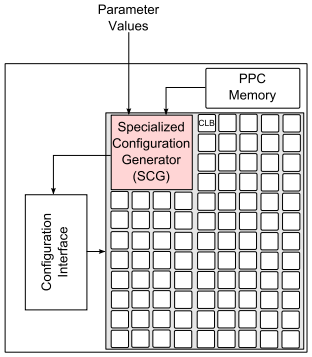

Figure 2: DCS for FIR filter implemented on an FPGA

the FPGA fabric. If a soft-core processor is used then a processor will be mapped onto a set of CLBs. We evaluate the performance of DCS on different Xilinx FPGAs by comparing the above three *micro-reconfiguration* cost parameters for each FPGA and illustrate how the DCS tool flow evolves with the evolution of Xilinx FPGA architectures.

4. XILINX FPGA ARCHITECTURES

In our experiment, we have used Virtex-II Pro, Virtex-5 and Zynq 7000 family SoC architectures. We compare the device resources and the configurations that are related to DCS.

Table 1 shows the device names and the corresponding boards we used for our experiments. The number of inputs to a LUT for the corresponding FPGAs is also present in the same table. The number of inputs will influence the memory size of the PPC functions and the time taken to evaluate the boolean functions within the LUT entries.

The embedded processors of both type (soft-core and hard-core) and their respective clock configurations, that are used in the Xilinx FPGAs are tabulated in Table 1. These processors are used to generate specialized configurations by evaluating the boolean functions. To describe the specialization procedure, we use a standard C program with Xilinx SDK.

The ARM Cortex-A9 processor within the Zynq SoC is a dual core processor but we use only a single core. It has instruction and data caches each of size 32kB [14]. The PowerPC processor for both Virtex-II Pro and Virtex-5 was configured with instruction and data caches each of size 32kB [10] [12]. The MicroBlaze can also be configured to enable instruction and data caches. However, these caches in a softcore processor are just a reserved memory space in BRAMs and are not an actual or physical silicon. Since the PPC functions already reside in BRAMs we did not enable caches for the MicroBlaze.

The HWICAP is used as a configuration interface and is responsible for loading the specialized bitstreams into the FPGA configuration memory. Table 1 shows the configurations for the HWICAP that we used for our experiments. The HWICAP throughput is also tabulated and it shows the rate at which the frames are read from the configuration memory and the rate at which the frames are written into the configuration memory.

The size of one frame for an individual FPGA and the bus that we used to connect our parameterized design on a

Table 1: Xilinx FPGA device details

	Virtex-II Pro (XC2VP30)	Virtex-5	Zynq SoC
Device name	XC2VP30 -FF896-7C	XC5VFX70T -FFG1136	XC7Z020 -CLG484-1
Board name	XUPV2P Development System	ML507 Evaluation Platform	ZedBoard
LUT inputs (k)	4	6	6
LUT entries	16	64	64
Hard-core Processor	PowerPC 405 Core	PowerPC 440 Core	ARM Cortex-A9
Soft-core Processor	MicroBlaze (6.00.b)	MicroBlaze (8.20.b)	MicroBlaze (8.40.a)
Hard-core CPU clock (MHz)	300	400	667
Soft-core CPU clock (MHz)	100	100	100
HWICAP type	OPB HWICAP (1.00.b)	XPS HWICAP (5.01.a)	AXI HWICAP (2.03.a)
HWICAP clock (MHz)	66.67	100	100
HWICAP throughput (non-DMA) (MB/s)	10	19	19
HWICAP port width (bits)	8	32	32
Frame size (32-bit words)	206	41	101
Bus type	OPB + PLB	PLB	AXI
Bus clock (MHz)	66.67	100	100

self reconfigurable platform, are tabulated in Table 1. The size of a frame describes the minimum number of words (1 word = 32 bits) in a bitstream that needs to be replaced for each reconfiguration process and has direct influence on the reconfiguration time. For the Virtex-II Pro family, the frame size is not the same for the entire family and varies for different FPGA devices [13]. The specification of the bus can be seen in [11] [8] [9] and the bus interconnections in a typical embedded design are described in [1].

5. PARAMETERIZED DESIGN

We used a 16-tap FIR filter of 8-bit data width as a parameterized design for showing the benefits of DCS. The general benefits of using DCS for FIR filter implementation are discussed in [5]. All coefficients are the parameterized inputs and hence for each infrequent change in the coefficient value, a specialized bitstream is generated and the filter taps containing multipications are reconfigured accordingly.

The multiplications for the FIR filter are designed for a (T)LUT implementation keeping in mind that they should suit for all three FPGAs. Since Virtex-II Pro has a LUT input size of 4, we make use of 4-bit multiplications only. The filter requires 16 8-bit multiplications. Two 4-bit multiplications are combined to form one 8-bit multiplier and therefore, a total of 32 4-bit multiplications were used to build a complete FIR filter design [5]. It is known that a 4-bit multiplier is mapped onto 12 TLUTs, therefore 1-tap of the FIR filter contains 24 TLUTs.

6. RESULTS

In this section, the results of our experiments are tabulated and compared. Table 2 shows the time taken by the processors to evaluate the boolean functions during the specialization phase. Table 3 shows the time spent during reconfiguring one multiplier of the FIR filter which is composed of 12 TLUTS. Table 4 shows the PPC memory size values. These values are the size of PPC functions only. They are compiled with "-O2" optimization and without debug option.

Table 2: PPC evaluation time in microseconds

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor	6.6	4.6	1.7
Soft-core Processor	18.7	28.3	28.9

Table 3: Reconfiguration time in milliseconds

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor	0.5	1	2.8
Soft-core Processor	1.89	1.4	4.0

Table 4: PPC memory size in KB

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor	7	10	11
Soft-core Processor	8	11	12

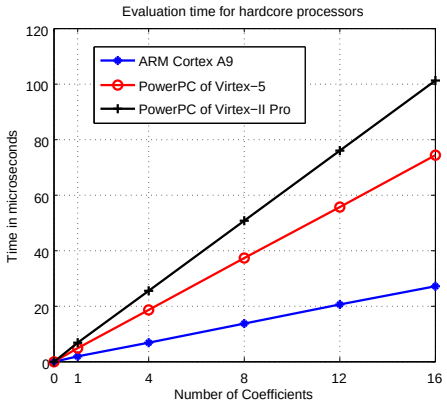

Figure 3: Evaluation time comparison of Hard-core processors

6.1 Evaluation time - Hard-core Processors

We compare the performance of evaluating boolean functions on all 3 hard-core processors - PowerPC of Virtex-II Pro, PowerPC of Virtex-5 and ARM Cortex-A9 of the Zynq SoC. Figure 3 shows the plots and it is clear that ARM Cortex-A9 takes the least amount of time compared to both PowerPCs and hence it can claim to be very efficient. The PowerPC in a Virtex-5 is more efficient than the PowerPC in a Virtex-II Pro, showing the improvements in the newer processor architectures with higher clock frequency support. It is to be noted that the number of boolean functions that needs to be evaluated increases for the Zynq SoC and the Virtex-5 compared to the Virtex-II Pro because of an increase in LUT inputs and corresponding LUT entries.

6.2 Evaluation time - Soft-core Processors

We also compare the performance of evaluating boolean functions on all 3 soft-core processors - MicroBlaze of Virtex-II Pro, MicroBlaze of Virtex-5 and MicroBalze of Zynq SoC. Figure 4 shows that the efficiency of the MicroBlaze of both

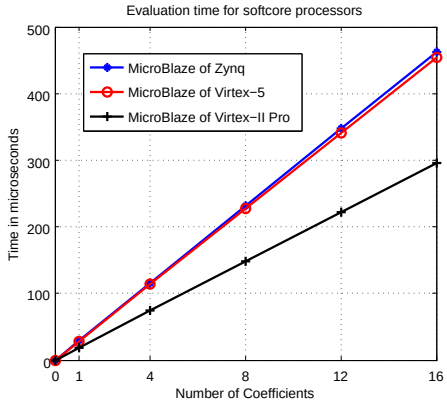

Figure 4: Evaluation time comparison of Soft-core processors

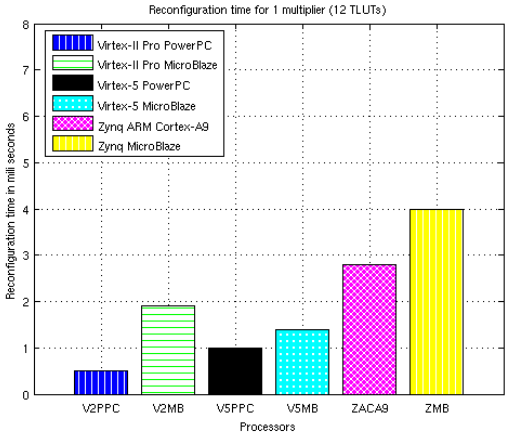

Figure 5: Reconfiguration time comparison

the Virtex-5 and the Zynq SoC are almost the same. Interestingly, the MicroBlaze in the Virtex-II Pro consumes less time and hence proves to be a very efficient soft-core processor. However, the main reason for this behaviour is that the number of LUT entries for the Virtex-II Pro (16) is smaller than for the Virtex-5 and Zynq SoC (64).

Comparing the hard-core processor with the soft-core processor in each FPGA shows that hard-core processor is always more efficient than the soft-core. The main advantage of using the hard-core processors is that they support a very high clock frequency which influences the evaluation of boolean functions and also slightly the reconfiguration time. However, in some FPGAs it is inevitable to use soft-core processors due to lack of availability of hard-core processors in their FPGA architecture.

6.3 Reconfiguration time

Reconfiguration time is defined as the time required to update the configuration of all TLUTs using new specialized bitstreams. The updating process involves read, modify and write steps as explained in section 3. Figure 5 shows the bar graph of the reconfiguration time for the 3 different FPGAs using respectively hard-core and soft-core processors. The normalized values are tabulated in Table 5, the normalization is with respect to FPGAs mentioned in the parenthesis. It is clear that the reconfiguration time overhead is not same for Virtex-II Pro and Virtex-5. The resources that influence the reconfiguration time overhead are the frame size, interconnect bus speed, HWICAP port width and HWICAP clock. The Virtex-II Pro has a lower capacity of these resources compared to the Virtex-5. However, the number of frames to be reconfigured for a TLUT also has a direct influence on the reconfiguration time and should be considered during the comparison. Clearly, the number of frames to be reconfigured in the Virtex-5 is higher than for the Virtex-II Pro and therefore it consumes more amount of time during reconfiguration.

The reconfiguration time overhead of the Zynq SoC is almost double the overhead of the Virtex-5. The main reason for the increase in reconfiguration time is the increase in the number of words per frame for the Zynq SoC compared to the Virtex-5. It is to be noted that the interconnect bus speed, HWICAP port width and HWICAP clock frequency are same for the Virtex-5 and the Zynq SoC. The same note

Table 5: Normalized Reconfiguration time

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor(Virtex-II Pro)	1	2	5.6
Soft-core Processor (Virtex-II Pro)	1	0.7	2.1
Hard-core Processor (Virtex-5)	0.5	1	2.8
Soft-core Processor (Virtex-5)	1.35	1	2.8

Table 6: Normalized Reconfiguration time

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor	1	1	1
Soft-core Processor	3.5	1.4	1.4

applies to the MicroBlaze results in all 3 FPGAs. The maximum clock frequency of the HWICAP that can be used for a reliable implementation is shown in Table 1. The HWICAP proves to be the bottleneck for the reconfiguration process. The hard-core processor clock speed is much higher than for the HWICAP, so obviously the HWICAP cannot process and synchronize directly with the processor. The processor will stall some clock cycles until the frames are swapped in and out of the configuration memory at the speed of the HWICAP clock.

Observing these results, we see that the DCS tool flow encounters a higher reconfiguration time overhead for the newer Xilinx FPGA architectures. In order to reduce the reconfiguration time overhead, one would have to improve the bus architecture and the HWICAP speed. The hard-core processors are fast enough to reduce the evaluation time and processing the frames during read, modify and write-back operation during reconfiguration, even when considering the increase in the LUT entries size. The Virtex-II Pro has a reconfiguration time overhead that is relatively smaller compared to the Virtex-5 and the Zynq SoC but it has a lower number of LUT entries compared to the newer FPGAs.

Normalizing the reconfiguration time with respect to the respective hard-core processors in all 3 FPGAs, shows that the use of hard-core processors is more efficient in reconfiguration than using the respective soft-cores. Table 6 shows the normalized values of the reconfiguration time with respect to their hard-core processors.

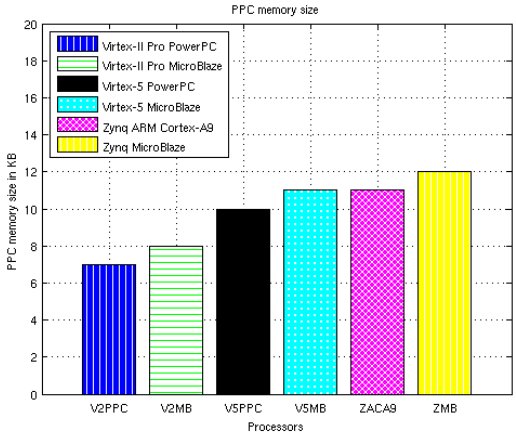

Figure 6: PPC memory size comparison

Table 7: Normalized PPC memory size

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor(Virtex-II Pro)	1	1.4	1.6
Soft-core Processor (Virtex-II Pro)	1	1.4	1.5
Hard-core Processor (Virtex-5)	0.7	1	1.1
Soft-core Processor (Virtex-5)	0.7	1	1.1

Table 8: Normalized PPC memory size

	Virtex-II Pro	Virtex-5	Zynq SoC
Hard-core Processor	1	1	1
Soft-core Processor	1.1	1.1	1.1

6.4 PPC memory size

Figure 6 shows the bar graph of the memory size of the PPC functions. The normalized values with respect to the corresponding FPGAs shown in parenthesis are tabulated in the Table 7. We believe that the increase in PPC memory size of the Virtex-5 and the Zynq SoC compared to the Virtex-II Pro is caused by the increase in number of LUT entries from 16 to 64 for individual LUTs. It is also observed that the code density of the PowerPC is almost the same as that of the ARM Cortex-A9.

Normalized PPC memory size with respect to the hard-core processor is tabulated in Table 8. It reveals that the code density of the hard-core processors is a little bit higher than that of the soft-core processor which is in agreement with [7]. It can be seen that the increase in LUT entries affects the PPC memory size, reconfiguration time and the evaluation time. This affect can be negated by using an efficient bus structure, high speed HWICAP and highly sophisticated processor architecture.

7. CONCLUSION

The DCS tool was implemented on three different Xilinx FPGAs: Virtex-II Pro, Virtex-5 and Zynq SoC. Their corresponding resources related to run time reconfiguration were described. A 16-tap, 8-bit FIR filter was used as a parameterized design to evaluate the performance of DCS on three different FPGAs. The evaluation time, reconfiguration time and the PPC memory size were the three metrics used to measure the overhead and the performance of DCS. It is clear that newer FPGA architectures tend to include more LUT resources and features. The added features increase the size of the LUT entries and the frame size. This creates the overhead on reconfiguration time, evaluation time and the PPC memory size. The MicroBlaze soft-core processor proves to be inefficient in newer FPGA architectures for DCS because the processor clock speed, memory and ISA are limited and do not compete with those of hard-core processors. However for the edge cases, MicroBlaze in old FPGA architecture such as Virtex-II pro would be more suitable to implement the DCS.

8. REFERENCES

[1] K. Bruneel, F. Abouelella, and D. Stroobandt. Automatically mapping applications to a self-reconfiguring platform. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 964–969, April 2009.

[2] K. Bruneel, W. Heirman, and D. Stroobandt. Dynamic data folding with parameterizable configurations. *ACM Transactions on Design Automation of Electronic Systems*, 16(4), 2011.

[3] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Invited paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAS. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, Aug 2006.

[4] F. Mostafa Mohamed Ahmed Abouelella, K. Bruneel, and D. Stroobandt. Efficiently generating fpga configurations through a stack machine. In *Field Programmable Logic and Applications, 20th International conference, Abstracts*, Milano, Italy, 2010.

[5] F. Mostafa Mohamed Ahmed Abouelella, T. Davidson, W. Meeus, K. Bruneel, and D. Stroobandt. How to efficiently implement dynamic circuit specialization systems. *ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS*, page 38, 2013.

[6] E. Vansteenkiste, K. Bruneel, and D. Stroobandt. A connection router for the dynamic reconfiguration of fpgas. In *Proceedings of the 8th International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ARC'12, pages 357–364, Berlin, Heidelberg, 2012. Springer-Verlag.

[7] V. Weaver and S. McKee. Code density concerns for new architectures. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 459–464, Oct 2009.

[8] LogiCORE ip Processor Local Bus (PLB) v4.6 (v1.05a).

[9] LogiCORE IP On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.00d).

[10] PowerPC Processor Reference Guide (ug011).

[11] AXI Reference Guide (ug761).

[12] Virtex-5 Family Overview (ds100).

[13] Virtex-II Pro and Virtex-II Pro X FPGA User Guide (ug012).

[14] Zynq-7000 All Programmable SoC Overview (ds190).