Module based Partial Reconfiguration on Bitstream Relocation Filter

M. Angelin Ponrani
M.Tech student
Karunya University

G. Manoj
Assistant professor
Karunya University

R. Rajesvari
Assistant professor
Karunya University

ABSTRACT

The concept of partial bit stream relocation activity, the run time relocation can be obtained by using a system component. It is able to update the bitstream information and moving the reconfigurable module to a desired position. In Dynamic partial reconfiguration, it provides the possibility to change certain parts of the hardware while the other parts of the system remain in use. It provides a methodology to generate bitstreams for removal of old hardware modules, and placement and routing of new hardware modules. Relocation filter is implemented as a software component. The data errors can be detected and corrected by using the cyclic redundancy check and it is then fed into a system. Reconfiguration process is done over here by keeping cyclic redundancy check as a static part and bitstream relocation filter as a reconfigurable one. Finally the area require to store the Bitstream is analysed. Using module based partial reconfiguration; we can dramatically increase the functionality of a single FPGA in terms of the memory that is used to store the bitstreams. Important applications for this technology include reconfigurable communication and cryptographic systems.

Keywords

Partial relocation, Bit stream Relocation Filter, cyclic redundancy check.

1. INTRODUCTION

Partial Reconfiguration is the process of changing a portion of reconfigurable hardware circuitry while the other part is still running/ operating. Field programmable gate arrays are often used as a support to partial reconfiguration. Partial reconfiguration allows for critical parts of the design to continue operating while a controller either on the FPGA or off of it loads a partial design into a reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between designs. Partial reconfiguration is the cornerstone for power- efficient, cost effective software-defined radios (SDRs).
Through the JTRS Program, SDRs are becoming a reality for the defense industries as an effective and necessary tool for communication. Partial reconfiguration can also be used in many other applications. Another example is in mitigation and recovery from single-event upsets. In-orbit, space-based, and extra-terrestrial applications have a high probability of experiencing SEUs. By performing partial reconfiguration, in conjunction with read back, a system can detect and repair SEUs in the configuration memory without disrupting its operations or completely reconfiguring the FPGA. (Read back is the process of reading the internal configuration memory data to verify that current configuration data is correct.)

From the functionality of the design, partial reconfiguration can be divided into two groups:

- Dynamic partial reconfiguration, also known as an active partial reconfiguration - permits to change the part of the device while the rest of an FPGA is still running.
- Static partial reconfiguration - the device is not active during the reconfiguration process. While the partial data is sent into the FPGA, the rest of the device is stopped (in the shutdown mode) and brought up after the configuration is completed. There are two styles of partial reconfiguration of FPGA devices from Xilinx:
    - Module- based partial reconfiguration
    - Difference-based partial reconfiguration

Module-based partial reconfiguration permits to reconfigure distinct modular parts of the design. Module-based partial reconfiguration requires performing a set of specific guidelines during at the stage of design specification. Finally for each reconfigurable module of the design, separate bit-stream is created. Such a bit-stream is used to perform the partial reconfiguration of an FPGA.

Difference-based partial reconfiguration can be used when a small change is made to the design. It is especially useful in case of changing LUT equations or dedicated memory blocks content. The partial bit-stream contains only information about differences between the current design structure (that resides in the FPGA) and the new content of an FPGA.

To obtain partial reconfiguration it is necessary to provide a configuration file called bitstream. It is necessary to assign each module at design time, a placement on target device. Therefore a designer can define the shape that has to be used to implement the desired core and its placement on the device.

This paper is organized as follows. Section II gives an overview of some related works of various reconfiguration technique used to relocate the Bitstream while Section III presents the design methodology followed in order to attain the partial reconfiguration process. Section IV presents the implementation of the Bitstream relocation filter. The results are shown and analyzed in the Section V and then we conclude with the work done in section VI.

2. RELATED WORK

There are different types of relocation filters found in architecture. However, it is important to note that the memory resources are utilized can be much low as possible. Partial reconfiguration allows the modification of hardware functionalities at runtime. It provides the possibility for great improvements in the concept of reconfigurable computing.

Column-wise partial reconfiguration can be realized by means of a space allocation manager that determines the columns where single modules should be placed and a component which modifies the bitstream to place it in the correct position. It describes the development of such component in the form of the Bitstream Relocation Filter [1].

The possibility of dynamically change the functionalities hosted on the device only when needed and while the rest of the system keeps working. It describes and compares two different solutions, hardware and a software one, to perform Bitstream relocation: BiRF and Banco de Materials Light [2].The feature of partial reconfiguration provided by Field Programmable Gate Arrays (FPGAs) makes it possible to change hardware modules while others keep working. The combination of this feature and the high gate capacity enables the integration of dynamic systems that can be adapted to changing demands during runtime.In order to prevent any extra configuration overhead for the relocation process, the REPLICA filter is used [3].

Dynamically reconfigurable system-on-chips can change its hardware as well as software functionalities during run-time. A co-design and coverification methodology is proposed for dynamically reconfigurable SoC. First it is modeled using unified modeling language and then it is mapped to the systemC SoC using a functional architecture co design methodology [4].Adaptive FIR filter system design uses dynamic partial reconfiguration. It is responsible for providing the best solution for realization and autonomous adaptation of FIR filters [5].

3. DESIGN METHODOLOGY

The various design methodologies to generate bit streams are to remove the old hardware modules and placement and routing of new hardware modules within a FPGA. It is possible to replace a hardware core with another one by manipulating only the Bitstream. But it is a lengthy process. To avoid this inconvenience we are going for the partial reconfiguration process. Because of this technique it is possible to change a portion of the circuit while the other parts are still running. Hence for performing the partial reconfiguration process we assigned cyclic redundancy check as a static part and the Bitstream relocation filter as a reconfigurable one. The CRC generator is used to generate the partial bitstreams.

3.1 Systematic flow of reconfiguration

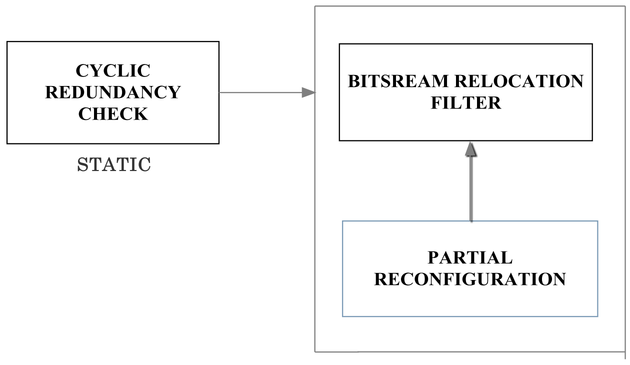

Fig 1: Block diagram of system reconfiguration

These bitstreams are fed into the bitstream relocation filter for reconfiguration is shown in Figure 1.

3.2 Cyclic Redundancy Check

The Cyclic Redundancy Check (CRC) is an efficient technique for detecting errors during digital data transmissions between a source and a destination. The destination device calculates the CRC of the received data. If the CRC calculated by the destination device does not match the one calculated by the source device, then the received data contains an error. This technique is used in a wide variety of applications from Ethernet transmission to daily file transfers. It provides quick and easy insurance of data integrity within digital communication systems.

The CRC is based on polynomial manipulations which treat each received message as a binary number. The received message is then divided by a fixed value, also known as the generator polynomial, using modulo-2 arithmetic. The characteristic of the CRC implementation is determined by the generator polynomial selection. The generator polynomials are selected to maximize the error detection capability without using too many resources.

CRCs are based on the theory of cyclic error-correcting codes. Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors, contiguous sequences of erroneous data symbols in messages. This is important because burst errors are common transmission errors in many communication channels, including magnetic and optical storage devices. The length of the remainder is always less than the length of the generator polynomial, which therefore determines how long the result can be. In practice, all commonly used CRCs employ the finite field. This is the field of two elements, usually called 0 and 1, comfortably matching computer architecture. The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC: it uses the generator polynomial $x+1$.

3.3 Designing CRC Polynomials

The selection of generator polynomial is the most important part of implementing the CRC algorithm. The polynomial must be chosen to maximize the error-detecting capabilities while minimizing overall collision probabilities. The most important attribute of the polynomial is its length (largest degree (exponent) +1 of any one term in the polynomial), because of its direct influence on the length of the computed check value.The most commonly used polynomial lengths are:

- 9 bits (CRC-8)
- 17 bits (CRC-16)
- 33 bits (CRC-32)
- 65 bits (CRC-64)

The design of the CRC polynomial depends on the maximum total length of the block to be protected (data + CRC bits), the desired error protection features, and the type of resources for implementing the CRC, as well as the desired performance.

3.4 Functional Description

A polynomial called generator polynomial must be chosen before the user computes the CRC of a transmitted message. The generator polynomial must have a degree greater than zero and a non-zero coefficient in the MSB and LSB positions. The selection of generator polynomial is the most important part of implementing the CRC algorithm. The polynomial must be chosen to maximize the error-detecting capabilities.

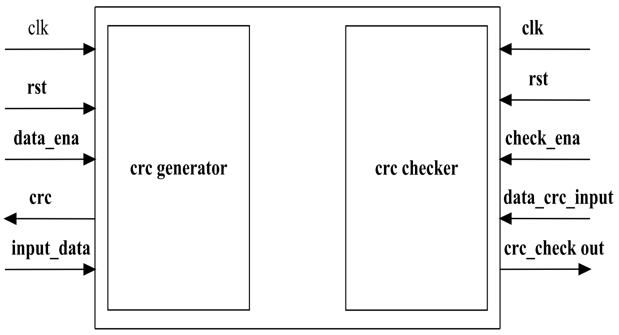

Fig 2: Functional Block Diagram of CRC

An attribute of the generator polynomial is that its length is equal to the degree +1. For example, in CRC-8, the degree is 8 and the length is 9. The degree of the generator polynomial determines the length of the CRC code.

3.5 Applications of CRC

A CRC-enabled device calculates a short, fixed-length binary sequence, known as the check value or improperly the CRC, for each block of data to be sent or stored and appends it to the data, forming a code word. When a codeword is received or read, the device either compares its check value with one freshly calculated from the data block, or equivalently, performs a CRC on the whole codeword and compares the resulting check value with an expected residue constant. If the check values do not match, then the block contains a data error. The device may take corrective action, such as rereading the block or requesting that it be sent again. Otherwise, the data is assumed to be error-free (though, with some small probability, it may contain undetected errors; this is the fundamental nature of error-checking).

3.6 CRC Generator and checker

A polynomial called generator polynomial must be chosen before the user computes the CRC of a transmitted message. The generator polynomial must have a degree greater than zero and a non-zero coefficient in the MSB and LSB positions. An attribute of the generator polynomial is that its length is equal to the degree +1. For example, in CRC-8, the degree is 8 and the length is 9. The degree of the generator polynomial determines the length of the CRC code. For example, if the degree of the generator polynomial is 16, then the length of the CRC code is 16. In this reference design, a *n* number of zero (‘0’) bits (*n* is the degree of the generator polynomial) is appended to the transmitted message before the *n*-bit CRC code is computed. Modulo-2 arithmetic (XOR operation) is implemented when computing *n*-bit CRC code, as shown in the example below. In this example, the generator polynomial is chosen as CRC-16-IBM (11000000000000101) and the transmitted message is chosen as 0xAA (10101010). Before the CRC code is computed, 16 bits of zeros are appended to the 0xAA and line the bits in a row. To check the CRC after the data transmission, the user passes the transmitted message appending the CRC code through the CRC checker module. The calculation of the CRC checker is the same as the CRC generator.

4. IMPLEMENTATION OF BITSTREAM RELOCATION FILTER

4.1 Partial Reconfiguration

Partial Reconfiguration is the process of changing a portion of reconfigurable hardware circuitry while the other part is still running/ operating. Field programmable gate arrays are often used as a support to partial reconfiguration. Normally, reconfiguring an FPGA requires it to be held in reset while an external controller reloads a design onto it. Partial reconfiguration allows for critical parts of the design to continue operating while a controller either on the FPGA or off of it loads a partial design into a reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between designs. Partial reconfiguration is not supported on all FPGAs. A special software flow with emphasis on modular design is required. Typically the design modules are built along well defined boundaries inside the FPGA that require the design to be specially mapped to the internal hardware. Vertex-II/ Vertex-II Pro, Virtex-4, Virtex-5 and Virtex-6 are the FPGA’s which support partial reconfiguration.

4.2 FPGA Configuration Methods

FPGAs are volatile memory-based programmable devices, the programming needs to be reloaded each time the device powers up. The process through which the device receives its program is called configuration. Xilinx FPGA devices can have the configuration loaded through one of the following modes:

Slave or master serial mode: Serial mode allows for daisy-chain configurations and is supported by all Xilinx FPGA families. In this mode, an external clock, a microprocessor and download cable is required. Data is loaded at one bit per clk.

Slave or master Select MAP mode: Select MAP mode is supported by the Vertex families. Select MAP mode allows parallel reading and writing through byte wide ports. An external clock source, microprocessor, download cable are required. The data is loaded one byte per CCLK. This mode is typically used as a configuration mode on devices when configuration speed is an important factor.

Boundary-Scan mode: It is an industry standard (IEEE 1149.1, and 1532) serial Programming mode. External logic from a cable, microprocessor, or other device is used to drive the JTAG specific pins, Test Data In (TDI), Test Mode Select (TMS), and Test Clock (TCK) and sense device response on Test Data Out (TDO). This mode is the most popular mode of configuration due to its standardization and ability to program FPGAs, PLDs, and PROMs through the same four JTAG pins. The data in this mode is loaded at one bit per TCK.

Usually, the device is reconfigured externally using a parallel cable, which is called Xilinx Parallel Cable. It connects the hosts PC to the JTAG connection of the target system. Its purpose is to download the bit stream to the FPGA and can also support debugging. Another mean of configuring a FPGA externally is to use a System ACE controller which is a controller to manage multi-bit stream, having access to the JTAG interface, the MPU and a compact Flash solution.

4.3 BiRF as a 2-D Relocation Filter

The 2-D version of the BiRF core is an evolution of its 1-D predecessor. It consists of a logical unit, the CRC, used to recomputed the CRC on the modified bitstream, and of a finite-state machine (FSM) that parses the bitstream and performs the necessary manipulations in order to exploit relocation. Main inputs of the filter are the configuration bitstream, in the form of a sequence of 32-bitwords (DATAIN), and the address used to specify the location where the bitstream has to be moved (TARGETPOSITION); as can be seen from the figure, the bitstream is taken from the bitstream memory through the bus, and the target position is received from the GPP through the bus too. The output of the filter is the manipulated bitstream, sent word by word (DATA OUT) to the bus.

The new FSM is an adaptation and an optimization, tailored for Vertex 4 and Vertex 5 devices, of the BiRF previous version. This version does not separate commands with a single parameter from commands with multiple parameters. Furthermore, some code optimizations have been done in order to reduce the area and increase the core performance in term of frequency. The target address, provided as a14-bit word, is an input of BiRF: the first bit is the top-bottom bit indicator, the next five bits contain the row address, and the last eight bits specify the column address. The calculation of the new FAR parameter is computed by the parser and is different in Vertex 4 and Vertex 5 devices, as the two compositions of FAR are different too. In order to perform the CRC computation, the standard parallel implementation has been used to provide the new checksum within the minimum amount of time to obtain better performance.

4.4 BiRF Parser

The need of a parser is due to the necessity to identify the FAR (Frame Address Register) and CRC (Cyclic Redundancy Check) commands in the bitstream, in order to perform the modification of the parameters that follow them. These two commands are the only ones which have to be altered in the relocation process; for this reason, when any other command or padding is recognized, the parser has to leave the bitstream unchanged, simply feeding the input through the output. Therefore, a finite state machine has been created, depicted in Fig. 3.

This FSM is an adaptation and an optimization of the BiRF one for Virtex-4 and Virtex-5 devices. In the adaptation process, the first change has been introduced: this version does not separate commands with a single parameter to commands with multiple parameters. Furthermore, some code optimizations have been done, in order to reduce the area consumption and to increase the core performance in term of frequency. The behavior of the FSM is briefly described in the following.

When BiRF Square is reset, the first phase that performs the recognition of the beginning of Bit stream commands, Starts; the FSM enters the Dummy state, which waits for Dummy word that may signal the beginning of the command words. After the recognition of such word, the Parser waits for Sync word: if it is received immediately after the Dummy word, this phase is terminated, and the commands recognition starts entering in the

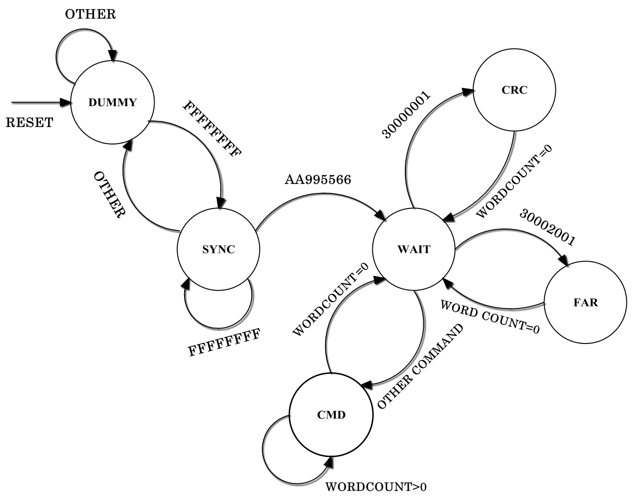

Fig 3: BiRF Parser FSM

Wait state. If the current word is the CRC or the FAR command, the parser feeds through output the next word which contains the new CRC or FAR. When a generic command is recognized, the FSM also has to calculate the number of parameters, in order to keep the bitstream the same for the right amount of clock cycles.

5. RESULTS AND DISCUSSION

In this section we perform the partial reconfiguration process using the plan ahead tool. At first we have to give the reset signal to the parser FSM. Then we have to mention the address to which state we want to make transition. According to the address we enter into the finite state machine it gets shifted from dummy state to sync state and so on and in each state, corresponding to the opcodes it perform the read and write operation. Whenever it finds the frame address register or else cyclic redundancy check register it switches from one state to another else if it meets with other commands the input is directly fed into the output.

5.1 Waveform *of BiRF*

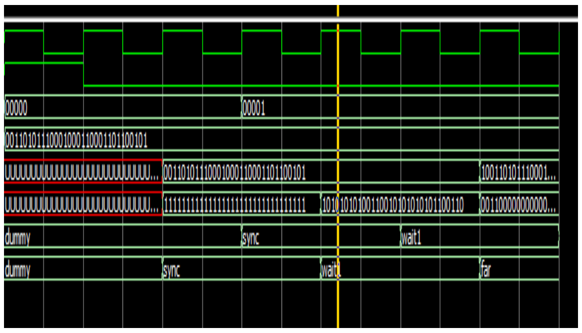

Fig 4: Waveform of BiRF

5.2 Device Utilization of BiRF

After synthesis process the number of LUT's, slices, registers and global clock buffer utilized in reconfiguration process is obtained.

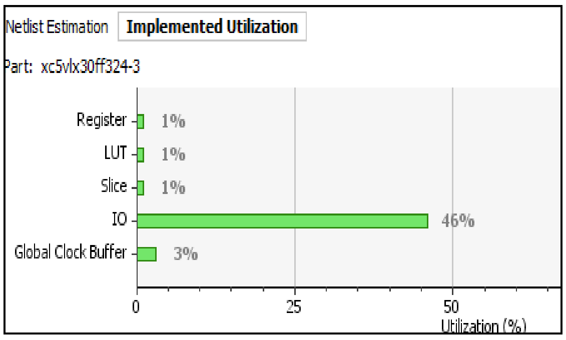

Fig 5:Device utilization of BiRF

5.3 Reconfigured Floor planning of BiRF

Before reconfiguration process, the number of slices and look up tables used for the bit stream reconfiguration filter utilized is more and hence the memory space required is too large. But after the partial reconfiguration process the utilization of look up tables and slices is much reduced. For the reconfiguration process, we have to select the input and output in the floor planning step. After that we have to define the partial reconfiguration region using the pblock in the plan ahead tool. Using the design rule check we have to check whether errors are present if that so the correction is made. When implementation is run, place and route are done from scratch with the net list, options, and constraints provided for that module. Then the import copies the results from a Promoted location for this module, preserving the exact results. Hence the place and route has successfully completed in the selected Configuration. After the partial reconfiguration verification the bit files are generated.

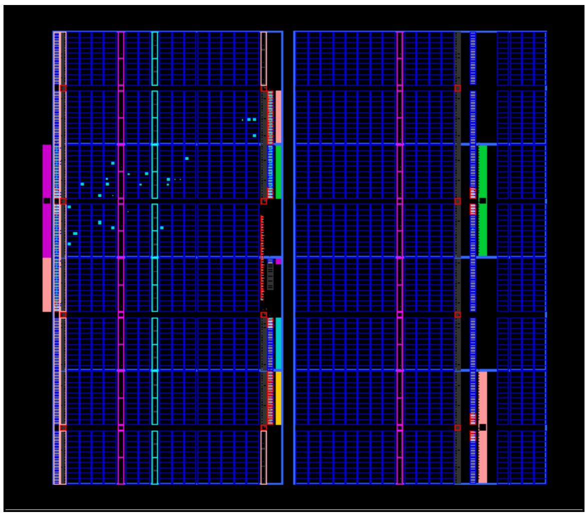

Fig 6: Reconfigured floor planning *of BiRF*

5.4 Utilization of LUT's and Slices

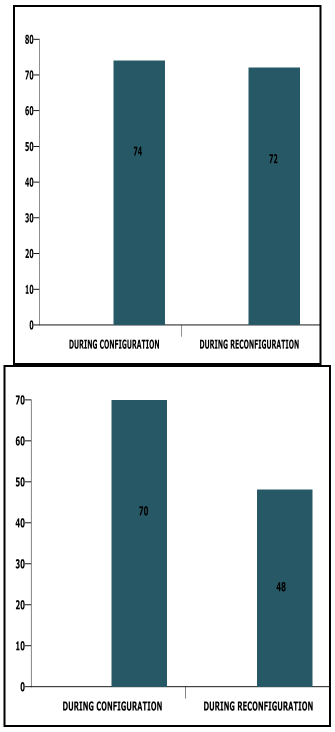

Fig 7: Utilization of LUT's and Slices

6. CONCLUSION

The cyclic redundancy check consists of CRC generator which generate a stream of bits and this bitstream is taken as a packet and then the different opcodes have been checked and corresponding to states, the 32-bit data is obtained and whenever the cyclic redundancy check (CRC) or Frame address register (FAR) is recognized the bitstream changes its location and hence its relocated.

Thus the partial reconfiguration process is done by assigning the cyclic redundancy check as a static part and Bitstream Relocation Filter as a partially reconfigurable part. After performing the reconfiguration technique the number of LUT's and slices is analyzed. The area of the system is much reduced after performing the partial reconfiguration. System reconfiguration can be done in the future.

7. ACKNOWLEDGMENTS

I praise and thank Almighty God whose blessings have bestowed in me the will power and confidence to carry out my project. I feel it a pleasure to my guide for his valuable support, advice and encouragement. I would also like to thank my dear friend and my parents who have prayed and helped me during the project work.

8. REFERENCES

[1] A Ahmad, B Krill, A Amira and H Rabah," Efficient architectures for 3D HWT using dynamic partial reconfiguration" Journal of Systems Architecture,2010.

[2] M. D. Santambrogio, “Hardware/software codesign methodologies for dynamically reconfigurable systems,,” M.S. thesis, Dipt.Elet.Inform.,Politecnico di Milano, Milano, Italy, 2008.

[3] S.Corbetta, F. Ferrandi, M. MOrandi, M. Novati, M.D. Santambrogio, and D. Sciuto, “Two novel approaches to online partial bitstream relocation in a dynamically reconfigurable system,” in *Proc. IEEE Comput. Soc.Ann.Symp. VLSI*, May 2007, pp. 457–458.

[4] S. W. S. Raaijmakers, “Run-time partial reconfiguration for removal, placement and routing on the virtex-ii pro,” in *Proc. Int. Conf. Field Program.Logic Appl.,Aug.2007*,pp. 679–683.

[5] C.S. Choi and H. Lee, “A self- reconfigurable adaptive FIR Filter system on partial reconfiguration platform,” IEICE Trans. Inf. Sysr., vol. E90-D,no. 12, pp. 1932-1938, 2007.

[6] P.A. Mudry, F. Vannel, G. Tempesti, and D. Mange, “Confetti: A reconfigurable hardware platform for prototyping cellular architectures,” in *Proc. IEEE Int.Parallel Distrib.Process.Symp. (IPDPS)*,Mar. 2007, pp.1–8.

[7] P. C. T. Becker and W. Luk, “Enhancing relocatability of partial bitstreams for run-time reconfiguration,” in *Proc.15th Annu. IEEE Symp. Field-Program.Custom Comput.Mach.*, Apr. 2007, pp. 35–44.

[8] F. Ferrandi, M. Morandi, M. Novati, M. D. Santambrogio and D.Sciuto, “Dynamic reconfiguration: Core relocation via partial bitstreams filtering with minimal overhead,” in *Proc. 8th Int. Symp. Syst.-on-Chip*, Nov. 06, pp. 1–4.

[9] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P.Lysaght, “Modular dynamic reconfiguration in virtex FPGAS,” *Proc. Inst. Electr.Eng. Comput.Digit.Tech.*,vol. 153, no. 3, pp. 157–164, May 2006.

[10] M. P. H. Kalte, G. Lee, and U. Rückert, “Replica: A bitstream manipulation filter for module relocation in partial reconfigurable systems,” in *Proc. 12th Reconfigurable Arch. Workshop*, 2005, p. 151b.

[11] H. Kalte, M. Porrmann, and U. Rückert, “System-on-programmablechip approach enabling online fine-grained 1D-placement,” in *Proc. 18th Int. Parallel Distrib. Process.Symp. (IPDPS)—Reconfigurable Archit.Workshop (RAW)*, 2004, p. 141.

[12] E. Horta and J. W. Lockwood, “Parbit: A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAS),” WUCS-01-13, 2001.