Performance-Area Improvement by Partial Reconfiguration for an Aerospace Remote Sensing Application

L.A. Cardona*, J. Agrawal*, Y. Guo*†, J. Oliver* and C. Ferrer*‡
*Departament de Microelectrònica i Sistemes Electrònics, (IEEC-UAB), 08193 Bellaterra (Barcelona), Spain
†Institut de Ciències de l'Espai (IEEC-CSIC), 08193 Bellaterra (Barcelona), Spain
‡Institut de Microelectrònica de Barcelona (CNM-CSIC), 08193 Bellaterra (Barcelona), Spain
Email: *{LuisAndres.Cardona, Jharna.Agrawal, Joan.Oliver, Carles.Ferrer}@uab.cat, †guo@ieec.uab.es

***Abstract*—Dynamic Partial Reconfiguration (DPR) allows modification of certain parts of an FPGA while the rest of the device continues to operate and remains unaffected by the partial reprogramming. DPR for FPGA-based designs is an increasingly important feature in many systems that must adapt quickly to changing run-time requirements. Particularly in aerospace applications, the reuse and on-line reprogramming is a key aspect as the access to the system to re-design the hardware it is not a trivial task. We propose an autonomous reconfigurable digital signal processing architecture tailored for the utilization of FPGA resources efficiently. This paper deals with an efficient way of DSP computing by means of DPR applied to a hardware module and discusses execution time, flexibility and area savings. Reconfiguration on a co-processor is introduced to perform mathematical operations separately but using the same resources. Some overhead in timing is justified by the reduction in area obtained. Software application and hardware module performances are analyzed in terms of area and time to compare the benefits of the approach.**

***Keywords*-Partial Reconfiguration; FPGA; DPR; Fast Simplex Link; MicroBlaze; Co-Processor; System-On-Chip;**

I. Introduction

A typical FPGA-based platform can be seen as a system that in addition to the field programmable logic cells integrates different types of resources such as embedded CPUs, memories of diverse types (Flash, SRAM), general purpose I/O ports, communication buses, standard and customized peripherals, etc. This type of systems are well suited to tailor complex applications since these present certain advantages compared to traditional processor-based systems. Flexibility, possibility of fault tolerance capabilities, parallel processing, and the capability to include and adapt open cores and open sources (i.e. embedded OS) are aspects that have lead to the increased usage of FPGA to design embedded systems even as final implementation platforms rather than only prototype targets [1].

Recent advances in FPGA devices have brought Dynamic Partial Reconfiguration (DPR) as a way to achieve more flexibility in the management of resources [2]. DPR allows the reconfiguration of certain parts of the device without affecting the remaining parts of the design. The design process of partial reconfigurable (PR) architectures can be managed at different granularity levels in the FPGA. It can be applied to collection of basic logic components such as CLBs in Xilinx devices or to more complex blocks, such as IP cores. The selection of the level where to apply DPR is determined by the application requirements. In our case, an IP core, connected through a dedicated interface to the processor, is the target component to implement partial reconfiguration. The target application for which our approach was tailored has been designed to implement data processing as part of a remote sensing instrument [3]. Since the instrument is planned to operate at aerospace level, a limited bandwidth link should be optimally used to avoid loss of data. Currently, a software-based application (C code) implements a compression algorithm applied to the data collected by the instrument. Starting from this SW-based approach, HW/SW partitioning is used to reduce the execution time. This task was achieved by the inclusion of a co-processor to implement the algorithm. But this method imply considerable overhead in area. Here our approach to have better performance-area trade-off is the deployment of DPR. The HW functions implemented by the co-processor are partitioned to result in reutilizing resources of the FPGA. Besides area utilization of the different implementations, the influence in performance is analyzed separately for hardware and another time for hardware having reconfiguration. The rest of the paper is organized as follows: Section 2 describes related work. Next section establishes the target application. Section 4 describes the steps followed in this work. Next section presents the basic tests and the corresponding results. Finally, Section 6 exposes the main conclusions and future work.

II. Related Work

Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than only hardware solutions [4]. This approach has been increasingly used in the design of complex systems by embedding IP cores into FPGA devices and connecting and modifying them with certain flexibility. Thus, designers are able to explore and analyze different characteristics of systems, such as area utilization,

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.69

performance, power consumption, etc. In addition, the use of DPR as a method to quickly configure a part of the device with a minimal amount of data transfer has contributed to explore different features of this type of implementations. As the amount of information needed to configure an entire FPGA can be rather large, it can be quite time consuming, in addition to power consumption. Therefore DPR appears as a more efficient way to deal with these challenges. Even though it is a promising aspect in the design of flexible systems, careful analysis of limitations regarding especially with reconfiguration time should be addressed [5]. Therefore, based on previous implementations of a test application using FPGA devices [6], [7], our efforts here are oriented to the inclusion of DPR to analyze the area required and the effects on execution time by using this approach.

III. Target Application

The DPR approach is followed to improve the execution time but using efficiently the available resources of an FPGA device that implements an application for data processing. This application has been planned to be part of a post-processing system integrated to the GOLD-RTR instrument [3]. This instrument was designed to compute and store in real time, waveforms generated by cross-correlating local copies of GPS signals with reflected GPS signals in a surface of interest such as the ocean. This data can be used to obtain geophysical parameters such as sea level and tides, sea surface mean-square slopes, ice roughness and thickness, soil moisture, etc. The GOLD-RTR schedules consecutive time slots of 1 ms operating to a sample frequency of 40 MHz, and using up to ten channels. For each channel, 64 lags with complex values (I and Q) are generated every 1 ms and control information about the satellites and timing parameters are added to generate a waveform. Therefore the waveform is composed by 32 B of header and 128 B of complex data, as shown in Fig. 1. $[I(j), Q(j)]$, $0 \leq j < 64$, represents the 64 complex signal pairs.

The instrument will operate to aerospace level and due to intensive computation for the waveforms and limited bandwidth link between instrument and earth station, a type of compression by using an on-board real-time data processing has been designed to cope with these limitations [7]. Then, once a waveform is available, the 64 complex signal pairs should be processed according to the designed algorithm to obtain real-time behavior and minimize the memory requirements in space level. To do this, the algorithm integrates one identical waveform by adding incoherently the different points of the reflected waveform for 1 s, as stated in (1).

$$W(j) = \sqrt{\left(\sum_{i=0}^{N_w-1} I_i(j)\right)^2 + \left(\sum_{i=0}^{N_w-1} Q_i(j)\right)^2} \Big/ N_w \quad (1)$$

Equation 1 then, represents the function applied to the data for each channel; *j* represents lag index (0 to 63); *i* is related with waveform index; and Nw represents the total number of waveforms for each lag. As the algorithm was designed to compare every 1s reflected and directly received signals, this imply that 1000 waveforms have to be processed every second ($Nw = 1000$), and *i* varies in the range from 0 to 999.

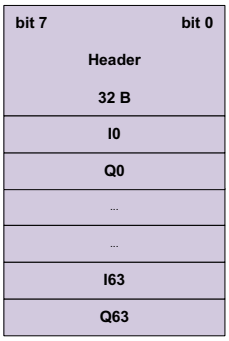

Figure 1. Waveform of 160B generated by GOLD-RTR every 1ms

The post-processing module is the responsible for implementing the algorithm corresponding to (1). This algorithm has been implemented in C language [7], using a Xilinx FPGA-based platform and the soft-core MicroBlaze processor. It has served as the starting point for our approach. We focused on the processing of data contained in the waveforms once the algorithm has used the control information from the header of each waveform to decide if the data are processed or discarded according to certain conditions. With this in mind, the main tasks to be executed on the waveform data are: To classify the 64 I-Q pairs every 1 ms. Accumulated addition of I and Q values from the arriving waveforms (1000 for 1 s of data). Once all waveforms were processed, the calculations of square, addition, square root and division have to be realized.

Considering the huge amount of data to be processed in constrained time, hardware acceleration has been considered as a way to improve the performance of the system. We designed a co-processor to execute the mathematical operations corresponding to (1). In this way, we changed the execution of the algorithm from software to hardware. The next section describes the process carried out to do this also as the details to include DPR in the co-processor to increase flexibility in management of resources.

IV. Implementation Approach

The steps followed during the process and the tools required in it are hereby presented. Our complete design consists of the GPS Open-Loop Differential Real-Time Receiver (GOLD-RTR) instrument, and the Xilinx XUP Virtex5 FPGA board with the MicroBlaze processor, and a co-processor as the main components. The XUP-V5 is a general purpose development board that includes a wide range of peripherals connected to the Virtex-5 XC5VLX110T FPGA

[8]. As stated in previous section, hardware acceleration was considered to improve the performance of the system. One possible option was the utilization of user-defined instructions. But it is a fairly complex solution. Therefore, as the MicroBlaze processor, using a more robust and versatile co-processor solution based on the Fast Simplex Link (FSL) offers greater performances, it was the selected option. In addition to the hardware acceleration, the precise focus of this approach was the introduction of DPR to improve the management and utmost utilization of available resources.

Three implementations were developed. The first one, based on the previous developments, executes the algorithm as a C code. The second one uses a co-processor to execute the algorithm. And the last one introduces DPR to the co-processor. Xilinx tools EDK - XPS [8], were used in the design of hardware implementations. And PlanAhead tool was employed to design the DPR-based architecture. 100 MHz was the operation frequency of the processor, buses and co-processor in all configurations. And to test and compare the three implementations, we used data corresponding to the waveforms for 1 s, which were stored in DDR2_SDRAM memory.

A. Architecture 1

In this implementation, the application for data processing, described as a C code was executed on MicroBlaze processor. Typical components for embedded system were used. External SDRAM, UART interface, timers, PLB bus, etc. This architecture was the starting point to design the HW/SW co-processing approaches.

B. Architecture 2

Further we added a Co-Processor for the same application using FSL bus as depicted in Fig. 2. Xilinx ISE tool was used to change the application from software/ application (C code) to hardware/ module (VHDL). The computations corresponding to (1) were described in VHDL to design the co-processor behavior. In this way data to be processed were read from memory and sent to the co/processor connected through FSL link. Once processed the results were stored in SDRAM memory.

C. Architecture 3

For the previous architecture, DPR was introduced in the co-processor to fragment the execution of the application

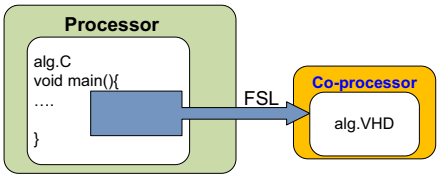

Figure 2. Changing execution from C to VHDL by using a Co-Processor

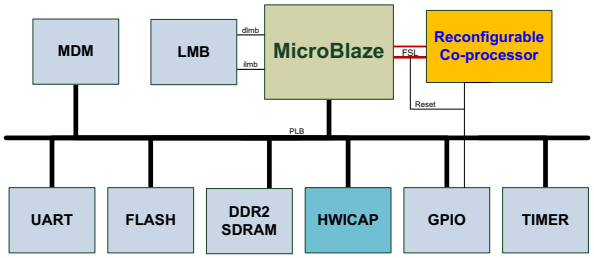

Figure 3. DPR applied to the co-processor

and therefore reduce the resources required to implement the algorithm. As we could identify the parts of the algorithm suitable to be time-multiplexed, we divided the VHDL code into different parts. Accumulated addition, square and addition as the first module (Logic1). And root square and division for the second module (Logic2). These two descriptions were used to generate the partial bitstreams to dynamically reconfigure the co-processor.

In addition, the HWICAP peripheral was included in the system as it is the responsible for reading the partial bitstreams from Flash memory and partially reconfigure the system. The operation frequency of this module was set to 100 MHz. By using PlanAhead, the size of the reconfigurable co-processor was chosen considering that its minimum size was determined by the greatest size of the partial descriptions. Fig. 3 presents the general view of this implementation. With this approach we have the possibility of changing the function implemented by the co-processor partially reconfiguring it with small or partial bitstreams located in FLASH memory. It implies that we have 2 implementations, each one depending on the function carried out by the reconfigurable partition. After all data were received by the co-processor and accumulated adding were performed, the square computation and addition were implemented by the logic 1. As the root square and division are only required after the previous steps, the logic 2 is implemented in the co-processor to perform these two operations.

For the three described architectures, measurements of execution time and quantity of resources required were obtained. Next section elaborates on this.

V. Results

The three architectural implementations were tested with data for 1 s according to previous description. The time to read them and to write back the results to the same memory was considered part of the execution time. Timers were controlled from software functions to get theses measurements.

Timing behavior of the three implementations is depicted in table I. The algorithm corresponding to (1) and described in C, was implemented in architecture 1. The resources used for this implementation are presented in table II.

Table I
PERFORMANCE COMPARISON

Architecture	Time (ms)
Arch 1 (SW-based)	54.598
Arch 2 (co-processor)	14.439
Arch 3 (DPR co-processor)	22.589

Table II
AREA ARCHITECTURE 1

FPGA resources	Used	(%)	Available
Number of occupied slices [slices]	2841	(16%)	17280
Number of slice registers [FF]	5594	(8%)	69120
Number of slice LUTs [LUT]	4492	(6%)	69120
Number of DSP48E	3	(4%)	64

Table III summarizes the area results for the co-processor based implementation and table IV presents the corresponding results for DPR-based implementation.

Now with these results we find that compared with architecture 1, (Execution time Arch 1/Execution time Arch 2 = 3.78), architecture 2 presents a gain of 3.78x and architecture 3 presents a gain of 2.42x in performance.

Regarding the area, also comparing with architecture 1, we observed an average overhead of 2.6x in the architecture 2. From these results it is foreseen that area resources required by the co-processor can be considerable if the application is extended to many channels. This challenge is an aspect in favor of the DPR approach as a good option to get better management of resources. This is seen when we compared architecture 3 with architecture 2. Depending on the part of the application to be implemented, an improvement in area of at least 10% is achieved. An expected degradation in execution time of 1,56x (Execution time Arch 3/Execution time Arch 2 = 1.56) is observed due to the time required for reconfigure and write/read data to/from memory.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, a novel approach was presented which simplifies the usage of area and time proficiency for a remote sensing application. By dynamically reconfiguring a co-processor, significant improvement in area utilization with acceptable degradation in time performance was obtained. This DPR approach is considered a starting point to incorporate Fault Tolerance features which are a mandatory requirement in this application designed to operate at aerospace level. DPR could be employed in future for fault mitigation mechanisms. Another aspect to take into account is the detailed analysis of the reconfiguration time and the exploration of possible alternatives to manage the HWICAP core to improve the reconfiguration time. For instance, by changing the connection from PLB bus to FSL.

Table III
AREA ARCHITECTURE 2

FPGA resources	Used	(%)	Available
Number of occupied slices [slices]	7355	(42%)	17280
Number of slice registers [FF]	16968	(24%)	69120
Number of slice LUTs [LUT]	22445	(32%)	69120
Number of DSP48E	11	(17%)	64

Table IV
AREA ARCHITECTURE 3

FPGA resources	Log 1	(%)	Log 2	(%)
Number occupied slices [slices]	5601	(31%)	3976	(23%)
Number slice registers [FF]	13134	(19%)	8984	(13%)
Number slice LUTs [LUT]	15899	(23%)	8293	(12%)
Number of DSP48E	11	(17%)	3	(4%)

ACKNOWLEDGMENT

Support funded by Spanish Ministry of Science and Innovation under DELPHIS project: TEC 2009-09712.

REFERENCES

[1] S. Anvar, O. Gachelin, P. Kestener, H. Le Provost, and I. Mandjavidze, "Fpga-based system-on-chip designs for real-time applications in particle physics," *Nuclear Science, IEEE Transactions on*, vol. 53, no. 3, pp. 682–687, june 2006.

[2] W. Lie and W. Feng-yan, "Dynamic partial reconfiguration in fpgas," in *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, vol. 2, nov. 2009, pp. 445–448.

[3] O. Nogues-Correig, E. Cardellach Gali, J. Sanz Campderros, and A. Rius, "A gps-reflections receiver that computes doppler/delay maps in real time," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 45, no. 1, pp. 156–174, jan. 2007.

[4] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Comput. Surv.*, vol. 34, pp. 171–210, June 2002. [Online]. Available: http://doi.acm.org/10.1145/508352.508353

[5] J. H. Pan, T. Mitra, and W.-F. Wong, "Configuration bitstream compression for dynamically reconfigurable fpgas," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, nov. 2004, pp. 766–773.

[6] A. Cardona, Y. Guo, and C. Ferrer, "Partial reconfiguration of a peripheral in an fpga-based soc to analyse performance-area behaviour," in *Proc. SPIE*, vol. 8067, 2011, p. 806705. [Online]. Available: http://dx.doi.org/doi/10.1117/12.887096

[7] G. Yi, D. Atienza, A. Rius, S. Ribo, and C. Ferrer, "Ht-pcp: Gnss-r multi-channel cross-correlation waveforms post-processing solution for gold-rtr instrument," in *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, june 2010, pp. 157–163.

[8] Xilinx. [Online]. Available: http://www.xilinx.com