A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration

Christopher Claus[1], Florian H. Müller[1], Johannes Zeppenfeld[1] and Walter Stechele[1]

[1]Technische Universität München
Lehrstuhl für Integrierte Systeme (LIS)
Theresienstrasse 90, 80333 München, Germany
{Christopher.Claus, Florian.Mueller, Zeppenfe, Walter.Stechele}@tum.de

Abstract

The Xilinx Virtex family of FPGAs provides the ability to perform partial run-time reconfiguration, also known as dynamic partial reconfiguration (DPR). Taking this concept one step further, partial dynamic self-reconfiguration becomes possible through the Internal Configuration Access Port (ICAP). In this paper a framework for lowering reconfiguration times using the combitgen tool [2] to reduce the overhead found within bitstreams, along with a completely new, very simple and area efficient ICAP controller that is connected directly to the Processor Local Bus (PLB) and is equipped with Direct Memory Access (DMA) capabilities is presented. Using this PLB Master ICAP controller, it is possible to reach the maximum practical throughput that can be achieved with the ICAP interface of Virtex-II Pro devices. Compared to an alternative realization using the OPBHWICAP provided by Xilinx (a slave attachment on the On-Chip Peripheral Bus), it is possible to achieve improvements concerning reconfiguration times by a factor of 20.

Keywords: dynamic partial self-reconfiguration, ICAP, reconfiguration time

1. Introduction and Related work

Modern Systems on Chip (SoCs) are subject to rapid changes concerning their functionality and other requirements. In order for them to react more flexibly to environmental changes or new tasks, a simple method must be found to allow such systems to adapt to their surroundings. One possible concept for this builds on the assumption that only those parts of the system (system components) affected by the new tasks or environment must be updated. Following this idea a little further, one can imagine that certain system components could be replaced during run-time, while the remaining, unaffected parts of the system remain fully operational. A developer could use one chip for different tasks and switch between them during run-time. Thus the so called Dynamic Partial Reconfiguration (DPR) leads to a fast and cost efficient application development. The Xilinx Virtex FPGA family additionally provides the possibility of dynamic partial self-reconfiguration, the use of which is gaining importance for achieving faster reconfiguration times. Much research is going on in this field e.g. Blodget et al. [1], Donato et al. [4], Upegui et al. [10] and Williams et. al. [11]. The Autovision system [3] deals with video processing for future driver assistance systems, where hardware accelerators (coprocessors) should be exchanged. The following scenario should describe why it is necessary to reduce the reconfiguration time. Using a frame input rate of 25 frames per second results in a maximum allowable time of 40 ms to process one image. If the image processing can be done in 35 ms, 5 ms are left over for reconfiguring a coprocessor. Assuming a reconfiguration can be accomplished within this timeframe, no frames must be dropped. The authors in [6] mention fairly high reconfiguration times (6.2 seconds for a 35KB bitstream) if the bitstream is loaded over the RS232 connection. With standard techniques and tools available on the market it is possible to reconfigure 110 frames in 19.39ms on a Virtex-II Pro FPGA. Both of these values are clearly to high, so a new method for reconfiguration had to be found. In section 2 Dynamic Partial Self-Reconfiguration and the Internal Configuration Access Port (ICAP) are described. Additionally an insight into the Reconfiguration Flow being used and the combitgen Tool [2] which is used to create optimized partial bitstreams is given. Section 3 gives an overview of the system, including subsection 3.4 in which the PLB Master ICAP Con-

troller that is used to achieve shorter reconfiguration times is described. In addition, an example SoC is mentioned which is used to verify the correctness of this approach. Some experimental results, namely reconfiguration times are discussed in section 4. Finally the paper is concluded in section 5.

2. Dynamic Partial Self-reconfiguration

In Virtex-II and Virtex-II Pro devices a configuration frame represents the atomic unit which can be reconfigured. A configuration frame is a 1 bit wide collection of bits that reaches from top to bottom of the FPGAs configuration memory. This has changed in Virtex-4 devices, but as the paper focuses on reconfiguration times of Virtex-II Pro devices, the Virtex-4 will not be further considered. Nevertheless, the framework can easily be adapted to Virtex-4 devices with some slight modifications. Each of the FPGA's CLBs spans multiple configuration frames, 22 in the case of an XC2VP30 FPGA. To change the functionality of a system using dynamic partial reconfiguration, the set of frames included within the bitstream must be written to the configuration memory of the FPGA. In addition to the frames containing data within the bitstream, there may be some pad frames necessary to shift the last frame into the configuration memory. This process is described more fully in [16].

2.1. Internal Configuration Access Port (ICAP)

The configuration architecture of the Virtex-II series is explained in [12] and [17]. Details about the Internal Configuration Access Port (ICAP) interface can be found in [14], [15] and [16]. The ICAP Interface behaves just like the SelectMAP Interface of the FPGA in slave mode, which is well documented in [16]. The ICAP allows internal read and write access to the configurable FPGA logic. Thus it allows self-reconfiguration of Virtex-II devices. The ICAP interface consists of separate 8-bit data ports for reading and writing, write and chip enables, a busy signal, and a clock input. The ICAP is physically located in the lower right corner of the Virtex FPGAs, and users must make sure not to reconfigure the circuitry controlling the ICAP. Thus the ICAP does not allow full reconfiguration of the entire FPGA. In contrast to SelectMAP, ICAP does not support multiple modes. Beside the mode pins (M2, M1, M0) that can be found in the SelectMAP Interface other pins are missing too, such as DONE, INIT, and PROGRAM. The SelectMAP CS pin has been renamed CE on the ICAP but it provides exactly the same function. The Xilinx Embedded Development Kit (EDK) provides a peripheral called the OPBHWICAP which wraps the ICAP with additional logic to read and write frames to a block ram (BRAM) [15]. The OPBHWICAP is connected to the On-Chip Peripheral Bus (OPB) as a slave peripheral. The partial bitstream files to update the configuration memory can be provided from outside or inside the circuit. In this approach the partial bitstreams are stored in the DDR SDRAM. Additionally, the ICAP controller is connected to the Processor Local Bus (PLB) as a master, using an in-house PLB IP Interface (IPIF). An initial version of a PLB ICAP controller from Xilinx can be found in [14].

2.2. Reconfiguration Flow

In this work the Early Access Partial Reconfiguration (EAPR) flow is used. It was newly introduced by Xilinx and is installed via a special patch for ISE 8.2.01i. This flow is also used in the Planahead Tool of Xilinx. More detailed information can be found in the work of Lysaght et. al. [9] and [5]. The EAPR flow is based on the well known module-based reconfiguration flow which is explained in [13]. The major difference between the module-based reconfiguration flow and the EAPR flow is that the EAPR flow allows nets in the base design to cross through a partially reconfigurable region without the use of a bus macro, as can be seen in Figure 1. Bus macros are predefined units of logic and wiring that guarantee the correct routing between the reconfigurable module and the unchanged part of the system before and after the reconfiguration. Instead of using tristate buffer based bus macros as recommended in [13], LUT-based bus macros are utilized in this work which were introduced primarily by Hübner et al. [7]. The nets that cross the reconfigurable region from left to right and vice versa are mainly connections to the DDR SDRAM. Of course it would be possible to simplify the layout for partial reconfiguration by placing the reconfigurable module on the right side of the device, but in this example the intent was to show that the EAPR flow also works under more difficult conditions.

2.3. combitgen

The combitgen tool generates optimized partial bitstreams. It combines the advantages of existing reconfiguration flows, namely module-based, PartialMask and difference-based, while avoiding their disadvantages. Details and a comparison between these reconfiguration flows can be found in [2]. The bitstream size is proportional to the reconfiguration time, hence if superfluous data is removed it is possible to reconfigure the device more quickly. The module-based flow is based on the PartialMask flow, which creates partial bitstreams consisting of complete CLB columns. If only one frame is different between two toplevels inside a column, one can imagine that

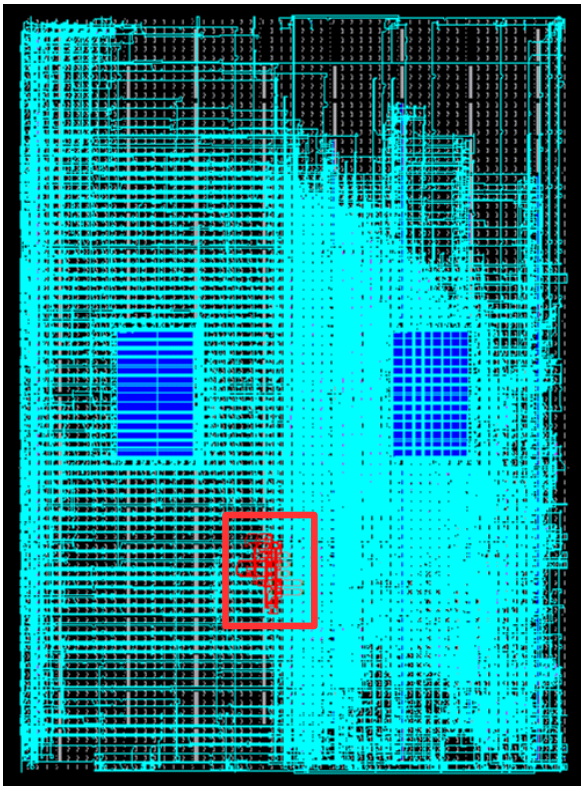

Figure 1. Layout of an example reconfigurable system. No bus macros are used to span the reconfigurable area. The reconfigurable module is marked with a rectangle.

this results in a huge overhead concerning the number of frames in the partial bitstream.

With the Xilinx bitgen tools difference-based method of generating partial bitstreams, which only writes those frames which are actually different, n toplevel bitstreams require $n(n-1)$ partial bitstreams. This originates from the fact that the different frames between e.g. toplevel A and B are not the same as between A and C. combitgen [2] works around this problem by comparing each frame across all toplevels, and marks it as having to be written for all toplevel bitstreams if it is different between any two of them. This way, the different frames of A and B or A and C are just a subset of all marked frames. These frames are then written from each toplevel bitstream into an associated partial bitstream, which can be used to configure from any other toplevel to this one. As a result, only n partial bitstreams are needed for n toplevels. Another advantage of combitgen [2] is its ability to exploit the so-called Multi Frame Write (MFWR) method to minimize bitstream sizes, thereby reducing reconfiguration times. The Xilinx bitgen tool uses MFWR only for the intended purpose of writing identical frames to multiple locations in configuration memory (for details about the configuration of Virtex-II FPGAs see [16]). When writing one or more unique frames, bitgen simply writes the frame to the FDRI register and then shifts it into the desired frame address. This shifting normally occurs automatically when sequential frame addresses are written. However, a dummy pad frame must be written in addition to the useful data in order to shift in the last frame of a continuous range of frames. Since each frame contains a fairly large amount of data (206 32-bit words in an XC2VP30 FPGA), this can result in a hefty overhead, especially when only a few frames need to be written. The overhead necessary to issue an MFWR command (e.g. when only a single frame must be written) is much lower in comparison, requiring only 13 32-bit command words per frame. When writing $\lfloor \frac{206}{13} \rfloor = 15$ or fewer consecutive frames it is therefore more efficient to write each frame individually using MFWR. Recapitulating these facts, combitgen [2] uses the MFWR feature for writing identical frames as well as for writing single frames in an efficient manner. This process is automated by combitgen, making it an excellent addition to the EAPR flow.

3 System Overview

A block diagram of an example system using the PLB Master ICAP controller is shown in Figure 2.

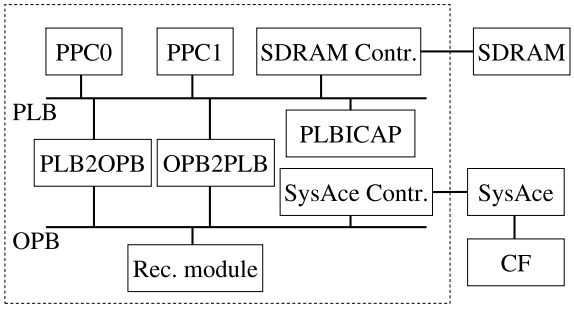

Figure 2. Block diagram representing the architecture of a reconfigurable system.

In this example system an OPB peripheral (Rec. module) will be exchanged, which represents an adder or subtractor depending on the configuration loaded. The layout of the system is depicted in Figure 1. The partial bitstreams are preloaded from a Compact Flash card to the DDR SDRAM during initialization.

3.1. Reconfiguration times: a theoretical consideration

After the design is placed and routed there are three methods of reducing the reconfiguration time:

- reduce bitstream size
- optimize the way bitstreams are written to the configuration memory (remove pad frames)
- optimize the bitstream transfer from memory to the ICAP

The first possibility is to reduce the size of the partial bitstreams. Unnecessary frames are removed by combitgen [2]. Secondly, the way in which partial bitstreams are written to the configuration memory can be optimized. Single frames are written with the Multiple Frame Write (MFWR) feature as explained in section 2.3 and [2]. Finally, reconfiguration times can be reduced further by optimizing the bitstream transfer from memory to the ICAP. This can be achieved using a PLB Master ICAP Controller, which will be presented in section 3.4.

3.2. Maximum theoretical throughput

Based on the assumption that the ICAP can process incoming data every clock cycle, the maximum theoretical throughput to the ICAP can be calculated by:

$$MTT = \frac{IDIW}{ClockPeriod} \quad (1)$$

where $IDIW$ is the ICAP Data Input Width (8 bits in Virtex-II and Virtex-II Pro devices and 32-Bit in Virtex-4 Devices). MTT is the maximum theoretical throughput that can be achieved were the ICAP is capable of processing incoming data every clock cycle. Considering a clock period of 10ns (100 Mhz) and the Virtex-IIs ICAP data input width of 8-bits, a maximum theoretical throughput of 100 KBytes/ms would be possible.

3.3. Maximum practical throughput

In reality it is not possible for the ICAP in Virtex-II and Virtex-II Pro devices to process new data every clock cycle, and indeed when the ICAP is clocked with frequencies above 50 MHz (100 MHz in Virtex-4 devices) it is necessary to respect the ICAPs handshaking (busy) signal. This signal indicates whether the ICAP has accepted the incoming data or not. In Figure 3 the `BUSY_ICAP` signal is shown recorded from a reconfiguration sequence using the PLB ICAP controller. It can be seen that ICAP is busy quite often which is indicated by the handshaking signal `BUSY_ICAP`. Thus the maximum theoretical throughput cannot be achieved. The maximum practical throughput of ICAP in Virtex-II Pro devices is in the range of 94% to 96% of the maximum theoretical throughput as can be seen in Table 2.

3.4. PLB ICAP Controller

The PLB Master ICAP controller provides the interface necessary to transfer bitstreams to and from the ICAP. In order to avoid unnecessary overhead, the ICAP controller is equipped with DMA capabilities, which allow the controller to access bitstream data directly from the main memory. Not only does this minimize the amount of data which must be transferred, it also reduces the load on the CPU. After receiving the memory address and length of a bitstream from the controlling processor, the Master ICAP controller begins bursting in the required bitstream data directly from main memory. Incoming data is stored within a FIFO, from where it can be fed one byte at a time into the ICAP. ICAP handshaking is also respected to allow for operation at the native bus speed (usually 100MHz in a system). As most of the data found within the bitstream header (e.g. design name, target device, date, etc.) is superfluous, combitgen [2] can be used to automatically generate a stripped-down version of the bitstream, allowing the data stored in memory to be sent directly to the ICAP while simultaneously reducing the amount of data that must be transferred. The modified bitstreams therefore begin with the sync word (0xAA995566). In order to further reduce the complexity of the ICAP controller (and thereby its size), the bitstreams stored in memory are padded to multiples of 128 bytes, allowing fixed burst lengths regardless of the actual size of the bitstream. In order to be accessible to the PLB Master ICAP controller, all bitstreams must be stored in main memory before they can be used to reconfigure the FPGA. Since a Compact Flash card was used for non-volatile bitstream storage, any required bitstreams are preloaded into main memory when the system boots. This is necessary because loading the bitstream data directly from the Compact Flash card would slow down the whole reconfiguration process. Once the bitstreams have been placed in memory, a start signal can be sent to the ICAP controller to initiate a reconfiguration. Currently this signal is provided by pressing an external push-button switch. In future designs this signal will be sent along with the bitstreams memory address and size via the Device Control Register (DCR) bus directly to the ICAP controller. The controller then requests the bitstream data from memory and forwards it to the ICAP as described above. Once implemented, reading frames from the ICAP will proceed in a similar manner, except that frames will be accessed one at a time and stored internally within

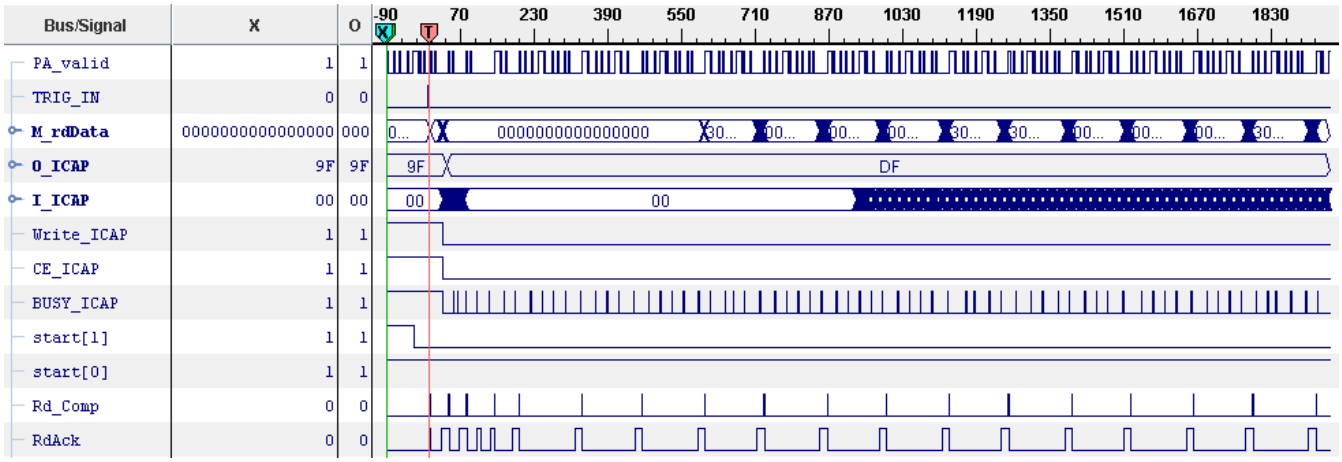

Figure 3. Waveforms of the PLB ICAP controller using a 100 Mhz clock obtained from Chipscope

the ICAP controller rather than being written out to main memory. The CPU will then be able to read the frame data directly from the ICAP. This readback of configuration data is not implemented yet but forms a part of the future work.

4. Experimental Results

In this section the results obtained when comparing the ordinary OPBHWICAP and unmodified bitstreams from Xilinx with the PLB Master ICAP module and bitstreams created with combitgen [2] are presented. In all of the designs the partial bitstreams are stored in the DDR SDRAM.

4.1. Reconfiguration times

As explained in section 3 a simple OPB-Peripheral is reconfigured. In Table 1, results from measuring the reconfiguration times in software are presented. In this case the xtime.h library and the OPBHWICAP from Xilinx are used. The same timing results can be obtained when using an opb timer hardware module. Compared to the bitstreams that are created by the conventional flows, a decent improvement of bitstream size can be achieved when using combitgen [2]. The first column shows the creation method of the bitstream. The second column contains the partial bitstream size in bytes. The number of data frames in the partial bitstream is depicted in column three. This number does not include the pad frames that must be written using the conventional (non-MFWR) method. The time in ms to partially reconfigure the device is shown in column four and finally the calculated throughput in KBytes/ms is depicted in column five. As can be seen, combitgen was not able to achieve an appreciable improvement compared to the difference-based flow. In this case, the first three frames were written in a row with MFWR for single frames. The following 18 frames were also written in a row, using the conventional method instead.

BS creation method	Size in byte	# of data frames	reconfig. times (ms)	throughput (KB/ms)
EAPR	92445	110	19.39	4.77
Difference-based	19966	21	4.18	4.77
combitgen	19290	21	4.03	4.78

Table 1. Reconfiguration times using the original OPBHWICAP and opb_timer from Xilinx [15]

However, if more than two reconfigurable modules were to be used, the difference-based flow would not be applicable any more. The average throughput which can be achieved with the OPBHWICAP is nearly 5 KB/ms. Compared to this, see the results with the ICAP Controller in Table 2.

BS creation method	Size in byte	# of data frames	reconfig. times (ms)	throughput (KB/ms)
EAPR	359168	806	3.75	95.77
Difference-based	62208	69	0.653	95.26
combitgen	58240	69	0.614	94.85

Table 2. Reconfiguration times using the new PLB Master ICAP Controller

Here the time was also measured in software. Interrupts thrown by the PLB Master ICAP indicate the end of a write process. The difference in bitstream size in Table 1 and Table 2 results from varying reconfiguration area constraints. The reconfigurable module itself is the same. As already explained in section 3.3 it is not possible to achieve the maximum theoretical throughput of 100KB/ms due to the ICAP's busy signal. A calculated average of 95 KBytes/ms is the maximum practical throughput that can be achieved. However, compared to the OPBHWICAP the throughput can be increased by a factor of 20. Table 3 depicts some estimated results comparing the EAPR results when using the OPBHWICAP controller and the combitgen results when utilizing the PLB ICAP controller. In the first column the partial bitstream creation method is stated, followed by a column containing the size of the corresponding partial bitstream in bytes. These values are derived from Table 1. The third column shows which ICAP controller is used. The fourth column contains the measured throughput of the corresponding ICAP controller. These values are derived from the results in Table 1 and Table 2. The last column contains the estimated reconfiguration times.

BS creation method	Size in byte	ICAP version	measured tp (in KB/ms)	rec. time (est. in ms)
EAPR	92445	OPB-HWICAP	4.77	19.38
combitgen	19290	PLB-ICAP	94.85	0.20

Table 3. Comparison between EAPR results with OPBHWICAP and combitgen results with PLB ICAP

In this simple design the reconfiguration time can be reduced by a factor of 96.9, just by using combitgen and the PLB ICAP controller. Of course the main improvement is caused by the PLB ICAP controller but combitgen delivers an additional factor that further reduces the reconfiguration time. Using the PLB Master ICAP the bus is no longer the bottleneck of the system. Instead, the ICAP input width of the Virtex-II and Virtex-II Pro devices is the limiting factor. The Virtex-4 family comes with an ICAP input width of 32 bits, which would reduce the reconfiguration times by an additional factor of 4 assuming that the transfer of bitstreams from main memory to the input FIFO of the PLB ICAP Controller remains fast enough. Preliminary observations of the bus load indicate that this would be the case.

4.2. Synthesis results

In this section the synthesis results of the PLB Master ICAP Controller are presented. The development platform used is the XUP Virtex-II Pro Development System with an XC2VP30 FPGA from Digilent. In Table 4, the synthesis results of the PLB Master peripheral are shown.

Number of Slices:	98	out of	13696	0%
Number of Slice Flip Flops:	94	out of	27392	0%
Number of 4 input LUTs:	185	out of	27392	0%
Number of BRAMs:	2	out of	136	1%
Minimum period: 3.788ns (Maximum Frequency: 264.023MHz)				

Table 4. Device Utilization and timing summary of the PLB Master ICAP Controller on a Virtex-II Pro (2vp30ff896-7) excluding the PLB IPIF

Having demonstrated the performance gain possible with the PLB Master ICAP Controller, now the synthesis results based on the Xilinx XUP Virtex-II Pro Development System with an XC2VP30 FPGA are presented. Results for the original Xilinx OPBHWICAP are shown in Table 6, those for the PLB Master peripheral in Tables 4 and 5, with and without the PLB IPIF respectively. Although the additional slices required by the IPIF push the PLB Master ICAP's total area requirement above that of the OPB ICAP controller, this is considered a small price to pay for the greatly improved performance. In Table 5 the synthesis results of the modified PLB IPIF are depicted.

Number of Slices:	223	out of	13696	1%
Number of Slice Flip Flops:	347	out of	27392	1%
Number of 4 input LUTs:	326	out of	27392	1%
Minimum period: 4.135ns (Maximum Frequency: 241.815MHz)				

Table 5. Device Utilization and timing summary of the modified PLB IPIF on a Virtex-II Pro (2vp30ff896-7)

Compared to this see the synthesize results of the OPBWHICAP including the OPB IPIF in Table 6.

Number of Slices:	112	out of	13696	0%
Number of Slice Flip Flops:	154	out of	27392	0%
Number of 4 input LUTs:	177	out of	27392	0%
Number of BRAMs:	1	out of	136	0%
Minimum period: 3.985ns (Maximum Frequency: 250.916MHz)				

Table 6. Device Utilization and timing summary of the OPBHWICAP Controller on a Virtex-II Pro (2vp30ff896-7) including the OPB IPIF

Of course a Master attachment to the PLB occupies more slice resources but as shown in the tables above the imple-

mentation of the PLB Master ICAP Controller and the PLB IPIF is still very resource efficient.

5. Conclusion and further work

In this paper a new ICAP controller with a master attachment to the PLB, allowing for reconfiguration of Virtex-II devices independently of the CPU at speeds very close to the theoretical maximum was presented. In addition several methods were described to reduce the reconfiguration overhead in partial bitstreams, thereby further reducing reconfiguration times. Future work includes extending the PLB Master ICAP controller to allow for readback, as well as providing it with an interface to the Device Control Register bus. Adaptions to other architectures, such as the 32-bit data port width of the Virtex-4 ICAP, would require minor additional changes. It is planned to implement a generic version of the PLB ICAP Controller so that it can also be used in Virtex-4 devices. Two ICAPs and a 32-bit data port width in Virtex-4 or Virtex-5 devices should enable a calculated maximum practical throughput of up to 800 KBytes/ms. Bitstream compression could be used to load the configuration data more quickly into the input FIFO of the PLB Master ICAP Controller. The compression could also be used to lower the load on the PLB Bus. One possible approach to this would be using a real-time decompression module such as that described in [8]. In addition it is planned to reconfigure complex modules in more advanced systems e.g. reconfigure hardware coprocessors for image processing in the Autovision system [3].

6. Acknowledgements

This work is supported by the German Research Foundation DFG (Deutsche Forschungsgemeinschaft) in the focus program No. SPP1148. We want to thank the Institute for Information Processing Technology (Institut für Technik der Informationsverarbeitung - ITIV) in Karlsruhe, especially professor J. Becker and M. Hübner for their great support in the field of dynamic partial reconfiguration. Additionally we want to thank Brandon Blodget, Jeff Mason and Tobias Becker for their support on the EAPR Flow and ICAP. Finally we want to thank Xilinx for providing development boards for our research activities.

References

[1] B. Blodget, S. McMillan, and P. Lysaght. "A Lightweight Approach for Embedded Reconfiguration of FPGAs". *Date*, 01:10399, 2003.

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

[3] C. Claus, F. H. Müller, J. Zeppenfeld, and W. Stechele. "Using Partial-Run-Time Reconfigurable Hardware to accelerate Video Processing in Driver Assistance System". In *Proceedings of the Design, Automation and Test in Europe Conference (DATE07)*, Nice, France, April 2007.

[4] A. Donato, F. Ferrandi, M. Redaelli, M. D. Santambrogio, and D. Sciuto. "Caronte: A Complete Methodology for the Implementation of Partially Dynamically Self-Reconfiguring Systems on FPGA Platforms". *Field-Programmable Custom Computing Machines Proceedings (FCCM 2005)*, 00:321–322, 2005.

[5] N. Dorairaj, E. Shiflet, and M. Goosman. "PlanAhead Software as a Platform for Partial Reconfiguration". *Xcell Journal*, 55:68–71, 2005.

[6] R. J. Fong, S. J. Harper, and P. M. Athanas. "A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguation". *rsp*, 00:117 pp, 2003.

[7] M. Hübner, T. Becker, and J. Becker. "Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration". *Proceedings of the 17th symposium on Integrated circuits and system design (SBCCI 04)*, pages 28–32, 25-29 April 2004.

[8] M. Hübner, M. Ullmann, F. Weissel, and J. Becker. "Real-Time Configuration Code Decompression for Dynamic FPGA Self-Reconfiguration". *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 3*, 04:138b, June 21-24 2004.

[9] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgeford. "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration on XILINX FPGAS". In *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL06)*, Madrid, Spain, August 2006.

[10] A. Upegui and E. Sanchez. "On-chip and On-line Self-Reconfigurable Adaptable Platform: the Non-Uniform Cellular Automata Case". *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS06)*, page 4 pp, 25-29 April 2006.

[11] J. W. Williams and N. Bergmann. "Embedded Linux as a Platform for Dynamically Self-Reconfiguring Systems-on-Chip". *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04*, pages 163–169, June 21-24 2004.

[12] Xilinx, Inc. "XAPP151: Virtex Series Configuration Architecture User Guide". v1.7, 20th October 2004.

[13] Xilinx, Inc. "XAPP290: Two Flows for Partial Reconfiguration: Module Based or Difference Based". v4.0, 9th September 2004.

[14] Xilinx, Inc. "XAPP662: In-Circuit Partial Reconfiguration of RocketIO Attributes". v2.4, 26th May 2004.

[15] Xilinx, Inc. "OPB HWICAP (v1.00.b) Product Specification". pages 1–13, 4th March 2005.

[16] Xilinx, Inc. "UG012: Xilinx Virtex-II Pro and Virtex-II Pro X FPGA User Guide". v4.0, 23 March 2005.

[17] Xilinx, Inc. "XAPP138: Virtex FPGA Series Configuration and Readback". v2.8, 11th March 2005.