PARTIALLY RECONFIGURABLE FPGA IMPLEMENTATION OF A REAL-TIME SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BENGİSU TENGİLİMOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2014

Approval of the thesis:

PARTIALLY RECONFIGURABLE FPGA IMPLEMENTATION OF A REAL-TIME SYSTEM

submitted by **BENGİSU TENGİLİMOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof.Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU _____

Dr. Fatih Say
ASELSAN Inc. _____

Date: **20/06/2014**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Bengisu TENGİLİMOĞLU

Signature :

ABSTRACT

PARTIALLY RECONFIGURABLE FPGA IMPLEMENTATION OF A REAL-TIME SYSTEM

Tengilimoğlu, Bengisu

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı

June 2014, 60 pages

The use of reconfigurable logic has increased in different kinds of applications during the last decades. Compared to other logic solutions reconfigurable logic has advantages such as easy update and lower time to market. The concept of partial reconfiguration in Field Programmable Gate Arrays (FPGAs), which are the most commonly used reconfigurable logic, has recently been introduced by leading FPGA vendors. Partial reconfiguration is a technique that allows reconfiguring a specific part of an FPGA during runtime. This method allows switching between design modules that are not necessary to function at the same time without interrupting the FPGA's processing of the current task and therefore larger applications may be implemented.

This thesis study mostly concentrates on partially reconfigurable architectures. Properties of partially reconfigurable architectures and systems are examined. As a case study, partial reconfiguration has been implemented on an FPGA, which is used as a hardware accelerator in a real-time target detection and tracking system. Using partial reconfiguration, switching between different algorithm components takes place during runtime without interrupting the object-tracking capability. By this approach, FPGA resources are used efficiently and power consumption is reduced.

Keywords: Runtime Partial Reconfiguration, Partial Reconfiguration, Target Tracking and Detection, Video Processing

ÖZ

GERÇEK ZAMANLI BİR SİSTEMİN KISMİ YENİDEN YAPILANDIRILABİLİR FPGA İLE GERÇEKLEŞTİRİLMESİ

Tengilimoğlu, Bengisu
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Cüneyt F. Bazlamaçcı

Haziran 2014, 60 sayfa

Son yıllarda, farklı uygulamalarda yeniden yapılandırılabilir mimari kullanımı artmıştır. Diğer çözümler ile karşılaştırıldığında yeniden yapılandırılabilir mimari kullanımının kolayca güncellenebilme ve daha kısa zamanda piyasaya sürülebilme gibi avantajları bulunmaktadır. En çok kullanılan yeniden yapılandırılabilir mimari olan Alan Programlanabilir Kapı Dizinleri'ne (FPGA), başlıca FPGA üreticileri tarafından kısmi yeniden-yapılandırma özelliği getirilmiştir. Kısmi yeniden-yapılandırma, çalışır durumda olan FPGA'nın belirli bir bölümünün yeniden programlanmasına / yapılandırılmasına olanak sağlayan bir tekniktir. Bu yöntem sayesinde aynı anda çalışması gerekmeyen FPGA uygulamaları arasında FPGA'nın çalışması kesilmeden geçiş yapılabilir ve aynı FPGA ile daha büyük uygulamalar gerçeklenebilir.

Bu tez çalışması yeniden yapılandırılabilir mimarileri üzerine yoğunlaşmıştır. Yeniden yapılandırılabilir mimari özellikleri ve sistemleri incelenmiştir. Örnek olarak, gerçek zamanlı bir hedef takip ve tespit sisteminde donanımsal hızlandırıcı olarak kullanılan FPGA'da kısmi yeniden-yapılandırma tekniğinin uygulanması gerçekleştirilmiştir. Sistemde, kısmi yeniden-yapılandırma kullanılarak hedef tespit

ve takip işlemi kesilmeksizin (görüntü karesi kaçırılmaksızın) farklı algoritma bileşenleri arasında geçiş yapılmıştır. Bu yöntem ile FPGA özkaynakları daha etkin kullanılırken aynı zamanda güç tüketimi azaltılmaya çalışılmıştır.

Anahtar Kelimeler: Gerçek Zamanlı Kısmi Yeniden-Programlama, Kısmi Yeniden-Programlama, Hedef Takip ve Tespit, Görüntü İşleme

To my family

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my father Yakup Tengilimoğlu and my mother Leyla Tengilimoğlu for their love, support and patience over the years.

I sincerely thank my supervisor Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı for all his guidance and support throughout my study.

I would like to thank to my employer, ASELSAN for supporting me.

I also express my gratitude to TÜBİTAK BİDEB "National Scholarship Program for MSc Students".

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BLE	Basic Logic Elements
BRAM	Block Ram
CLB	Configurable Logic Block
CPU	Central Processing Unit
DSP	Digital Signal Processor
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPL	Field Programmable Logic
FSM	Finite State Machine
GPP	General Purpose Processor
HDL	Hardware Description Languages
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
LUT	Look Up Table
Non-PR	Non-Partial Reconfigurable
OS	Operating System

PAL	Phase Alternating Line
PCIe	Peripheral Component Interface Express
PL	Programmable Logic
PLB	Processor Local Bus
PR	Partial Reconfiguration
QDR	Quad Data Rate
RAM	Random Access Memory
RM	Reconfigurable Module
RP	Reconfigurable Partition
SRAM	Static Ram
VHDL	Very High-speed Integrated Circuit Hardware Design Language
VTU	Video Tracking Unit

CHAPTER 1

INTRODUCTION

Designers of embedded systems can choose among a variety of hardware platforms. The open question then is which hardware platform is suited best for a certain task. Each platform has its own advantages and disadvantages. Today many applications exist in which both system performance and design cost are critical, so for designers the choice of the right hardware platform for their applications is not always a simple task since they consider many parameters such as time to market, performance, cost, development ease, power, feature flexibility, etc.

The most common classical hardware platform is the general purpose processor (GPP), which is generally based on Von-Neumann Model in which the behavior of the digital circuit is controlled by a sequence of instructions that defines the data processing and control operations. GPP based computation has advantages such as design flexibility, architecture compatibility and high level application programming. However, execution time is bounded by a limited cycle time due to sequential processing of the instructions. Using the same generic instruction set for all types of algorithms may also be regarded as a limitation. Therefore, a need for an alternative solution for computing intensive applications exists.

In Application Specific Integrated Circuits (ASIC), the operations are realized with a fixed digital circuit. ASICs are therefore low unit cost and high performance devices. Besides its very high initial development cost, its static nature is a major

drawback of the technology. After it is manufactured, the functionality of an ASIC cannot be changed. If the application changes, a new ASIC has to be designed. FPGAs are used as a fast prototype and low cost replacement of ASICs with a high degree of re-programmability. As depicted in Figure 1.1, FPGAs provide an intermediate solution between GPP and ASIC in terms of design cost and performance for embedded systems.

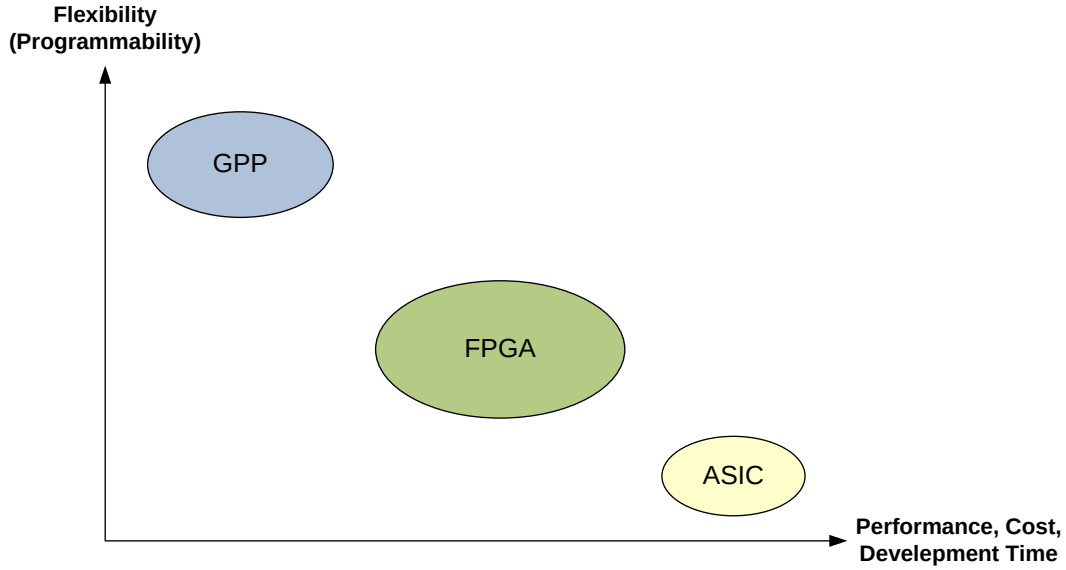


Figure 1.1: Comparison of GPP, ASIC, and programmable hardware

FPGAs belong to Programmable Logic (PL) family. Main advantages of this family are programmability of hardware. Some FPGAs can provide additional flexibility and hardware by supporting partial reconfiguration feature. With this feature, it is possible to rearrange a part of a running FPGA, called as partial reconfigurable area, without affecting the operation of other static areas. Partial reconfiguration is done using a bit stream of the components created by firmware to perform the required algorithm. The logic in an FPGA design is generally divided into two different parts, namely reconfigurable and static as illustrated in Figure 1.2.

As an example, FPGA-fabric may initially contain A1 function at the reconfigurable part (A1.bit is loaded to the reconfigurable block). At some point of execution A2 function may be needed so reconfigurable logic can be replaced by the contents of the partial bit file A2 while static logic part keeps functioning and is completely unaffected by the loading of a new partial bit file. In this scenario it is worth noting that static part contains logic that are always needed for the user application and A1 and A2 are modules that are not necessary to function all the time and at the same time. Using this approach, FPGA resources can be used in a time-division multiplexed way and an application can be built using a smaller FPGA. This may result in a reduction in the overall cost and power consumption and may provide a shorter overall configuration time.

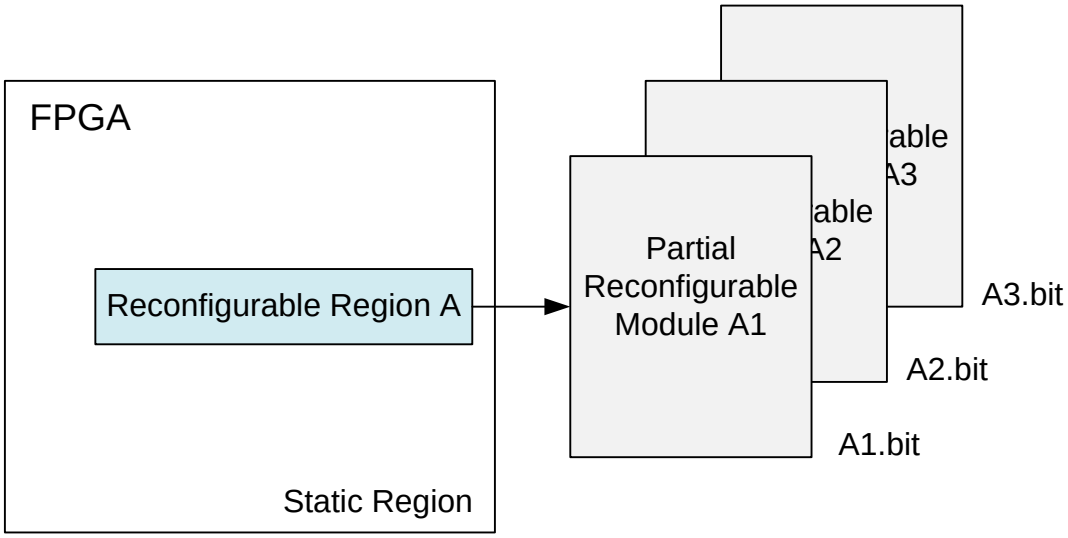


Figure 1.2: Partial Reconfiguration

In the above paragraph, it is stated that there should be time-multiplexing among partially reconfigurable modules. This is not the only requirement for a partially reconfigurable design. To take advantage of partial reconfiguration, one should

analyze design specifications, consider requirements and limitations associated with partially reconfigurable systems. This may simplify both the design and debugging processes and avoid potential future risks of malfunctioning. System level analysis is also a must.

From a system analysis point of view, the following is a set of sample questions that should be considered before using partial reconfiguration concept.

- Is there a signal or input to FPGA that can be used for the triggering / initialization of partial reconfiguration process?
- Is there a proper internal or external memory for the storage of partial bit files?
- How and when a partial bit file will be transferred to the partial bit file storage memory?
- During partial reconfiguration process, how will a partial bit file be transferred to the FPGA's configuration memory?
- Which configuration access port will be used?
- Is partial reconfiguration time overhead critical in the system?

1.1.1 Scope of the Thesis

The present research work is conducted as part of an ongoing process in a video tracking project at Aselsan Inc.

The Video Tracking Unit (VTU) being developed by Aselsan Inc., Defense and Systems Technologies is a real-time target detection and tracking system. VTU consists of five sections; cameras, monitor, host-PC, FPGA and the processor. In

VTU, FPGA is used as a hardware accelerator and it implements a series of operations that are time-consuming to perform in the processor. Within the operations that are implemented in FPGA, there exist tracking and detection components, which are not required to run simultaneously. By employing partial reconfiguration feature of Xilinx Virtex-5 XC5VFX70T FPGA, we aim to reduce power consumption and increase the effective use of FPGA resources for possible future additional functions and add more flexibility by switching among components at run-time.

The present work focuses on the reconfigurable parts of the VTU. As part of this work, use of partial reconfiguration, partial reconfigurable system design in Xilinx tools and performance evaluation techniques of partially reconfigurable systems has also been investigated. To sum up, this thesis evaluates the feasibility of using partial reconfiguration especially in video processing systems using VTU system developed at Aselsan Inc. as a case study.

1.1.2 Motivation and Contribution

In embedded systems development, efficient use of silicon area and minimization of energy consumption are two important challenges. Compared to software execution, hardware accelerated tasks can reduce energy consumption of several orders of magnitude, but since they are always active, these tasks use silicon area and consume power even when they are unused. Dynamic Partial Reconfiguration (DPR) feature of FPGAs brings an interesting solution to this problem by allowing to share a part of silicon area between different applications and thus it brings the opportunity for reducing power consumption.

The motivation behind this thesis work is to lower system power and cost by using dynamic partial reconfiguration feature of FPGA used in Video Tracking Unit as well as to use FPGA resources effectively. Also, partially reconfigurable architecture provides more flexibility to our system. Because there are modules implemented on FPGA accelerator that we may want to change according to the changing environment in the future. Another motivation of this work is, while gaining experience on FPGA new feature and its software tools, to provide FPGA programmers an evaluation of partial reconfiguration tools and techniques for a real-time system.

As for contribution, for realization of a proper non-interrupted partially reconfigurable tracking system, a fast ICAP controller was implemented with reasonably low resource usage. Partial reconfiguration was done in a limited reconfiguration time for a real-time system realization.

1.1.3 Thesis Organization

The rest of the thesis is organized as follows. First, in Chapter 2, a brief introduction to reconfigurable computing is given. Xilinx FPGAs and partial reconfiguration feature in Xilinx FPGAs are discussed. This chapter also includes application areas of reconfigurable architectures. Chapter 3 presents an extensive literature survey of existing related video processing applications on reconfigurable computing platforms and use of partial reconfiguration on computer vision systems. Starting at Chapter 4, Video Tracking Unit (VTU) is introduced and it is explained that how it is made a partially reconfigurable system. Architectural details of the implemented reconfigurable VTU system is given. Also implementation steps of partial reconfiguration process on the employed FPGA are presented in Chapter 4. In

Chapter 5, the implementation results are shared. Logic requirements of our management circuitry are given and the performance of our final system is evaluated in terms of power consumption and resource utilization with a comparison to non-reconfigurable system. The thesis is summarized and concluded in Chapter 6, which also presents observations made throughout the study and some future works.

CHAPTER 2

BACKGROUND

2.1 RECONFIGURABLE COMPUTING

Today, many applications have very demanding requirements on computation performance, power consumption and flexibility. Some hybrid FPGA/CPU computers are offered by vendors [1]. Other reconfigurable computing architectures target multimedia applications [2] [3]. Reconfigurable computing combines the flexibility of a microprocessor with the performance of application specific hardware. A reconfigurable computing fabric consists of a flexible array of configurable computational units and a configurable interconnect as illustrated in Figure 2.1. The fabric can be configured to function like an application specific hardware. The high performance and power efficiency of reconfigurable computing come from custom datapath configurations. The datapath is built in such that there is no sequential flow like in Von-Neumann model. The high degree of parallelism can be realized by pipelining.

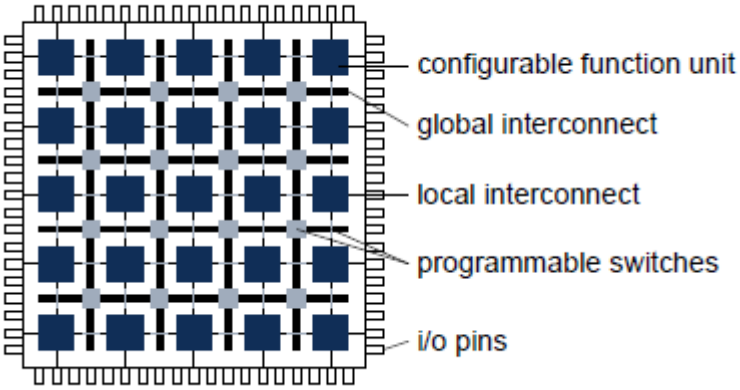


Figure 2.1: A general FPGA architecture.

2.2 CLASSIFICATION OF RECONFIGURABLE ARCHITECTURES

In [4] [5], current reconfigurable architectures are classified as described below. It is worth noting that a reconfigurable architecture may have more than one of these properties.

2.2.1 Partial Reconfiguration (PR)

Partial reconfiguration is reconfiguration of a subset of the device logic. Partial reconfiguration can be achieved by using the configuration interface that allows writing new configuration data to configuration memory of the device. This concept is explained in detail in Section 2.5 using Xilinx FPGAs.

2.2.2 Runtime Reconfiguration

This term describes that the architecture can be reconfigured while the device is active. If it is not performed at the right time to the FPGA it can destroy or interfere with other pre-existing configuration and the active circuitry. Some design tools provides great help to avoid this problem and it will be covered in the following sections.
If devices are configured before they are activated (at power-up), it is called static configuration.

2.2.3 Multi-Context Configuration

Current SRAM-based FPGAs, in which state of the configurable interconnections are stored in SRAM based switches, contain an on-chip memory that stores a single, active device configuration. Speed of runtime reconfiguration can be improved if a device stores more than one device configurations at the same time, so called multi-context configuration. With multi-context configurations, it is possible to reduce reconfiguration rates to nanoseconds.

2.2.4 Fine-Grain Logic

Fine-grained architectures are intended to implement bit level logic circuits. Calculations that have arbitrary bit width can be done by using fine grained architectures. The advantage of fine-grained architectures is that it can map any logical circuit on the hardware. However, the overhead of routing resources increases as a cost of this flexibility. The well-known example for a fine-grained

architecture is FPGA. FPGAs are commercially available reconfigurable devices and most of reconfigurable computing researches are done on them.

2.2.5 Coarse-Grain Logic

Coarse Grained architectures are composed of array of Processing Elements (PEs). Processing Elements are designed to compute word-level computations. They contain coarse grain structures such as an ALU or a small processor. Therefore, a datapath calculation can be easily mapped on coarse grain architectures. Coarse-grain architectures are optimized for data processing and it is not possible to implement any arbitrary digital logic. Because of this lack of flexibility, configuration data is significantly reduced.

2.3 RECONFIGURABLE COMPUTING SPECIFIC DESIGN ISSUES

We have already emphasized that a large amount of configuration data is required to program a device with fine grain architecture while which is not an issue in coarse grain architecture. The configuration task can be eased with partial reconfiguration. With partial reconfiguration, only resources which require changes to implement new functionality, are reconfigured. Thus, the configuration contains only data corresponding to the reconfigurable parts. Implementation of partial configuration requires support by the configuration architecture of the device. Also the technic should be supported by software design tools. The development software tools should provide a mechanism to generate partial configuration data for the reconfigurable parts.

There is another requirement for the development software tools used for partial reconfiguration. When doing partial reconfiguration there exists a risk for corruption of previously existing functions so, the software must handle selection of part of FPGA for reconfiguration such that there is no possibility of corruption. Also, during reconfiguration reconfigurable circuit is transformed into a new circuit. During the process, the active circuit may not perform a meaningful operation since the device is configured column by column. The busmacros may prevent the interfere of being configured part with the active part of the device. In Figure 2.2 steps of reconfiguration concept is illustrated.

This problem is also solved with the development software since it place and route the design such that intermediate reconfiguration have no affect the static part. The designer must implement the reconfigurable modules such that the modules can be isolated from the device during runtime reconfiguration, i.e. by disabling the bus interface during configuration.

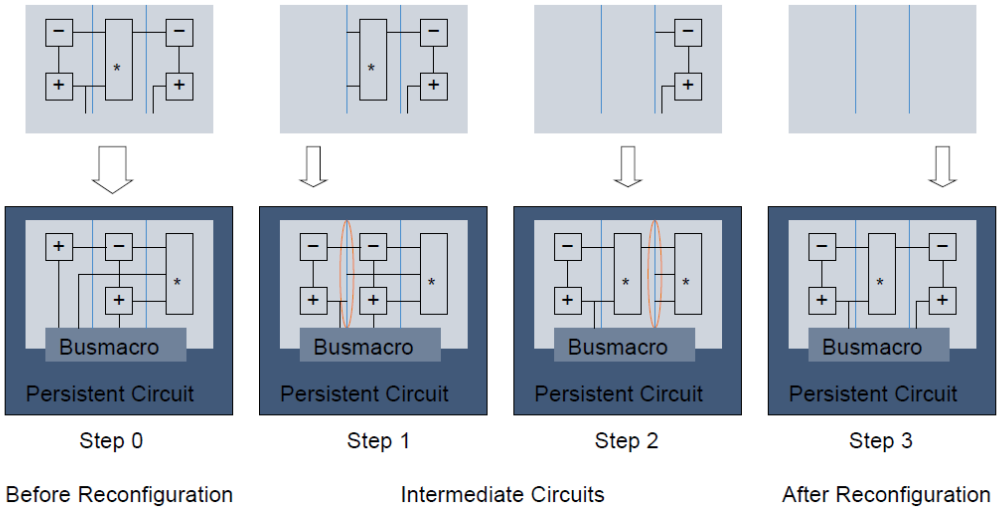


Figure 2.2: Partial reconfiguration process in a single context device. [6]

The bandwidth of configuration interface is limited, which determines the speed of loading new data to the device. Configuration caching and configuration prefetching techniques can overcome this limitation. Data compression can also be used to increase speed of reconfiguration. Data compression reduces the configuration data by discarding its redundant parts. The configuration architecture of a device or the software tool to generate configuration data file may support compression.

2.4 MAIN STRUCTURE OF XILINX FPGAS

Modern FPGAs were first introduced in the middle of the 1980's by the American company Xilinx [13] but the concept can be traced back to the 1960's. The first Xilinx FPGAs only contained a few thousand logic cells while modern FPGA Integrated Circuits (ICs) can contain several millions.

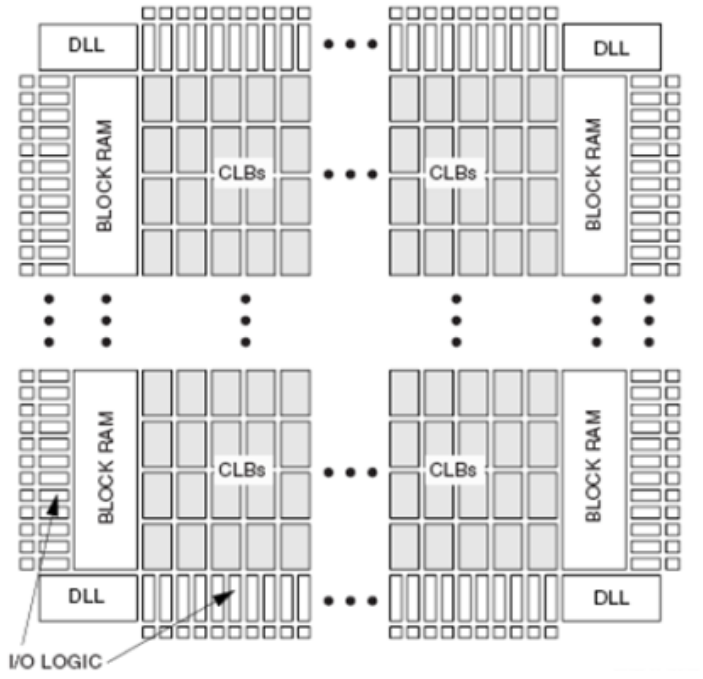


Figure 2.3: General Structure of Virtex FPGAs. [37]

An FPGA is composed of three basic elements: logic or memory blocks, I/O cells, and interconnection resources. Configurable Control Blocks (CLBs) provide the core logic and storage capabilities of the FPGA. Today, most commercial CLBs are LUT-based. Each CLB consists of several Basic Logic Elements (BLEs) arranged in a special fashion. In Xilinx's FPGAs a CLB consists of a number of so called slices, which consists of several BLEs. A BLE contains an N-input LUT and a D-type-flip-flop.

The programming of the FPGA is basically just connecting the CLBs in the right fashion. Several different programming methods exist with some being static while others being changeable. The most common one today is the use of some kind of static memory to hold the configuration while older technologies used fuses and anti-fuses to create permanent connections. Hardware Description Languages (HDLs) were created in order to increase the development speed of implementations on FPGAs. Using a synthesis tool the HDL code is translated into a bit stream containing the configuration data of the FPGA. This bit stream can then be uploaded to the FPGA from a computer or via a dedicated programmer. The two most popular HDLs are Verilog and VHDL. Both are commonly used within both academia and industry. In later years graphical development tools for embedded systems on FPGAs have been released by most FPGA-manufacturers enabling developers to rapidly develop complex systems. A survey of various FPGA-architectures can be found in [15].

2.5 RUN-TIME RECONFIGURABILITY OF FPGAS WITH A FOCUS ON XILINX FPGAS

Many of the modern FPGAs support run-time reconfigurability to some extent, with partial reconfiguration being the most common one. The potential benefits of run-

time reconfiguration are many, for example, the ability to dynamically move demanding functionality from software to hardware may improve overall performance in many applications. However, one should consider the time it takes to reconfigure a section on the FPGA and weigh this against potential speed-up. Even though the possibility of run-time reconfiguration has existed for over two decades now, few FPGA manufactures have yet to provide a complete design flow including run-time reconfiguration. The main reason is that the number of logic blocks on FPGAs has increased rapidly during this time and hence no direct need appeared strongly for partial reconfiguration due to the much easier process of implementing a fully static system instead of using function swapping. The second reason is the added development time needed for the implementation and verification of systems that employs partial reconfiguration. However, as the FPGSs grow larger so does the time needed to program these. This proves to be troublesome for example in applications where start-up timing is crucial. Static power consumption has also increased due to the increased number of transistors in each package. By utilizing smaller FPGAs in combination with partial reconfiguration it is possible to achieve lower power consumption and, in some cases, higher performance.

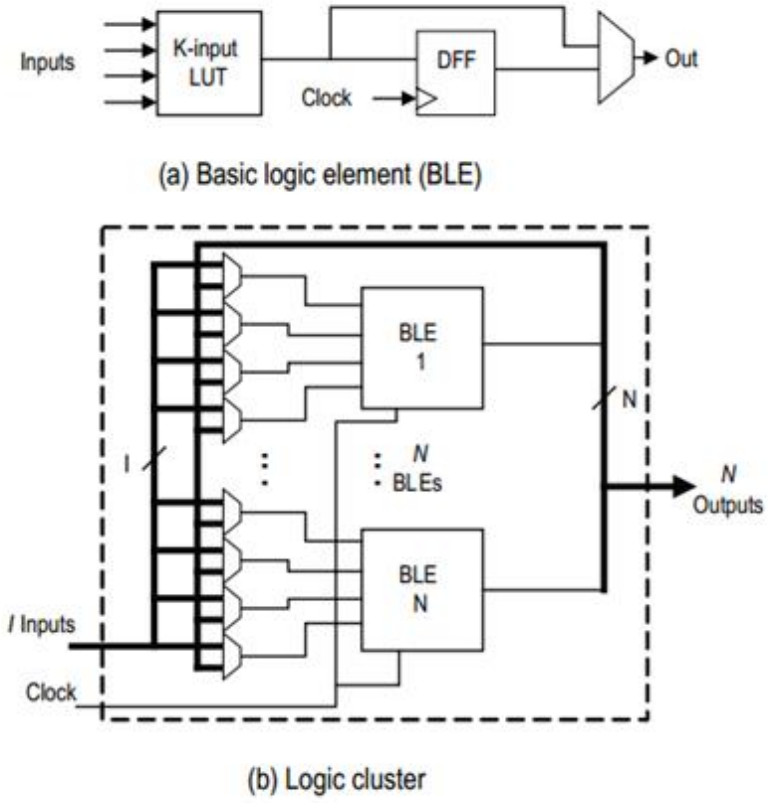


Figure 2.4: General concept of an FPGA-device [15]

Koch presents some crucial ideas behind the concept of partial reconfiguration [16]. In active reconfiguration an FPGA is reconfigured during run-time without disturbing the rest of the FPGA. In passive reconfiguration the entire FPGA is stopped (meaning all clocks in the FPGA are stopped) during reconfiguration. In the present work only active partial reconfiguration is considered and hence will be used synonymously with partial reconfiguration. Koch identifies three possible benefits of partial reconfiguration as performance improvement, area and power reduction and fast-system start-up.

An example of partial reconfiguration, where only crucial components are loaded on the FPGA at start-up and the rest of the functionality is loaded at a later stage is given in Xilinx Application Note 883 [17]. In this application an FPGA connected to the Peripheral Component Interconnect (PCI)-Express bus is partially configured at boot-up in order to meet the strict timing constraints of the bus.

The partial reconfiguration methods can generally be divided into two categories: i) difference based reconfiguration for small net list changes and ii) module reconfiguration for large module based changes. This thesis work will focus on module reconfiguration. However Xilinx reports that their latest FPGAs support reconfiguration of CLBs (ip ops, look-up tables, distributed random access memory (RAM), multiplexers, etc.), block RAM, and digital signal processor (DSP) blocks and all associated routing resources [18]. This implies that a high level of granularity can also be achieved during difference based partial reconfiguration. Partial reconfiguration can be regarded as a specific implementation of context switching but, as Koch points out, one must consider the entire system state before explicitly labeling partial reconfiguration as context switching [16].

Some key words used in the context of partial reconfiguration is needed to be explained and elaborated on before a more technical discussion of partial reconfiguration can occur. A short summary of commonly used terms in the context of partial reconfiguration are presented below as described by Koch [16] and Xilinx [19].

Reconfigurable Partition: A physical part of the FPGA constrained by the user to host reconfigurable modules.

Reconfigurable Module: A net list that is set to reside inside a reconfigurable partition at some point. Several modules can share the same reconfigurable partition.

Reconfigurable Logic: Logic elements that make up the reconfigurable module.

Static Logic: Logic implemented in such a way that it is not reconfigurable.

Proxy Logic: Logic inserted by the design software in order to provide the system with a known communication path between static logic and reconfigurable partitions.

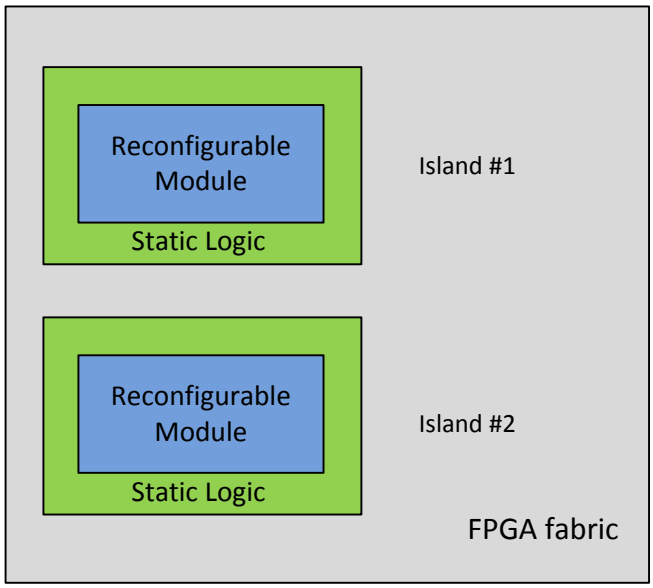


Figure 2.5: A typical island style partitioning strategy.

The techniques for placement of reconfigurable modules and techniques for the interaction with reconfigurable modules within reconfigurable partitions differ among manufactures. Three of the most common ones are i) island style, ii) grid

style and iii) slot style. In this report, we will be focusing on the island style, which is the simplest model for partial reconfiguration and the only one supported by Xilinx so far. Island style placement is illustrated in Figure 2.5.

It is worth pointing out the static region around the reconfigurable module. This is needed in order to provide a safe and efficient way for routing signals in and out of the reconfigurable module. The static region is extra important when the reconfigurable module is connected to a bus as it makes sure that the bus is not disturbed during reconfiguration. In Xilinx FPGAs the most common implementation of the static region was done by the so called bus macros, which were added manually by the user in the Integrated Software Environment (ISE) tool suite during the design phase. These have since been replaced by proxy LUTs that are automatically inserted during the synthesis. Island style placement only allows for one reconfigurable module per island. This means that a certain degree of fragmentation will occur when swapping modules as resources within the island will be wasted if not all resources are used by the new module. This is further enhanced by the fact that reconfigurable partitions must be predefined by the user and hence finding a perfect partition size in order to avoid fragmentation may be quite difficult if not impossible. Furthermore, the current tools on the market require net lists to be generated for a unique pair of module and island. This means that even if the same module can be placed in two different islands, two net lists and bit streams must still be generated. For example, a system that has 6 modules and 3 islands where all modules can rest in any island should have 18 unique net lists and bit streams. This is clearly a time consuming issue for the designer.

Xilinx states in their user guide for Partial Reconfiguration [19] that all logic can be reconfigured during run-time except "clocks and clock modifying logic , I/O and I/O related components, serial transceivers (MGTs) and related components and

individual architecture feature components such as BSCAN, STARTUP, etc. ", which must remain in the static region. Furthermore, bidirectional interfaces between static logic and reconfigurable logic are not allowed. Some specific IP components might also function erratically if used in combination with partial reconfiguration.

Another design consideration to notice is that the interface between the reconfigurable partition and the static logic must be static, as was stated earlier, implying that all reconfigurable modules to reside within a reconfigurable partition must have the same interface out towards the rest of the FPGA. Ports or bus connections cannot be created on the fly. An extensive list of design considerations for partial reconfiguration can be found in [19].

In order to use partial reconfiguration on a Xilinx FPGA, one should use Xilinx ISE design suite. The work flow to be used to employ partial reconfiguration on a Xilinx FPGA will be presented in the method-chapter of this report. Partial reconfiguration concept on a Xilinx FPGA is illustrated in Figure 2.6. The concept is explained in depth in [20]. Designers are forced to use so called "Bottom-Up-Synthesis" in order to implement a reconfigurable system successfully. This implies that all modules should have separate net lists for each possible instantiation and no optimization is allowed for the interface between the module and the rest of the FPGA. Bottom-Up-Synthesis is explained in Partial Reconfiguration Guide [19] and the Hierarchical Design Methodology Guide [21] both by Xilinx.
Xilinx states the following performance issues in [19]:

- Performance will vary between designs but in general expect 10% degradation in clock frequency and not to be able to exceed 80% in packing density.

- Longer runtimes will be observed during synthesis and implementation due to added constraints.
- Too small reconfigurable partitions may result in routing problems.

From the user point of view, reconfiguration of the FPGA during run-time is only a matter of writing a partial bit stream to the associated reconfiguration port. The most commonly used reconfiguration port in Xilinx-based systems is the ICAP-interface, which can be instantiated as a soft IP-core in the FPGA-fabric. This report focuses on the actual usability of the partial reconfiguration flow and hence the low-level technical reconfiguration process will not be discussed here. A good introduction to the partial reconfiguration work flow and limitations for Xilinx FPGAs can be found in Partial Reconfiguration User Guide by Xilinx [19] and the flow is further explained in Section 4.1.2.

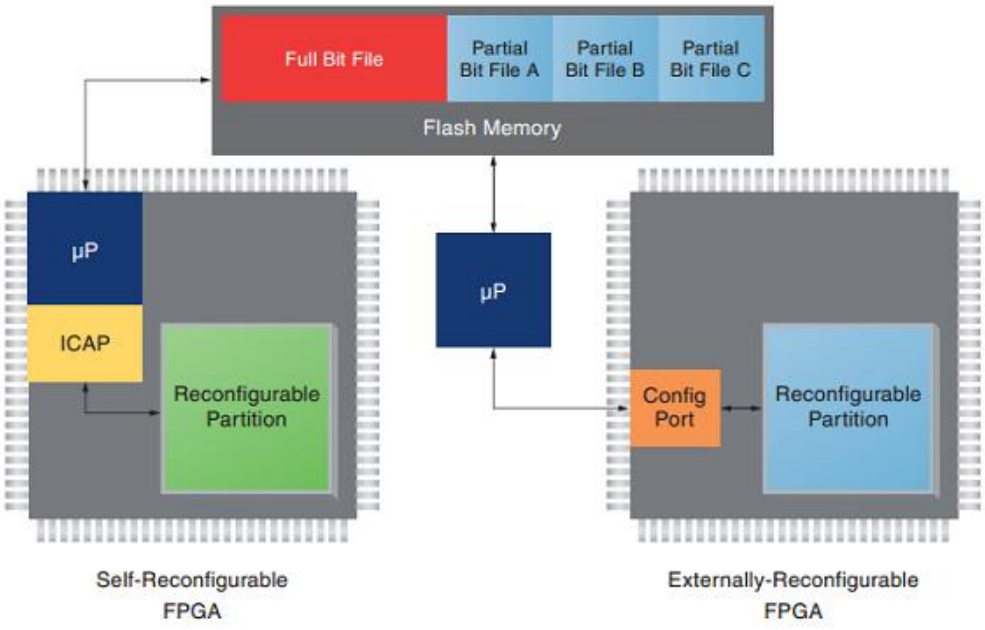


Figure 2.6: Outline of Xilinx RP [19]

Xilinx claims that the configuration time scales fairly linearly since the bitstream size grows with the number of reconfigurable frames with small variances depending on the location. If the reconfiguration time is linear with respect to the bitstream size, this would imply that partial reconfiguration can be used in time-critical systems since the worst case scenario can be calculated and verified with a high degree of certainty. This property may be useful in high-speed applications such as video processing or other streaming data applications [19].

An article written by Xilinx employees describes the general work-flow in the ISE-tool-suite and much of the information there still applies to the current versions. [22].

CHAPTER 3

RELATED WORK

3.1 Implementations of Reconfigurable FPGA-Systems

Utilizing the computational power of an FPGA in a mono or stereo vision system is no new concept. This concept has been investigated by many researchers throughout the years for reasons already accounted for. One interesting and relevant implementation of computer vision on an FPGA is the work in [23]. Authors in [28] compared the performance of a hard logic solution against a soft CPU-core (Altera Nios II) implemented on an FPGA for the task of filtering a 64 x 64 pixel or 256 x 256 pixel grey scale image using an n x n coefficient matrix. It was reported that the logic FPGA-implementation was immensely faster than the similar implementation on a CPU. The authors found that the logic based solution could perform up to 80 times faster.

Dynamic partial reconfiguration is an important feature for the realization of many reconfigurable computing platforms such as presented in [24]. In [24] a complete reconfigurable computing platform for embedded application is presented where the context switching of hardware tasks is required. An optimal partitioning for managing a partially run time reconfigurable hardware resource is proposed. For the implementation of the platform Virtex6 LX760 FPGA is used and for dynamic reconfiguration of the FPGA, HWICAP core is used.

In [25], a comparison between different ICAP-components is made and the reconfiguration speed versus bit-file-size is compared. It is reported that by making radical changes to the standard ICAP the reconfiguration time could be lowered by an order of magnitude one. In Figure 3.1 one can see the almost linear relationship between bit file size and reconfiguration time.

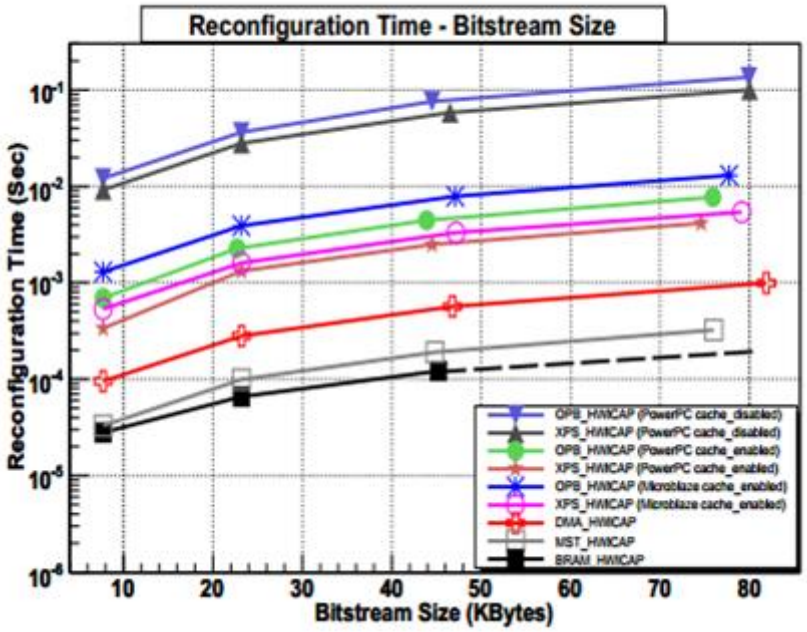


Figure 3.1: Reconfiguration time vs Bitstream size [25]

Koch et al. [26] present the concept of partial reconfiguration and demonstrate some commonly used tools. Some possible applications are presented one of which is "self-adaptive reconfigurable video-processing system. The modules depicted in the figure can be dynamically loaded and unloaded during run-time. This system was implemented on a Xilinx Virtex-II FPGA. No performance indicator was given but it seems that the performance of the system is acceptable for non-real time applications thus proving the concept to be implementable. [27] presents the GoAhead tool. GoAhead aims to provide developers with a simple user-interface in

order to create systems containing reconfigurable modules. Several other tools and work flows have lately emerged for creating run-time reconfigurable systems [28] [29].

3.2 Implementations of Reconfigurable FPGA-Systems Running Computer Vision Algorithms

Ackermann et al. [30] implemented a self-reconfigurable video processing system on an FPGA. They present two different implementations of the system. In the first one a Xilinx Virtex-IV FPGA is used and reconfiguration was performed with the help of the ICAP and Microblaze processor. Some common image processing algorithms were implemented as modules used for the reconfiguration testing. The first image processing algorithm implemented was binarization, the second was edge detection using 2-D gradients and the third one was edge detection using horizontal derivative. The bit-streams here are stored on a CompactFlash-card accessed by the MicroBlaze processor. Image data is retrieved from a camera with a resolution of 2048 x 2048 pixels at 16 frames / second. For both implementations only one image processing algorithm was allowed on the FPGA.

In the second implementation, the MicroBlaze processor has been removed and replaced by a small bit-stream controller connected to a Static Random Access Memory (SRAM) memory where all the bit-streams are stored. It is clear from the results that run-time reconfiguration is possible even in such demanding systems of video processing.

Another implementation of a dynamic reconfiguration in a multimedia application appeared in [31]. Xilinx Virtex-4 FPGA is used to implement a test-platform, where a VGA-camera stream is fed into the FPGA, passed through a reconfigurable filter

module and then output to a monitor. Also available on the FPGA was an audio module that had a reconfigurable filter slot. Three programming techniques utilizing the ICAP were tested. The first two, OPB-HWICAP and XPS-HWICAP, are made by Xilinx while the last one, SEDPRC, is designed by Bhandari et al. [33]. The novel ICAP interface seems to perform much better than the Xilinx alternatives, since performance of the ICAP-interface can be improved by utilizing custom-made components.

A run-time reconfigurable "multi-object-tracker" was implemented by Perschke et al. [32]. The implemented system is based on a Xilinx Virtex 4 FPGA with a PowerPC-CPU and video is retrieved from a camera running at 384 x 286 pixels at 25 FPS. Similar to Bhandari et al. [35], Perschke et al. have implemented a new ICAP-controller in order to improve the performance of the system since the standard Xilinx ICAP-controller was deemed too slow to be used. Perschke et al. [37] conclude that components are switched between frames and hence no delay in the object-tracking algorithm is produced. Considering the resource utilization of components and the speed of switching, one can conclude that the method is suitable to be used in high-speed vision systems.

An application for partial reconfiguration within the automotive industry can be found in [33] [34]. These studies show that FPGAs and real-time reconfiguration can be useful within the automotive industry as well where time-constraints are strict and overall reliability of the system is a critical factor.

CHAPTER 4

IMPLEMENTATION OF A PARTIALLY RECONFIGURABLE REAL-TIME SYSTEM

4.1 BACKROUND

4.1.1 VTU (Video Tracking Unit) System Architecture

Video Tracking Unit is a system that is being developed by Aselsan Inc., Defense Systems Group. The system can be divided into five main components, which are i) cameras, ii) host-PC, iii) FPGA iv) microprocessor and v) monitor. In the present work, we focus on the FPGA part but in this section the whole system is described shortly for a better understanding of the system.

In VTU, FPGA is used as a hardware accelerator and it implements a series of operations that are time-consuming to perform in a processor. XC5VFX70T of Xilinx Virtex-5 Family FPGA is used in the system.

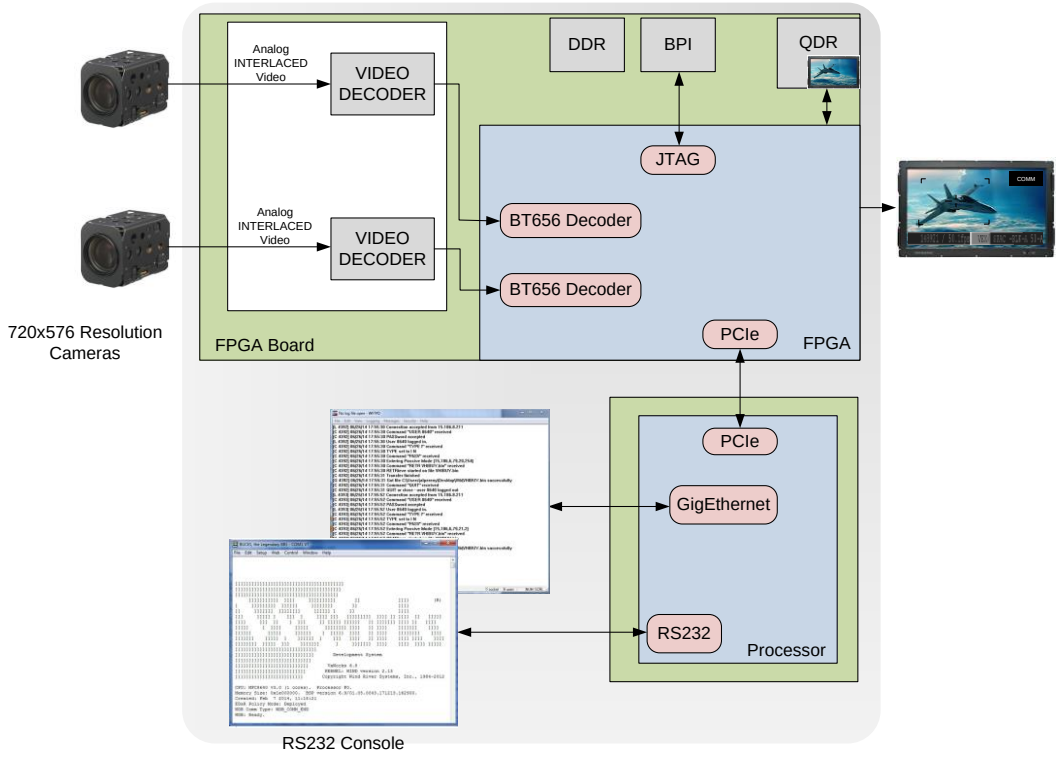


Figure 4.1: Video Tracking Unit (VTU)

In VTU, there are two cameras, only one of which is active at a time. Cameras are directly connected to the FPGA. The analog video standard of cameras is PAL (Phase Alternating Line). PAL is the common format for analog video display and is based on a 625 line, 50 field/25 frames a second, 50Hz system. In PAL video images, frames are 720 × 576 pixels. Video signals from the cameras are connected to video decoders on FPGA Board. Video decoder converts analog video data in PAL standard into digital video in BT.656 format. BT.656 video consists of color and brightness information in YCrCb format. Digitized video signals are connected directly to the FPGA. BT.656 format decoder component implemented in the FPGA extracts information such as Y, Cb, Cr and active area from the digital video signals. YCrCb format is a scaled and offset version of YUV (luminance, blue-luminance

red-luminance). YUV signals are generated from RGB (red, green and blue) source. The weighted values of R, G, and B are added to produce Y signal representing the brightness (or luminance) and U and V signals representing the color information. The active area corresponds to the duration in which the video data is being transmitted. The remaining is the blanking portion.

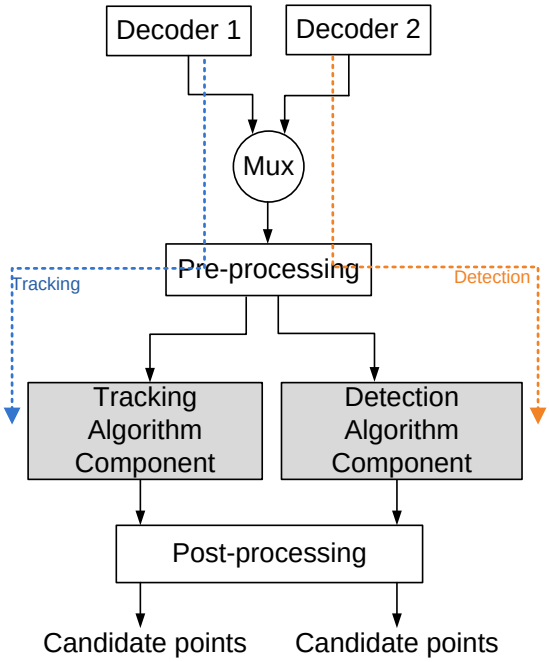


Figure 4.2: Video Processing Flow on FPGA in Non-PR System.

In the VTU system in which partial reconfiguration is not applied (Non-PR VTU), the video processing chain applied to each image frame received from the BT656 component is shown in Figure 4.2. On an image, pre-processing such as noise suppression is carried out first. In detection and tracking phases, outputs of target or target type specific operations are converted into candidate target points by post-processing. Outputs of both components, which are the candidate target points, are sent to the processor via the PCIe interface. Algorithms running on the processor use either the outputs of detection component or the outputs of tracking component

according to the system state, i.e., whether the system is in detection or tracking state and a symbol is generated accordingly. Generated symbol information is then sent to the FPGA to be displayed on the user console monitor.

In parallel to the flow shown in Figure 4.2, each frame arriving from the active camera is written to the external QDR memory placed on the FPGA board. Then, this buffered frame is fed to the BT656 encoder component to be monitored with symbol information coming from the processor.

Communication between the processor and the FPGA is handled via PCIe and to communicate with the Host-PC, the system uses RS232 interface. Information such as which camera is in active state and current state of the system (detection or tracking) is written to the memory on the CPU via the Host-PC and sent to FPGA by the processor. Some other parameters that are necessary for the processes implemented by FPGA software are also sent to FPGA by the processor and FPGA sends candidate target points to the processor after post-processing. Candidate target points are processed by the algorithms running on the processor. A target is determined (if there is any) and a symbol information is created. This symbol information is sent to the FPGA.

In the detection state, filtering, thresholding and spatial analysis components are used to determine possible targets. Following these processes, candidate targets are determined using pre-processing. In the tracking state, a filter with adaptable coefficients is used. Candidate targets are determined with post-processing on the filtered image.

4.1.2 Xilinx Tools and Implementation

This section describes how to create and implement a partially reconfigurable FPGA design using partitioning. Partial reconfigurable design steps can be shortly described as;

- Synthesize Netlists from HDL Source
- Create a PlanAhead Project
- Create Reconfigurable Partitions (PR)
- Add Reconfigurable Modules (PR) Netlist to PRs
- Add Additional RMs
- Add Black Box Modules (optional)
- Floorplanning Reconfigurable Partitions
- Partition Pins and RP Interface Timing
- Partial Reconfiguration Design Rule Checks
- Implement and Promote a Configuration
- Create and Implement additional Configurations
- Run PR Verify
- Generate and Download Bit Files

To implement a partial reconfigurable design, Xilinx ISE (Integrated Software Environment) and PlanAhead tools are used and Partial Reconfiguration Design

Flow [21] is followed. In addition, the restrictions given in Xilinx Application Note [17] are also taken into consideration.

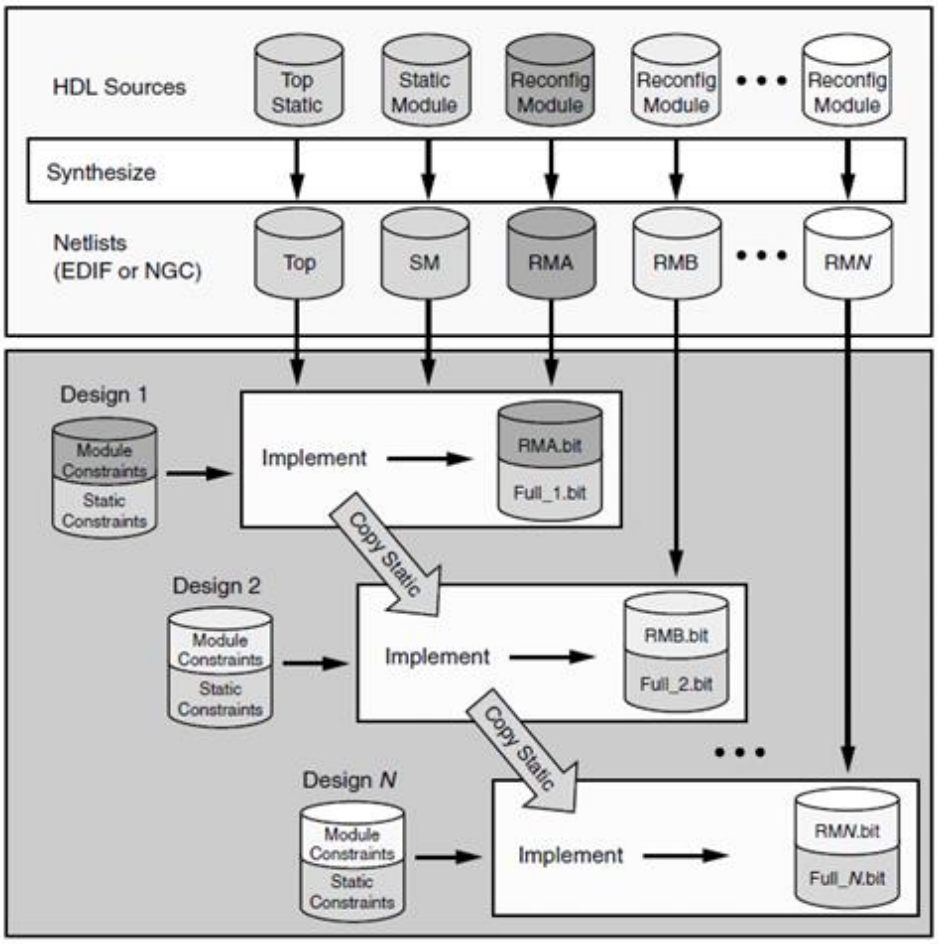


Figure 4.3 Overview of the Partial Reconfiguration Software Flow [19]

Figure 4.3 illustrates software flow of a partially reconfigurable design using Xilinx tools. First step is the synthesis of HDL sources which means converting HDL code to a gate-level netlist. The top light gray box represents the bottom-up synthesis. To correctly structure and synthesize a partially reconfigurable FPGA design, bottom-up synthesis approach is required. Each reconfigurable module is synthesized individually to generate its own netlist. The reconfigurable module netlists are not

included in the top-level netlist since the top-level design synthesize when the reconfigurable modules as black boxes. In this work the Xilinx ISE 14.5 is used to synthesize the design, and the PlanAhead 14.5 to implement the design.

As it can be concluded from the previous paragraph, partial reconfiguration design requires hierarchy as it is illustrated in Figure 4.4. Later we see that floorplanning is required to define reconfigurable regions and this hierarchy is also required in order to floorplan the design i.e. to create a reconfigurable partition for RMs.

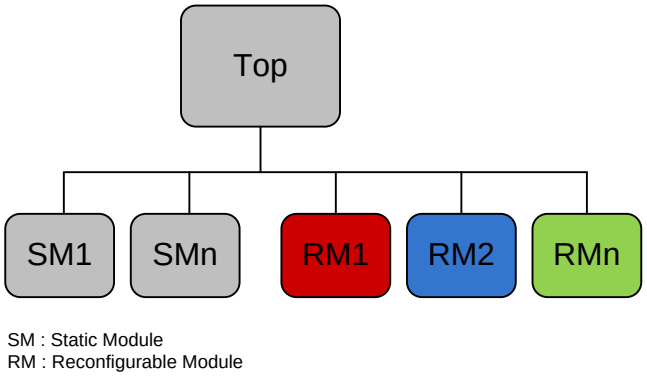


Figure 4.4 The hierarchy for a partial reconfiguration design

Netlist files generated using Xilinx ISE built-in synthesizer XST (Xilinx Synthesis Technology) are inputs of Xilinx PlanAhead netlist project for the implementation. After synthesis step, partitioning the design for partial reconfiguration is the next step.

A Partition is a logical section to be considered for design reuse, defined by the user at a hierarchical boundary. Previously it is stated that while synthesizing the top-level design the reconfigurable modules are left as black boxes. These black boxes are defined as reconfigurable partitions in PlanAhead and netlists of reconfigurable

modules are added to these related partitions. In our work there is only one reconfigurable partition and two reconfigurable modules of it. Then, a logical section of the FPGA is selected for partitions and area constraints are defined for each partitions. It is important to note that selection of partition section is the most critical step in reconfigurable designs since it require advanced knowledge about the used device. This step has to do manually and partition must include at least the largest PR module resource requirement.

There are three implementation stages in Xilinx design flow: translate, map and place and route. Translate is performed by the NGDBUILD (Native Generic Database Build) program. During the translate phase an NGC netlist generated by XST is converted to an NGD netlist. The NGC netlist is based on the UNISIM component library which is designed for behavioral simulation while NGD netlist is based on the SIMPRIM library designed for timing simulation. Mapping is performed by the MAP program. In this phase the SIMPRIM primitives from an NGD netlist are mapped on specific device resources such as LUTs, FFs, BRAMs.

As seen from dark gray box in Figure 4.3, a partially reconfigurable FPGA design implementation is similar to implementing multiple non-PR designs that share a common logic which is called static logic. The first configuration (design 1 in Figure 4.3) implements the static logic, and the user promotes this result. All other configurations (design 2, design N in Figure 4.3) import the static logic from the promoted configuration.

4.2 PARTIALLY RECONFIGURABLE VTU SYSTEM

In order to add partial reconfigurable property to the VTU system developed, several changes have been made and some components that are required for partial reconfigurable system are added.

As it is briefly explained in 4.1.2 section, partial reconfiguration requires a hierarchical design that must be followed during the coding process. Hence, first step was to make system hierarchically proper for PR design. For this tracking and detection components are recoded as a module and these PR modules were carried to right under top module and decoupling logic are added to inputs and output of PR Module. Partition pins are automatically managed, and some of the aspects of the functionality of previously used busmacros replaces this automation, though, use of decoupling logic in reconfigurable module pins is highly recommended since by this way it is possible to disconnect the reconfigurable region and the static portion during partial reconfiguration. It is necessary for preventing non-predictable behavior during reconfiguration. Hierarchical change is illustrated in

Figure 4.5

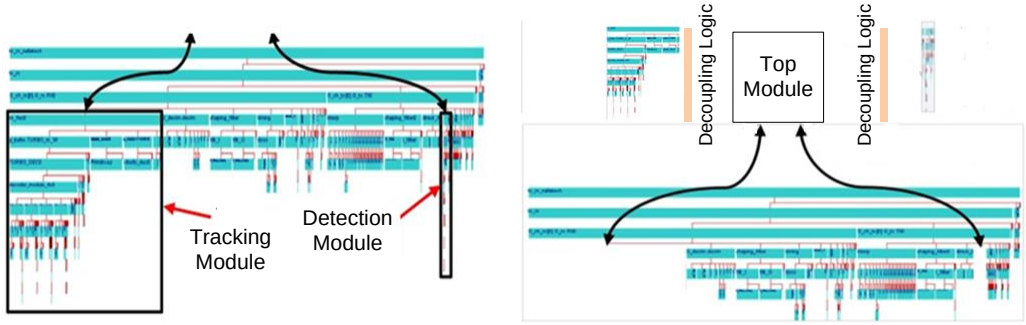


Figure 4.5: Hierarchical change of FPGA code

Overall block diagram of PR system after modifications is given in Figure 4.6. Partial reconfiguration is performed first using Internal Configuration Access Port

(ICAP) primitive. An ICAP-controller component is implemented for this purpose. The ICAP is an internal configuration interface of the FPGA, which can also be used for self-configuration.

The partial bit files that are needed for reconfiguration are stored in the onboard quad data rate (QDR) memory. ICAP has direct access to the QDR memory controller. At power-up, partial bit files are sent from the Host-PC to the unused area of the QDR memory, which is also used for buffering the incoming video frame before it is monitored with its symbol information. In our system, the implemented ICAP-controller has a direct access to QDR memory controller.

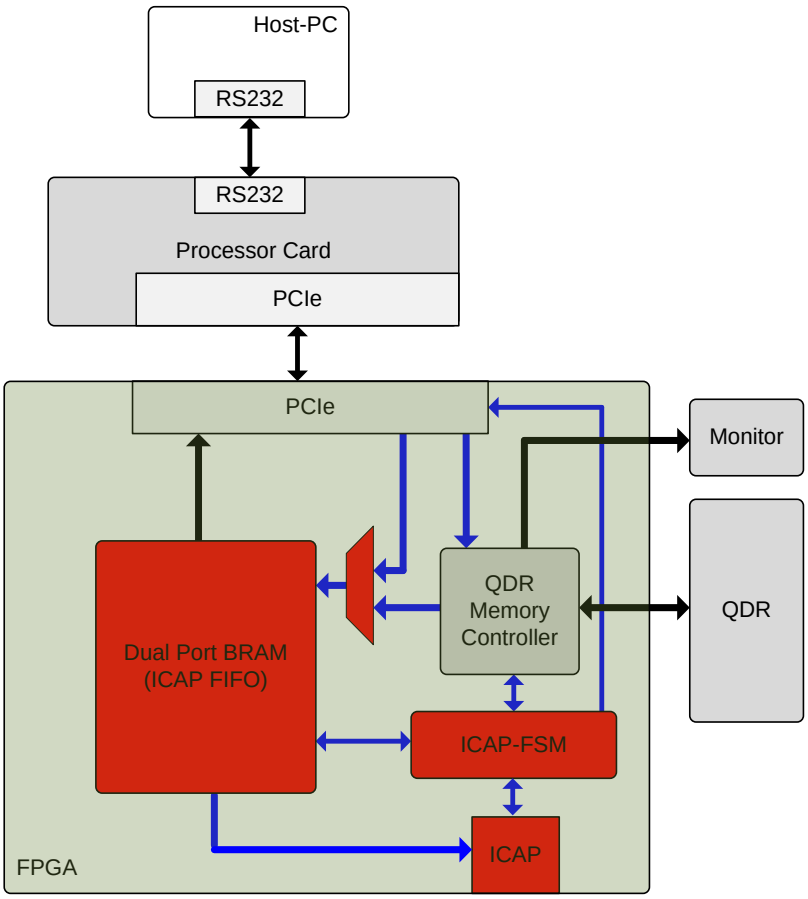


Figure 4.6: Simplified Block Diagram of PR System.

In the partially reconfigurable VTU (PR-VTU) system, run-time switching between tracking and detection components, i.e., the start of reconfiguration, is triggered when the system switches from detection state to tracking state or vice versa. The trigger comes from the Host-PC to the processor and is send to the FPGA via PCIe link afterwards. With this triggering, the necessary partial bit files are read from the QDR memory and fed into the ICAP FIFO when it is not full. This process lasts until the end of the bit file. QDR memory is also accessed in order to read the buffered frame. Thus, reading of the partial bit file from the QDR memory is performed in horizontal blanking of the buffered frame. During vertical blanking of the incoming video frames, sending new configuration data from ICAP FIFO to FPGA configuration memory via ICAP-controller is initiated, i.e. reconfiguration is performed.

In the following section, ICAP controller is briefly explained and some critical and implementation specific parameters, such as configuration time, are explained.

4.2.1 Configuration Time

There is a limitation on reconfiguration time due to real-time behavior of our system, thus reconfiguration time is very critical. The reconfiguration time has to be minimized for real-time tracking. In an ideal situation, we must reconfigure the system between two video frames, i.e. in vertical blanking interval.

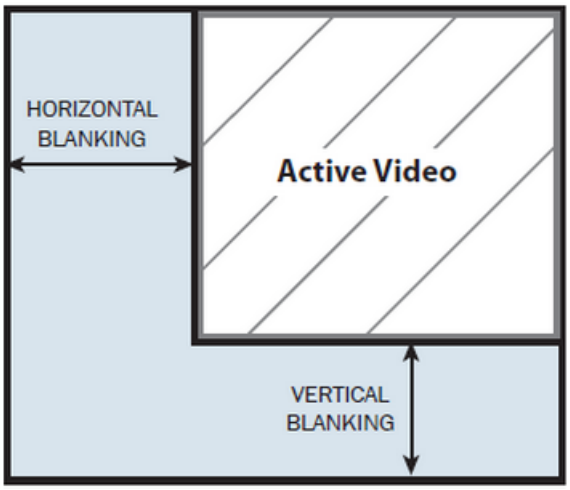


Figure 4.7: The horizontal and vertical blanking interval, and active region.

Vertical Blanking Interval is an interval of time between the last line of a given frame and the beginning of the next frame during which the incoming data stream is not displayed on the screen (Figure 4.7). This term is based on CRT screens. It is actually the time required for analog TV electronic gun beam to move from bottom of the frame to the top of next frame.

As it was stated earlier, analog video is converted to BT.656 standard for 625/50 system in our system. The BT.656 video signal data format describes a protocol for the interfaces and data stream format required to send uncompressed PAL (Phase Alternating Line), SECAM (Sequentiel Couleur avec Mémoir) or NTSC (National Television System Committee) video formats. The protocol defines both parallel and serial interfaces and also the blanking, sync and multiplexing schemes used in transmission of video signal. The BT.656 parallel interface uses 8 or 10 bits of multiplexed YCbCr data and a clock with 27MHz. Figure 4.8 shows the composition of the 8 bit parallel BT.656 interface data stream for a single horizontal line of video data. The end of active video data in the current line marked by the EAV (End of

Active Video) and the start of the active video data in the current line is marked by SAV (Start of Active Video). These marks eliminate the need for the HSYNC, VSYNC, and BLANK timing signals that are normally used in a video system. The EAV and SAV sequences are shown in Table 4-1. Both EAV and SAV codes are a sequence of four bytes. The first three bytes in the sequence are a fixed preamble: FFh – 00h – 00h. The fourth byte which is labeled 'XY', contains information about which field being transmitted (Field 1 or Field 2 in an interlaced video signal), the state of field blanking (Vertical Blanking) line blanking (Horizontal Blanking). The protection bits used for detection and correction of 1-bit errors and the detection of 2-bit errors. The status of P3, P2, P1 and P0 determined according to the states of bits Field (F), Vertical (V) and Horizontal (H).

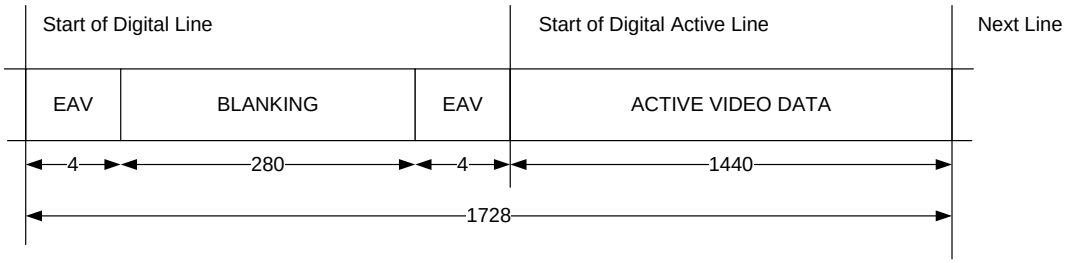


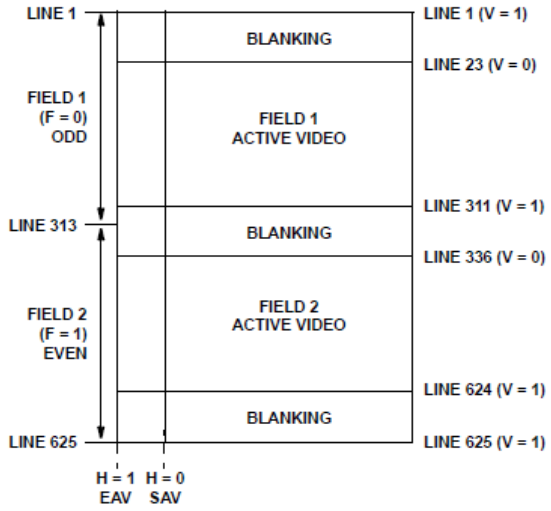
Figure 4.8: BT.656 8-bit parallel interface data format for 625/50 video systems

Table 4-1: EAV SAV Sequences

	8-BIT DATA								10-BIT DATA	
	D9 (MSB)	D8	D7	D6	D5	D4	D3	D2	D1	D0
Preamble	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
Status Word	1	F	V	H	P3	P2	P1	P0	0	0

For standard 625/50 video systems, one line is 1728 byte long and there are 25 lines in each frame in vertical blanking time as it can be seen in Figure 4.9 (line 1-23, line 311-313 for Field 1 and line 313-336, line 624-625 for Field 2). Knowing that operating frequency of the interface is 27MHz, maximum configuration time is calculated as;

$$25\ x\ 1728 \approx 1.6\ ms \tag{4.1}$$



LINE NUMBER	F	V	H (EAV)	H (SAV)
1-22	0	1	1	0
23-310	0	0	1	0
311-312	0	1	1	0
313-335	1	1	1	0
336-623	1	0	1	0
624-625	1	1	1	0

Figure 4.9: Typical BT.656 vertical blanking intervals for 625/50 video systems

The algorithms (detection and tracking) cannot deliver any results for frames received in this reconfiguration time and if the reconfiguration process exceeds 1.6 ms, then since it is not possible to start tracking in the middle of a frame, according to the exceeding reconfiguration time, frame or more than one frame may be lost.

4.2.2 The ICAP Controller

The Internal Configuration Access Port (ICAP) is a primitive found in Xilinx Virtex FPGAs. The ICAP gives access to the FPGA configuration memory via the Configuration Registers [35]. The ICAP interface is depicted in Figure 4.10.

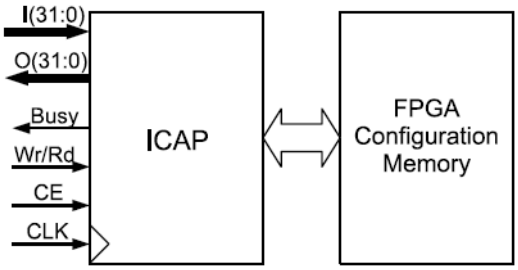


Figure 4.10: The internal Configuration Access Port (ICAP).

ICAP-controller is made up of three modules: ICAP-FSM, ICAP-FIFO and ICAP itself. Finite State Machine (FSM) controls the ICAP signals and Write FIFO are used to stream data to the ICAP. Figure 4.11 is the block diagram of our system based ICAP controller. Our ICAP-controller is connected directly with memory-controller without using the PLB-bus and the standard memory-controller. This is beneficial in real-time, high performance applications since the hard and soft processors does not used to move data to the ICAP controller during a reconfiguration operation thus, ICAP controller with high throughput is achieved. The ICAP FIFO is a dual port BRAM memory primitive with one port connected to the FSM and the other port multiplexed between the processor and the external memory interface. The partial bit files are stored in the onboard QDR memory and they are written at each power-up before system starts to operate. The memory controller, which is directly connected to the ICAP-controller, works at a frequency of 200 MHz and reads the data in the form of 9 Bytes. Bitstream data read from

QDR are first are stored in ICAP-FIFO, which is used for caching the data. ICAP-FIFO is avoided to run empty during the reconfiguration process for 100% throughput.

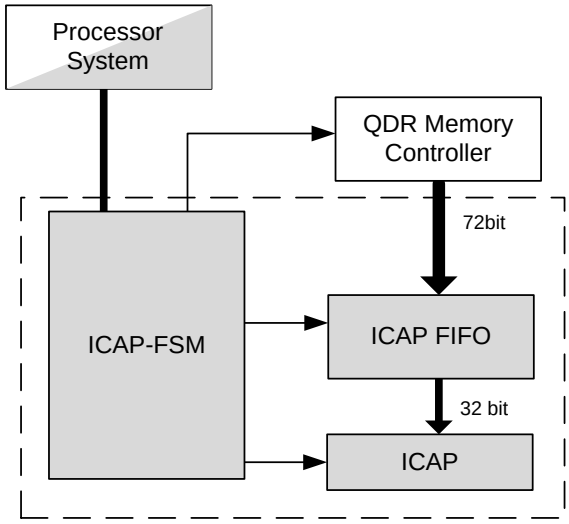


Figure 4.11: ICAP Controller

ICAP Controller is implemented in such a way that if the input clock frequency is 100MHz, which is the recommended maximum value of Xilinx for Virtex-5 ICAP primitive, ICAP reaches its maximum throughput of 400 MB/s within the system during partial reconfiguration. However, because of the timing constraint in reconfiguration time and the final partial bit file sizes, we had to overclock ICAP primitive at 125MHz in our module. Most IC manufacturers build their chips within a tolerance when specifying the operating characteristics. Taking this into account the ICAP port for this system is overclocked by 25%, when the device is in nominal operating conditions. The overclocking technique can be used in systems; however, the clocking for the system would need to be controlled with a BUFGMUX primitive [36]. This primitive allows for glitchless clock switching.

4.2.3 Selected Reconfigurable Partition Properties

Following the addition of the necessary components and make the system hierarchical, next step is to implement a partial reconfigurable design using Xilinx tools. In previous section 4.1.2, partial reconfigurable design flow is explained in detail. It is said that the most critical part in the design flow is the selection of partial reconfigurable partition. This step has to do manually and the choice of this physical block in FPGA should be done according to the resource requirements of each reconfigurable module i.e. partition must include at least the largest PR module resource requirement that is tracking module in our system. In Table 4-2, resource utilization of tracking and detection modules which are reconfigurable modules in the PR-VTU system is given together with utilization percentages according to the available FPGA resources. From the Table 4-2; it can be seen that partition must include at least 8986 LUTs, 13584 FFs, 19 RAMB36 and 64 DSP48. However in practice, taking into consideration of required routing resources and achieving required timing performances, partitions are recommended to be chosen such that highest utilization is about 80% for each resource type [37]. There are also other restrictions and recommendations concerning the selection of these areas by Xilinx [37] that are followed in our design as well. Many iterations is done in partition selection step and the available resource information of final partition selected is given in Table 4-3.

Table 4-2: Resource utilization percentages of PR module according to FPGA.

	FFs	**6 input- LUTs**	**RAMB36**	**DSP48**
Tracking RM	13584 (30%)	8986 (20%)	19 (12%)	64 (50%)
Detection RM	1831 (4%)	2138 (4%)	12 (8%)	0 (0%)

Table 4-3: Resource utilization percentages of PR modules according to partition.

	RP	Tracking RM		Detection RM	
	Available	**Required**	**%**	**Required**	**%**
FFs	23360	13584	58.2	1831	7.8
6 input- LUTs	23360	8986	38.5	2138	9.2
RAMB36	64	19	29.7	12	18.8
DSP48	80	64	80	0	0

As depicted in section 4.1.2, after netlists are generated and the partition area is determined, in each design (configuration) appropriate netlists are implemented and the full and partial bit files for that configuration of the FPGA are generated. The static logic from the first implementation is shared among all other design implementations. Final layout of the two configurations (target and detection) after placement and routing, are given in Figure 4.12. Left is the layout of target configuration and right is the detection. Reconfigurable Partition is highlighted in red.

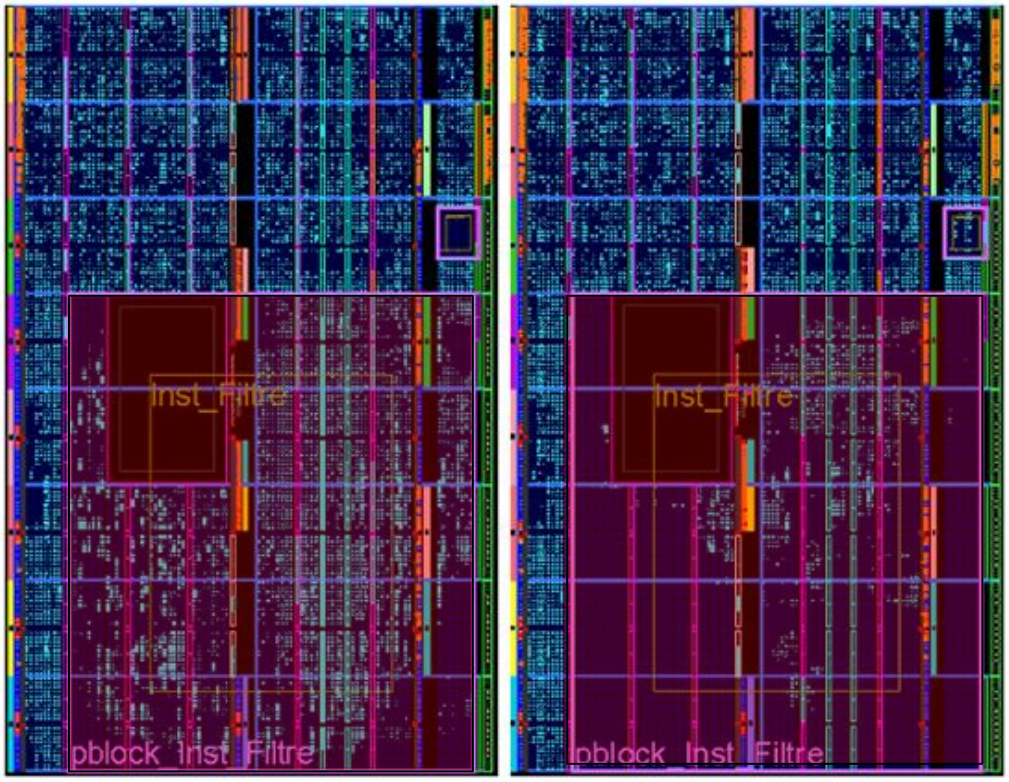


Figure 4.12 Layout of Target (Left) and Detection (Right) Configurations

In Table 4-4, partial bit file sizes of partial reconfigurable target and detection components are given. Since partial reconfiguration should last at most as the vertical blanking time, i.e., 1.6ms and throughput of our ICAP-controller is 500MB/sec. Maximum partial bit file size become:

$$500\ MB\ x\ 1.6\ ms = 800KB \tag{4.2}$$

It can be concluded that those bit files are quite large (larger than 800KB). As a solution, bit stream compression feature of BitGen tool is used. The BitGen compression feature may be enabled to reduce the size of the bit files. This option looks for repeated configuration frame structures in reconfigurable partition after

pace and route to reduce the amount of configuration data that must be stored in the partial bit file. This savings is consolidated as reduced configuration file and reconfiguration time. When the compression option is activated in PR design, all of the bit files (full and partial) are created as compressed bit files. Partial bit stream file sizes are given in Table 4-4 before and after the compression.

Table 4-4: Partial Bitfile Sizes

	Partial Bit File Sizes	
	Before Compression	After Compression
Detection RM	1296KB	690KB
Tracking RM	1296KB	780KB

CHAPTER 5

EVALUATION AND IMPLEMENTATION RESULTS

5.1 EVALUATION OF THE SYSTEM

5.1.1 Measurement of Reconfiguration Throughput and Time

The theoretical reconfiguration throughput of the ICAP is calculated as:

$$Theoretical\ Throughput\ = ICAP\ data\ width\ x\ clock\ frequency. \quad (5.1)$$

However the ICAP is not always able to process the incoming data. When this happens it raises a busy signal and stalls the reconfiguration process. Moreover, the system could not be able to provide a data per clock cycle, thus it could not use the ICAP at its theoretical throughput. In the light of these considerations, the reconfiguration time can be computed as:

$$Reconfig.Time = Bitstream\ size\ x\ Data\ Throughput + tbusy. \quad (5.2)$$

where $tbusy$ is the time lapsed during the busy state of the ICAP and data throughput is the rate at which data are sent to the ICAP.

For ICAP controller, ICAP data width is set to 32bit and that the frequency of operation is 125MHz since we do overclocking, the theoretical ICAP throughput is

500 MB/sec and it is previously explained that ICAP controller is design such that ICAP can process an incoming data at every clock cycle.

In order to find out whether this theoretical throughput is achievable, we performed experiments to configure different regions of the chip, to repeatedly writing NOPs to ICAP, and to stress the configuration circuit by repeatedly configuring one region. During all these tests, we found out that ICAP always ran at full speed such that it was able to consume four bytes of configuration data per cycle, regardless of the semantics of the configuration data. This confirms that configuration time is a pure function of the size of the bitstream file, i.e. $tbusy$ =0.

Also to measure reconfiguration time a hardware counter is added to the ICAP-FSM Module. It measures the number of clock cycles required for reconfiguration process. This counter is monitored using Chipscope and reconfiguration time is measure to be 1.578 ms. It is less slightly that expected because unused header part of partial reconfiguration is not fed to the ICAP.

5.1.2 Evaluation of System Performance

In the system, feeding data to ICAP primitive starts with the start of vertical blanking interval and with the tests explained in section 5.1.1 it is guaranteed that it lasts less than the interval, i.e. no video frame is lost during reconfiguration process.

In order to test whether our tracking and detection reconfigurable modules are function properly after the partial reconfiguration, we applied to one of the system cameras a test pattern using a pattern generator. Using full bit files of detection and tracking configuration, the outputs (candidate points) of both modules for this pattern is saved in the Host-PC. After partial reconfiguration process, outputs (candidate points) of both modules are compared with saved ones using Matlab. Results showed that functionality of the modules is not affected from the process.

5.2 IMPLEMENTATION RESULTS

5.2.1 Hardware Utilization Analysis

The comparison of non-PR VTU and two configurations of PR VTU according to resource utilization is shown in Figure 5.1 and resource utilization gains of two configurations according to the non-PR VTU is given in Figure 5.2.

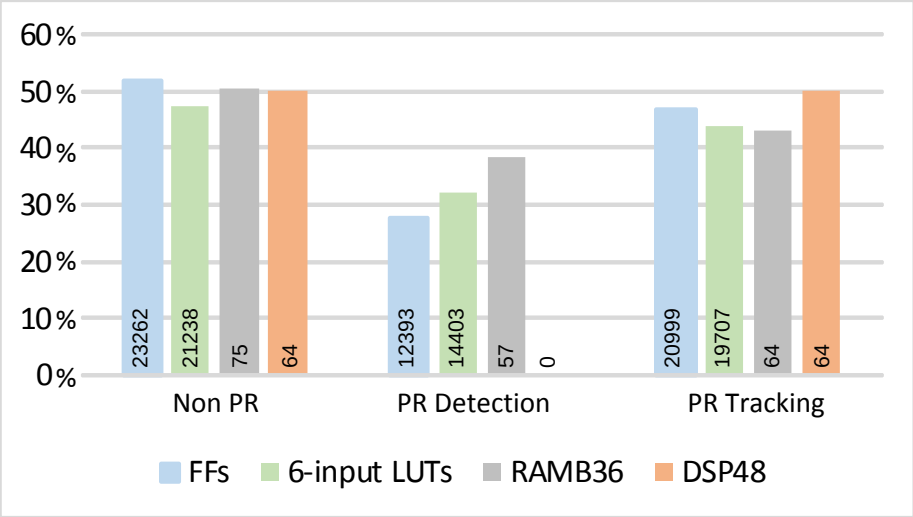


Figure 5.1 Resource utilization percentages according to the available resources in XC5VFX70T

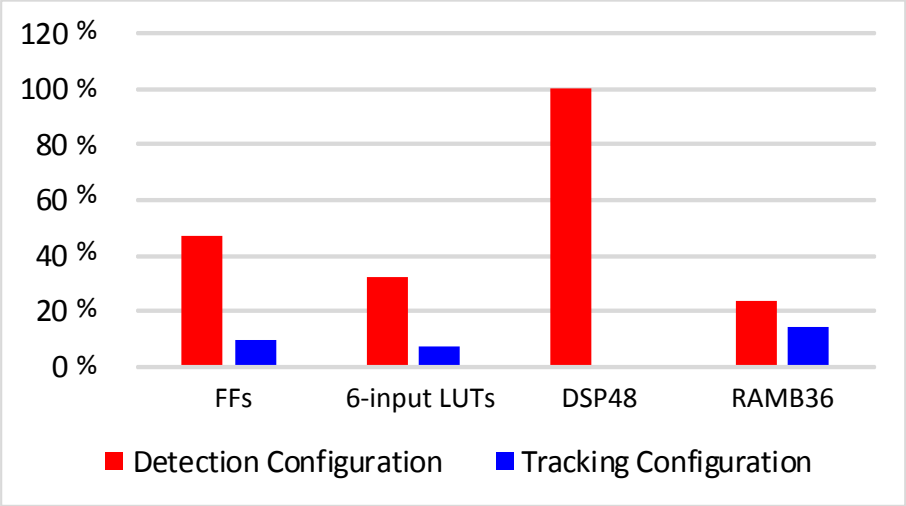


Figure 5.2 Resource utilization gain according to the non-PR VTU

In detection configuration, FF and LUT savings are quite good with 32.18 % and 46.72% while in tracking configuration they are 7.21 % and 9.73 respectively. Because detection module is quite small in size (Table 4-2), tracking configuration saving become relatively low and it prevent us from choosing more small FPGA. However this saving is important for lowing overall power and releasing resources of FPGA for the possible use for another purposes in the system.

5.2.2 Power Consumption Analysis

In order to measure the chip power consumption, we placed a 0.033 Ohm shunt resistor into the 1.0 V core power supply rail on the FPGA board the voltage drop across the shunt resistor is measured with a multimeter. Then, the current that was drawn into the FPGA chip is derived from the voltage drop and by multiply this current by 1.0 V the chip core power consumption under different configurations is derived.

Table 5-1 shows the chip power consumption under different states of the system. The first column shows the current drawn into the FGPA chip, the second column shows chip power consumption. During partial reconfigurations (pr), the static part is trying to reconfigure the reconfigurable partition to switch from detection to tracking, during this time the chip consumes 3.501 W of power and 5.102 W of power while switching from tracking to detection. If the static is active and the detection module region is loaded in the partition (detection), the chip power consumption is 3.428 W. The chip power consumption becomes 5.009 W if the tracking module is loaded. The chip power consumption is 5.611 W if the non-PR design bit file is loaded in the system.

Table 5-1: Chip Power Consumption

	Current (A)	**Power (W)**
pr (detection to tracking)	3.501A	3.501W
pr (tracking to detection)	5.102A	5.102W
Detection	3.428A	3.428W
Tracking	5.009A	5.009W
Non-PR	5.611A	5.611W

Partial reconfiguration brings a power overhead [37]. However in our system during the configuration phase there is a little power increase because high ICAP throughput results very low power overhead [37]. Level of differences between partial bit files also have an impact on this value [37], hence it can be concluded that the differences in detection and tracking partial bit files are low. Moreover, in our

system both partial reconfiguration power overhead are much lower that the Non-PR system case and reconfiguration time is quite small (1.6ms).

Since this a target detection and tracking system, an assumption can be made such that it is mostly in detection it can be said that the power gain is 2,183W (5.611W-3.428W) i.e. 38% of total FPGA power consumption.

Another comment of this result is that since it is a system that is supposed to be active nearly all the time, the energy reduction gained by using partial reconfiguration.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Partial reconfiguration is a technique that allows reconfiguring a specific part of an FPGA during runtime [6]. This method allows switching between design modules that are not necessary to function at the same time without interrupting the current task of the static FPGA part.

Video Tracking Unit (VTU) that is being developed by ASELSAN Inc., Defense and Systems Technologies is a real-time target detection and tracking system. In this system, an FPGA is used as a hardware accelerator and it implements a series of operations that are time-consuming to perform in a processor. In the operations that are implemented on FPGA hardware, there exist tracking and detection components, which are not needed to run simultaneously. As a case study, partial reconfiguration has been implemented on an FPGA, which is used as a hardware accelerator in a real-time target detection and tracking system. Using partial reconfiguration, different algorithm components are switching during runtime without interrupting object-tracking. Using partial reconfiguration feature of Xilinx Virtex-5 XC5VFX70T FPGA is used in the system, reduction in power consumption is observed and the FPGA resources are used more effectively for possible future additional functions.

In the future, the architecture is predicted to benefit from increased flexibility by implementing different tracking and detection modules according to the changing environment or target types.

REFERENCES

[1] “http://www.cray.com” [Online], Cray Inc., June 2004, Available: http://www.cray.com [Accesed 10 June 2014].

[2] “www.xilinx.com,” [Online], Xilinx Inc., June 2014, Available: http://www.xilinx.com [Accesed 1 June 2014].

[3] “www.altera.com,” [Online], Altera Inc., June 2014, Available: http://www.altera.com [Accesed 1 June 2014].

[4] Bobda C., Introduction to Reconfigurable Computing: Architectures, algorithms and applications. Springer Publishing Company, 2007.

[5] Hartenstein R., “A decade of reconfigurable computing: A visionary retrospective,” in Design, Automation and Test in Europe Conference and Exhibition (DATE 2001), pp. 642–649, 2001.

[6] Rullmann M., Merker R., Dynamically Reconfigurable Systems, Springer, pp 161-181, 2009.

[7] Liu, M., Kuehn, W., Lu, Z., Jantsch, A., “Run-time partial reconfiguration speed investigation and architectural design space exploration” in IEEE Int. Conf. Field Programmable Logic and Applications, pp. 498-502, 2009.

[8] Santambrogio, M.D., Sciuto, D., “Task scheduling with configuration prefetching and antifragmentation techniques on dynamically reconfigurable systems” in ACM Proc. Conf. Design, Automation and Test in Europe (DATE ’08), pp. 519–522, 2008.

[9] LLanos C., Jacobi R.P., Rincón M.A., Hartenstein R.W., “A Dynamically Reconfigurable System for Space-Efficient Computation of the FFT”, in International Conference on Reconfigurable Computing and FPGAs (ReConFig'04), pp 360-369, 2004.

[10] Ullmann, M.; Huebner, M.; Grimm, B.; Becker, J., “An FPGA run-time system for dynamical on-demand reconfiguration”, in 18th International Parallel and Distributed Processing Symposium, pp. 135- 142, 2004.

[11] Jianwen, L., Chuen, J.C., “Partially reconfigurable matrix multiplication for area and time efficiency on FPGAs ” in Euromicro Symposium on Digital System Design (DSD 2004),. pp. 244-248, 2004

[12] Hennessy J. A., Patterson D. L., Computer Architecture: A Quantitative Approach, Morgan Kauffmann, 1990.

[13] Gholamhosseini H., Hu S., “A high speed vision system for robots using fpga technology” 15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP 2008), pp.81 -84, 2008.

[14] Kuphaldt T. Lessons In Electric Circuits, Volume IV. Digital. Open Book Project, 2007.

[15] I. Kuon, R. Tessier, and J. Rose. “Fpga architecture: Survey and challenges. Foundations and Trends” in Electronic Design Automation, vol. 2, no 2, pp.135-253, 2007.

[16] D. Koch, Partial Reconfiguration on FPGAs. Springer, 2012.

[17] Tam S., Kellermann M., Fast configuration of pci express technology through partial reconfiguration, Xilinx Inc., 2010.

[18] Dye D., Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite. Xilinx Inc.,2012.

[19] Xilinx Inc., Partial Reconfiguration User Guide. Xilinx Inc., 2012.

[20] Khalaf A., Jagtiani M., “Framework for self reconfigurable system on a xilinx fpga” in Embedded Systems Design, pp.22-27, 2011.

[21] Xilinx Inc., Hierarchical Design Methodology Guide. Xilinx Inc., 2012.

[22] Lysaght P., Blodget B., Mason J., Young J., Bridgford B. “Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic

reconfiguration of xilinx fpgas" in IEEE International Conference on Field Programmable Logic and Applications (2006. FPL'06), pp. 1-6, 2006.
[23] H. Gholamhosseini, S. Hu, "A high speed vision system for robots using fpga technology". In 15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP 2008), pp. 81 -84, 2008.
[24] Say, F., Bazlamacci, C. F., "A reconfigurable computing platform for real time embedded applications", Microprocessors and Microsystems, vol. 36, no. 1, pp. 13-32, September 2011.
[25] Ming Liu, Kuehn W., Z. Lu,, Jantsch A., "Run-time partial reconfiguration speed investigation and architectural design space exploration" in IEEE International Conference on Field Programmable Logic and Applications (FLP 2009), pp. 498-502, 2009.
[26] D. Koch, J. Torresen, C. Beckhoff, D. Ziener, C. Dennl, V. Breuer, J. Teich, M. Feilen, and W. Stechele, "Partial reconfiguration on fpgas in practice - tools and applications", in ARCS Workshops (ARCS), pp. 1 -12, 2012.
[27] C. Beckhoff, D. Koch, and J. Torresen, "Go ahead: A partial reconfiguration framework", in IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 37-44, 2012.
[28] Otero A., Torre E., Riesgo T., "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems", in 2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1-8, 2012.
[29] Ochoa-Ruiz G., Labbani-Narsis O., Bourennane E., Cherif S., Meftali S., Dekeyser J., "Facilitating IP deployment in a MARTE-based MDE methodology using IP-XACT: a XILINX EDK case study", in 2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1-8, 2012.
[30] Ackermann K.F., Hoffmann B., Indrusiak L.S., Glesner M., "Enabling selfreconfiguration on a video processing platform", in International Symposium on Industrial Embedded Systems (SIES 2008), pp. 19 -26, 2008,

[31] Bhandari S., Subbaraman S., Pujari S., Cancare F., Bruschi F., Santambrogio M.D., Grassi P.R.. “High speed dynamic partial reconfiguration for real time multimedia signal processing”, in 15th Euromicro Conference on Digital System Design (DSD), pp.319 -326, 2012.

[32] Rummele-Werner M., Perschke T., Braun L., Hubner M., Becker J., “A fpga based fast runtime reconfigurable real-time multi-object-tracker” in IEEE International Symposium on Circuits and Systems (ISCAS), pp. 853 -856, 2011.

[33] Claus C., Zeppenfeld Muller J., F., and Stechele W., “Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system” in Design, Automation Test in Europe Conference Exhibition (DATE '07), pp. 1-6, 2007.

[34] Claus C., Ahmed R., Altenried F., Stechele W., Reconfigurable Computing: Architectures, Tools and Applications. Springer, pp 55-67, 2010.

[35] Xilinx Inc., Virtex 5 FPGA Configuration Guide. Xilinx Inc., 2009.

[36] Xilinx Inc., Virtex 5 FPGA User Guide. Xilinx Inc., 2012.

[37] Shaoshan L.; Pittman, R.N.; Forin, A; Gaudiot, J.-L., “On energy efficiency of reconfigurable systems with run-time partial reconfiguration,” in 21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP), pp.265-272, 7-9 July 2010