Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model

KYPRIANOS PAPADIMITRIOU, APOSTOLOS DOLLAS
Technical University of Crete
and
SCOTT HAUCK
University of Washington

Fine-grain reconfigurable devices suffer from the time needed to load the configuration bitstream. Even for small bitstreams in partially reconfigurable FPGAs this time cannot be neglected. In this paper we survey the performance of the factors that contribute to the reconfiguration speed. Then, we study an FPGA-based system architecture and with real experiments we produce a cost model of partial reconfiguration (PR). This model is introduced to calculate the expected reconfiguration time and throughput. In order to develop a realistic model we take into account all the physical components that participate in the reconfiguration process. We analyze the parameters that affect the generality of the model and the adjustments needed per system for error-free evaluation. We verify it with real measurements, and then we employ it to evaluate existing systems presented in previous publications. The percentage error of the cost model when comparing its results with the actual values of those publications varies from 36% to 63%, whereas existing works report differences up to two orders of magnitude. Present work enables a user to evaluate PR and decide whether it is suitable for a certain application prior entering the complex PR design flow.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Modeling techniques

General Terms: Experimentation, Measurement, Performance, Theory, Verification
Additional Key Words and Phrases: Reconfigurable Computing, Field Programmable Gate Arrays, Partial Reconfiguration, Reconfiguration Time

1. INTRODUCTION

During recent years many applications that exploit the dynamic nature of reconfigurable devices have been developed. Due to their SRAM-based technology, both fine-grain and coarse-grain reconfigurable devices can be reprogrammed potentially unlimited times. This reconfigurability can serve as the vehicle to customize the design of applications even during operation in order to improve their performance

Author's address: K. Papadimitriou and A. Dollas are with the Micropocessor and Hardware Laboratory, ECE Dpt., Technical University of Crete, Kounoupidiana Campus, 73100, Greece. S. Hauck is with the Adaptive Computing Machines and Emulators Laboratory, EE Dpt., University of Washington, Seattle, WA 98195, USA.
Part of this work was conducted while K. Papadimitriou was with the University of Washington as a visiting predoctoral student.

in terms of different aspects, such as speed, power, and area. In this work we focus on fine-grain partially reconfigurable (PR) devices. Previous published work has shown that i)specialized circuits can operate at higher speeds vs. their general static counterparts [McKay et al. 1998], ii)chip area is saved by programming only the physical resources that are needed in each operation phase [Gholamipour et al. 2009], iii)power can be saved by programming only the circuit that is needed, which allows for static leakage reduction, and by programming optimized circuits, which allows for dynamic power reduction [Noguera and Kennedy 2007; Paulsson et al. 2008].

The configuration data produced to program a reconfigurable device is called a bitstream. When only a portion of a PR device is to be reconfigured a partial bitstream is produced. Although fine-grain devices like FPGAs offer customization at the bit-level allowing for great flexibility, at the same time they require large bitstreams to be configured as opposed to the coarse-grain devices. This induces a considerable reconfiguration time, which cannot be neglected even for devices supporting partial reconfiguration. Usually, published reconfiguration times refer to the configuration port of the chip only [Lysaght et al. 2006]. There exist works proposing the incorporation of PR in their systems, which use such theoretical values and due to unrealistic assumptions the results are questionable. When incorrectly used the reconfiguration time might deviate one or two orders of magnitude from the realistic value [Griese et al. 2004; Galindo et al. 2008]. For "real-world" systems a more holistic approach is needed, taking into account all the system components that contribute to the total reconfiguration overhead, such as the internal and external memory, the internal configuration port, the reconfiguration controller, the FPGA configuration memory, and the connections means. As today FPGA platforms are used as final products and not just for rapid system prototyping, partial reconfiguration can play a significant role, and thus it would be useful to effectively precalculate its overhead.

Performance evaluation frameworks are being developed in order to leverage existing partial reconfiguration design flows with design space exploration at the early stages of development [Hsiung et al. 2008]. Our work relies on experiments with a "real-world" platform used as reference, and with straightforward math, we construct a model to evaluate reconfiguration time for new platforms without being necessary to enter the tedious PR design flow. The model is verified by comparing the results with real measured data and is used to evaluate previously published systems. Our contributions are:

—a survey with the most recent works on measuring reconfiguration times of PR FPGAs, and an investigation of the performance margins for different system setups.

—an analysis of the system factors that contribute to the reconfiguration overhead of which a designer should be aware to evaluate the PR system.

—the formulation of a cost model[1] to evaluate reconfiguration in platforms using an embedded processor to control the reconfiguration; the proposed model does

[1]available on line in [PRCC 2010].

not rely on the throughput of the configuration port alone, and it can be used to compute the reconfiguration time without performing experiments.

The paper is structured as follows: Section 2 has the basics of partial reconfiguration and discusses recent works that include measurement of reconfiguration time. In Section 3 we categorize different system setups of published works, and we analyze the characteristics that affect the performance of reconfiguration. Section 4 describes the reference system architecture. Section 5 analyzes the components that contribute to the total reconfiguration time and proposes the cost model. In Section 6 we first verify the model and then use it on different platforms and compare its results with real measurements appeared in previous publications of other researchers, in order to extract the percentage error. Also we discuss the merits and limitations of the model. Finally, Section 7 summarizes the benefits of our work.

2. STATE OF THE ART

Several vendors are striving to develop devices supporting partial reconfiguration. Xilinx offers the largest FPGA matrices which incorporate this feature, and most research efforts on this field use devices from this vendor. We focus on these devices and we provide some background on partial reconfiguration technology along with the corresponding terminology. Then we discuss recent works that measure the reconfiguration time and throughput on "real-world" platforms with modern FPGAs. Some of them include theoretical formulas to calculate the expected results.

2.1 Background on Partial Reconfiguration

The generic structure of the Xilinx Virtex FPGAs is shown in Figure 1. It is a Virtex-II/Pro device [Xilinx Inc. 2007a], one of the most widely used FPGA devices with partial reconfiguration capability. The newer Virtex-4 and Virtex-5 FPGAs have similar structure. It comprises an embedded processor such as a hardcore PowerPC or a softcore Microblaze, a reconfigurable array, the Processor Local Bus (PLB) [IBM Inc. 2000] and the less efficient On-Chip Peripheral Bus (OPB) [IBM Inc. 2001]. The processor is attached on the PLB bus, which communicates with the OPB through a bridge. The modules implemented in the array logic can be attached to the buses and act as peripherals to the processor. The array is 2-D fine-grain heterogeneous, mainly composed of configurable logic blocks (CLBs), hardcore memory blocks (BRAMs), hardcore DSP units (MULTs) and I/O resources. Each CLB contains look up tables (LUTs), flip-flops, multiplexers and gates that are configured to implement the design logic. The array can be configured either externally or internally, by a processor or a dedicated reconfiguration controller. The serial JTAG, the SPI and the parallel SelectMAP allow for external configuration, whereas the parallel Internal Configuration Access Port (ICAP) allows for internal - partial only - configuration. The ICAP in Virtex-II/Pro has a width of 8-bits, whereas Virtex-4 and Virtex-5 families support 16-bit and 32-bit transfers as well. The maximum operational frequency of ICAP suggested by the vendor is 100 MHz. A BRAM attached to the ICAP caches the configuration bits before they are loaded to the FPGA configuration memory. An IP core called OPBHWICAP [Xilinx Inc. 2006] is provided by the vendor which is connected on the OPB bus as a slave peripheral, and enables a processor to access the configu-

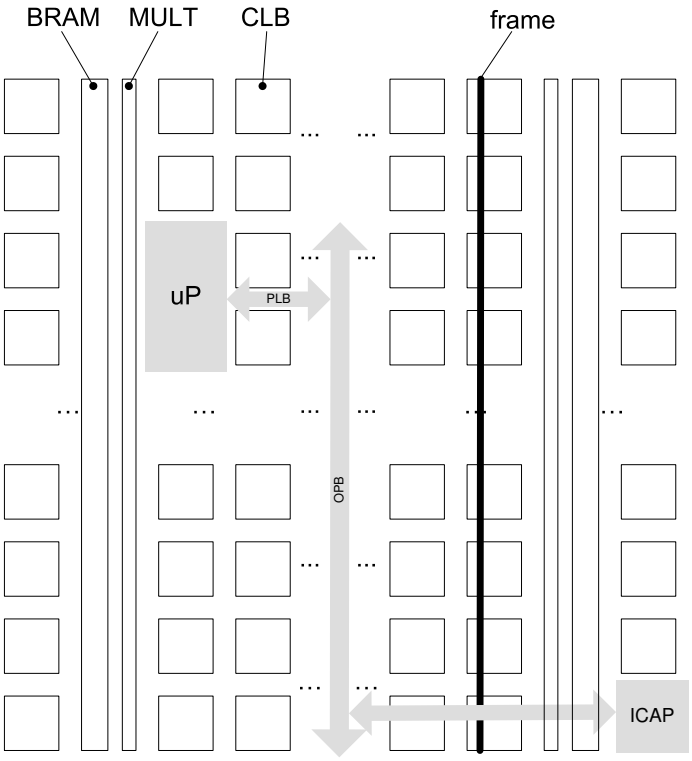

Fig. 1. The Virtex II-Pro FPGA resources and configuration frames.

ration memory through the ICAP, by using a library and software routines. For the Virtex-4 and Virtex-5 FPGAs, the XPSHWICAP [Xilinx Inc. 2007b] has been released, which works similarly with the OPBHWICAP, but it is connected on the PLB bus allowing for lower-latency reconfiguration.

The configuration memory of Virtex-II/Pro is arranged in vertical frames that are one bit wide and stretch from the top edge to the bottom edge of the device. They are the smallest addressable segments of the device's configuration memory space, so all operations must act on entire configuration frames. They do not directly map to any single piece of hardware; rather, they configure a narrow vertical slice of many physical resources [Xilinx Inc. 2007a]. A pad frame is added at the end of the configuration data to flush out the pipeline of the reconfiguration processing machine in order for the last valid frame to be loaded. Therefore, to write even one frame to the device it is necessary to clock in two frames; the configuration data frame plus a pad frame. In the newer Virtex devices, the structure and the PR technology is essentially the same, with a few changes. In the previous Virtex-II/Pro family the frame size is not fixed as it is proportional to the entire height of the device, which differs amongst the parts. By contrast, in Virtex-4 and -5 families the frame spans a fixed height of the device; its width is one bit. Moreover, the OPB bus is not suggested anymore due to its low performance, and although it will not

likely be supported in the future it remains optional within the infrastructure. The PLBv4.6 (also called XPS) is now the default, which provides more bandwidth, less latency and it is more deterministic and stable as compared with the OPB. Also, PLB allows for simultaneous read and write transfers and can be instantiated as either a shared bus, or for point-to-point connections; the latter option reduces the latency of transfers between the modules connected on the bus.

The PR design flow suggested for complex designs is the module-based flow [Lysaght et al. 2006]. One or more areas of the FPGA are defined as partially reconfigurable regions (PRR). For each PRR more than one module called partially reconfigurable modules (PRM), can be created and imported. A partial bitstream is generated to create each PRM. During operation, the PRMs can be swapped in and out of the corresponding PRR without interrupting the remaining hardware.

2.2 Reconfiguration Time and Throughput

Table I has some of the most representative works measuring the reconfiguration time, sorted according to the publication year[2]. It contains the characteristics of "real-world" platforms, along with the respective reconfiguration time and throughput measured with hardware and/or software means, such as a logic analyzer and software time-stamps. The first column has the *Reference*. The *Storage* column has the memory means from which the bitstream is loaded to the configuration memory. The next column consolidates the type, the bit-width and the operational frequency in MHz of the *Configuration port*, e.g. ICAP8@100 refers to the 8-bit ICAP port running at 100 MHz. In case a characteristic is missing it means the information is not provided by the authors. The column *Cntlr* refers to the type of reconfiguration controller. The prefixes "v-" and "c-" define whether the solution is furnished by the "vendor" or by the authors as a "custom" solution respectively. Thus the c-OPB and the c-PLB refer to a custom reconfiguration controller located on the OPB or the PLB bus respectively, whereas the v-OPB and the v-XPS refer to the vendor's OPBHWICAP and XPSHWICAP solutions respectively that are controlled by an embedded processor (alternatively a custom core can be implemented to interface with the vendor's OPBHWICAP and XPSHWICAP). The last three columns have the *Bitstream Size (BS)* in KBytes, the corresponding *Reconfiguration Time (RT)* in msec and the system's *Actual Reconfiguration Throughput (ARTP)* in MBytes/sec. The values are included as published, and in the rare case a value is not explicitly reported in a reference we calculate it, e.g. the fraction of the bitstream size over the reconfiguration time calculates the actual reconfiguration throughput and vice-versa with the formula $ARTP = \frac{BS}{RT} \Leftrightarrow RT = \frac{BS}{ARTP}$. All experiments of Table I concern Virtex-II and Virtex-4 FPGAs, mainly the XC2VP30, XC4FX20 and XC4FX60 devices. The 8-bit ICAP corresponds to the Virtex-II, and the 32-bit to the Virtex-4 FPGA families.

Along with the Table I, we discuss partially reconfigurable systems by concentrating on i)the type of the external storage from which the partial bitstream is loaded to the array, ii)the type of the reconfiguration controller, iii)the measurement of the

[2]At this point we preferred to sort Table I according to the publication year (first column) than according to the reconfiguration throughput (last column). A categorization according to the throughput is provided in the following Table.

Table I. Reconfiguration-related characteristics and measured reconfiguration time and throughput. The Bitstream Size (BS) is in KBytes, the Reconfiguration Time (RT) in milliseconds, and the Actual Reconfiguration Throughput (ARTP) in MBytes/sec.

Reference	Storage	Conf. Port	Cntlr	BS	RT	ARTP
[Fong et al. 2003]	PC/RS232	ICAP8	c-HW	34.8	6,200	0.005
[Griese et al. 2004]	PC/PCI	SMAP8@25	c-HW	57.0	14.328	3.88
[Gelado et al. 2006]	$BRAM_{PPC}$	ICAP8	v-OPB	110.0	25.99	4.13
[Delahaye et al. 2007]	SRAM	ICAP8	v-OPB	25.7	0.510	49.20
[Papadimitriou et al. 2007]	$BRAM_{PPC}$	ICAP8@100	v-OPB	2.5	1.67	1.46
[Papadimitriou et al. 2010]	CF	ICAP8@100	v-OPB	14.6	101.1	0.15
[Claus et al. 2007]	DDR	ICAP8@100	c-PLB	350.75	3.75	91.34
[Claus et al. 2007]	DDR	ICAP8@100	v-OPB	90.28	19.39	4.66
[Claus et al. 2008]	DDR	ICAP8@100	c-PLB	70.5	0.803	89.90
[Claus et al. 2008]	DDR	ICAP8@100	v-OPB	70.5	15.0	4.77
[Claus et al. 2008]	DDR2	ICAP32@100	c-PLB	1.125	0.004	295.40
[Claus et al. 2008]	DDR2	ICAP32@100	v-OPB	1.125	0.227	5.07
[French et al. 2008]	DDR2	ICAP32@66	v-OPB	514.0	112.0	4.48
[Manet et al. 2008]	DDR2/ZBT	ICAP32@100	c-OPB	166.0	0.47	353.20
[Liu et al. 2009]	DDR2	ICAP32@100	v-OPB	79.9	135.6	0.61
[Liu et al. 2009]	DDR2	ICAP32@100	v-XPS	80.0	99.7	0.82
[Liu et al. 2009]	DDR2	ICAP32@100	v-OPB	75.9	7.8	10.10
[Liu et al. 2009]	DDR2	ICAP32@100	v-XPS	74.6	4.2	19.10
[Liu et al. 2009]	DDR2	ICAP32@100	c-PLB	81.9	0.991	82.10
[Liu et al. 2009]	DDR2	ICAP32@100	c-PLB	76.0	0.323	234.50
[Liu et al. 2009]	$BRAM_{ICAP}$	ICAP32@100	c-PLB	45.2	0.121	332.10

reconfiguration process and its phases, and iv)the theoretical formulas to calculate the expected reconfiguration time and throughput. Regarding the configuration port we focus on ICAP which offers the fastest way to partially reconfigure the array.

Commonly, the partial bitstreams are stored in an external non-volatile memory. Therefore a host PC can be used which, after the system startup, transfers the bitstream to the FPGA either through PCI [Griese et al. 2004] or serial port [Fong et al. 2003]. For embedded solutions a compact flash located on the FPGA platform is preferred, which is the case for most works of Table I. Other non-volatile memories that can be used are linear flash, SPI flash, and platform flash PROM. After the system boots, the bitstream can be loaded to a high-speed memory, either off-chip or on-chip, in order to perform faster reconfiguration. If only the compact flash is used, a longer reconfiguration time is required [Papadimitriou et al. 2007]. More on the advantages and disadvantages of different memory types are available in [Möller et al. 2006].

With respect to the type of reconfiguration controller various solutions exist, each of which can be employed depending on the application needs. Some works used the OPBHWICAP [Gelado et al. 2006; Papadimitriou et al. 2007; Delahaye

et al. 2007; Claus et al. 2007; Claus et al. 2008; French et al. 2008; Liu et al. 2009] whereas a few experimented with the XPSHWICAP [Liu et al. 2009]. On the other hand, customized reconfiguration controllers aim to speedup reconfiguration and/or release the processor time to other tasks, such as the reconfiguration management under the supervision of an operating system [Santambrogio et al. 2008]. This can be done either through Direct Memory Access (DMA) with a customized controller on the OPB [Manet et al. 2008] or the PLB bus [Claus et al. 2007; Claus et al. 2008; Liu et al. 2009], or with a PLB master connected directly with the controller of the external memory, allowing for burst transmissions [Liu et al. 2009]. Moreover, to speed up reconfiguration, partial bitstreams can be loaded immediately after the system startup into large on-chip memory implemented with BRAMs attached to the reconfiguration controller [Liu et al. 2009]. In general, the systems with the processor acting as the reconfiguration controller suffer from long delays due to the large time needed to access its program memory to call and execute the software instructions. Also, when several modules are connected to the bus the delay is longer and unstable due to the contention between reconfiguration and data transfers.

The distinct phases during reconfiguration can vary depending on the system setup. The measured times mainly concern the phase to pull the bitstream from the off-chip memory to the local memory of the processor, copy it from the local memory of the processor to the ICAP, and send it from the ICAP to the FPGA configuration memory [Gelado et al. 2006; Papadimitriou et al. 2007; French et al. 2008]. These subtasks iterate until the entire partial bitstream is written to the configuration memory [Papadimitriou et al. 2010]. For specific systems other subtasks have been measured, such as the time for configuration code analysis to guarantee safety, initialization and start time of the reconfiguration controller [Griese et al. 2004], the time to send the adequate instructions to the ICAP [Gelado et al. 2006], and the time to copy the configuration data from user space to kernel space in Linux [French et al. 2008]. In all systems the largest overhead comes from the bitstream transfer from the external memory to the on-chip memory [Griese et al. 2004; Papadimitriou et al. 2007; 2010], and from the processor to the ICAP through the bus [Gelado et al. 2006; French et al. 2008; Delahaye et al. 2007].

Although some of the above works evaluate the time spent in each reconfiguration phase, they do not provide a theoretical analysis taking into account the system characteristics. Some of them calculate the expected reconfiguration throughput based on the bandwidth of the configuration port only [Claus et al. 2007; Manet et al. 2008]. However, in [Griese et al. 2004] the authors demonstrated that the actual reconfiguration throughput deviates one order of magnitude from the theoretical bandwidth of the configuration port. In another work this deviation reaches up to two orders of magnitude [Galindo et al. 2008]. Following the same concept, Lysaght et al. [Lysaght et al. 2006] reports the reconfiguration times for Virtex-II Pro and Virtex-4 devices. For example the partial bitstream for 25% of a XC2VP30, which is equal to 353 KBytes, can be loaded in less than 6 ms when the ICAP operates at 100 MHz and is fully utilized. This has a considerable difference with the end-to-end system times, and depending on the system setup the actual reconfiguration times can take significantly longer, as demonstrated in Table I. Claus et al. [Claus et al. 2008] provide a more sophisticated model to calculate the expected

reconfiguration throughput and latency. However, this model is based on the characteristics of the configuration port itself only; these are the operational frequency, the interface width and the frequency of activation of the ICAP's "BUSY" handshaking signal. This model cannot be safely applied on the Virtex-4 devices due to the different behavior of the "BUSY" signal in the specific family. Although this model can be fairly accurate under specific circumstances, if a designer wants to evaluate a system, (s)he would need to built it first, then evaluate it, and then examine whether it meets the application needs. We are interested in a model that will provide better estimation regardless of the maximum throughput of the configuration port early in the design stage.

3. INVESTIGATION OF THE RECONFIGURATION PERFORMANCE

In this Section we analyze further the characteristics that affect the reconfiguration performance. Table II categorizes the existing approaches according to the configuration port and its bandwidth, the reconfiguration controller and the storage means. It also summarizes the expected range of the actual reconfiguration throughput according to the system characteristics. We observe that the throughput varies heavily depending on the three characteristics.

When comparing the *cases A and B* where the PowerPC (PPC) processor is the reconfiguration controller, loading the partial bitstream from the local memory of the processor implemented with BRAMs offers a speedup of 10x over the compact flash solution [Papadimitriou et al. 2007]. In *case I*, where on-chip memory implemented with BRAMs attached to the ICAP is used to fetch the entire partial bitstream prior reconfiguration takes place, much better results are produced. In this case a dedicated reconfiguration controller accounts for the high throughput, as it continuously pulls the prefetched configuration data; this allows for the almost full utilization of the configuration port [Liu et al. 2009].

In *case D* where a customized reconfiguration controller located on the PLB bus is utilized, the reconfiguration throughput reaches almost the bandwidth of the 8-bit ICAP configuration port [Claus et al. 2007; Claus et al. 2008]. However in

Table II. Comparison between the bandwidth of the configuration port and the actual reconfiguration throughput for different published system setups.

	Configuration Port				
case	Type	Bandwidth	Cntlr	Storage	ARTP
A	ICAP8@100	95.3 MB/s	v-OPB	CF	0.15 MB/s
B			v-OPB	$BRAM_{PPC}$	1.46 MB/s
C			v-OPB	DDR	4.54-4.77 MB/s
D			c-PLB	DDR	89.90-91.34 MB/s
E	ICAP32@100	381.5 MB/s	v-OPB	DDR2	0.61-11.1 MB/s
F			v-XPS	DDR2	0.82-22.9 MB/s
G			c-OPB	DDR2/ZBT	353.2 MB/s
H			c-PLB	DDR2	82.1-295.4 MB/s
I			c-PLB	$BRAM_{ICAP}$	332.1-371.4 MB/s

case H when the 32-bit ICAP is used, the same setup cannot achieve the configuration port bandwidth. This might be due to the DDR2 SDRAM controller which is not able to feed the reconfiguration controller fast enough in order to attain the ICAP bandwidth [Claus et al. 2008; Liu et al. 2009]. High throughput approaching the theoretical maximum is also achieved in *case G* with a custom reconfiguration controller located on OPB [Manet et al. 2008]. In this work, the authors reported that the throughput was ranging from 0.46 to 353.2 MB/sec for different system setups depending on the reconfiguration controller and the type of external memory. Their fastest solution was provided with a custom OPB ICAP reconfiguration controller when the partial bitstreams were loaded from a high-speed external memory; however the authors do not report whether the DDR2 SDRAM or the ZBT SRAM was used. They were not able to reach the maximum throughput due to the DMA overhead. Also, due to a clock skew problem they sent the configuration data on the falling edge of the ICAP clock. It is interesting to notice that the throughput of the best case of the custom OPB controller of [Manet et al. 2008] which is 353.2 MB/s is higher than the custom PLB controller of [Claus et al. 2008] which is 295.4 MB/s, despite the fact that the former is attached on the OPB bus and the latter on the more-efficient PLB bus (see Table I of Section 2). We see two possible reasons that the system with the OPB reconfiguration controller offers higher throughput; either i)the bitstreams are fetched from the ZBT SRAM which provides higher throughput than the DDR2 SDRAM and consequently allows for higher utilization of the ICAP bandwidth, or ii)if the bitstreams are fetched from the DDR2 SDRAM, the corresponding controller is implemented more efficiently.

When the processor acts as the reconfiguration controller its settings can affect significantly the reconfiguration speed. This holds for both the OPBHWICAP and the XPSHWICAP of *cases E and F* respectively. Such settings are the selection of a hardcore processor (e.g PowerPC), or a softcore processor (e.g. Microblaze), and the incorporation of separate instruction cache and data cache [Liu et al. 2009]. Another study shows that the change of the amount of memory allocated in the processor local memory for buffering reads and write calls to the compact flash caused variations in the reconfiguration time with respect to the number of reconfiguration frames [Papadimitriou et al. 2010]. Another action towards increase of reconfiguration performance is the replacement of the vendor's Application Program Interface (API) that handles the ICAP accesses with a better one [Möller et al. 2006].

Based on the above works we conclude that for recent PR technology on "real-world" systems:

—The reconfiguration throughput of the system cannot be calculated from the bandwidth of the configuration port alone. To evaluate the system holistically, the overhead added by the components participating in the reconfiguration process should be taken into account.

—The characteristics that affect the reconfiguration overhead depend on the system setup, such as the external memory and the memory controller, the reconfiguration controller and its interface with the configuration port, and the user space to kernel space copy penalty when an operating system running on the processor controls the reconfiguration.

—A bus-based system that connects the processor, the partially reconfigurable mod-

ule(s), the static module(s), and the configuration port, can be non-deterministic and unstable due to the contention between data and reconfiguration transfers.

—A large on-chip memory implemented with BRAMs attached to the configuration port that prefetches the bitstreams after the system boots can allow for fast reconfiguration. Due to the limited size of BRAMs in FPGAs, the utilization cost should be considered according to the application needs, i.e. using BRAMs to prefetch configuration vs. saving BRAMs to deploy the circuits.

—The settings of the processor can affect the reconfiguration time. Moreover, the size of the local memory of the processor can affect significantly the reconfiguration time. However, similar to the above case the cost of BRAM utilization should be considered.

—The selection of the reconfiguration controller depends on the application needs, the available resources, the speed and the power constraints. For example, in case area is of greater concern than speed, and an operating system executes on an embedded processor, the latter can undertake the reconfiguration process in order to avoid the consumption of logic resources to implement a dedicated reconfiguration controller. Previous works suggested ways to reduce the reconfiguration overhead if this is needed [Hauck 1998; Papadimitriou and Dollas 2006].

—Nearly the full bandwidth of the configuration port can be used with a dedicated reconfiguration controller that is either equipped with DMA capabilities, or acts as a master directly connected to the configuration port allowing for burst transmissions. Also, effective design choices at the system level should be made such as the usage of a high-speed memory to store the partial bitstreams after power-up and an efficient memory controller.

Obviously only the development of an efficient reconfiguration controller can allow for the actual reconfiguration throughput to approach the ICAP bandwidth. However, this is not the common case either due to the system components that bound the full utilization of the ICAP or the designer's choices and application needs. A cost model to precalculate the expected reconfiguration time can assist in the performance evaluation phase. Moreover, it would be beneficial if this model could be used early in the design flow in order to either i)prevent the designer from entering the tedious PR design flow, or, ii)timely change the settings of the system setup. The evaluation can be done either right after the bitstream estimation of the partially reconfigurable modules at the initial budgeting stage of the floorplanning stage, or even after the synthesis results. Although the latter method is less accurate, our experience so far showed that the cost model still produces satisfactory results.

Several works exploring applications that benefit from partial reconfiguration can be augmented with a cost model for partial reconfiguration in order to provide realistic data. For the dynamically reconfigurable network processor proposed in [Kachris and Vassiliadis 2006] the cost model can calculate the time needed for reconfiguration after which the system will meet the network workload. Bitstreams of different encoding, encryption and transcoding modules are fairly large and should be located in an external memory, so, the added overhead before the system is ready to execute is useful to know. In the field of multi-mode communication systems the

Table III. Size of BRAM resources for moderate-sized FPGAs of Virtex-II,-4,and -5 families. The Table illustrates the maximum size of the bitstreams that can be prefetched in the extreme case none of the actual circuits utilizes BRAMs.

FPGA	BRAM size (KBytes)
XC2VP30	306
XC4FX60	522
XC5VLX50T	270

cost model can be used to calculate the time to load the optimized configurations of the filters [Gholamipour et al. 2009].

Furthermore, assuming that unlimited amount of on-chip memory is available to fetch the entire bitstream and then load it at the ICAP's speed is not realistic. An analysis on the memory resources required by the circuit itself when deployed in the field should be performed firstly by the designer. Also, an analysis of the local memory of the processor and the bitstream size is needed, prior deriving the amount of BRAMs that can be allocated for bitstream prefetching. Table III shows that moderate-size FPGAs commonly used for research have limited BRAM resources. In many cases the amount of on-chip memory imposes limitations to the size of the bitstreams that can be prefetched; this is obtained from Table I that contains bitstream sizes for several "real-world" applications.

4. SYSTEM ARCHITECTURE

Initially we present the general architecture of a partially reconfigurable system, and then we describe in detail a realistic system with a Xilinx Virtex-II Pro FPGA. It is used as the reference system to develop the cost model.

4.1 General Architectural Model

Figure 2 illustrates the general architectural model of a PR FPGA-based system. The FPGA main components are the array, the configuration memory that programs the array, the reconfiguration controller, the configuration port and the on-chip memory buffer. The connection between them is made either point-to-point or through a bus. Other system components are the volatile high-speed memory from which the partial bitstreams are loaded and its memory controller. The memory controller can reside either off-chip or on-chip implemented as an IP module. Other necessary components are the non-volatile memory used as a repository for the partial bitstreams and its controller. However, these are omitted from Figure 2 as after the system boots, the bitstreams can be copied to the high-speed memory (for faster reconfiguration), and thus they do not need to be involved in the process again. If the off-chip high-speed memory is left out and each time the bitstream is loaded directly from the compact flash, throughput becomes flash-dominated and little transfer speed improvement is possible.

The configuration takes place in two distinct phases shown with the dashed lines. Once the memory controller is instructed to load the bitstream, it is copied from the off-chip memory to the on-chip memory buffer. Then the reconfiguration controller loads the bitstream to the FPGA configuration memory through the configuration port. These phases occur alternately in succession until the entire bitstream is

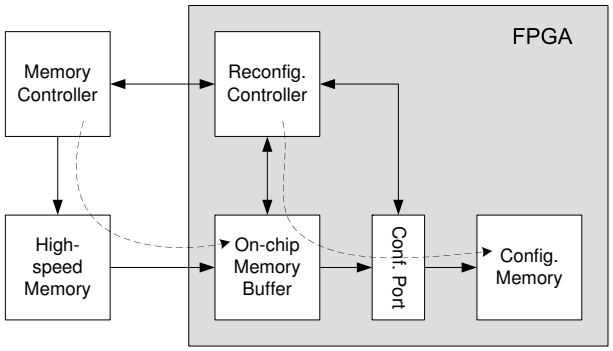

Fig. 2. General architectural model and flow of partial reconfiguration.

copied to the configuration memory, except for the case when the entire bitstream is prefetched in the on-chip memory. If this is desirable (depending on the user application) and feasible (depending on the available on-chip memory) the first phase can be executed before reconfiguration is actually needed, i.e. after the system boots, or, scheduled during system operation before the task is needed for execution. It is important to notice that the on-chip memory can be composed of separate memory blocks that communicate hierarchically; this holds for the following system.

4.2 The Reference System

Figure 3 shows a PR system implemented on a platform with a Virtex-II Pro FPGA [Papadimitriou et al. 2007]. The white boxes are part of the FPGA, while the grey boxes lie outside of the FPGA. The Device Control Register (DCR) bus is used for communicating with the status and control registers of the peripherals. The PPC is attached on the PLB bus and contains two interfaces capable of accessing memory, the Processor Local Bus (PLB) interface and the On-Chip Memory (OCM) interface; in the present case we used the first interface by attaching on the bus a PLB_BRAM controller connected with the PPC local memory. The OPBHWICAP incorporates a finite state machine for control, a configuration cache implemented with one BRAM (set 2 KBytes by the vendor) and the ICAP. The PRR and the static part are OPB peripherals. The OPB sysace peripheral is used to interface with the external System ACE controller in order to communicate with the compact flash. A DDR controller attached on the PLB controls the DDR SDRAM memory; however it is not used in the current setup.

The PR system is representative as it belongs to the largest vendor in the field that supports fine-grain partial reconfiguration. Also, although the PowerPC is currently used as the embedded processor to control the reconfiguration, the softcore Microblaze can be used instead, which also supports the reconfiguration functions. Other custom processors can be used as well but the appropriate functionality should be developed from the scratch to support reconfiguration.

At this point it is important to discuss the way the bus-based system works. The IBM CoreConnect infrastructure is the backbone of the internal system connecting the processor to the peripherals using the PLB, the OPB, and the DCR buses to build a complete system [IBM Inc. 2000; 2001; 2006]. The DCR bus is used for

Table IV. System settings.

Parameters	Size
CF sector	512 Bytes
PLB BRAM block	48 KB
stack	6 KB
buffer cache	0.5-4 KB
processor array	0.5-4 KB
ICAP BRAM cache	2 KB

accessing status and control registers within the PLB and OPB masters and slaves. It off-loads the PLB and the OPB from the lower performance read/write transfers from/to the status and control registers. It also decreases access latency during periods of high processor bus utilization. DCR is not part of the system memory map, and thus it removes configuration registers from the memory address map, allowing for data transfers to occur independently from, and concurrent with, PLB and OPB transfers.

In the setup of Figure 3, the PPC is the controller of the reconfiguration process carried out in three distinct phases that are repeated until the entire bitstream is written in the configuration memory [Papadimitriou et al. 2010]. These phases are shown with the dashed lines: i)First the PPC requests the bitstream from the external storage means (the compact flash in the present case) and writes it in its local memory (first level of the on-chip memory hierarchy) with block-by-block transactions, ii)then the PPC transfers the bitstream word-by-word to the ICAP configuration cache (second level of the on-chip memory hierarchy), and iii)once the configuration cache is full the PPC instructs the OPBHWICAP to load the bitstream to the FPGA configuration memory through the ICAP. These actions are determined with software commands residing in the PPC program memory. For sake of simplicity we will refer to them as SM-PPC[3], PPC-ICAP and ICAP-CM respectively.

Table IV has the system parameters we configured as fixed and those we varied to examine their affect on performance. The PPC's local memory was 48 KBytes and the stack was 6 KBytes. The buffer cache (bc) defines the amount of memory for buffering read and write calls to the System ACE controller. The processor array (pa) is allocated in the PPC local memory to store the configuration data, and determines the amount of data read from the compact flash per transaction[4]. The ICAP cache has fixed size set by the vendor one BRAM, which is equal to 2 KBytes. In order to configure a system with low resources we did not enable the instruction/data caches and interrupts of the PPC.

Table V has the time spent in the processor calls (which force the transactions)

[3]SM and CF acronyms are used throughout the paper depending on the type of storage we refer to, i.e. SM stands for storage means referring to a general type of memory, either volatile or non-volatile, high-speed or low-speed, and CF stands for the compact flash which is non-volatile low speed memory.

[4]Transactions are conducted in multiples of one sector per processor call. A sector is the smallest unit the compact flash is organized in and is equal to 512 Bytes. Thus we varied the processor array size in multiples of one sector.

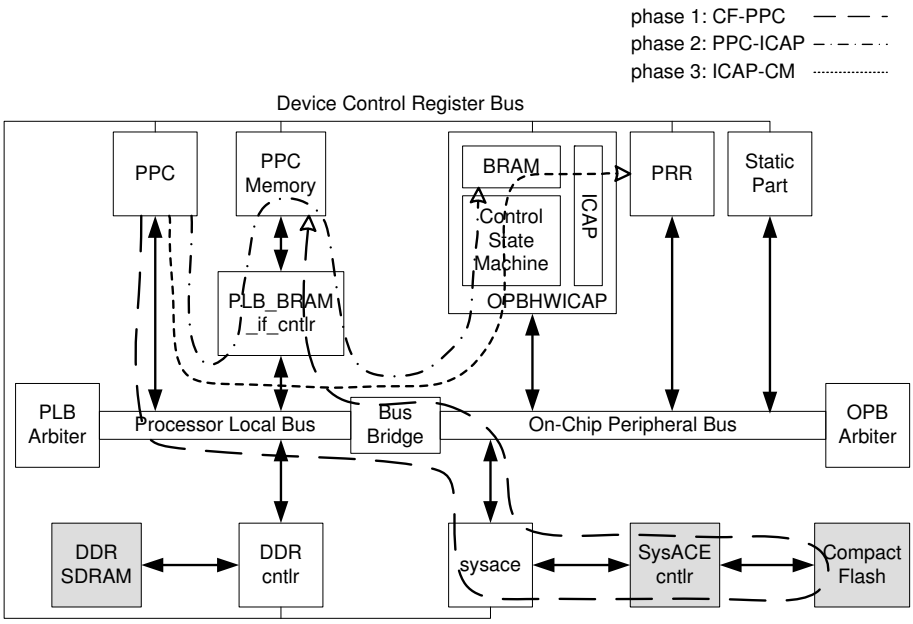

Fig. 3. Flow of partial reconfiguration in the reference system with a Virtex-II Pro.

during reconfiguration for different processor array sizes. They were measured with software time-stamps. It is observed that the processor array dictates the amount of data transferred during the SM-PPC phase. In the same way it dictates the transfer in the PPC-ICAP phase. However, in the case the processor array is larger than the ICAP cache, the amount of data per call is bounded by the latter's size. This is illustrated in the fifth row of Table V where the PPC-ICAP time is the same with the one shown in the above row. Moreover, if the processor array is larger than the bitstream, the latter is transferred with one processor call and stored in the processor local memory in its entirety prior the transfer to the ICAP. From the ICAP side, the configuration cache size is fixed and thus inhibits accommodation and transfer of the entire bitstream at once. Also, the ICAP cache should be filled up with 2048 Bytes before it is ready to write the configuration memory. This is the reason the ICAP-CM time is constant in all cases. For example, in the first row of Table V where processor array = 512 Bytes, iterations of the first and the second phases are executed in succession (with the corresponding processor calls) until the ICAP cache is filled up. In particular, four consecutive iterations of the two first phases occur until the ICAP cache will become full with 2048 Bytes. Then, in the third phase, the contents of the ICAP cache are loaded to the FPGA configuration memory with one transaction (caused by the corresponding processor call). The foregoing sequence is repeated until the entire bitstream is copied to the configuration memory.

The above observations revealed the way the reconfiguration operation works. We processed the data of Table V to obtain a metric that has a meaningful system-level

Table V. Execution time-per-processor-call for different sizes of the processor array. The amount of data transferred per call is dictated by the minimum of the processor array and the ICAP cache. In the ICAP-CM phase, the processor call needs 2048 Bytes to perform a transaction.

processor array (Bytes)	SM-PPC (ms)	PPC-ICAP (ms)	ICAP-CM (ms)
512	1.5	0.42	
1024	2.94	0.83	
1536	4.35	1.25	0.02526
2048	5.79	1.67	
2560 ($> ICAP\ cache$)	7.24	1.67	

Table VI. Average time-per-processor-call. In order to conduct realistic analysis we adhere to the way the phases are carried out by the corresponding processor calls; the minimum amount of data for the SM-PPC and PPC-ICAP phases to be carried out is 512 Bytes, while ICAP-CM phase needs exactly 2048 Bytes.

SM-PPC	PPC-ICAP	ICAP-CM
1.45 ms (per-512 Bytes)	0.42 ms (per-512 Bytes)	0.02526 ms (per-2048 Bytes)

design correspondent. Specifically, we extracted the "average time-per-processor-call". We observe that for the SM-PPC and PPC-ICAP phases, the time per 512 bytes (the smallest unit that can be transferred) is almost constant. For the SM-PPC this time is $\sim 1.45ms$, while for PPC-ICAP it is $\sim 0.42ms$. For the ICAP-CM phase the quantity of 512 Bytes is not the smallest quantity, as exactly 2048 Bytes are needed to initiate a transfer to the configuration memory. In turn, we formed the Table VI which will be used in Section 5 to construct the cost model.

4.3 Options Affecting Reconfiguration Performance

Based upon the foregoing observations we list the optional processor features that can improve reconfiguration speed:

—The processor array puts the upper limit on the amount of configuration data transferred from the compact flash per processor call. Moreover, when the processor array is smaller than the ICAP cache, the former puts a limit on the amount of configuration data sent to the ICAP cache per processor call.

—Enabling the I-Cache and D-Cache of the processor can improve significantly the reconfiguration throughput. Although their deactivation accounts for resource savings, it causes diminishing results in performance. A study for the Virtex-4 has proven that for a system the overall improvement ranges from 16.6 to 23.3 times [Liu et al. 2009].

—The Application Program Interface (API) provided by Xilinx allows for the software control of the IP core (OPBHWICAP or XPSHWICAP) for accessing ICAP. It requires the software controller on the processor to fetch 512-word blocks of the configuration data and store them in a BRAM attached as cache to the ICAP. Then the API instructs ICAP to program the configuration memory. This process is slow and an improved API can increase the performance. [Möller et al. 2006].

—The stack of the processor should be set at least 4 KBytes. Otherwise the system

hangs while reading from the compact flash.

—Reading from the compact flash is a very slow process. Fetching the configuration data to a volatile high speed memory after the system boots can improve the performance.

—Setting the PPC memory on the shared PLB through the PLB_BRAM controller might not be effective. The dedicated On Chip Memory (OCM) interface provides lower latency and faster access [Lund 2004].

The above conclusions, in combination with the Table II of Section 3, can support the designer to make initial decisions on the system setup.

5. DEVELOPMENT OF A COST MODEL

Initially we formulated the cost model on the reference system and then we extended it to suit more generic cases.

5.1 The Cost Model

The total reconfiguration time (RT) can be expressed by the sum of the times spent in each phase of the reconfiguration process:

$$RT = RT_{SM-PPC} + RT_{PPC-ICAP} + RT_{ICAP-CM} \tag{1}$$

The phases occur in a successive manner and are repeated until the entire bitstream is loaded to the configuration memory. Thus we are interested in the aggregate time spent in each phase. All phases are controlled with software instructions residing in the processor's program memory, and there is no overlap between the phases as the instructions are executed in-order. The processor calls the routines to transfer the configuration data from the storage means (compact flash in present case) to its local memory, then to the ICAP configuration cache, and then to the configuration memory. If for every phase, the number of executed processor calls, the amount of configuration data per call, and the time per call are known, we will be able to compute the aggregate reconfiguration time per phase, and finally the total reconfiguration time. Thus the Equation (1) becomes

$$RT = SM_{calls} \times SM_{time} + IC_{calls} \times IC_{time} + CM_{calls} \times CM_{time} \tag{2}$$

To compute the number of processor calls to the compact flash, SM_{calls}, first we need to know the amount of data (measured in bytes) to be transferred. If fs the frame size (measured in bytes) of the corresponding FPGA (e.g. 824 bytes for XC2VP30 and 164 bytes for all Virtex-4 and Virtex-5), then for n frames, the amount of reconfiguration bytes including the pad frame is $fs \times (n + 1)$. At this point we recall from Section 4 the way transactions from the compact flash to the processor are performed. The amount of configuration data transferred with one transaction depends on the size of the processor array, denoted as pa, allocated in the processor memory. Its size is defined in multiples of one sector, and thus transactions are conducted in multiples of one sector with the corresponding processor call. The number of calls for a given number of reconfiguration frames is

$$SM_{calls} = \frac{fs \times (n+1)}{pa} \tag{3}$$

The time per processor call to the compact flash, SM_{time}, was measured with software time-stamps. It depends directly on the *pa* size. From Table VI the time per 512 bytes has an average value of 1.45 ms. Thus the time per call is

$$SM_{time} = \frac{pa \times 1.45ms}{512} = \frac{pa}{353}ms \tag{4}$$

From Equations (3) and (4) the aggregate time of the 1st phase of reconfiguration process is

$$SM_{calls} \times SM_{time} = \frac{fs \times (n+1)}{353}ms \tag{5}$$

To compute the time spent in the processor calls for transferring the bitstream to the ICAP cache we follow the same concept. The amount of configuration data transferred from the processor local memory to the ICAP cache with one transaction depends on the *pa* size, unless the latter is larger than the ICAP cache. In this case the size of the ICAP cache, denoted as ic_{size}, puts the upper limit on the amount of bytes transferred per transaction. The reason is that a new write transfer to the ICAP cache is accepted only if the transfer of the previous configuration data to the configuration memory has been completed. We introduce a quantity named *block* which is

$$block = min(pa, ic_{size})$$

The number of the corresponding processor calls to transfer the bitstream is

$$IC_{calls} = \frac{fs \times (n+1)}{block} \tag{6}$$

The time per processor call to the ICAP cache, IC_{time}, was measured with software time-stamps. It depends directly on the *block* size. As obtained from Table VI the time spent per 512 bytes has an average value of 0.42 ms. Also, it stops increasing after the amount of configuration data exceeds the size of ICAP cache as the latter cannot accept more than 2 KBytes per transaction. Thus the time per call is

$$IC_{time} = \frac{block \times 0.42ms}{512} = \frac{block}{1219}ms \tag{7}$$

From Equations (6) and (7) the aggregate time of the 2nd phase of reconfiguration process is

$$IC_{calls} \times IC_{time} = \frac{fs \times (n+1)}{1219}ms \tag{8}$$

Similarly, we compute the time spent in the processor calls for loading the contents of the ICAP cache to the configuration memory. The ic_{size} puts the upper limit on the amount of data written per transaction. The number of the corresponding processor calls to transfer the bitstream is

$$CM_{calls} = \frac{fs \times (n+1)}{ic_{size}} \tag{9}$$

The time per processor call to the configuration memory, CM_{time}, was measured with software time-stamps. As shown in Table VI it is steady due to the fixed volume transfers from the ICAP cache to the configuration memory, which directly

Table VII. Percentage of the time spent in each phase of reconfiguration and the corresponding measured throughput and theoretical bandwidth. The values concern the reference system.

	CF-PPC	PPC-ICAP	ICAP-CM
%RT	77.28%	22.38%	0.34 %
ARTP (calculated from Table V)	0.33 MB/s	1.17 MB/s	77.32 MB/s
Theoretical Bandwidth (provided in data-sheets)	64 MB/s	95.3 MB/s	95.3 MB/s

depends on the ic_{size}. In Virtex-II Pro the ICAP cache size is limited to one BRAM, thus $ic_{size} = 2048\ Bytes$. For a different ic_{size}

$$CM_{time} = \frac{ic_{size} \times 0.02526ms}{2048} = \frac{ic_{size}}{81077}ms \tag{10}$$

From Equations (9) and (10) the aggregate time of the 3rd phase of reconfiguration process is

$$CM_{calls} \times CM_{time} = \frac{fs \times (n+1)}{81077}ms \tag{11}$$

Therefore from the Equations (5), (8), (11), the Equation (2) becomes

$$\begin{aligned} RT &= fs \times (n+1) \times (\frac{1}{353} + \frac{1}{1219} + \frac{1}{81077})\ ms \\ &= fs \times (n+1) \times 3.66 \times 10^{-3}\ ms \end{aligned} \tag{12}$$

where fs is measured in bytes. Equation (12) holds for a PR system where the bitstreams are loaded directly from a compact flash under the control of a processor. It is observed that the variables pa and ic_{size} do not affect the reconfiguration time.

5.2 Model Extension

Using the above Equations, we derive the percentage of the reconfiguration time (RT) spent in each phase, shown in the first line of Table VII. Also, based upon the values of the Table V and from the fraction of the amount of bytes transferred over the transfer time we directly calculate the maximum achievable throughput (ARTP) of each phase, shown in the second line of Table VII. The throughput refers to the number of total bytes - including the overhead - successfully delivered per second. The values in the Table illustrate the magnitude of the difference between the throughput and the theoretical bandwidth, shown in the second and the third line of the Table VII respectively. The theoretical bandwidth is determined by the slowest physical component involved in the corresponding reconfiguration phase, and it is obtained from the data-sheets; for the first phase this is the compact flash system, for the second phase it is the access to the ICAP cache, and for the third phase it is the access to the configuration memory through the ICAP.

Presently, the slowest phase of the reconfiguration process is the SM-PPC, which consumes 77.28% (see Table VII) of the reconfiguration process. If instead of the compact flash we use a faster type of external memory the system performance would increase. To find this speedup we apply Amdahl's law in a straightforward manner. The law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S. The total speedup of applying the improvement

is

$$Speedup_{total} = \frac{Time_{old}}{Time_{new}} = \frac{1}{(1-P) + \frac{P}{S}} \tag{13}$$

We adjust the problem to our case which is concerned with *the reconfiguration speedup achievable from an improvement to the reconfiguration time that affects a proportion P (which corresponds to the SM-PPC phase) of that reconfiguration time where the improvement has a speedup of* S_{SM-PPC}, which results in

$$\frac{RT_{CF}}{RT_{SM}} = \frac{1}{(1-P) + \frac{P}{S_{SM-PPC}}} \Leftrightarrow RT_{SM} = (1 - P + \frac{P}{S_{SM-PPC}}) \times RT_{CF} \tag{14}$$

where RT_{CF} and RT_{SM} are the reconfiguration times for the reference system with the compact flash and the optimized system with the faster external storage respectively. For $P = 77.28\%$ and by using the Equation (12) to calculate the RT_{CF}, the Equation (14) becomes

$$RT_{SM} = fs \times (n+1) \times (0.83 + \frac{2.83}{S_{SM-PPC}}) \times 10^{-3}\ ms \tag{15}$$

Equation (15) determines the performance margins when the improvement is applied on the first phase of the reconfiguration process. It is more generic than Equation (12) in that it calculates the reconfiguration time not only for compact flash but for other storage means such as a DDR SDRAM or a ZBT SRAM. If an identical compact flash to the one of our reference setup is used to load directly the bitstream, the speedup factor would be $S_{SM-PPC} = 1$. In case a different storage is used, a proportional analysis regarding the performance gained over the compact flash is needed, in order to quantify the S_{SM-PPC}. This analysis along with the usage and effectiveness of the formula is described in Section 6.

Equation (15) can be used to calculate the reconfiguration throughput of useful frames, i.e. without the pad frame. The bitstream size that corresponds to the useful configuration is $BS = fs \times n$. Thus

$$useful\ ARTP_{SM} = \frac{BS}{RT_{SM}} = \frac{n}{n+1} \times \frac{10^6}{0.83 + \frac{2.83}{S_{SM-PPC}}}\ Bytes/s \tag{16}$$

For the compact flash where $S_{SM-PPC} = 1$, Equation (16) becomes

$$useful\ ARTP_{CF} = \frac{n}{n+1} \times 0.26\ MBytes/s \tag{17}$$

Similarly, we can compute the reconfiguration throughput of each phase from the corresponding aggregate reconfiguration time provided in Equations (5),(8) and (11).

5.3 Discussion

Table VII shows clearly that for each phase the throughput vs. the theoretical bandwidth is degraded considerably. This is due to the distinct components involved in the reconfiguration process. For example, in the PPC-ICAP phase, the way the PPC sends the configuration data to the ICAP cache has a negative impact on the reconfiguration time. The processor transfers the data from the PPC local

memory to the ICAP cache in a word-by-word fashion, which is iterated until the ICAP cache is filled up. We found that to transfer one word 633 PPC cycles are required when the PPC runs at 300 MHz, which translates to 2.11 μs. We found this time to be fairly constant. The reason this large amount of cycles is needed is that the transfer from the PPC local memory to the ICAP is bus-based plus the function calls residing in the processor local memory are dictated by an Application Program Interface (API). Moreover the buses operate at 1/3 of the PPC frequency, i.e. 100 MHz, which accounts for the large number of clock cycles in the processor. Also, we measured the time spent for the processor call in the ICAP-CM phase. For our case where the ICAP cache was 2048 Bytes, 7541 cycles of the PPC running at 300 MHz were needed, which translates to 25.26 μs (see Table VI). We repeated this measurement and we found it to be constant as well. The above delays add up to the total reconfiguration overhead every time a processor function is called, as the reconfiguration process is not pipelined, but it is controlled with sequential functions. Moreover, the throughput is bounded by the slowest CF-PPC phase, during which several components shown in Figure 3 interact, such as the compact flash, the external System ACE controller, the OPB sysace module, the PPC memory and its memory controller, the bus bridge, and the library maintained in the PPC memory to supervise the transfers.

It is obtained from Equation (15) that the parameters pa and ic_{size} are irrelevant. In particular, the user needs to be aware only of the frame size, fs, and the number of frames, n, to calculate the expected reconfiguration time. The former is available in the data-sheets. The quantity $fs \times (n+1)$ can be replaced with the number of reconfiguration bytes, rb, of the partial bitstream

$$rb = fs \times (n+1) = (fs \times n) + fs = partial\ bitstream\ size + fs \qquad (18)$$

where the *partial bitstream size* is extracted after running the place & route tool, or can be estimated from the synthesis results by using the percentage of occupied area and the size of configuration memory of the corresponding FPGA. Finally, the S_{SM-PPC} factor is calculated with a proportional analysis described in the next Section.

6. VERIFICATION AND USAGE

First we verify the cost model on the base compact flash system, and then we use it to investigate the performance of systems where the bitstreams are loaded from a DDR memory.

6.1 Verification

We used a platform for cryptography applications that we developed on a Virtex-II Pro FPGA with the Xilinx PR design flow. Two cryptography modules, the Advanced Encryption Standard (AES) and the Device Encryption Standard (DES/3DES), are switched on-the-fly without disturbing the remaining logic. The setup is the same with the one presented in Figure 3 except that the PPC operates at 100 MHz instead of 300 MHz. The PPC is the reconfiguration controller and the bitstreams are loaded directly from the compact flash. Table VIII has the details for the two designs as well as for the blanking bitstream which is used for power-saving when none of the cryptography modules is needed. The first two columns show the type

Table VIII. Comparison between the calculated and measured reconfiguration times for a partially reconfigurable cryptography system.

Partial bitstream	BS (Bytes)	Calculated RT (ms)	RT (ms)	Diff (ms)	%Error
AES	749,737	2,748.19	3,732.16	983.97	26.36%
DES/3DES	744,037	2,727.30	3,649.75	922.45	25.27%
Blanking	673,895	2,470.19	3,359.19	889.00	26.46%

of the partially reconfigurable modules and the measured bitstream size. The third and the fourth columns have the reconfiguration time as calculated with Equation (15) and as measured with the software time-stamps, respectively. The fifth column shows the absolute difference between the calculated and the measured times. The sixth column has the percentage error of the cost model. This is the metric we used to express the magnitude of the deviation between the theoretical value (also called approximate) and the measured value (also called actual). The following formula was used

$$\%error = \frac{|approximate\ value - actual\ value|}{actual\ value} \tag{19}$$

It is observed that there is a discrepancy between the cost model results and the measured times. However, the percentage error is fairly constant for all bitstreams. A factor that contributes to this error is the lower frequency of the processor in the cryptography system (100 MHz vs. 300 MHz in the reference system), which accounts for the larger reconfiguration time. As an extension the cost model could include the processor clock as a factor. Nevertheless, in its present form it estimates the expected reconfiguration time with a good accuracy, far better than the one and two orders of magnitude reported in other works [Griese et al. 2004; Galindo et al. 2008].

6.2 Reaping the Benefits of the Cost Model

We employ the cost model to evaluate PR systems with different setups. In particular, we study systems in which after boot-up the partial bitstreams are placed in a DDR; this allows for faster reconfiguration. A DDR controller, either as a discrete component or implemented as an IP core within the FPGA, is required. In both cases the DDR controller interfaces with the internal system through the on-chip buses, i.e. OPB or PLB. The reconfiguration is controlled by the PPC running at 300 MHz. The OPB is 32-bit and the PLB is 64-bit, and with a typical clock of 100 MHz they sustain a maximum throughput of 400 MB/s and 800 MB/s respectively. We calculate the reconfiguration time and throughput using the Equations (15) and (16) for different setups. The speedup factor S_{SM-PPC} affects the phase of reconfiguration process during which the bitstream is loaded from the external storage means. We calculate it as a fraction of the available bandwidth (BW) - which is dictated by the slowest component amongst the on-chip bus and the selected DDR - over the bandwidth of the compact flash; this is expressed with Equation (20). In the reference system we used to develop the cost model, the theoretical bandwidth

Table IX. Reconfiguration time (RT) and throughput (ARTP) for setups with various DDR memories. The values are calculated with the cost model for a partial bitstream of 80 KBytes. They concern a system, where the DDR controller is implemented as an IP core attached on the OPB (400 MB/s) or the PLB (800 MB/s) bus.

Storage	Clock(MHz)/ Width(bits)	Bandwidth (MB/s)	S_{SM-PPC}		RT_{SM} (ms)		$ARTP_{SM}$ (KB/s)	
			OPB	PLB	OPB	PLB	OPB	PLB
CF	32/16	64	1.00	1.00	299.83	299.83	266.82	266.82
DDR	100/8	200	3.13	3.13	142.18	142.18	562.67	562.67
	133/8	266	4.16	4.16	123.77	123.77	646.34	646.34
	166/8	332	5.19	5.19	112.68	112.68	709.95	709.95
	100/16	400	6.25	6.25	105.09	105.09	761.27	761.27
	133/16	532	6.25	8.31	105.09	95.88	761.27	834.35
	166/16	664	6.25	10.38	105.09	90.34	761.27	885.55
	100/32	800	6.25	12.50	105.09	86.54	761.27	924.42
	133/32	1064	6.25	12.50	105.09	86.54	761.27	924.42
	166/32	1328	6.25	12.50	105.09	86.54	761.27	924.42

of the compact flash was 64 MB/s.

$$S_{SM-PPC} = \begin{cases} \dfrac{DDR\ BW}{CF\ BW} & \text{if } DDR\ BW \leq on-chip\ Bus\ BW \\ \dfrac{on-chip\ Bus\ BW}{CF\ BW} & \text{if } DDR\ BW > on-chip\ Bus\ BW \end{cases} \quad (20)$$

Table IX has the expected reconfiguration times for a partial bitstream of 80 KBytes (the size is indicative; a different size could be chosen) and the corresponding throughput for various DDR memories when the DDR controller is attached on the OPB or the PLB bus. These results are drawn in Figure 4. It is observed that for the fastest DDR memories, the DDR controller on the PLB offers higher performance as compared to the OPB. Also, in both cases there is a point beyond which the transfer from the external storage is no longer the bottleneck of the system. Therefore, even though the available bandwidth of the external storage increases considerably, the benefit to the reconfiguration time is negligible.

To explore the effectiveness of our approach we use it to evaluate previously published systems and we compare our results with the published measured times. In particular, based on the information included in published works in which the embedded processor acts as the reconfiguration controller, we calculate the expected reconfiguration time with our cost model. Then, we compare the result with the measured times of Table I. These data and their comparison are consolidated in Table X. In [Liu et al. 2009] the PPC running at 300 MHz is the reconfiguration controller. A DDR controller is implemented as an IP core attached on the PLB bus, and has 32-bit interface and 100 MHz clock, which results in a theoretical bandwidth of 800 MB/s. According to Equation (20) this offers a speedup of 12.5 over the reference system. Equation (15) produces a reconfiguration time of 82.3 ms, denoted in the first row of Table X, which deviates from the published measured time [Liu et al. 2009] by 39.3%. In that work it was explicitly reported that the ICache and DCache of the PPC were disabled, which holds for our reference system

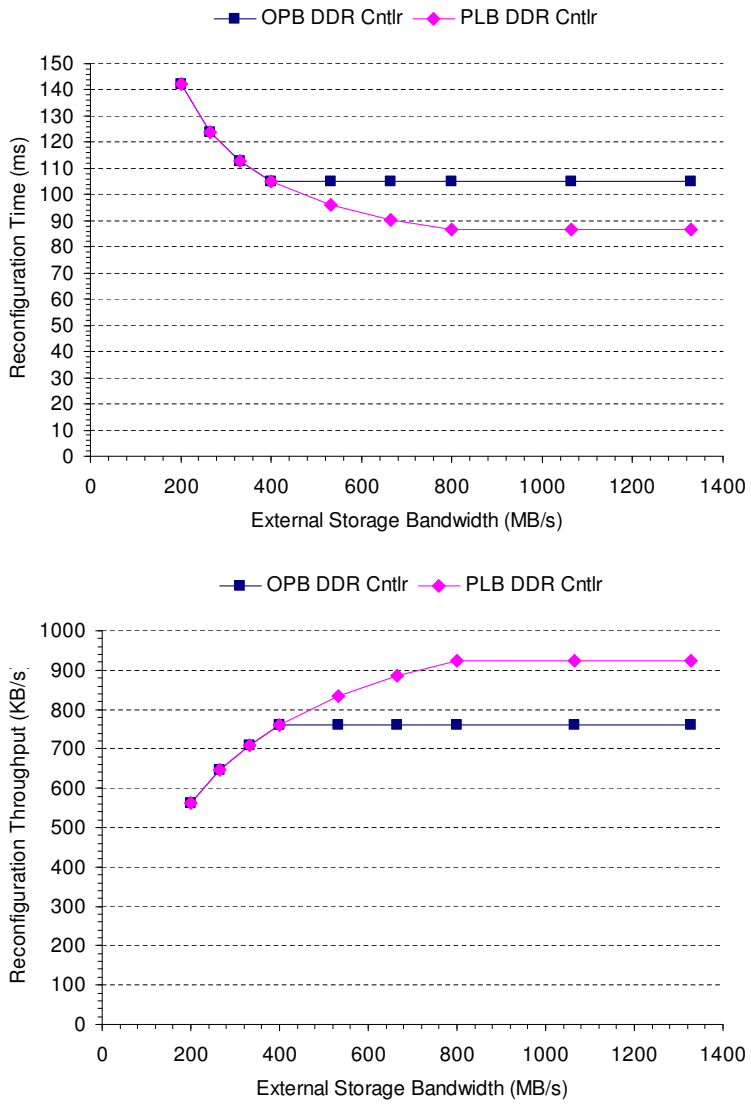

Fig. 4. Reconfiguration time for a partial bitstream of 80 KBytes and the corresponding throughput for different DDR memory bandwidths. The data are taken from Table IX.

too. In the second system of Table X, the ICache and DCache of the processor were enabled. According to [Liu et al. 2009] the activation of the processor caches offers an overall enhancement of 16.6 times in the reconfiguration time. Using the cost model and by dividing the result with 16.6 we obtained the time reported in the second row of the Table, equal to 5 ms, which deviates from the published measured time by 35.9%. In the same way we extracted the values for all systems of Table X.

Table X. Comparison between the calculated and published reconfiguration times for different setups of partially reconfigurable systems using the processor to control reconfiguration.

Reference	Storage	BS(KB)	Calculated RT(ms)	Published RT(ms)	%Error
[Liu et al. 2009]	DDR2@800MB/s	79.9	82.3	135.6	39.3%
[Liu et al. 2009]	DDR2@800MB/s	75.9	5.0	7.8	35.9%
[Claus et al. 2007]	DDR@400MB/s	90.3	7.2	19.39	63.1%
[Claus et al. 2008]	DDR@400MB/s	70.5	5.6	15.13	62.9%
[Papadimitriou et al. 2010]	CF@64MB/s	14.6	57.8	101.1	42.8%

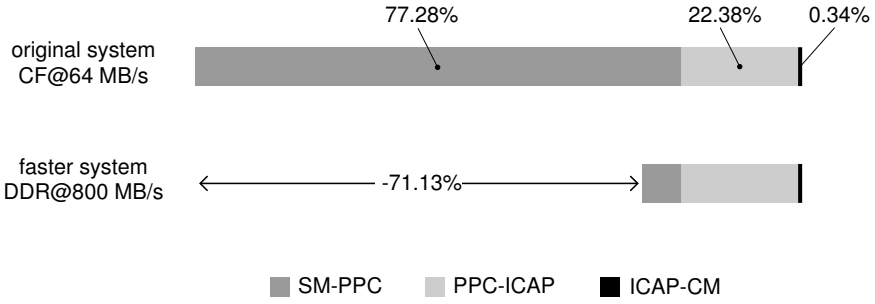

Fig. 5. The reconfiguration operation has three independent phases. In the original system, the SM-PPC phase (dark-gray part) takes 77.28% of the total reconfiguration operation (see Table VII). Making this part 12.5 times faster and leaving intact the rest of the operation (light-gray and black parts) in the faster system reduces the total reconfiguration time by 71.13%.

Our analysis allows for the examination of the enhancement margins when using the DDR memory, which is illustrated in Figure 5. The speedup of $S_{SM-PPC} = 12.5$ concerns the SM-PPC phase only, which takes the 77.28% of the reconfiguration time in the original system (see Table VII). This offers a total speedup of 3.46 times (the speedup is calculated using Equation (13) for $P = 77.28\%$ and $S = 12.5$). It is obvious that the total improvement from the increase of the external storage bandwidth becomes saturated, and the transfer from the PPC to the ICAP bounds now the reconfiguration performance.

6.3 Strengths, Weaknesses and Potential Extensions

Present work is novel in that except for the verification of the cost model, it examines its effectiveness on systems presented in previous works. The cost model provides results with a percentage error ranging from 36% to 63% when evaluated upon published measurements. This error is small considering that other works relying on the throughput of the configuration port show a discrepancy of two orders of magnitude between the theoretical and the measured values [Griese et al. 2004; Galindo et al. 2008]. Other theoretical models based on the characteristics of the configuration port, e.g. the ICAP's "BUSY" signal, cannot be applied to other FPGA families except for the Virtex-II/Pro, and they focus on custom reconfigu-

ration controllers targeting ICAP full utilization [Claus et al. 2008]. Our approach does not rely on the throughput of the configuration port, nor on characteristics that vary with the FPGA technology, e.g. behavior of "BUSY" signal differs in Virtex-II, Virtex-4 and Virtex-5 FPGAs.

The proposed cost model can be augmented with more factors in the same way the simple model for the compact flash on the reference system was extended to support other types of external memories. Specifically, the second part of Section 5 was devoted to extend the model with the optimization factor S_{SM-PPC}. Following the same concept, Equation (15) can be enriched with optimization factors for the PPC-ICAP and the ICAP-CM phases (namely $S_{PPC-ICAP}$ and $S_{ICAP-CM}$ respectively). To do this further experimentation with a different setup is needed. In particular, Figure 4 shows that after a specific point the larger bandwidth of the external memory does not necessarily result in a higher throughput. Instead, the PPC-ICAP phase becomes the bottleneck as illustrated in Figure 5. Therefore, other alternatives can be explored such as the usage of the On Chip Memory (OCM) instead of the PLB memory, which is likely to provide lower latency communication. Furthermore, the cost model can be extended with a factor that relates to the impact of changing the processor clock. Experimentation will quantify this as well to reduce even more the uncertainty for the expected reconfiguration time.

The presented approach considers systems in which the reconfiguration is controlled by the processor. Therefore, it serves users who desire to pre-evaluate their system without putting the effort to implement a custom reconfiguration controller. The values we used to develop the cost model were obtained with real measurements previous published in [Papadimitriou et al. 2007; 2010]. Even though they are of small scale they were proven adequate to foresee well the reconfiguration performance. After we gained insights from the reference system, we developed the model without delving into details such as the latency of the distinct components involved in the reconfiguration process. The only factors that someone should consider of is whether the processor is the reconfiguration controller and its caches are enabled, and the type of external storage the bitstreams are loaded from. In particular, for a given system setup, simple characteristics gathered from data-sheets such as the available bandwidth, the frame size, and the bitstream size are enough to predict the reconfiguration time.

7. DISCUSSION SUMMARY

This paper delivers an up-to-date survey and an exploration of parameters affecting reconfiguration performance. A real system is analyzed to identify the features that increase reconfiguration speed. Using this system as reference a simple cost model for the early prediction of reconfiguration time is developed. The model is verified, then used to evaluate several "real-world" systems, and the results are compared with published measurements. Also, the study quantifies the improvement of reconfiguration performance given the speedup of data transfer offered by the DDR memory over the compact flash. Finally, the strengths and weaknesses of our approach are discussed.

We conclude that experimental evaluation is necessary for revealing the details of system-level operation. With data gathered from experiments we devised a method

and a cost model currently applied in a category of real PR systems, which can be enhanced with more parameters to broaden its range of applicability.

Nowadays FPGA platforms are delivered as end-products and the benefits of partial reconfiguration technology are still being explored in numerous application domains. With the proposed model an opportunity arises for people not involved with partial reconfiguration yet but are interested to study whether their applications can benefit from this technology. We aim to assist researchers from different domains such as telecommunications, cryptography and networking to estimate the reconfiguration overhead without entering the complex and rigid PR design flow, and also to make effective choices of the system setup. Hence, based on specific constraints they can decide if it is worthwhile to proceed in designing an application with PR technology by either assigning this task to experts having the know-how of PR design flow, or putting this effort on their own.

ACKNOWLEDGMENT

The authors wish to thank Dr. Christoforos Kachris for his valuable feedback that helped improving the initial manuscript.

REFERENCES

Claus, C., Muller, F. H., Zeppenfeld, J., and Stechele, W. 2007. A New Framework to Accelerate Virtex-II Pro Dynamic Partial Self-Reconfiguration. In *IPDPS*. 1–7.

Claus, C., Zhang, B., Stechele, W., Braun, L., Hübner, M., and Becker, J. 2008. A Multi-Platform Controller Allowing for Maximum Dynamic Partial Reconfiguration Throughput. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. 535–538.

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.

Fong, R. J., Harper, S. J., and Athanas, P. M. 2003. A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguation. In *Proceedings of the IEEE International Workshop on Rapid System Prototyping (RSP)*. 117–123.

French, M., Anderson, E., and Kang, D.-I. 2008. Autonomous System on a Chip Adaptation through Partial Runtime Reconfiguration. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 77–86.

Galindo, J., Peskin, E., Larson, B., and Roylance, G. 2008. Leveraging Firmware in Multichip Systems to Maximize FPGA Resources: An Application of Self-Partial Reconfiguration. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 139–144.

Gelado, I., Morancho, E., and Navarro, N. 2006. Experimental Support for Reconfigurable Application-Specific Accelerators. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA 2006) in conjuction with ISCA*. 50–57.

Gholamipour, A. H., Eslami, H., Eltawil, A., and Kurdahi, F. 2009. Size-Reconfiguration Delay Tradeoffs for a Class of DSP Block in Multi-mode Communication Systems. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

Griese, B., Vonnahme, E., Porrmann, M., and Ruckert, U. 2004. Hardware Support for Dynamic Reconfiguration in Reconfigurable SoC Architectures. In *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPL)*. 842–846.

Hauck, S. 1998. Configuration Prefetch for Single Context Reconfigurable Coprocessors. In

Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA). 65–74.

Hsiung, P.-A., Lin, C.-S., and Liao, C.-F. 2008. Perfecto: A Systemc-based Design-space Exploration Framework for Dynamically Reconfigurable Architectures. *ACM Transactions on Reconfigurable Technology and Systems 1,* 3, 1–30.

IBM Inc. 2000. 128-bit Processor Local Bus. Architecture Specifications Version 4.6.

IBM Inc. 2001. On-Chip Peripheral Bus. Architecture Specifications Version 2.1.

IBM Inc. 2006. Device Control Register Bus 3.5. Architecture Specifications.

Kachris, C. and Vassiliadis, S. 2006. Performance Evaluation of an Adaptive FPGA for Network Applications. In *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping (RSP)*.

Liu, M., Kuehn, W., Lu, Z., and Jantsch, A. 2009. Run-Time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration. In *Proceedings of the IEEE International Workshop on Field Programmable Logic and Applications (FPL)*.

Lund, K. 2004. PLB vs. OCM Comparison Using the Packet Processor Software. Application Note: Virtex-II Pro Family XAPP644 (v1.1), Xilinx.

Lysaght, P., Blodget, B., Mason, J., Young, J., and Bridgeford, B. 2006. Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration on Xilinx FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. 1–6.

Manet, P., Maufroid, D., Tosi, L., Gailliard, G., Mulertt, O., Ciano, M. D., Legat, J.-D., Aulagnier, D., Gamrat, C., Liberati, R., Barba, V. L., Cuvelier, P., Rousseau, B., and Gelineau, P. 2008. An Evaluation of Dynamic Partial Reconfiguration for Signal and Image Processing in Professional Electronics Applications. *EURASIP Journal on Embedded Systems 2008*, 11 pages.

McKay, N., Melham, T., and Susanto, K. 1998. Dynamic specialisation of XC6200 FPGAs by partial evaluation. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 308–309.

Möller, L., Soares, R., Carvalho, E., Grehs, I., Calazans, N., and Moraes, F. 2006. Infrastructure for Dynamic Reconfigurable Systems: Choices and Trade-offs. In *Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design, SBCCI*. 44–49.

Noguera, J. and Kennedy, I. O. 2007. Power Reduction in Network Equipment through Adaptive Partial Reconfiguration. In *Proceedings of the IEEE International Workshop on Field Programmable Logic and Applications (FPL)*.

Papadimitriou, K., Anyfantis, A., and Dollas, A. 2007. Methodology and Experimental Setup for the Determination of System-level Dynamic Reconfiguration Overhead. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 335–336.

Papadimitriou, K., Anyfantis, A., and Dollas, A. 2010. An Effective Framework to Evaluate Dynamic Partial Reconfiguration in FPGA Systems. *IEEE Transactions on Instrumentation and Measurement 59,* 6, 1642–1651.

Papadimitriou, K. and Dollas, A. 2006. Performance Evaluation of a Preloading Model in Dynamically Reconfigurable Processors. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 901–904.

Paulsson, K., Hubner, M., and Becker, J. 2008. Cost-and Power Optimized FPGA based System Integration: Methodologies and Integration of a Low-Power Capacity-based Measurement Application on Xilinx FPGAs. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 50–55.

PRCC. 2010. Partial Reconfiguration Cost Calculator. `http://users.isc.tuc.gr/~kpapadimitriou/prcc.html`.

Santambrogio, M. D., Rana, V., and Sciuto, D. 2008. Operating System Support for Online Partial Dynamic Reconfiguration Management. In *Proceedings of the IEEE International Workshop on Field Programmable Logic and Applications (FPL)*. 455–458.

Xilinx Inc. 2006. OPB HWICAP (v1.00.b) Product Specification DS280 July 26, 2006.

Xilinx Inc. 2007a. Virtex-II Pro and Virtex-II Pro X FPGA User Guide. Datasheet. UG012 (v4.2) November 5, 2007.

Xilinx Inc. 2007b. XPS HWICAP (v1.00.a) Product Specification DS586 November 05, 2007.