# A Self-Reconfigurable Platform for General Purpose Image Processing Systems on Low-Cost Spartan-6 FPGAs

Salih Bayar *, Arda Yurdakul†
Computer Engineering
Boğaziçi University
P.K. 2 TR-34342 Bebek,
Istanbul, TURKEY
Phone: +90 212 359 7780
Fax: +90 212 287 2461
Email: salih.bayar@boun.edu.tr *
Email: yurdakul@boun.edu.tr †

Mehmet Tukel
Dept. of Electronics and Communications Engineering
Istanbul Technical University
Maslak,
Istanbul, TURKEY
Phone: +90 212 285 67 32
Fax: +90 212 285 35 65
Email: tukel@itu.edu.tr

***Abstract*—There is still no partial reconfiguration tool support on low-cost Field Programmable Gate Arrays (FPGAs) such as old-fashioned Spartan-3 and state-of-the-art Spartan-6 FPGA families by Xilinx. This forces the designers and engineers, who are using the partial reconfiguration capability of FPGAs, to use expensive families such as Virtex-4, Virtex-5 and Virtex-6 which are officially supported by partial reconfiguration (PR) software. Moreover, Xilinx still does not offer a portable, dedicated self-reconfiguration engine for all of the FPGAs. Self-reconfiguration is achieved with general-purpose processors such as MicroBlaze and PowerPC which are too overqualified for this purpose. In this study, we propose a new self-reconfiguration mechanism for Spartan-6 FPGAs. This mechanism can be used to implement large and complex designs on small FPGAs as chip area can be dramatically reduced by exploiting the dynamic partial reconfiguration feature for on-demand functionality loading and maximal utilization of the hardware. This approach is highly attractive for designing low-cost compute-intensive applications such as high performance image processing systems. For Spartan-6 FPGAs, we have developed hard-macros and exploited the self-reconfiguration engine, compressed Parallel Configuration Access Port (cPCAP) [1], that we designed for Spartan-3. The modified cPCAP core with block RAM controller, bitstream decompressor unit and Internal Configuration Access Port (ICAP) Finite State Machine (FSM) occupies only about 82 of 6,822 slices (1.2% of whole device) on a Spartan-XC6SLX45 FPGA and it achieves the maximum theoretical reconfiguration speed of 200MB/s (ICAP, 16-bit at 100MHz) proposed by Xilinx. We have also implemented a Reconfigurable Processing Element (RPE) whose arithmetic unit can be reconfigured on-the-fly. Multiple RPEs can be utilized to design a General Purpose Image Processing System (GPIPS) that can implement a number of different algorithms during runtime. As an illustrative example, we programmed the GPIPS on Spartan-6 for switching between two applications on-demand such as two-dimensional filtering and block-matching.**

## I. Introduction

Due to recent enhancements in embedded applications, designs are getting larger and more complex. The larger the system, the larger the chip is required to fit the design into the target device by designers. Hence, the large digital hardware systems can be partitioned into smaller modules so that each module runs on the target device at different time intervals by utilizing the same hardware resources. Thus, a very large digital hardware system can fit into a smaller chip. Furthermore, the power dissipation of digital systems plays an important role especially in large designs. Effective energy saving methods have to be incorporated in modern Integrated Circuit (IC) designs nowadays. Moreover, when image processing is concerned, high performance and power consumption are major issues especially in portable devices.

By exploiting the partial run-time reconfiguration of an FPGA, a large design can fit into a smaller device. As the hardware can be shared between various applications, the resource utilization is increased. Even though the power dissipation during reconfiguration cannot be neglected, the total power consumption of the system can be decreased. It has already been shown that the system power or energy can be decreased by exploiting run-time reconfiguration capability of FPGAs with using power reduction methods such as clock scaling, clock gating, reconfigurable hardware deactivation and removal, etc. [2][3][4][5] Another advantage of partial reconfiguration is that a reconfigurable system can be updated locally or remotely at any time, so the hardware servicing can be provided continuously. In order to realize partial reconfiguration, reconfigurable architectures have to be preferred to conventional design architectures. Especially, partial reconfiguration of Field Programmable Gate Arrays (FPGA) has an important role in such design techniques. There have been different works to achieve partial reconfiguration of Xilinx FPGAs. What seems to be disregarded thus far is the ability of self-reconfiguration of low cost FPGAs and the ability of self-reconfiguration on processor independent environments. So far, most of the works related to self-reconfiguration on Xilinx FPGAs are implemented with a custom core such as MicroBlaze, PowerPC, etc. However, in

this study, we have implemented a processor-independent self-reconfiguration platform.

To achieve a safe communication between the fixed area and the reconfigurable area, there should be routing resources, which are not affected during reconfiguration process. To do this, completely fixed and static hard bus macros are preferred.

This study is an extended version of our cPCAP [1] core which controls the partial reconfiguration flow through SelectMAP/ICAP port and supplies configuration clock for reconfiguration. The adapted cPCAP core with block RAM controller, bitstream decompressor unit and ICAP FSM manages the self-reconfiguration process through ICAP interface on a Spartan-6 FPGA. cPCAP uses block RAMs (BRAMs) to store partial configuration bitstreams. BRAMs provide on-chip fast memory in FPGAs. Our custom core maximizes the utilization of BRAMs by storing compressed partial bitstreams (PBs) at initial configuration time and decompressing them during self reconfiguration. On-chip BRAM can be used as a cache memory; storing only PB(s) for the next partial configuration and removing past PB(s) from BRAM. By utilizing the cache approach, the on-chip memory storage cost can be decreased to one Nth, where N is the number of PBs used by the system. Because of being written entirely in VHDL, the cPCAP core is highly portable and can also be used for all other Xilinx FPGA architectures. Thus it is not necessary to use an external intelligent agent to control the partial reconfiguration flow. The details of design flow, generation of PBs, the way we store the PBs can be found in [1][6].

High performance image processing applications can be implemented on different platforms such as DSPs, FPGAs, analog and digital ASICs. An analog solution is the Cellular Neural Network-Universal Machine (CNN-UM) [7]. But it is an expensive system and cannot fully utilize the word length offered by todays high resolution digital cameras. The digital version of the CNN is known as Discrete Time CNN (DTCNN). It is cheaper and more flexible than the analog one due to its modular structure. The most common image processing application on a DTCNN is the 2-D filtering with a threshold [8]. An implementation of DTCNN on a Virtex-II FPGA can process more than 200 Megapixels per second [9]. By modifying the coefficients (templates), it is possible to carry out hole-filling, dilation, erosion, etc. on the same architecture [10] [11]. However, it is not proved that all image processing applications can be implemented by DTCNNs. Therefore we cannot talk about a GPIPS that consists of only DTCNNs. However, we can combine the configurability of the FPGA with the modular structure of the DTCNN to provide a high resolution GPIPS. Since the minimum partial reconfiguration time for FPGAs is still in the order of microseconds, it is obvious that switching from one application to another application will take milliseconds. Therefore, the DTCNN-based modular GPIPS architecture proposed in this paper will provide high performance during runtime of each application, but the switching time from one application to another application will be slow due to todays slow reconfiguration times.

The rest of the paper is organized as follows: In section II, studies about self-reconfigurable platforms and image processing systems on reconfigurable architectures are addressed. Section III gives brief information about Spartan-6 and it's configuration architecture. In section IV, the proposed processor-independent self-reconfiguration platform for low-cost Spartan-6 FPGA is mentioned. Section V shows some tiny self-reconfigurable examples. Section VI demonstrates an implementation of an image processing application on our self-reconfiguration platform. Finally, in section VII, we stress the fact that the proposed GPIPS platform has not been completed yet, but the results give an idea about how such a system can be implemented with partial reconfiguration.

## II. Related Works

*1) Related works on partial- (self-) reconfigurable platforms:* In [12], Koch et al. have demonstrated partial reconfigurable systems on Spartan-6 FPGAs. As the reconfiguration interface, they have used the ICAP with 16-bit Mode at 100MHz. To control the reconfiguration flow, they have used a host PC and the UART interface to access the ICAP. Since such a system does not reconfigure itself without any external engine, it is not in the class of self-reconfigurable systems. To put it clearly, a self reconfigurable system should have an internal agent/ processor/ FSM which controls the configuration flow of the FPGA device. When the reconfiguration process control engine is an external agent, then the whole system is not in the class of self-reconfigurable system but in the reconfigurable system. They have also mentioned maximum attainable reconfiguration speed (200MB/s) on Spartan-6 FPGAs, but not about the reconfiguration speed in their own system.

In [13], a self-reconfigurable system on a Virtex-4 FPGA is proposed. The preferred reconfiguration engine is the MicroBlaze soft core and the configuration interface is ICAP. The authors did not mention about the ICAP interface bit width that they have preferred in their study. Partial reconfiguration times that they obtained vary from 108ms to 460ms while partial bitstream (PB) sizes are in the range of 58KB to 244KB. Therefore the reconfiguration speed is about 0.54MB/s.

In [14] Hübner et al. proposed a fast dynamic and partial reconfiguration data path for Virtex-4 FPGAs. In their study, they have compared three different ICAP approaches (FSL-ICAP, XPS-ICAP, and pure ICAP) on the soft-core MicroBlaze and the hard core PowerPC. The maximum speed they have attained is 295.4MB/s with ICAP (32-bit, 100MHz) on PowerPC. For the PB repository, they have chosen external DDR2-SDRAM memory. This memory has different bus connections such as XCL bus and PLB bus over MPMC controller to the MicroBlaze and PowerPC respectively. The different speed values between these ICAP designs come from the bus type and the way they connect the bus to the DDR2-SDRAM memory.

François et al. have presented a fast ICAP controller on a Virtex-5 FPGA in [15]. In their study, they have reached the ICAP upper bound throughput of 800MB/s by overclocking the ICAP to 200MHz (32-bit mode). Actually there is

no specific data-sheet value for the ICAP maximum clock frequency, but it is clear that this value should never exceed the maximum clock frequency for any external configuration port (100MHz for serial/SelectMAP interface of Virtex-4 and Virtex-5 devices). Since exceeding this values may result in incorrect operation, we do not recommend overclocking the ICAP module. In addition to this, they have compared to two different ICAP approaches, $xps\_hwicap$ and $xps\_hwicap$ with direct memory access (DMA) at 100MHz. The maximum achieved ICAP bandwidth for two different approaches are 128MB/s and 174MB/s respectively.

In [16], Ming Liu et al. proposed to use DMA, Master burst (MST) and a dedicated Block RAM cache in order to reduce the reconfiguration time on a Virtex-4 FPGA. They made a comparison among different ICAP (32-bit, 100MHz) designs such as $opb\_hwicap$, $xps\_hwicap$ with their proposed architectures $dma\_hwicap$, $mst\_hwicap$ and $bram\_hwicap$. As a maximum reconfiguration speed among these approaches, they have attained 371.4MB/s for $bram\_hwicap$.

All above mentioned designs either suffer from speed or do not offer a processor-independent self-reconfiguration platform for the low-cost state-of-the art FPGAs. In addition to these, overclocking of ICAP module of an FPGA is not recommended by the device vendor. To run safely, the speed limit of ICAP is 100MHz. Therefore, the maximum throughput that can be achieved is 400MB/s (32-bit mode, not supported in Spartan-6) theoretically. Since Spartan-6 supports reconfiguration through ICAP only in 16-bit interface, the maximum theoretical reconfiguration speed through ICAP at 100MHz is 200MB/s for these devices. In our case, we propose a processor-independent self-reconfiguration platform for low-cost Spartan-6 FPGA with a safe throughput of 200MB/s (Spartan-XC6SLX45 ICAP, 16-bit mode, 100MHz).

*2) Previous Studies on general purpose image processing systems.:* Todays high performance image processing systems utilize multiple cores. In [17] a parallel architecture which has capability of reconfiguration in runtime is proposed. The architecture RAMPSOC is built on a Network-on-Chip (NoC) called "Star Wheels". To manage the hardware resources at runtime, to schedule and allocate the tasks to the individual PEs, a special purpose OS, is integrated on one of the processors. This OS is called CAP-OS and is responsible for running ICAP. It also has dedicated libraries for some image processing algorithms and communication, etc. In the user end, algorithm or code is written with C/C++, and with the toolchain of the platform it is mapped to the PEs [18]. From the top view, this architecture is more like a platform that is designed to handle software and hardware partitioning and reconfiguration.

In [19] Malki and Spaanenburg have presented a NoC based parallel image processing architecture. They discuss the effects of different types of communications to performance. Image processing algorithms of the architecture are based on Discrete Time Cellular Neural Network. They have introduced new methods to handle the boundary conditions of the image with the new architecture. The proposed architecture facilitates multiple operations on a single image, and single operations on multiple images, with access to the external image memory. This architecture is not capable of partial reconfiguration.

In [20] a mixed signal image processing platform is proposed. Image processor part is an analog Cellular Neural Network (CNN UM). It has dedicated blocks for control and dynamic reconfiguration. The architecture comprises a RAM based, a multicontext dynamically reconfigurable lookup table section and includes coarse grain logic cells targeted for synthesis programs, it also includes programmable analog blocks with configurable interconnectivity and an interface to the on-chip 8051 microprocessor core. The microprocessor can read and write both the configuration and the analog-digital internal signals during runtime. The microprocessor can switch applications (two configurations) based on dynamic reconfiguration activated by a microprocessor command.

## III. Configuration on Spartan-6 FPGAs

The Spartan-6 family is built on a 45-nm, 9-metal layer, dual-oxide process technology [21] [22]. The Spartan-6 was marketed in 2009 as a low-cost solution for automotive, wireless communications, flat-panel display and video surveillance applications [22].

The partial self-reconfiguration on a Spartan-6 FPGA can be achieved through ICAP module of the device up to 100MHz with a 16-bit interface only. Spartan-6 FPGAs do not support 8-bit and 32-bit for ICAP. Therefore, the maximum self-reconfiguration speed that can be achieved on a Spartan-6 FPGA is 200MB/s. In addition to this, the external SelectMAP configuration interface can also be used for initial or partial reconfiguration. Note that some of Spartan-6 FPGAs (e.g. XC6SLX4 devices or devices using TQG144 or CPG196 packages.) do not offer the SelectMAP interface.

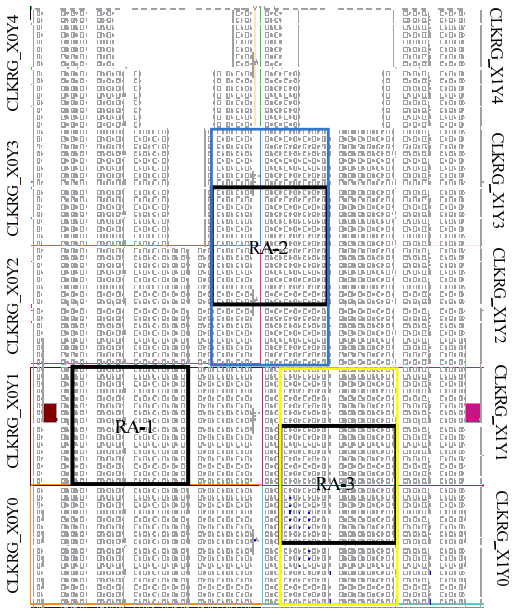


Fig. 1: Selected Reconfigurable Areas on a Spartan-6 FPGA

Opposite to the Spartan-3 FPGAs, Spartan-6 FPGAs offer the partial reconfiguration in a two dimensional fashion [12]. The atomic unit that can be reconfigured is a single frame within a clock region (in terms of CLB configuration). The configuration is in frames that covers 16 CLBs in the height. Therefore, for selecting reconfigurable area, the designer must take into account that the selected area should be within the same clock region, i.e. in the same 16-CLB-row. In Figure 1, three identical selected reconfigurable areas RA-1, RA-2, RA-3 are demonstrated on a Spartan-XC6SLX4 (the smallest member of Spartan-6 family) FPGA. The Spartan-XC6SLX4 is used only for demonstrative purposes, no actual design is implemented on this device. All implementations in this study are done on Spartan-XC6SLX45 FPGA. These areas have the same size but they are located in different portions of the FPGA. The RA-1 consists of 11CLB columns with 16-CLB-height in the same clock region. The RA-2 has also 11CLB columns with 16-CLB-height and is located in four different clock regions. The RA-3 has also 11CLB columns with 16-CLB-height and is located in two different clock regions. The generated PB for RA-1 will have 11CLB×(#Frames/CLB) frames information. However RA-2 will have 4CLB×(#Frames/CLB) frames in clock regions $X0\_Y2$ and $X0\_Y3$ separately and 7CLB×(#Frames/CLB) frames in clock regions $X1\_Y2$ and $X1\_Y3$ individually. So RA-2 will have 22CLB×(#Frames/CLB) frames information. In the same way, RA-3 will have 11CLB×(#Frames/CLB) frames in clock regions $X1\_Y0$ and $X1\_Y1$ individually which results in 22CLB×(#Frames/CLB) frames totally. Therefore, generated bitstream for RA-2 and RA-3 will be two times greater than RA-1, even if all portions does the same job. As a result, the reconfigurable area should fit into a clock region. Otherwise the number of frames included into the bitstream information will increase and the size of PB will be greater than expected. When the logic occupation in a clock region is not enough, then the neighbor clock regions can also be utilized.

## IV. Self-Reconfiguration Platform

Our reconfiguration engine cPCAP [1] can use either SelectMAP or ICAP interface in any bit-width combination (i.e. 8, 16, 32-bit). We have adapted cPCAP core to run on Spartan- XC6SLX45 FPGA with configuration interface ICAP 16-bit mode at 100MHz. Since cPCAP core is entirely written in VHDL, it can be adapted to any other Xilinx FPGA to control the reconfiguration flow. As there is no SelectMAP interface on Spartan-XC6SLX4 FPGA devices (i.e. XC6SLX4, XC6SLX45, XC6SLX45T), we have used ICAP 16-bit mode with cPCAP core.

As shown in Figure 2, cPCAP core reads compressed PBs from on-chip BlockRAM and decompress them on-the-fly at the time of reconfiguration. Therefore there is no downtime during decompression. As we use block RAMs to store compressed PBs and do not use any custom bus type, we have achieved the maximum throughput attainable by ICAP, 200MB/s (16-bit mode, 100MHz). The compression of PBs is done at design time. The method used for compression is the run-length encoding and does not require a complex decompression process. The architecture details of cPCAP and the compression/ decompression mechanisms can be found in [1] and [6].

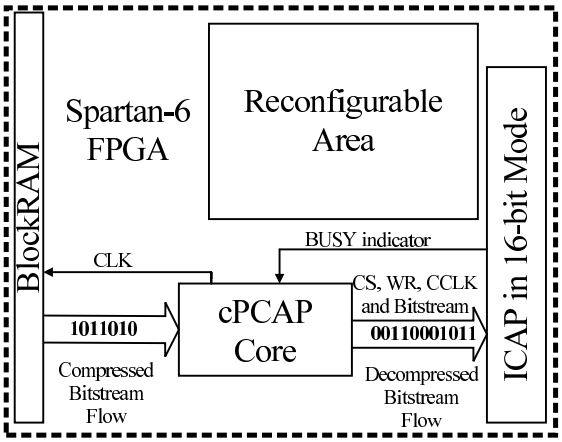


Fig. 2: Hardware architecture of the system over internal configuration port (ICAP) on a Spartan-6 FPGA

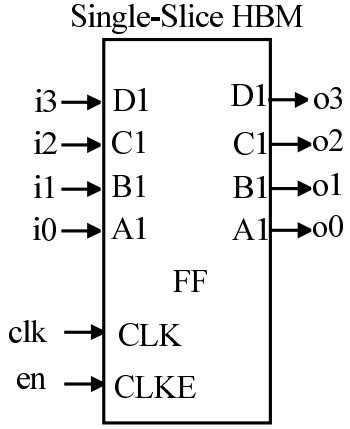


Fig. 3: 4-Bit Single Slice Spartan-6 Hard Bus Macro (HBM)

For the glitchless data flow between fixed area and reconfigurable area, we have used 4-bit slice based hard bus macros. Actually, there are no slice based hard bus macros offered by Xilinx for Spartan-6 FPGAs. However, there are some bus macros which are directly used for some target FPGA families (e.g. Virtex-5, Virtex-6) from Xilinx. Instead of designing a new bus macro we have manipulated and adapted Virtex-5 bus macros to the Spartan-6 bus macros. The 4-bit-width single slice hard bus macro component is illustrated in Figure 3. In Figure 4, the internal structure and connections of our single slice synchronous Spartan-6 bus macro can be seen.

## V. Test Results

At first we have implemented our cPCAP core for dynamically reconfiguring a few small sample circuits. The first one we have tested is a 4-bit forward counter with different speeds.

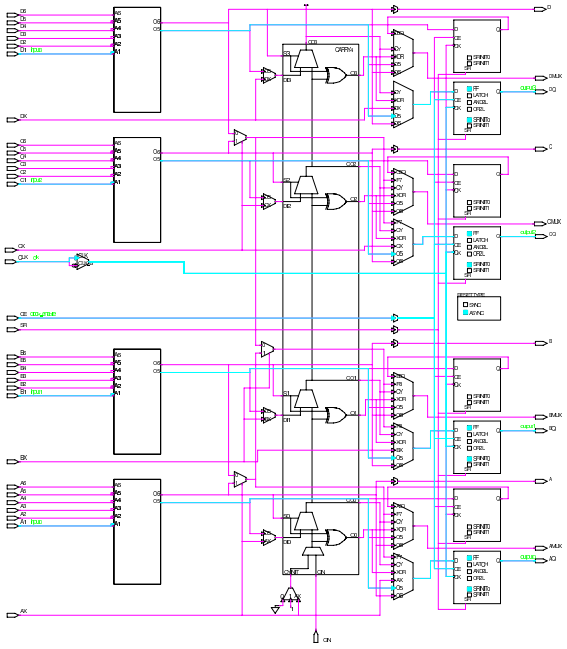

Fig. 4: Inside of Spartan-6 Slice Based Hard Bus Macro

To switch between different speeds we have reconfigured the clock input of the forward counter. According to this small example, there are 4 different speeds: 1Hz, 5Hz, 10Hz and 20Hz ( to be observable by human eye) respectively. We have written a process to obtain such small clock periods. For the reconfiguration view, we have actually changed the trigger clock of this process. To be able to have different clock values, we have generated different clock values from digital clock manager (DCM) of Spartan-6 FPGA and changed these values at runtime by reconfiguring the counter circuit through ICAP under the control of cPCAP core. There are various applications, where clock frequency of a system should be changed at run-time. Clock scaling method in [2] is one of them, which is mostly used to decrease FPGA power consumption by changing clock frequency of different components of system at run-time. Adjusting data transmission speed of a system, and other typical applications include speed drives, inverters, computers and computer controlled equipment, deep well pumps, industrial machinery, ships, aircraft. Actually, we are aware of the dynamic reconfiguration port (DRP) to reconfigure DCM outputs. But we just wanted to verify the correctness of our reconfiguration platform on a Spartan-6 FPGA. In this example, each PB is composed of only one frame. In Table I, the original, compressed PB sizes and the reconfiguration times are available. In this example, we have achieved to put four different compressed PBs in a single block RAM. We have only used two Bytes (two consecutive 00 bytes)separators between each compressed PBs (total overhead is six Bytes). Note that, we have not excluded header information from original PB files generated by bitgen tool of Xilinx; before the compression process, we directly applied run length encoding compression technique to the PB files at design time.

TABLE I: PB Sizes and Reconfiguration Times for Forward Counter Example

| PB | Original PB Size (Bytes) | Compressed PB Size (Bytes) | # of BRAMs | Space Savings | Reconfiguration Time$[\mu s]$ |
|---|---|---|---|---|---|
| P1 (1Hz) | 473 | 401 | 0.196 | 15.22% | 2.365 |
| P2 (5Hz) | 473 | 401 | 0.196 | 15.22% | 2.365 |
| P3 (10Hz) | 473 | 401 | 0.196 | 15.22% | 2.365 |
| P4 (20Hz) | 473 | 401 | 0.196 | 15.22% | 2.365 |
| Total | 1892 | 1604 | 0.783 | 15.22% | 9.46 |

The second small example is used to test the functionality of our hard bus macros. In this tiny example, we have tested two different small logical circuits, which gives different four-bit output values to the on-board LEDs. In this example, each PB is a composition of 2 frames. In Table II, the original, compressed PB sizes and the reconfiguration times are available.

TABLE II: PB Sizes and Reconfiguration Times for Hard Bus Macro Tester Example

| PB | Original PB Size (Bytes) | Compressed PB Size (Bytes) | # of BRAMs | Space Savings | Reconfiguration Time$[\mu s]$ |
|---|---|---|---|---|---|
| P1 | 603 | 349 | 0.170 | 42.12% | 3.015 |
| P2 | 603 | 348 | 0.169 | 42.29% | 3.015 |
| Total | 1206 | 697 | 0.340 | 42.2% | 6.03 |

## VI. The Low-Cost GPIPS: An Introduction

The proposed reconfiguration core can be used to change the tasks (modes) of a Reconfigurable Image Processing Element (RPE). The GPIPS consists of multiple RPEs connected with a network similar to that of a DTCNN. An RPE ( Fig. 5) consists of a static and a reconfigurable part. The reconfigurable part of the RPE is just its arithmetic unit and the rest is static. Register file is a module that stores temporary values for calculations in the RPE. Template coefficients memory stores the template coefficients for DTCNN which is explained below. The information of which template is to be used is received from *temp-no*. *Control input* is the signal which is sent by a global control unit. *in-data* and *in-address* buses carry the data and source address of the data. *out-data* and *out-address* carry the processed data and destination address of the data. Both addresses and data are received from and sent to the communication network. It may be wondered why communication network is not in the reconfigurable part. In many image processing applications, a group of pixels in a specified block (neighborhood) are computed. Providing a 2-D communication network between RPEs usually makes it sufficient to reconfigure just arithmetic unit on the RPEs. Obviously, local control block has to manage all the cases that RPE requires. In this study, local control manages block-match and DTCNN.

Once the static part of the RPE is determined and fixed, the whole algorithmic design process reduces to the design of the

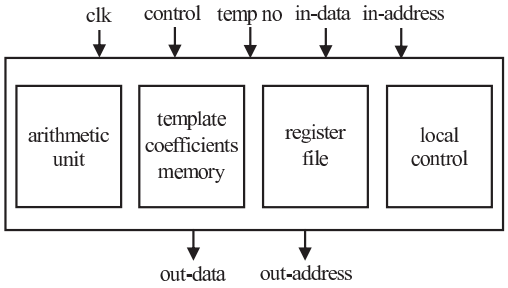


Fig. 5: Ports and sub-blocks of a RPE

arithmetic unit of the RPE and writing the necessary code for each application. In this part, we will try to explain how the arithmetic unit of an RPE can be designed to switch between block matching and any application that can be done with DTCNN on-the-fly. The code development is out-of-the-scope of this work.

To compute overall gain in the area occupation, the cost of one RPE area occupation has to be multiplied by the number of RPEs in the whole network. Similarly, the reconfiguration time of the RPEs in the network has to be calculated by multiplying reconfiguration time of one RPE by the number of PEs. Due to ongoing works of our project, in this paper total reconfiguration time is given in that manner.

Image processing algorithms can be implemented in several ways. Due to changes in standards and needs of systems, several algorithms and platforms have to be used in image processing applications. Cellular Neural Network was invented in 1988 by Chua-Yang [23]. After that this paradigm was realized on an analog chip with on-board peripherals and given a name CNN-UM [7]. It was shown that due to its parallel analog structure it could process image in a short time regardless of the image dimensions. It is hard to implement an analog chip in large resolutions, so researchers have tried to emulate analog CNN by Discrete Time Cellular Neural Network (DTCNN) digitally. Digital implementations of DTCNN are slower than the analog one but are still faster than so many platforms because of its parallel structure. A DTCNN module has to realize the following equation:

$$x\left(k+1\right)=\sum a_d y^d\left(k\right)+\sum b_d u^d\left(k\right)+i; d\in N_r \quad (1)$$

where, $x\left(k+1\right)$ is next state, $y^d\left(k\right)$ are output (processed) values of the image pixels in the neighborhood, $a_d$, $b_d$ are the elements of templates A and B, $u^d(k)$ is input image pixel, $i$ is the threshold value [19]. In literature there are 1-D [24],[25] and 2-D [19], [26] implementations of this mathematical formula. In Fig. 6 how template dot product, e.g ($\sum a_d y^d$), is computed is shown.

Regardless of the placement of the modules, arithmetic modules are similar in the processing elements of different implementations. Basically, arithmetic units of the processing elements have to perform multiplication and cumulative addition. Fig. 7, shows arithmetic unit that is used in RPEs of

$$\sum a_d y^d = \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{bmatrix} \otimes \begin{bmatrix} y_0 & y_1 & y_2 \\ y_3 & y_4 & y_5 \\ y_6 & y_7 & y_8 \end{bmatrix}$$
$$= a_0y_0 + a_1y_1 + a_2y_2 + a_3y_3 + a_4y_4 + a_5y_5 + a_6y_6 + a_7y_7 + a_8y_8$$

Fig. 6: Dot product computation

DTCNN in this work.

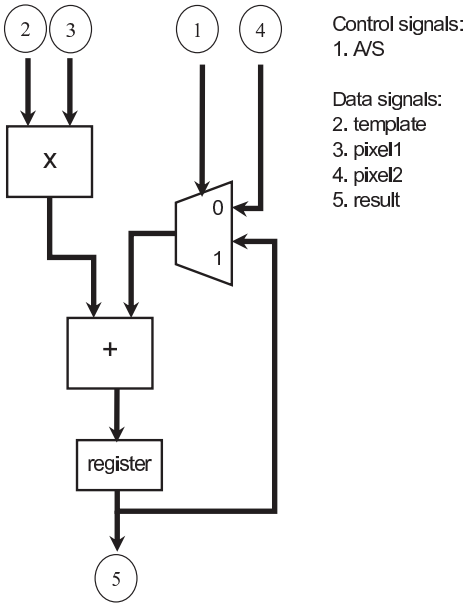


Fig. 7: Arithmetic unit of a RPE for DTCNN

*Template* and *pixel1* is multiplied and the product is summed with the partial sum (PS) or *pixel2*. PS/B signal selects whether *partial sum* or *pixel2* is added. "partial sum" is the output of "register" in Fig. 7 and it's the "result" after the computation of Eq. 1 is completed. Instructions that drive the arithmetic unit for DTCNN are written below with a pseudo code, and concurrent instructions are listed in a parenthesis with the word **concurrent**. *reg1*, *reg2* are the registers in register file of a RPE, in-data is a port of RPE which receives input data, and *tempElem(i)* is a template value (coefficient) stored in *template coefficient memory*. *register* is used as an accumulator. *i* is a counter that counts the number of pixels in the neighbourhood that are computed in one iteration of a pixel.

// i = 0, register = 0
**move** *reg1, in-data*; //store in register file
**move** *reg2, in-data;*
**concurrent** {**move** *pixel2*, *reg1*;
**move** *pixel1*, *reg2*;
**move** *template*, *tempElem(i)*;
**move** *PSB, 1*; **add** *i, 1*; }
*Ntimes*: //statement below is operated N times
**concurrent** {**move** *pixel1, in-data*;
**move** *template, tempElem(i)*;
**move** *PSB, 0*; **add** *i, 1*; }

//N +1: number of pixels calculated in one DTCNN iteration.

Inherently, DTCNN performs 2-D filtering with a threshold. Besides, one can do hole-filling, dilation, erosion, etc. [7]. Image processing algorithms are not limited with the algorithms that DTCNN can do. For instance, block-matching is a widely used image processing algorithm that cannot be done just by DTCNN. Block-matching is generally implemented on hardware due to its computational intensity. If DTCNN and block-matching algorithms are desired to be performed on an image processing platform, they have to be implemented separately or they can share the same area by being reconfigured as we show in this study. Mathematical model of block-matching is as follows:

$$s\left(m,n\right)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|x\left(i,j\right)-y\left(i+m,j+n\right)\right|; \quad (2)$$

where x(i,j) and y(i,j) correspond to pixel values in the reference block in the current frame and the candidate block in the search window in the previous frame, respectively. Addition and subtraction of pixels in candidate and reference blocks are calculated and the minimum one among the results of all blocks shows the motion vector. Fig. 8 shows how block-matching is done and motion vector is calculated for two sequential frames.

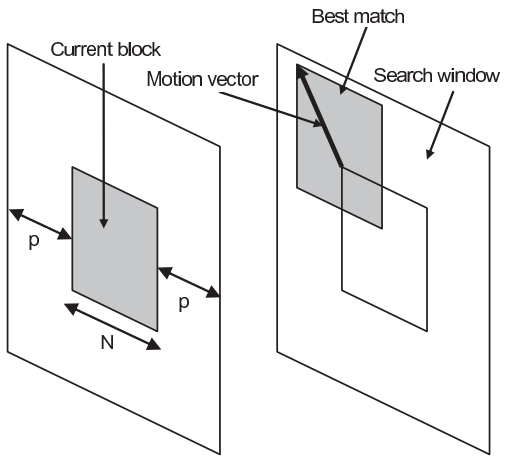


Fig. 8: Block-matching and computation of motion vector

Block-matching can be done in several ways, such as systolic arrays in 1-D and 2-D. The main arithmetic computations in a block-match RPE are addition, subtraction and absolute value. In Fig. 9 a block-match arithmetic unit is shown. A/S signal selects whether addition or subtraction of two pixels are computed.

Instructions that drive the arithmetic unit for block-matching are written with a pseudo code, and concurrent instructions are listed in a parenthesis with the word **concurrent**. The symbol || used in the Fig. 9 is used to show an absolute value unit.

**move** *reg1, in-data*; //store in register file
**move** *reg2, in-data*;

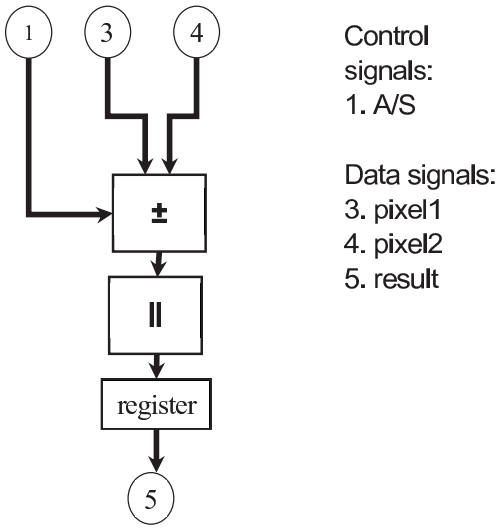


Fig. 9: Arithmetic unit of a RPE for block-matching

**concurrent** {**move** *pixel2, reg1*;
**move** *pixel1, reg2*;
**move** *AS, 1*; } //subtraction case
**move** *reg2, result*;
**move** *reg1, indata*;
**concurrent** {**move** *pixel2, reg1*;
**move** *pixel1, reg2*; **move** *AS, 0*; }
//addition case

In this study, arithmetic unit of a RPE has been partially reconfigured as DTCNN and block-matching and all other blocks remain unchanged. Note that, local control has been designed to be able to manage both DTCNN and block-matching algorithms. Although both block-matching and DTCNN can be implemented on 1-D or 2-D processor arrays, it is better to use same allocation for both ones to decrease design complexity and reconfiguration file size.

### *A. Implementation Results*

We have implemented only the arithmetic unit of the RPE. This is sufficient to estimate the latency for switching from one application to another application. In Table III, reconfiguration cost summary of a single RPE is shown. To reconfigure a single RPE, the required reconfiguration time is about 246 microseconds when ICAP-16 bit interface at 100MHz is used. Here, it is worth to mention that this reconfiguration time is valid when the reconfigurable module is located within a single clock region (the first two rows in Table III). More precisely, this is the case where the arithmetic unit of the RPE resides on the FPGA as RA-1 does in Fig. 1. As it can be seen obviously from Table III, a huge difference occurs in the size of partial bitstreams (thereby reconfiguration time) when the same arithmetic unit of the RPE is placed on the boundaries of two clock regions like RA-3. Since DTCNN contains a multiplier block and there is no DSP block column on Spartan-XC6SLX45 in the middle of FPGA, we did not test the case, where the reconfigurable unit resides on 4 different

clock regions as RA-2 does in Fig. 1. But, we are aware that it also would give the similar result like RA-3. The inputs 3 and 4 for both configurations of RPE are set to 18-bits and 36-bits respectively. In addition to these inputs, the input 2, which is used only in DTCNN configuration, is also set to 36-bits. The width of output (5) is the same for both configuration and set to 36-bits. The total reconfiguration time with multiple RPEs is calculated by multiplying reconfiguration time of a single RPE by the number of RPEs in the whole system. Assume that there exists 100 RPEs in the system; so switching from one configuration (Block-Matching/DTCNN) to other configuration (DTCNN/Block-Matching) will take about 24.6 milliseconds. Indeed, we are aware of the reconfiguration overhead of switching from one configuration mode to another one. However, the switching operation does not occur in the middle of an running algorithm, it takes place on-demand.

TABLE III: PB Sizes and Reconfiguration Times for a single RPE

| Nr. of CLK Regions | Partial Bitstream | Nr. Of Frames | Original PB Size (Bytes) | Compressed PB Size (Bytes) | # of BRAMs | Space Savings | Reconfiguration Time[$\mu s$] |
|---|---|---|---|---|---|---|---|
| 1 | P1 (Blockmatch) | 299 | 49299 | 15604 | 7.62 | 68.35% | 246 |
| 1 | P2 (DTCNN) | 299 | 49294 | 17490 | 8.54 | 64.52% | 246 |
| 2 | P1 (Blockmatch) | 406 | 66432 | 19111 | 9.33 | 71.23% | 331 |
| 2 | P2 (DTCNN) | 406 | 66427 | 21501 | 10.5 | 67.63% | 331 |

## VII. Conclusions and Future Work

In this study, we have demonstrated a reconfiguration engine which allows a designer to do self-reconfiguration on Xilinx Spartan-6 FPGAs at run time through ICAP interface (16-bit) at 100MHz. The proposed engine was already used in Spartan-3 and Virtex-4 previously. In this study, we have adapted our cPCAP core to be able to run on Spartan-6 FPGAs.

As in previous cPCAP design, we have used block RAM for the compressed PB storage. The PBs, which are compressed at design time, are stored on block RAM and decompressed on-the-fly during reconfiguration. While they are being decompressed, there is no down-time in the reconfiguration process.

As a future work, we are planning to reconfigure not only a RPE but the multiple set of RPEs (hundreds of) in the whole design. We are also planning to enlarge the number of algorithms of the RPE and to give the designer a software to be able to develop his/her project easily. Thus, we will be able to achieve switching two different configurations (DTCNN, Block-matching, etc) of a general purpose image processing system on-demand in approximately 25 milliseconds.

## Acknowledgment

This work is fully supported by Boğaziçi University Scientific Research Projects (Project Nr.:5578) and State Planning Organization of Turkey, DPT under grant no 2007K120610.

## References

[1] S. Bayar and A. Yurdakul, "*Self-Reconfiguration on Spartan-III FPGAs with Compressed Partial Bitstreams via a Parallel Configuration Access Port (cPCAP) Core*," PRIME2008, June 2008.

[2] K. Paulsson, M. Hübner, S. Bayar, and J. Becker, "*Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems*," ReCoSoc2007, Montpellier, France, June 2007.

[3] S. Liu, R. N. Pittman, and A. Forin, " *Energy Reduction with Run-Time Partial Reconfiguration*." Technical Report of Microsoft Research: MSR-TR-2009- 2017, September 2009.

[4] W. Osborne, W. Luk, J. Coutinho, and O. Mencer, "Energy reduction by systematic run-time reconfigurable hardware deactivation," *Transactions on HiPEAC*, vol. 4, no. 4, 2009.

[5] M. Lorenz, L. Mengibar, M. Garca-Valderas, and L. Entrena, "emph-Power Consumption Reduction Through Dynamic Reconfiguration," In Proceedings of FPL, 2004, pp. 751–760.

[6] S. Bayar and A.Yurdakul, "An efficient self-reconfiguration core for run-time reconfigurable fpga interconnects," 2011, journal of Systems Architecture, JSA' 2011 (revision).

[7] T. Roska and L. O. Chua, "*The CNN Universal Machine: an analogic array computer*," *IEEE Trans. on Circuits and Systems-II*, vol. 40, pp. 163–173, 1993.

[8] L. O. Chua and T. Roska, *Cellular Neural Networs and Visual Computing*. Cambridge, UK: Cambridge University Press, 2002.

[9] S. Malki and L. Spaanenburg, "*CNN Image Processing on a Xilinx Virtex-II 6000*," vol. 3, Proceedings ECCTD'03, 2003, pp. 261–264.

[10] K. Murugesan and P. Elango, "emphCNN based Hole filler template design using numerical integration techniques," ICANN'07 Proceedings of the 17th international conference on Artificial neural networks, 2007.

[11] A. Zarandy, A. Stoffels, T. Roska, and L. Chua, "Implementation of binary and gray-scale mathematical morphology on the cnn universal machine," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 45, no. 2, pp. 163–168, February 1998.

[12] D. Koch, C. Beckhoff, and J. Torrison, "*Advanced Partial Run-time Reconfiguration on Spartan-6 FPGAs*," FPT'10, December 2010.

[13] S. U. Bhandari, S. S. S. Pujari, and R. Mahajan, "*Internal dynamic partial reconfiguration for real time signal processing on FPGA*," *Indian Journal of Science and Technology*, vol. 3, no. 4, pp. 365–368, 2010.

[14] M. Hübner, Diana, Göhringer, J. Noguera, and J. Becker, "*Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs*," IPDPS'10, 2010.

[15] F. Duhem, F. Muller, and P. Lorenzini, "*FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA*," ARC'2011, 2011.

[16] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "*Run-Time Partial Reconfiguration Speed Investigation And Architectural Design Space Exploration*," In Proc. of the International Conference on Field Programmable Logic and Applications, August 2009.

[17] D. Göhringer and J. Becker, "*High performance reconfigurable multi-processor-based computing on FPGAs*," IPDPSW '10, 2010, pp. 1–4.

[18] D. Göhringer, M. Birk, M. Hübner, and J. Becker, "*High-Level Design for FPGA-based Multiprocessor Accelerators*," Workshop on Design Methods and Tools for FPGA-Based Acceleration of Scientific Computing (DATE'2011), 2011.

[19] S. Malki and L.Spaanenburg, "*A CNN-Specific Integrated Processor*," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, p. 1, 2009.

[20] R. Montufar-Chaveznava, D. Guinea, M. C. Garcia-alegre, and V. M. Preciad, "*CNN computer for high-speed visual inspection*," Proceedings SPIE, 2001, pp. 236– 243.

[21] P. Clarke, "*Xilinx launches Spartan-6, Virtex-6 FPGAs*," EETimes, 2 February 2009.

[22] Xilinx, "*The low-cost Spartan-6 FPGA family delivers an optimal balance of low risk, low cost, low power, and high performance*," Company Release, 2009.

[23] L. O. Chua and L. Yang, "Cellular neural networks: Theory and applications," *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1257–1290, 1988.

[24] M. Tükel and M. E. Yalcin, "*A new architecture for Cellular Neural Network on reconfigurable hardware with an advance memory allocation method*," 12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA'2010), 2010.

[25] K. Kayaer and V. Tavsanaoglu, "*A new approach to emulate CNN on FPGAs for real time video processing*," 10th. IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings (CNNA2008), 2008, pp. 23–28.

[26] Z. Voroshazi, Z. Nagy, A. Kiss, and P. Szolgay, "*Implementation of embedded emulated-digital CNN-UM global analogic programming unit on FPGA and its application*," *International Journal of Circuit Theory and Applications*, vol. 36, pp. 589–603, 2008.