Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 314358, 15 pages
http://dx.doi.org/10.1155/2015/314358

Research Article

AC_ICAP: A Flexible High Speed ICAP Controller

Luis Andres Cardona[1] and Carles Ferrer[1,2]

[1]*Departament Microelectrònica i Sistemes Electrònics, Universitat Autònoma de Barcelona (IEEC-UAB), Bellaterra, 08193 Barcelona, Spain*
[2]*Institut de Microelectrònica de Barcelona (CNM-CSIC), Bellaterra, 08193 Barcelona, Spain*

Correspondence should be addressed to Luis Andres Cardona; luisandres.cardona@e-campus.uab.cat

Received 21 August 2015; Revised 12 November 2015; Accepted 19 November 2015

Academic Editor: Michael Hübner

The Internal Configuration Access Port (ICAP) is the core component of any dynamic partial reconfigurable system implemented in Xilinx SRAM-based Field Programmable Gate Arrays (FPGAs). We developed a new high speed ICAP controller, named AC_ICAP, completely implemented in hardware. In addition to similar solutions to accelerate the management of partial bitstreams and frames, AC_ICAP also supports run-time reconfiguration of LUTs without requiring precomputed partial bitstreams. This last characteristic was possible by performing reverse engineering on the bitstream. Besides, we adapted this hardware-based solution to provide IP cores accessible from the MicroBlaze processor. To this end, the controller was extended and three versions were implemented to evaluate its performance when connected to Peripheral Local Bus (PLB), Fast Simplex Link (FSL), and AXI interfaces of the processor. In consequence, the controller can exploit the flexibility that the processor offers but taking advantage of the hardware speed-up. It was implemented in both Virtex-5 and Kintex7 FPGAs. Results of reconfiguration time showed that run-time reconfiguration of single LUTs in Virtex-5 devices was performed in less than 5 μs which implies a speed-up of more than 380x compared to the Xilinx XPS_HWICAP controller.

1. Introduction

Field Programmable Gate Array (FPGA) devices persist as fundamental components in the design and evaluation of electronic systems. They are continuously reported as final implementation platforms rather than only prototype elements [1]. FPGAs have moved according to VLSI scaling technology pace making it possible to develop these devices in state-of-the-art fabrication processes. For instance, 7-series family of Xilinx SRAM-based FPGAs are built on 28 nm, high-k metal gate process technology [2], Xilinx Virtex UltraScale+ uses 16 nm FinFET+, and Altera Stratix 10 devices are produced using Intel-14 nm Tri-Gate (FinFET) process technology [3]. This is one of the reasons that favor the increasing presence of such devices as programmable alternatives to ASICs.

In addition, technical improvements in design and fabrication of FPGAs have produced more robust and flexible components embedding larger RAM memory blocks (BRAMs), DSP blocks, processors, and dedicated hardwired components. The inherent reconfigurable characteristics that FPGAs offer are among one of the most important advantages in the actual hardware implementation and redesign of systems.

We focus on Xilinx devices because in addition to supporting Dynamic Partial Reconfiguration (DPR), it is possible to modify the bitstream. It implies that reverse engineering on the bitstream structure can be performed, which is essential in our approach to perform DPR on LUTs as will be explained in Section 3.

Xilinx SRAM-based FPGAs support DPR by means of the Internal Configuration Access Port (ICAP). This hardwired element, depicted in Figure 1, allows the configuration memory to be accessed at run time. Therefore, it is possible to modify specific parts of the system while the rest continue operating without being affected by the specific run-time modification. Dynamic Partial Reconfiguration can be used at different granularity levels. Considering the architecture of the device, it can be employed to modify basic logic components, such as Look-Up-Tables (LUTs), or bigger blocks, such

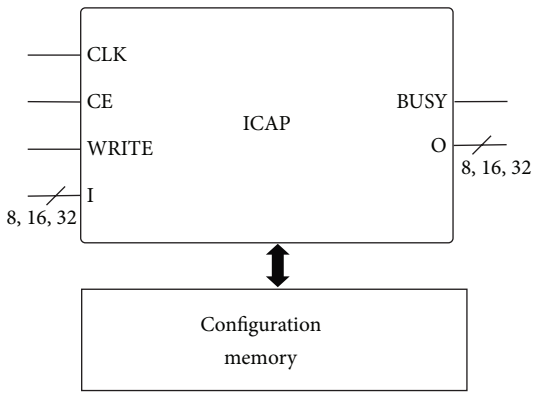

Figure 1: ICAP hardwired primitive.

as IP cores. Therefore, DPR is employed in a wide range of applications involving the design of self-adaptive systems and the evaluation of critical systems that need to be exhaustively tested before final production.

Xilinx tools such as PlanAhead or command line "*bitgen*−*r*" take the difference between two implementations to produce partial bitstreams that allow for modifying the specific parts that have been defined to change at run-time. The partial bitstreams are then copied in external or internal memory of the FPGA and from there are sent to the ICAP when a new hardware task is required by the system. In addition to this type of run-time reconfiguration that is especially suitable for coarse grain modules, there exist alternatives to dynamically modify basic elements, such as LUTs, using certain software functions executed in an on-chip processor.

With this in mind, the hardwired ICAP primitive and its associated controller become fundamental and inseparable modules in the design of dynamic run-time reconfigurable systems. The ICAP controller is responsible for performing all the commands to access and modify the configuration memory. Therefore, it is desirable that such controller meets at least two basic requirements: high reconfiguration throughput and flexibility.

Xilinx tools provide general controllers to drive the ICAP but they perform most of the processing as software routines in the processor. It implies flexibility but avoids to reach the maximum supported reconfiguration throughput. Diverse alternatives to these controllers have been reported to improve the reconfiguration speed. Most of them have been oriented to manage partial bitstreams generated at design time and also to manipulate frames that are the minimum addressable configuration memory.

Going deeper into the granularity of the device, any dynamic modification on the LUTs of an implemented design should also be available to increase the flexibility of the system. For instance, it can be used in cryptographic modules to modify the logic behavior of the module and increase the resistance against some type of external attacks. Therefore, an efficient mechanism that allows for modifying LUTs at run-time is also required as LUTs are basic components to implement any logic function in FPGAs. The ICAP controller should offer a way to perform DPR in LUTs at maximum supported speed, not only limited to pregenerated partial bitstreams, but presenting a simple interface doing the complexity of the architectural device transparent to the user.

In this paper, we present a novel run-time reconfiguration controller fully implemented in hardware and supporting partial reconfiguration of LUTs in Xilinx FPGAs. The main contributions of this work are

(i) design and implementation of the AC_ICAP controller that supports DPR of LUTs, validated in Virtex-5 and Kintex7 devices,

(ii) transparent on-chip translation of LUT coordinates and LUT configuration values into frames locations,

(iii) speed-up of the LUT-DPR and similar reconfiguration speed (compared to existing solutions) for partial bitstreams located in BRAM or flash memory,

(iv) FSM standalone operation and IP versions adapted to different embedded microprocessor interfaces (PLB, FSL, and AXI).

The rest of the paper is organized as follows. In Section 2, we review the most relevant works in the design of ICAP controllers. In Section 3, we present the main considerations regarding fine-grain partial reconfiguration. In Section 4, we detail the new AC_ICAP controller. In Section 5, the extension of the controller to be accessible from on-chip processors is presented. In Section 6, we describe the considerations to follow in porting the controller to a newer family of devices. In Section 7, we present the results of the reconfiguration time and area required by the controller. This is followed by Section 8 where the controller is used to modify LUTs in a cryptographic module as a way to implement countermeasures against external attacks. Finally, Section 9 concludes the paper and suggests future work.

2. Related Work

In this section, we outline some of the most relevant implementations of ICAP controllers used in FPGA Dynamic Partial Reconfiguration.

Partial reconfiguration has been widely used in diverse applications [5–7] that exploit the possibility to adapt hardware modules at run-time. A common requirement when using this technique is that the switching of hardware modules should be performed with minimal time overhead.

The most frequent approach to implement systems with DPR capabilities is by using the ICAP controllers available in Xilinx tools. XPS_HWICAP [4], depicted in Figure 2, AXI_HWICAP, and OPB_HWICAP are IP cores designed to be connected to the PLB [8], AXI, and low speed OPB buses, respectively. They are used as part of embedded processor systems (PicoBlaze or MicroBlaze) and the support for partial reconfiguration is given through a collection of software functions provided with the processor API. The functions allow for processing partial bitstreams located in memory, accessing configuration frames (*XHwIcap_DeviceReadFrame*, *XHwIcap_DeviceWriteFrame*), and modifying LUTs (*XHwIcap_SetClbBits*, *XHwIcap_GetClbBits*). An example of

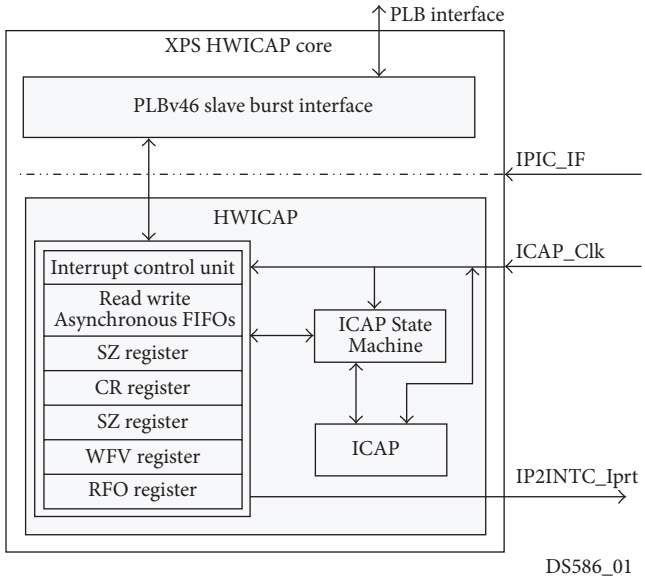

Figure 2: XPS_HWICAP [4].

the utilization of the functions to modify specific LUTs is detailed in [9] and authors in [10] use the functions to modify frames to emulate faults on the configuration memory.

The Xilinx functions perform most of the operations as software routines in the processor. Then, the commands to manage the ICAP and process the partial bitstream header executed in the processor, along with the bus latency, affect the speed of the partial reconfiguration process. Therefore, diverse alternative controllers have been developed to overcome these limitations. Authors in [11] explore different ICAP controllers analyzing the reconfiguration speed and propose three variations to speed up the processing of partial bitstreams but all of them require the presence of a processor. It is also the case of [12, 13]. In the latter case, the controller is integrated in the processor data path using the FSL link to minimize the bus latency. In contrast, [14, 15] present controllers for Virtex-5 devices able to load partial bitstreams from BRAM and flash memory totally implemented in hardware and independent of the processor. In a similar way, [7, 16] report implementations of processor-independent ICAP controllers for Virtex-4 FPGAs. Authors in [17] exploit DPR for the design of fault tolerant systems. Such approaches show improvements in reconfiguration speed that can reach the maximum supported throughput when using BRAMs. Further, some works, such as those presented in [7, 18], achieve throughput speed higher than the specified one in the technical documents by overclocking the ICAP.

All these works have been oriented to efficiently access the partial bitstreams and perform the tasks of hardware switching, but some further operations that a complete controller should support are not considered. A robust controller should be able to read back and write configuration frames and also give the possibility to modify LUTs besides to only control partial bitstreams. These last features are of paramount importance in the implementation of critical systems where the ICAP controller is a fundamental part of the design [21]. With this in mind, diverse approaches, such as those reported in [20, 22–24], use improved ICAP controllers as fundamental parts of fault tolerant systems in SRAM-based FPGAs. In such systems, the ICAP is used for the detection and correction of faults in the configuration memory. To do that, it is not enough with controlling precomputed partial bitstreams, they implement reading and writing of frames as the fault detection is performed at this level. For instance, once a frame is read, its CRC can be obtained to check if errors are present in its constituent bits. In case of erroneous values, the frame can be corrected and write back to the configuration memory with the right values. Therefore, these reported works include frame handling for both writing and reading of configuration frames.

To the best of our knowledge, the only work reported on performing run-time reconfiguration at LUT level implemented as part of the ICAP controller is presented in [25], but it is only valid for Xilinx Virtex-II devices where LUTs have four inputs and the architecture of the device is considerably different from newer Xilinx families. The frames cover the full height of the device and it is not detailed how the LUT configuration values are located on frames. In addition, this family is currently considered obsolete.

In this work, we develop a novel ICAP controller fully implemented in hardware with support for bitstreams management, reading and writing of frames, and LUTs modification. This approach offers improvement in LUT reconfiguration speed and it is performed without the need of precomputed partial bitstreams. In addition, it can be easily adapted to on-chip processors in diverse Xilinx FPGA families.

3. Dynamic Partial Reconfiguration for LUTs

In this section, we describe the general architecture of Xilinx FPGAs and the relevant concepts of partial reconfiguration taking as a reference the Virtex-5 XC5VLX110T device. But the general ideas are also valid for newer devices, especially when considering LUTs, as these remain the same; it is 6-input LUTs, for Virtex-5, Virtex-6, and 7-series families.

FPGAs are organized as an array of Configurable Logic Blocks (CLBs) connected to a switch matrix. Figure 3 shows the disposition of the XC5VLX110T FPGA where it can be observed that it is horizontally divided into two halves. In top (0) and bottom (1) halves, we find a fixed number of rows that depend on the size of the specific device. The Virtex-5 LX110T FPGA is divided into eight horizontal clock rows (HCLK): four in each half. Each HCLK includes a determined number of CLBs, BRAMs, DSPs, and I/Os. CLBs are distributed in 160 rows by 54 columns covering the whole device. Each CLB consists of two Slices and every Slice contains 4 LUTs, 4 flip flops, multiplexers, and carry logic. As a result, this FPGA has 17280 Slices, 69120 LUTs, and 69120 registers.

One CLB column is defined as a group of 20×1 CLBs that spans the HCLK height. It means that, in each CLB column inside the HCLK rows, there are 40 Slices and 160 LUTs.

The configuration memory is organized in frames. One frame is the smallest size of configuration memory able to be addressed. Therefore, any action on configuration memory

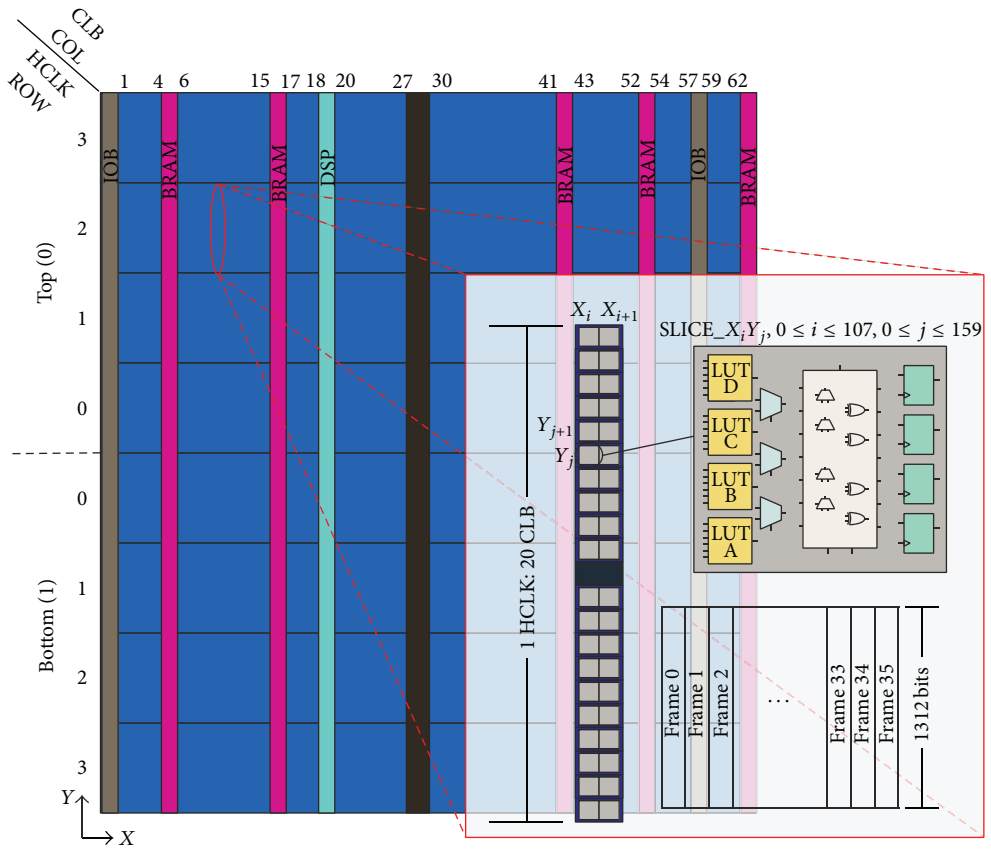

FIGURE 3: Relationship between X, Y coordinates and Frame Addresses for XC5VLX110T.

should be carried out taking frames as reference. One frame consists of 41 words of 32 bits each (1312 bits). Virtex-5 LX110T requires 23712 configuration frames to configure the whole chip. In consequence, the configuration file (bitstream) is composed of 972464 32-bit words (3.7 MB). It includes 272 words of control information in the header and the rest corresponds to configuration frames.

Every time we want to configure the whole device, the bitstream of 3.7 MB containing the description of the circuit to implement should be loaded into configuration memory.

Dynamic Partial Reconfiguration allows specific parts of the system to be modified; in consequence, the complete bitstream is not required but a smaller, partial bitstream, with the information of the specific region to modify, is used. Partial bitstreams are generated at design time using the difference-based approach. PlanAhead [26] or bitgen command line [27] is used to generate them. The command *bitgen* − *r config*1.*bit config*2.*ncd partial*2.*bit* takes as inputs the two different files for each configuration (*config*1 and *config*2) and the result is the partial bitstream *partial*2.*bit*, with the difference between them. The minimum size of partial bitstreams corresponds to one configuration frame increased with one extra dummy frame and control information.

To configure a column of CLBs, 36 frames are required. Inside the 36 frames, we have the information of every individual element present in the 20 CLBs. We focus on LUTs as these are the basic elements that implement all the combinational logic in FPGAs.

The LUTs or logic-function generators are six-input elements that require 64 bits to define the function to perform. The logic behavior of the LUT depends on the values (INIT value) configured in these 64 bits. To handle any individual LUT, it is necessary to define its location and its INIT value. The location uses three parameters: (X, Y, Bel). X and Y are the coordinates of the Slice and Bel is an index to select the individual LUT inside the Slice. The range of X and Y depends on the size of the FPGA (108 × 160 in the considered device). The Bel index ranges from 0 to 3, to select one of the 4 LUTs (LUT-A, LUT-B, LUT-C, and LUT-D) inside the Slice with coordinates (X, Y). Once the specific LUT has been identified, its INIT value can be modified through the 64 configuration bits. As explained in Section 2, this LUT parameter can be modified at run-time thanks to certain software routines provided by Xilinx API. The function *XHwIcap_GetClbBits* is used to read back the INIT value of the LUT and store it in memory. *XHwIcap_SetClbBits* copies any INIT value from system memory into the LUT configuration field. Both functions require the same type of parameters: the coordinates of the LUT (X, Y, and Bel) and the memory address to locate the INIT value. We found very limited information about these functions and the operations they perform. These are in format of object files (.o) and their source code is not available. In addition, the time required to

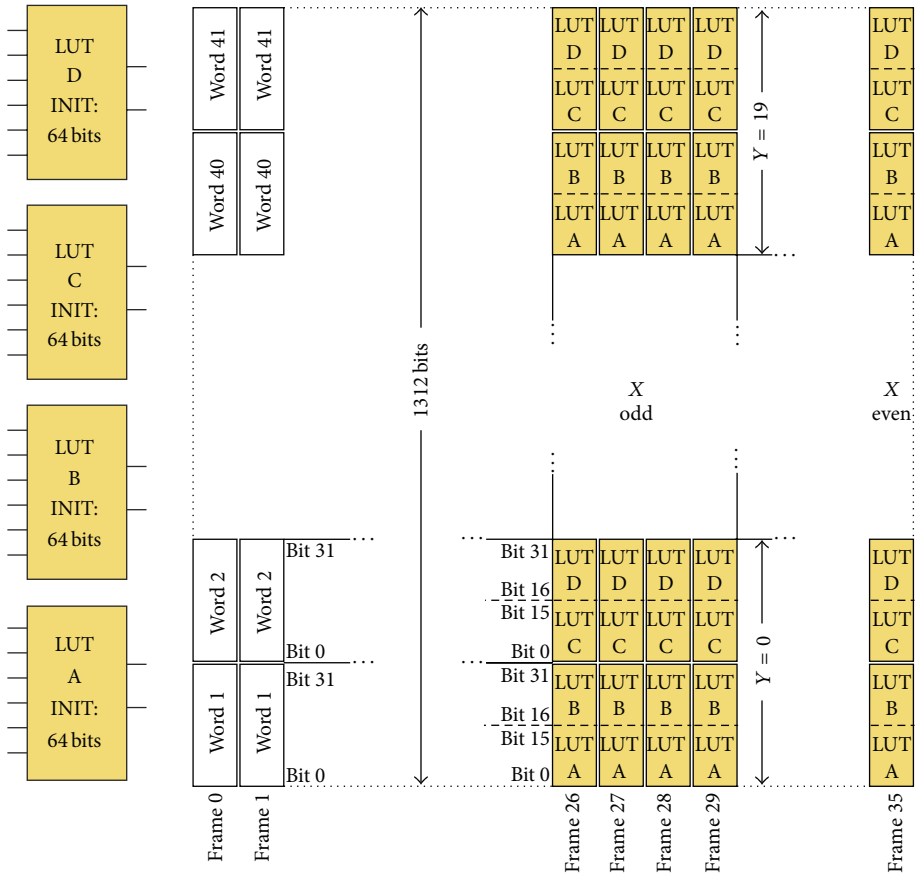

Figure 4: Frame bits for LUT configuration.

read and write the configuration value of a LUT using these functions is in the order of 2 ms while the time for reading and writing frames, using *XHwIcap_DeviceReadFrame* and *XHwIcap_DeviceWriteFrame* functions, is in the order of 30 μs. These numbers, experimentally obtained using a MicroBlaze-based system operating at 100 MHz, offered us opportunities to improve the reconfiguration time for LUTs. Therefore, we performed experiments to deduce the relationship between LUT parameters and configuration frames. By combining the *XHwIcap_SetClbBits* function to write to a specific LUT with the *XHwIcap_DeviceReadFrame* to analyze the programmed values on frames, we found that four frames are used to reconfigure a single LUT.

As shown in Figure 4, the 64 bits of the INIT value span four consecutive frames with each frame containing 16 INIT bits. The 40 Slices inside every CLB column can be seen as 2 columns of 20 Slices. One Slice column contains the 20 Slices with even values on X coordinate while the other 20 Slices presents odd values. The frames 26 to 29 enclose the LUT configuration values for the 20 Slices with odd-X coordinates while the frames 32 to 35 have the corresponding information for the 20 Slices when X coordinate is even. In a similar way, Slice-Y coordinate determines what specific word inside each frame to use. For any CLB column, Y takes 20 consecutive values. Depending on this value, a specific word in the frame corresponds to a single LUT. Two consecutive frame words have the partial information for the 4 LUTs of a Slice. 16 bits of INIT LUT-A and 16 bits of INIT LUT-B configuration values are in one 32-bit word. Similarly, LUT-C and LUT-D INIT values are located in the following word.

4. AC_ICAP Implementation

The AC_ICAP controller, detailed in Figure 5, offers similar functionality to the XPS_HWICAP and AXI_HWICAP available in Xilinx tools but AC_ICAP is fully implemented in hardware, instead of doing most of the tasks as software routines in the processor. It includes support for ReadFrames, WriteFrames, Modify LUTs, and load partial bitstreams from flash and BRAM memory. Compared to similar approaches that also implement reading and writing of frames in hardware [20], our controller is improved by the run-time reconfiguration of LUTs without the need for precomputed partial bitstreams. This last characteristic is of relevance in the implementation of self-adapted systems that may require fine modifications on the hardware according to values generated at run-time, not only based on precomputed values. This aspect will be addressed in more detail in Section 8.

The controller and its internal modules use Finite State Machines (FSMs) to operate on diverse configuration levels according to the values of the input Op_sel specified in Table 1.

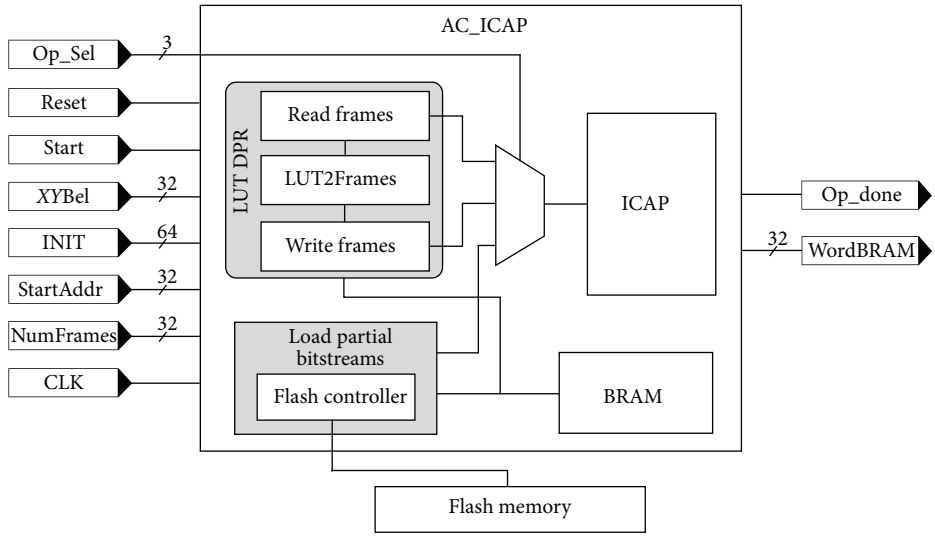

Figure 5: AC_ICAP detail.

Table 1: Coding of tasks.

Operation	Op_sel input
Read BRAM	000
ReadFrame	001
WriteFrame	010
Modify LUT	011
Recover LUT	100
Load partial bitstream from flash	101
Copy partial bitstream from flash to BRAM	110
Load partial bitstream from BRAM	111

The AC_ICAP was initially developed using a board equipped with the Virtex-5 LX110T FPGA and the implementation flow was performed in Xilinx tools, version 14.7. Even though the details are presented for Virtex-5 family, it should be noted that the controller is also implemented in 7-series family as described in Section 6.

As explained in Section 3, DPR of LUTs requires modifying specific parts of frames. Therefore, the two modules for read and write frames are indispensable in the implementation of LUT run-time reconfiguration. We designed the AC_ICAP controller with a space of BRAM able to store partial bitstreams that could reconfigure an area of 4 CLB columns. Then, the controller has low influence on the total BRAMs available in the device (148). In consequence, we included 7–36 Kbit BRAM elements (31.5 KB) configured as dual port memory. This memory space serves to store frames read and it is also used as the source of the frames to send to the ICAP. The initial 2800 Bytes are reserved to perform LUT modification and frame tasks. The remaining 28.7 KB can be used for both frame or partial bitstreams storage, as depicted in Figure 6. When partial bitstreams fit into the available BRAM, the load partial bitstreams from BRAM task can reach the maximum specified throughput because of the direct connection between the on-chip BRAM and the

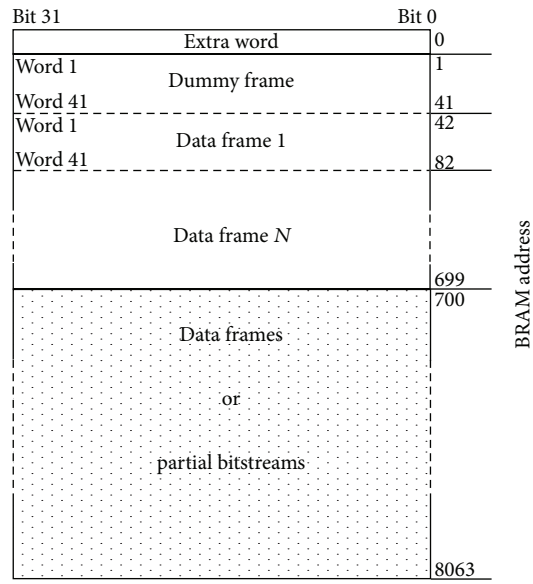

Figure 6: BRAM memory map.

ICAP through a link of 32 bits. By using a clock of 100 MHz, one 32-bit word is available with every clock cycle, which corresponds to the maximum ICAP supported throughput (3.2 Gbps). We adhere to the constraints specified in the technical documents regarding the maximum operation frequency of the ICAP: 100 MHz [4]. However, it should be taken into account that Hansen et al. [18] reported the correct operation of the ICAP when it is overclocked to achieve better reconfiguration throughput speed.

The constituent modules of the AC_ICAP controller are detailed next.

4.1. ReadFrames. ReadFrames module uses two parameters to define the location ($FAddr$) and number of frames (Nf) to

read. Nf takes the value 1 for a single frame read or any other value for multiple frame read. It is limited by the available BRAM memory on the controller. It should be noted that, for LUT modification tasks, one BRAM block is enough but we included six extra blocks to store frames or small partial bitstreams. We store all the read frames on BRAM and there they can be accessed to perform any operation on them. Alternatively, an external module able to process and store the read frames could acquire more frames than those limited by the size of the BRAM. For instance, the DDR memory present in the considered board has a capacity of 256 MB. It can be used to save configuration frames that occupy more than the 31.5 KB of the BRAM available with the AC_ICAP.

In the case of multiple frames ($Nf > 1$), *FAddr* is the address of the first frame where the reading process starts. From there, the routine will read Nf consecutive frames. The steps involved in ReadFrames routine are depicted in Figure 7. When op_sel = "001" and the Start signal is asserted, the ICAP is configured to read the specified frames. This is done by writing to certain registers of the ICAP as detailed in [28]. It is important to point out the correct assertion of the CE and WRITE inputs to define reading or writing operations on the ICAP. In both cases, WRITE should be modified before CE to avoid causing an abort sequence. It is detailed in the two boxes ICAP WRITE and ICAP READ in Figure 7.

The inputs *FAddr* and Nf are used in the two steps of the flow identified with the word Input. These two values are adapted to the format of the corresponding registers. *FAddr* should have the format of the Frame Address Register, that is, one 32-bit word with the fields: block type, Top, HCLK row, column, and frame inside the column. Nf is used to calculate the number of words to read (N) and generate a type 2 word to send to the ICAP. *FAddr* and Nf can be specified by the user through the inputs *StartAddr* and *NumFrames*, respectively. Or they can be generated by the *LUT2Frames* module, as will be explained in Section 4.3.

We must consider that any reading of frames includes one extra dummy frame generated at the beginning of the process and also one extra word. With this in mind, the number of words to read for the Virtex-5 device can be calculated as

$$N = 41 * (Nf + 1) + 1. \quad (1)$$

Equation (1) is valid for any Virtex-5 FPGA as in these devices all the configuration frames have the same size. It is 41 32-bit words. The dummy frame is represented by the addition of 1 to Nf. The last addition represents the initial word.

The state *READ N Words from FDRO* performs the actual read of the N 32-bit words that compose the frames. With every word read from the FDRO register of the ICAP, the BRAM address is increased to store the frames on this memory. Figure 6 shows the location of the frames and the extra word.

4.2. WriteFrames. This module was designed following the same approach as that presented in ReadFrames. The main differences are in the configuration commands required to prepare the ICAP to write to the configuration memory.

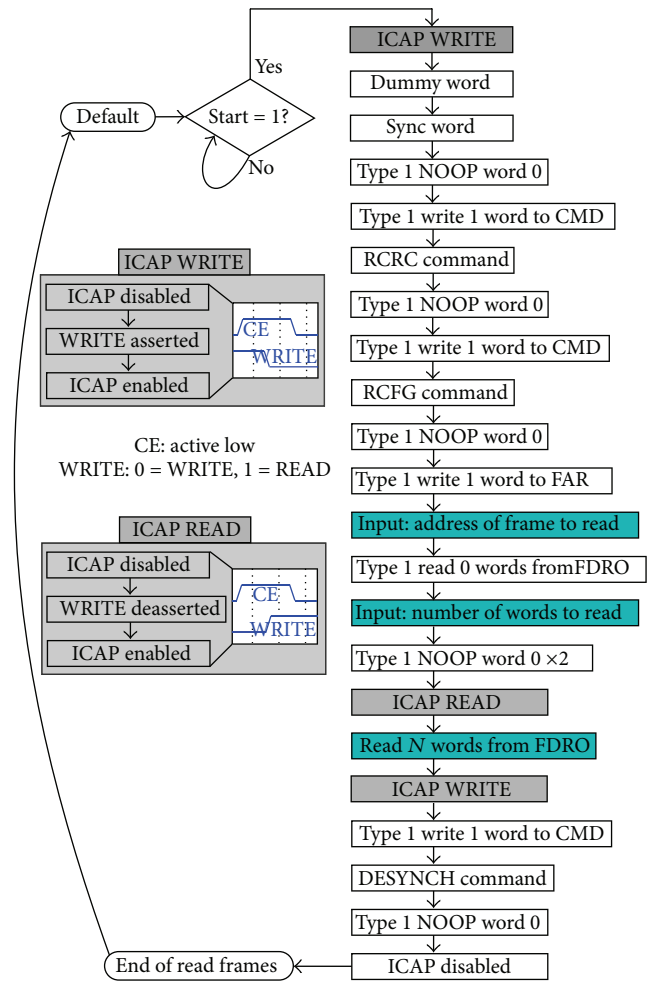

Figure 7: Read frame FSM.

WriteFrames module is activated when the Op_sel input, defined in Table 1, is "010" and the Start signal is asserted. To reach the maximum throughput speed, the preferred source for the frames to write is BRAM. If the frames are located in the BRAM of the AC_ICAP, one 32-bit word is available with every clock cycle.

As this module is normally used in combination with ReadFrames, the frames to be written have already been read and stored on BRAM. Then, WriteFrames module uses the same memory space, detailed in Figure 6, where ReadFrames placed the reading back frames.

In the same way that ReadFrames module needs to consider one dummy frame, in every write frames routine, the dummy frame should be sent to the ICAP at the last part of the process. Therefore, data frames start at BRAM_address = 42 and finish at address $41 * (Nf + 1)$. Immediately after data frames are sent, the dummy frame should follow. To do that, the starting address changes to 1 and finishes when 41 words (1 frame) are sent. The extra word at address 0 is not used in writing processes.

We generate the Op_done output to signalize the end of a writing process. It is necessary to guarantee that the ICAP tasks finish properly. After all the words are sent, it

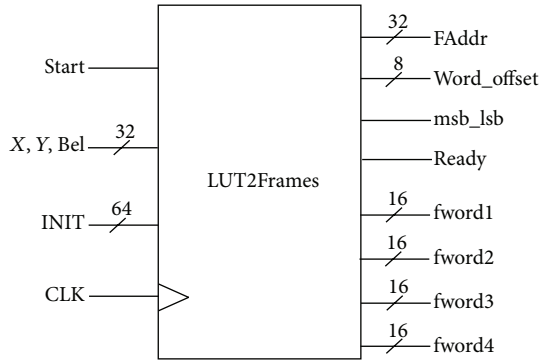

Figure 8: LUT2Frames module.

is necessary to send the DESYNC command and disable the ICAP. Op_done goes high when the ICAP receives and processes the DESYNC command. It is observed when the output port O changes from 0xDF to 0x9F. This process has a delay of 6 clock cycles independently of the value on the input CE.

4.3. DPR of LUTs with LUT2Frames Module. The LUT2Frames module allows the Dynamic Partial Reconfiguration of LUTs by doing the translation of the LUT parameters into frames representation. As described in Section 3, the LUTs are characterized by the coordinates (X, Y, Bel) and the INIT value. The LUT2Frames module, depicted in Figure 8, carries out two main tasks: (1) translating the X, Y, Bel coordinates into FAR format and (2) transforming the INIT (64 bits) LUT function into 4 words of 16 bits each.

The X, Y, Bel inputs, merged into one 32-bit word, and the INIT value are used by the LUT2Frames module when the Start input is set. Based on the coordinate values, one 32-bit word with the format of the Frame Address Register (*FAddr*) is generated to define the frame where the reading and writing start. In addition, X, Y and Bel values determine the word_offset that is the concrete word of each frame (the first one of the 2–41 words) that needs to be manipulated. From the 32-bit word, only 16 bits correspond to a specific LUT. Therefore, the signal msb_lsb indicates which part of the 32-bit word should be modified: 0 for the LSB part of the word (LUT-A or LUT-C) and 1 for the 16 MSB (LUT-B or LUT-D).

In parallel with the previous processing, LUT2Frames module generates four 16-bit words (fword1$\cdots$fword4) that correspond to the INIT value transformed and adapted to the four frames.

All the complexity of the frames location and addressing is transparent to the user. The LUT2Frames module implements all the translations and computes adequate addresses and memory management to allow for the user a simple operation when required to modify any LUT throughout the device.

When a LUT modification is required, the steps controlled by an FSM, as depicted in Figure 9, are executed. The process is triggered by the Start signal; then, the LUT2Frames module is activated. With the values generated by this

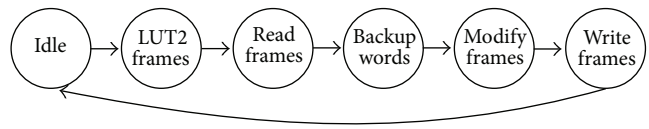

Figure 9: FSM for DPR of LUTs.

module, 4 frames starting at *FAddr* are read and stored in BRAM (read frames). Word_offset and msb_lsb signalize the specific words that should be modified. These 4 words are backed up (backup words), modified with the four words that LUT2Frames produced, and copied back to BRAM. At this point, the BRAM contains the frames with the new words, and the WriteFrames module performs the writing of the 4 frames corresponding to the LUT.

The Recover LUT routine uses the four backed up values obtained at the backup words stage to recover the LUT to its previous configuration value. Considering Figure 9, it only performs the last two steps of a LUT modification routine. It modifies the 4 frames on BRAM and then these are sent through WriteFrames module to recover the LUT to its previous INIT value. This routine is useful in applications that need to recover the previous function of a LUT before modifying another. By following this approach, we avoid reading four frames again as these are already on BRAM.

The correct operation of the controller was verified using ChipScope Pro Debugger [29]. Figure 10 shows the details for a LUT modification process. We specified the X, Y, Bel, and INIT values of the LUT to modify. The steps shown in Figure 9 can be identified in Figure 10. The LUT2Frames module requires only two clock cycles and the information it generates is used to address the four frames to read and to modify the four specific words in these frames.

4.4. Load Partial Bitstreams. This module follows an approach similar to the described in Section 2 in regard to the speed-up of partial reconfiguration by loading partial bitstreams. The load partial bitstreams module performs three main tasks: (1) load partial bitstreams from flash, (2) copy partial bitstreams from flash to BRAM, and (3) load partial bitstreams from BRAM. To do that, this module includes a memory access controller to read the partial bitstreams from flash memory. Therefore, the data read from flash can be directly sent to the ICAP I port or it can be copied into internal BRAM. When partial bitstreams are on BRAM, the maximum configuration speed on the ICAP can be reached. If partial bitstreams are on external memory, the reconfiguration time depends on the latency of the access to the memory. In this case, we use the Intel StrataFlash memory 28F256P30 which requires 26 clock cycles at 100 MHz to get a 32-bit word.

The size of the partial bitstreams that can be placed on BRAM is limited by the available BRAM memory on the controller. From the 7–36 Kbit BRAM present in the AC_ICAP, we reserved 2800 Bytes to perform LUT modification and frame tasks. Therefore, the maximum size of partial bitstreams that can be placed is 28.7 KB. It can be increased as

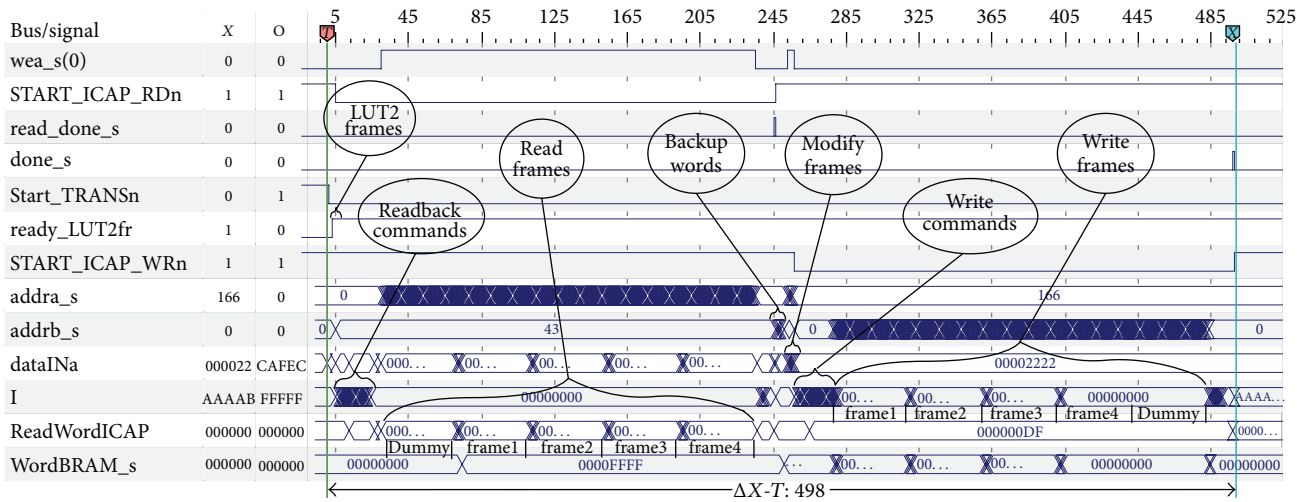

Figure 10: Chipscope detail of LUT-DPR with AC_ICAP in Virtex-5.

the FPGA includes more BRAMs (148 in the LX110T device) but it depends on the application constraints.

The partial bitstreams are generated following the standard Xilinx flow; it is using PlanAhead or bitgen tools. These configuration files include header information regarding the type of device, size of the configuration data, date and time of the generation of the bitstream, and so forth. We adapt the partial bitstreams to remove unnecessary information from the header and keep only the last header-field that corresponds to the size (in Bytes) of the partial bitstream not including the header. Therefore, our controller firstly reads the word that contains the size of the partial bitstream and uses this information to calculate the number of words (16-bit word for flash and 32-bit word for BRAM) to read from memory. With this approach, the only required parameter is the initial address where partial bitstreams are located. The controller automatically calculates the end address and performs the reading process. Depending on the operation selected by the input Op_sel, the data is sent to the ICAP or to BRAM. In a similar way, when Op_sel is set to "111," this module configures the ICAP control signals and BRAM address to allow high throughput partial reconfiguration.

5. AC_ICAP Adapted to On-Chip Processor

To make the controller able to be attached to processor-based designs, it was adapted to the Peripheral Local Bus and Fast Simplex Link interfaces used by MicroBlaze systems. To this end, the AC_ICAP was considered as a black box with the I/O ports depicted in Figure 5 and these were adapted to the respective buses. This approach offers increased flexibility as the controller can be easily commanded from the processor. We created a collection of functions adapted to each interface to perform the tasks presented in Table 1. Such functions, depicted in Code 1, use specific routines from the Xilinx API to access the PLB and FSL interfaces.

Code 1. Functions to drive the AC_ICAP IPs are as follows:

```
ReadBRAM (StartAddr);
ReadFrame (StartAddr, NumFrames);
WriteFrame (StartAddr, NumFrames);
ModifyLUT (XYBel, INIT);
RecoverLUT (XYBel);
LoadPBitsFlash (StartAddr);
CopyFlash2BRAM (StartAddr);
LoadPBitsBRAM (StartAddr);
```

The *StartAddr* parameter refers to a unique input that should be adapted according to the *op_sel* value. In the case of read and write frames, it corresponds to the address of the initial frame (*FAddr*). For the other functions, it is the memory address where data are stored. *NumFrames* is the number of frames to read or write and $X, Y, Bel, INIT$ are the parameters that control single LUTs. These are the only values required to command the AC_ICAP controller as this performs internally all the operations such as transforming the X, Y, Bel and $INIT$ into frame format, compute end address after reading the size of a partial bitstream, and so forth.

5.1. PLB IP. The PLB bus is used to connect peripherals to the MicroBlaze processor. The original AC_ICAP, designed in VHDL, is instantiated in a PLB wrapper to generate the custom PLB_AC_ICAP IP. The inputs and outputs of the controller are connected to signals of the PLB bus and then the processor can access them using register addresses. In consequence, the PLB_AC_ICAP can be attached to any MicroBlaze-based system such as the depicted in Figure 11. This architecture includes the flash memory where the full and partial bitstreams that modify the reconfigurable areas are located. The direct connection to the flash memory is also performed in the IP design by defining the AC_ICAP

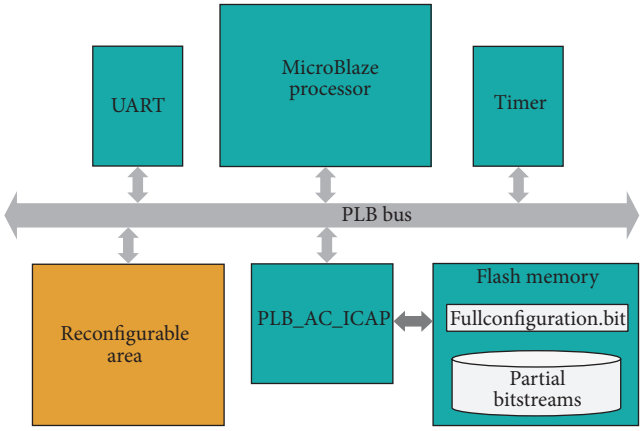

Figure 11: Architecture with PLB_AC_ICAP IP.

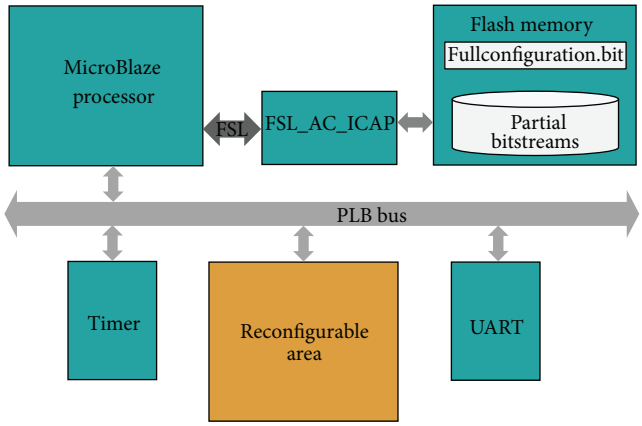

Figure 12: Architecture including FSL_AC_ICAP coprocessor.

connections to the flash as external ports. Once included in the hardware design in EDK, the software running in the processor is able to control the PLB_AC_ICAP peripheral by using the functions listed in Code 1. In consequence, a partial reconfiguration related task uses any of the functions specified in Code 1 and monitors the output op_done until it goes high as confirmation that the task has been completed.

5.2. FSL Coprocessor. Fast Simplex Link is an interface of the MicroBlaze processor that allows for including dedicated hardware routines with high execution priority and, therefore, implies low latency in the communication with the processor. In this approach, we adopted a solution similar to the presented in [13], in order to obtain minimal degradation in the performance of the controller due to the bus latency. Thus, the VHDL-based AC_ICAP was adapted to the FSL interface to be easily connected as a coprocessor and consequently exploit all the flexibility of the processor but taking advantage of the hardware acceleration in the ICAP related tasks. Figure 12 presents a system using the FSL_AC_ICAP coprocessor.

The FSL_AC_ICAP coprocessor is accessed in a similar way to that considered in the PLB_AC_ICAP IP, that is, by means of a collection of functions such as that presented in Code 1. The main differences are in the type of routines that these functions require to drive the FLS. In this case, we incorporate the blocking routines `putfsl` and `getfsl` available with Xilinx API as we consider that the reconfiguration tasks are of high priority.

6. Using the AC_ICAP in Newer Device Families

To validate the controller in 7-series devices, we use the KC705 board equipped with a Kintex7 XC7325T FPGA [30]. This FPGA contains 50,950 Slices; inside every Slice, 4 6-input LUTs and 8 FF are present. The 445 BRAMs correspond to 2002 KB and the bitstream size is 10.9 MB. To adapt the AC_ICAP, designed for Virtex-5, to 7-series devices, certain changes are required. The main differences are summarized as follows:

(i) The number of words per frame in 7-series family is 101 instead of 41 (Virtex-5). It is because the CLB columns in 7-series FPGAs are 50 high by 1 wide which implies that 100 Slices are present in the CLB columns. Similarly, the number of HCLK rows is different; for this specific device, it is 7 (3 top and 4 bottom).

(ii) The address of the frame where to start reading or writing is defined by the FAR register. For 7 series, this register uses 26 bits of the 32 available while in Virtex-5 FAR it uses 24. It is due to the increased size of the FPGA.

(iii) As opposed to Virtex-5, for 7 series, no extra word is required at the beginning of a read frames task. Therefore, the number of words (N_{words7}) to read/write from these devices can be computed according to (2) that is valid for any 7 series FPGA as in these devices all the configuration frames have the same size. The dummy frame is represented by the addition of 1 to the number of frames (Nf):

$$N_{\text{words7}} = 101 * (Nf + 1). \quad (2)$$

(iv) The word_offset that indicates what specific word on a frame should be modified in a LUT-DPR process now has a range of 0 to 100. It varies between 0 and 40 for Virtex-5. In a similar way, the skip columns (columns that contain resources different from CLBs: BRAMs, DSPs I/O) and the major column numbering require to be updated. The first column in Kintex7 has a major address of 2, while it is 1 for Virtex-5.

(v) In 7-family, the primitive ICAPE2 does not have the BUSY output. Instead, we should consider 3 clock cycles after CE assertion to get the valid data.

(vi) The WriteFrames module also required some changes. In Virtex-5, it is possible to bypass the CRC calculation by setting a configuration register (COR0-bit28) and loading the value 0xDEFC to the CRC register every time that the FAR is modified. In

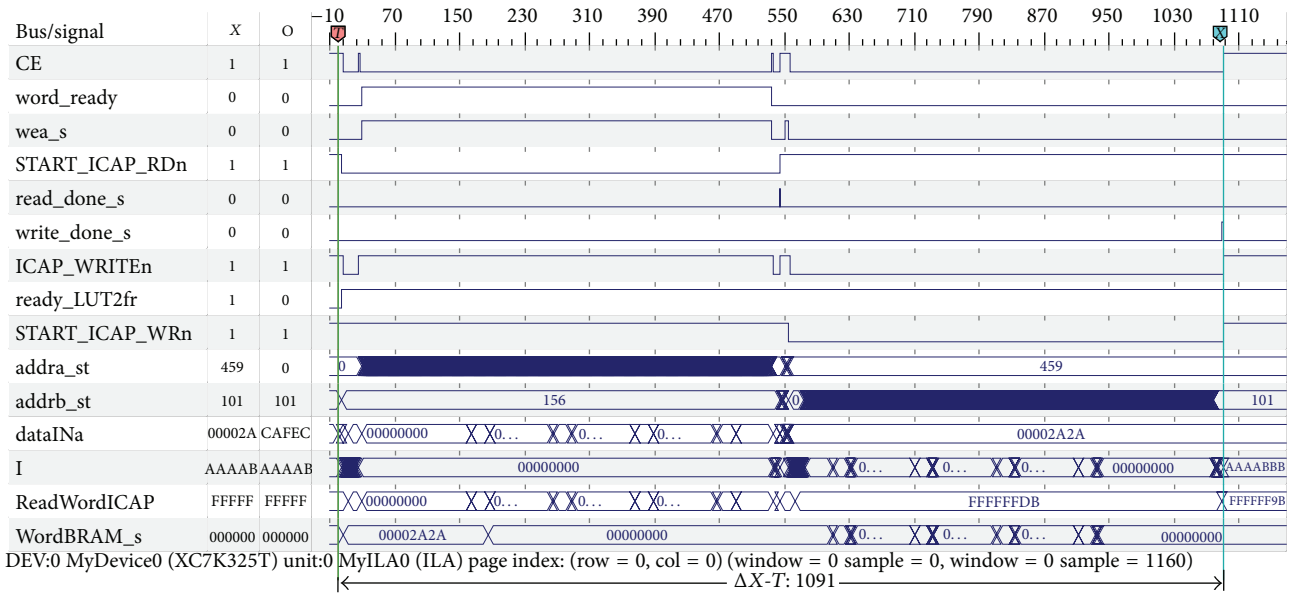

Figure 13: Chipscope detail of LUT-DPR with AC_ICAP in Kintex7.

7-family, such register is not present; by default, the new control register (COR1-bits15-16) is set to allow a continuous operation for the system after CRC is computed and therefore such steps were removed.

(vii) The flash memory available in this board is of the same type as the one present in the Virtex-5 but as the size is different, the flash controller was modified to include two extra address lines.

The number of frames required to configure a CLB column remain the same (36), so do the specific frames that contain the information for LUTs. We used 22 BRAM blocks to occupy a similar percentage (5%) as in Virtex-5.

Once the presented changes were performed in the AC_ICAP, it was implemented in the Kintex7 FPGA and tested with all the operations it supports. In Figure 13, we present again the details for DPR of one LUT as it involves diverse tasks available in the controller.

This new AC_ICAP was adapted to the AXI interface as this is used for all new Xilinx families. This IP is identified as AXI_AC_ICAP and supports the same functions presented in Code 1 that were adapted to the AXI API.

Based on the previous descriptions, we have different variations of the controller to evaluate: AC_ICAP, the standalone hardware version; PLB_AC_ICAP and AXI_AC_ICAP, adapted to PLB and AXI buses, respectively; and FSL_AC_ICAP, used as a coprocessor. We used PlanAhead 14.7 and Vivado 2015.3 to define reconfigurable partitions of different sizes (from 1 to 10 CLB columns) and generate different partial bitstreams.

For the Xilinx-based controllers, we implemented architectures such as those depicted in Figure 11 but instead of using the PLB_AC_ICAP we added the XPS_HWICAP or the AXI_HWICAP with the parameters that allow the best performance in reconfiguration throughput (Write FIFO Depth = 1024, Read FIFO Depth = 256, and FIFO type enabled). For these two cases, the Xilinx flash memory controller was also included to have access to the partial bitstreams located in this memory. In doing this, we can get accurate comparisons as we used the same tools version and synthesis options.

7. Experimental Results

This section summarizes the main results regarding the reconfiguration speed and resources utilization of the diverse versions of the AC_ICAP controller. We consider as primary reference comparing the Xilinx XPS_HWICAP for Virtex-5 and the AXI_HWICAP for Kintex7 as these are, among the reported alternatives, the ones that support most of the DPR tasks. We take into account that for partial bitstreams that configure up to 4 CLB columns it is possible to copy them into BRAM as it is limited to 28.7 KB for Virtex-5 and to 99 KB for Kintex7. To record the time performance of the AC_ICAP (standalone version), the ChipScope Pro was used. For versions adapted to processor interfaces, the timer included in the systems was used to register the number of clock cycles required for the specific tasks. These numbers are reported in Table 2. Here, we want to mention some issues in regard to the values obtained for Kintex7 FPGA. The LUT functions that the AXI_HWICAP includes do not support the 7-family. With the most recent version of the tools at the moment of performing the experiments (Vivado 2015.3 and driver hwicap_v10_0), the support is only given for Virtex6 and previous devices and we cannot modify them as the source code is not available. The functions for read and write frames using the AXI_HWICAP required the modification of some header files as these present some erroneous values. The file xhwicap_i.h: uses the values for Virtex6 in the 7-family but these should not be the same. For instance, it is declared that

Table 2: Timing behavior of AC_ICAP.

	Controller	LUT Reconf. [μs]	ReadFrame [μs]	WriteFrame [μs]	Reconf. throughput from BRAM [MB/s]	Reconf. throughput from flash [MB/s]
Kintex7*	AC_ICAP	10.91	2.39	2.33	380.47	14.66
	AXI_AC_ICAP	11.78	3.06	3.01	378.37	14.65
	AXI_HWICAP [19]	n/a	58.08	63.54	n/a	1.25
Virtex-5**	AC_ICAP	4.98	1.18	1.17	381.03	14.67
	PLB_AC_ICAP	5.88	1.90	1.90	378.73	14.66
	FSL_AC_ICAP	5.36	1.57	1.56	378.85	14.67
	XPS_HWICAP [4]	1912.17	29.21	32.16	n/a	1.32
	[15]	n/a	n/a	n/a	384.29§	6.57
	[14]	n/a	n/a	n/a	n/a	0.86
Virtex-4†	[20]	n/a	n/a	n/a	371.42	n/a
	BRAM_HWICAP [11]	n/a	n/a	n/a	371.4	n/a
	ICAP-I [16]	n/a	n/a	n/a	180	29‡

*All 7-series FPGAs are 6-input LUTs, frames of 101 32-bit words.
**Virtex-5: 6-input LUTs, frames of 41 32-bit words.
‡Valid for SD memory AT49BV322A.
†Virtex4: 4-input LUTs, frames of 41 32-bit words.
§Estimated value, not implemented.

the number of words in a frame for both families is 81. But for 7-series families, the correct value is 101. Something similar happens with the FAR creation. The driver creates the FAR with some parameters that are valid for Virtex-6 but not for Kintex7 and these were modified to obtain correct operation.

As can be observed from Table 2, the reconfiguration time of LUTs using the AC_ICAP is, to the best of our knowledge, the fastest reported alternative. Compared to the XPS_HWICAP in Virtex-5, it implies speed-up of more than 320 times for the PLB_AC_ICAP, the slowest version, and the standalone AC_ICAP offers improvement in reconfiguration time of LUTs of more than 380 times. In a similar way, the speed-up of read and write frame tasks, considering both Virtex-5 and Kintex7, experiences improvements of more than 18 and 21 times, respectively.

The reconfiguration throughput for load partial bitstreams from BRAM, for the AC_ICAP, is 380.47 and 381.03 MB/s for Virtex-5 and Kintex7, respectively. It is close to the maximum supported throughput of 400 MB/s and to the reported values on [15, 20]. For the work reported on [15], it should be noted that the value is estimated and not measured in a real implementation; since that controller does not include BRAMs. The deviation of our controller from the 400 MB/s value is due to the extra clock cycles required to start reading the BRAM and processing the DESYNC command (0x0D) by the ICAP. For every ICAP related task, we consider it finishes when the DESYNC command is acknowledged. It is done by monitoring the O port of the ICAP which changes from 0xDF to 0x9F in Virtex-5 and from 0xFFFFFFDB to 0xFFFFFF9B in Kintex7, as a confirmation of the success in completing the tasks. This implies six extra clock cycles after the last data is sent to the ICAP.

For the PLB, AXI, and FSL versions, there is some degradation in time due to the latency of the interfaces, but in all cases, they offer improvements of more than 11 times for load partial bitstream from flash.

The time to copy the partial bitstream from flash to BRAM is on the same range as the required one to load partial bitstream from flash. Instead of sending data to ICAP, these are stored on BRAM. Therefore, it can be especially useful when the application can copy the partial bitstreams to BRAM before the execution starts, for instance, at booting time.

In regard to resources utilization, Table 3 presents the details for every module of the AC_ICAP controller. It should be noted that the AC_ICAP includes the flash memory controller, which is not the case for XPS_HWICAP and AXI_HWICAP. Table 4 summarizes the resources required by the diverse options of the controller. The extra resources of PLB, AXI, and FSL versions of the AC_ICAP are due to the wrapper logic required to adapt the controller to these interfaces. It can be seen that the most resource demanding approach uses 5% of the Slices, which can be considered a reasonable size as all the operations are done in hardware.

Finally, in Table 5, we compare the resources required by complete MicroBlaze-based architectures including different versions of the ICAP controller. We can see that the systems using the AC_ICAP adapted to the PLB and FSL require on average 3% more resources of the Virtex-5 FPGA than the XPS_HWICAP alternative. This is the area overhead to pay in order to speed up all the reconfiguration tasks, such as the reconfiguration time of LUTs that is improved in 356x when the FSL_AC_ICAP is used. When we see the data for Kintex7, the area percentage is lower as the devices are bigger. Therefore, the speed-up of tasks takes increased relevance as the quantity of configuration data to manage has become bigger but the speed and bus width supported by the ICAP primitive remains the same since the Virtex-4 generation

Table 3: Resource utilization of AC_ICAP.

Module	Virtex-5			Kintex7		
	LUT	FF	BRAM	LUT	FF	BRAM
AC_ICAP	1667	1161	7	1286	1193	22
+Top FSM	691	471	0	452	585	0
++BRAM	36	3	7	27	3	22
++Load partial bitstreams	109	133	0	70	130	0
+++Flash controller	129	112	0	73	68	0
++Lut2Frames	119	118	0	39	136	0
++ReadFrames	230	138	0	232	113	0
++WriteFrames	353	186	0	393	158	0

Table 4: Resource utilization of ICAP controllers.

	Controller	Slices	LUTs	Flip flops	BRAM
Virtex-5	AC_ICAP	690 (3%)	1667 (2%)	1161 (1%)	7 (4%)
	PLB_AC_ICAP	952 (5%)	2375 (3%)	1609 (2%)	7 (4%)
	FSL_AC_ICAP	903 (5%)	2329 (3%)	1484 (2%)	7 (4%)
	XPS_HWICAP [4]	453 (2%)	714 (1%)	745 (1%)	3 (2%)
	[15]	*	96	87	0
	[14]	*	*	*	*
Kintex7	AC_ICAP	595 (1%)	1286 (1%)	1193 (1%)	22 (5%)
	AXI_AC_ICAP	734 (1%)	1578 (1%)	1332 (1%)	22 (5%)
	AXI_HWICAP	248 (1%)	546 (1%)	741 (1%)	2 (1%)

*Not reported.

Table 5: Resource utilization of full systems with different ICAP controllers.

	System using	Slices	LUTs	Flip flops	BRAM
V5	PLB_AC_ICAP	2084 (12%)	4556 (6%)	3516 (5%)	23 (15%)
	FSL_AC_ICAP	2094 (12%)	4643 (6%)	3450 (4%)	23 (15%)
	XPS_HWICAP	1631 (9%)	3077 (4%)	2981 (4%)	19 (12%)
K7	AXI_AC_ICAP	2054 (4%)	4160 (2%)	3725 (1%)	37 (8%)
	AXI_HWICAP	1471 (2%)	3311 (1%)	2708 (1%)	17 (3%)

(32-bits@100 MHz). From the presented data, we can summarize that the best performance-area trade-off is given by the AC_ICAP which uses 3% of the FPGA resources but offers speed-up of 380x in LUTs DPR.

Dynamic Partial Reconfiguration of LUTs using this approach presents the advantage that it does not require precomputed partial bitstreams for each modification to be performed. It allows run-time LUT modification with any boolean value and it is not limited by the availability of partial bitstreams in memory. This fine partial run-time reconfiguration is of increasing relevance in applications such as fault injection platforms and in cryptographic implementations where the hardware can be modified at LUT level to avoid certain types of attacks. A case application of these fine-grain modifications is presented in the next section.

8. AC_ICAP for LUT Evaluation of an AES Module

In this section, we use the AC_ICAP to evaluate an AES module available in [31]. The idea is to have a way to identify critical configuration values for LUTs. With such information, it is possible to design countermeasures strategies against external attacks. For instance, this approach could be employed to modify the logic behavior of certain LUTs to produce erroneous values without stopping the system. In doing so, the AES can be continuously working giving a false sense of correct operation that can be exploited as countermeasures against attacks such as differential power analysis.

If the partial bitstream approach is used to modify the LUTs, a partial bitstream is required for every LUT to modify. These should be generated at design time and copied to memory. Therefore, all the possible modifications of the LUTs should be defined at design time and once the system is operating it is very difficult to include any variation such as a new LUT modification because it implies time consuming process to generate new partial bitstreams. The advantage of DPR of LUTs supported by the AC_ICAP is that no partial bitstreams are required and any logic modification can be dynamically performed. To evaluate this approach, we used a pseudo random number generator (PRNG) to produce the 64 bits of the configuration memory for the LUTs to be modified. We do not focus on the details of AES or PRNG. Our goal is to offer a way to easily identify LUTs and its key values to be used in the evaluation and design of critical modules.

The architecture of the system is depicted in Figure 14 and it is implemented in a Virtex-5 FPGA. We included two replicas of the AES module to have online comparison of the outcomes and the BRAM stores the information of the LUTs. A single copy of the AES requires 8360 FF and 13952 LUTs. The DUT replica was constrained in area and defined as a partition to be used keeping the routing defined in the initial implementation. An area of 88 CLB columns (14080 LUTs) was defined to place the AES. As we can reuse the implemented partition in other designs, the values obtained with the DPR of LUTs remain valid for different implementations. The system, controlled by the FSM, uses the PRNG to get random configuration values to configure the LUTs and the AC_ICAP is used to modify the LUTs on the DUT area by using the X, Y coordinates of the Slices. Once a LUT is modified, some test bench inputs are applied to both the golden and DUT components and the outputs are analyzed to determine if the LUT modification produced erroneous values. After all input patterns are applied, the effect of such modification is classified. If this produced erroneous values, the LUT address and the configuration

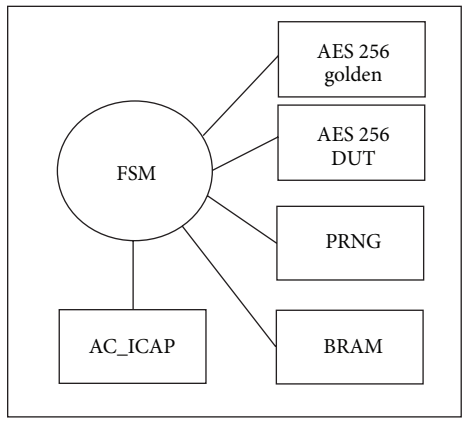

Figure 14: Architecture for AES with DPR of LUTs.

value are stored. The LUT is recovered to the previous value and a new LUT is tested. If no erroneous values were produced, it can be bypassed or tested with a new configuration value. Therefore, this approach allows a flexible alternative to evaluate the system exhaustively or in a more relaxed way. The information stored in BRAM is then used to decide what LUTs and its associated configuration values could be employed to intentionally modify logic functions when the system is being attacked.

9. Conclusion and Future Work

We presented the AC_ICAP, a new ICAP controller verified in Virtex-5 and Kintex7 FPGAs. It is able to load partial bitstreams, read and write frames, and also modify any LUT in the FPGA, in this last case without the need for pregenerated partial bitstreams. The controller was adapted to be easily included in systems with embedded processors using the PLB, FSL, and AXI links. Reconfiguration speed analysis of the processor-independent version shows improvement of more than 380 times in run-time reconfiguration of LUTs compared to XPS_HWICAP functions for Virtex-5 FPGAs. As our controller is fully implemented in hardware, it obviously requires more resources, but in any case it occupies more than 5% of the available elements on the XC5VLX110T device. Therefore, the AC_ICAP offers a complete high speed solution to perform diverse Dynamic Partial Reconfiguration tasks with acceptable FPGA footprint. It was used in the design of an AES module that can modify specific LUTs as a possible countermeasure against attacks.

As future work, we plan to extend the AC_ICAP with a DDR controller to speed up the reconfiguration tasks when these are based on precomputed partial bitstreams not able to be copied into BRAM due to their size. Therefore, DDR memory is as an alternative to overcome the limitations that available BRAM imposes.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] Keysight Technologies, *M9451A-DPD PXIe Measurement Accelerator*, 2015.

[2] Xilinx, *7 Series FPGAs Overview DS180 (v1.16.1)*, Xilinx, 2014.

[3] Altera, *A New FPGA Architecture and Leading-Edge FinFET Process Technology Promise to Meet Next Generation System Requirements WP-01220-1.1*, Altera, San Jose, Calif, USA, 2015.

[4] Xilinx, *LogiCORE IP XPS HWICAP (v5.01a) DS586*, Xilinx, 2011.

[5] L. A. Cardona, J. Agrawal, Y. Guo, J. Oliver, and C. Ferrer, "Performance-area improvement by partial reconfiguration for an aerospace remote sensing application," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '11)*, pp. 497–500, Cancun, Mexico, November-December 2011.

[6] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems," in *Reconfigurable Computing: Architectures, Tools and Applications*, P. Sirisuk, F. Morgan, T. El-Ghazawi, and H. Amano, Eds., vol. 5992 of *Lecture Notes in Computer Science*, pp. 55–67, Springer, Berlin, Germany, 2010.

[7] S. Bhandari, S. Subbaraman, S. Pujari et al., "High speed dynamic partial reconfiguration for real time multimedia signal processing," in *Proceedings of the 15th Euromicro Conference on Digital System Design (DSD '12)*, pp. 319–326, Izmir, Turkey, September 2012.

[8] IBM, *128-Bit Processor Local Bus Architecture Specifications*, IBM Corporation, Armonk, NY, USA, 2007.

[9] K. Glette and P. Kaufmann, "Lookup table partial reconfiguration for an evolvable hardware classifier system," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '14)*, pp. 1706–1713, Beijing, China, July 2014.

[10] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 965–970, 2007.

[11] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 498–502, August 2009.

[12] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 535–538, Heidelberg, Germany, September 2008.

[13] M. Hübner, D. Göhringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW '10)*, pp. 1–8, IEEE, Atlanta, Ga, USA, April 2010.

[14] S. Lamonnier, M. Thoris, and M. Ambielle, "Accelerate partial reconfiguration with a 100% hardware solution," *Xcell Journal*, no. 79, pp. 44–49, 2012.

[15] J. Tarrillo, F. A. Escobar, F. L. Kastensmidt, and C. Valderrama, "Dynamic partial reconfiguration manager," in *Proceedings of the IEEE 5th Latin American Symposium on Circuits and Systems (LASCAS '14)*, pp. 1–4, IEEE, Santiago, Chile, February 2014.

[16] V. Lai and O. Diessel, "ICAP-I: a reusable interface for the internal reconfiguration of Xilinx FPGAs," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '09)*, pp. 357–360, Sydney, Australia, December 2009.

[17] M. Straka, J. Kastil, and Z. Kotasek, "Generic partial dynamic reconfiguration controller for fault tolerant designs based on FPGA," in *Proceedings of the 28th Norchip Conference (NORCHIP '10)*, pp. 1–4, IEEE, Tampere, Finland, November 2010.

[18] S. G. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced ICAP hard macro," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW '11)*, pp. 174–180, Shanghai, China, May 2011.

[19] Xilinx, *AXI HWICAP v3.0*, Xilinx, San Jose, Calif, USA, 2015.

[20] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A novel high-performance fault-tolerant ICAP controller," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '12)*, pp. 259–263, IEEE, Erlangen, Germany, June 2012.

[21] A. Ebrahim, T. Arslan, and X. Iturbe, "On enhancing the reliability of internal configuration controllers in FPGAs," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '14)*, pp. 83–88, IEEE, Leicester, UK, July 2014.

[22] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Proceedings of the IEEE Aerospace Conference*, pp. 1–10, IEEE, Big Sky, Mont, USA, March 2008.

[23] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "Multiple-clone configuration of relocatable partial bitstreams in Xilinx Virtex FPGAs," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '13)*, pp. 178–183, Torino, Italy, June 2013.

[24] U. Legat, A. Biasizzo, and F. Novak, "SEU recovery mechanism for SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 59, no. 5, pp. 2562–2571, 2012.

[25] C. Schuck, B. Haetzer, and J. Becker, "An interface for a decentralized 2D reconfiguration on Xilinx Virtex-FPGAs for organic computing," *International Journal of Reconfigurable Computing*, vol. 2009, Article ID 273791, 11 pages, 2009.

[26] Xilinx, *Partial Reconfiguration User Guide UG702 (V14.7)*, Xilinx, 2013.

[27] Xilinx, *Command Line Tools User Guide UG628 (v 14.7)*, Xilinx, San Jose, Calif, USA, 2013.

[28] Xilinx, *Virtex-5 FPGA Configuration Guide UG191 (V3.11)*, Xilinx, 2012.

[29] Xilinx, *ChipScope Pro Software and Cores*, Xilinx, San Jose, Calif, USA, 2012.

[30] Xilinx, *Xilinx Kintex-7 FPGA KC705 Evaluation Kit*, Xilinx, San Jose, Calif, USA, 2015.

[31] Opencores, "AES project," 2015, http://opencores.org/project.