Dynamically and Partially Reconfigurable Embedded System Architecture for Automotive and Multimedia Applications

Naim Harb

Université de Valenciennes et du Hainaut-Cambrésis

LAMIH
LABORATOIRE D'AUTOMATIQUE DE MECANIQUE ET D'INFORMATIQUE INDUSTRIELLES ET HUMAINES

Dissertation
Presented and defended on: 23 September 2011

Dynamically and Partially Reconfigurable Embedded System Architecture for Automotive and Multimedia Applications

By

Naim M. Harb

For obtaining

Doctorate Degree

At the Université de Valenciennes et du Hainaut-Cambrésis

Ecole Doctorale Sciences Pour l'Ingénieur Université Lille Nord-de-France-072

Specialty: Automatique et Informatique des Systèmes Industriels et Humains

Domain : Embedded Systems

Referee	Pr. Georgi Gaydadjiev, Delft University of Technology
Referee	Pr. Lionel Torres, LIRMM Université de Montpellier
Examiner	Pr. Carlos Valderrama, Université de Mons
Examiner	MCF HDR Daniel Chillet, IRISA, Rennes
Superviser	Pr. Smail Niar, Université de Valenciennes
Co-Superviser	Pr. Mazen Saghir, Texas A&M University At Qatar

Numéro d'ordre 11/23

UNIVERSITE DE VALENCIENNES ET DU HAINAUT-CAMBRESIS
Le Mont Houy – LAMIH – 59313 VALENCIENNES Cedex 9
Téléphone (+33) 3 27 51 12 34 – Télécopie (+33) 3 27 51 11 00

UNIVERSITE DE VALENCIENNES ET DU HAINAUT-CAMBRESIS
Le Mont Houy – LAMIH – 59313 VALENCIENNES Cedex 9
Téléphone (+33) 3 27 51 12 34 – Télécopie (+33) 3 27 51 11 00

Acknowledgments

Summary

Application-specific programmable processors are widely used in embedded systems due to their customized instruction sets and microarchitectural features. This enables them to attain high levels of performance and energy efficiency. However, short time-to-market windows, high design and fabrication costs, and fast changing standards make application-specific processors a costly and risky investment for microelectronics manufacturers and embedded system designers.

To overcome these problems, embedded system designers are increasingly relying on Field Programmable Gate Arrays (FPGAs) as target design platforms. Today's FPGAs provide high levels of logic density and rich sets of embedded hardware components that enable the implementation of highly complex systems. They are also inherently flexible and can be easily and quickly modified to meet changing application or system requirements. FPGAs also support the reuse of standard or custom IP blocks, which further decreases system development time and costs. On the other hand, FPGAs are generally slower and consume more power than Application-Specific Integrated Circuits (ASICs), and this can restrict their use to limited application domains. However, recent advances in FPGA architectures, such as Dynamic Partial Reconfiguration (DPR), are helping bridge this gap. DPR enables a portion of an FPGA device to be reconfigured while the device is still operating. This reduces area and enables mutually exclusive subsystems to share the same physical space on a chip. It also reduces complexity, which usually results in faster circuits and lower power consumption.

The work in this PhD targets the exploration of the DPR feature in recent FPGAs in the advantage of an automotive and multimedia system. We target a Driver Assistant System (DAS) system based on a Multiple Target Tracking (MTT) algorithm as our automotive base system. On the other hand, we have selected the H.264 encoder as a multimedia targeted system. Both applications were fully detailed and analyzed before introducing our results and benefits of the use of DPR in each.

Starting with the automotive application, in this part of work a dynamically reconfigurable filtering hardware block for MTT applications in DAS was presented. Our system shows that there will be no reconfiguration overhead because the system will still be functioning with the original configuration until the system reconfigures itself. The free reconfigurable regions can be implemented as improvement blocks for other DAS system functionalities. Two approaches were used to design the filtering block according to driving conditions. The first approach was related to the number of targets (2 configurations) while in another approach, DPR was used on the basis of targets' proximity and danger level (4 configurations).

Regarding the H.264 multimedia system standard, we proposed a dynamically reconfigurable H.264 motion estimation computational unit whose architecture can be modified to meet specific energy and image quality constraints. By implementing 16 reconfigurable regions, we were able to support multiple configurations each with different levels of accuracy

and energy consumption. Image accuracy levels were controlled via application demands, user demands or support demands. Using a reconfiguration heuristic, our system can support up to 35 different configurations. With a maximum saving of up to 51% in energy consumption, the system can support all block sizes in an H.264 encoder. The data delivery was optimized to support a fully parallel architecture using a memory architecture that has negligible effect on quality.

Résumé

Les processeurs programmables sont largement utilisés dans la réalisation des systèmes embarqués en raison de leurs jeux d'instructions dédiés et leurs caractéristiques micro-architecturales. Ces caractéristiques leurs permettent d'atteindre de hauts niveaux de performance et d'efficacité énergétique. Cependant, les délais de plus en plus courts de mise sur le marché, les coûts de conception et de fabrication élevés et les standards qui changent rapidement exigent un investissement coûteux et risqué pour les fabricants de microélectronique.

Pour surmonter ces problèmes, les concepteurs de systèmes embarqués s'appuient de plus en plus sur les circuits reconfigurables (ou FPGA pour *Field Programmable Gate Arrays*) en tant que plateformes spécifiques de conception. Les récents FPGAs fournissent un haut niveau d'intégration des éléments logiques et un ensemble riche et complet de composants matériels intégrés (ou IP) qui facilite la mise en œuvre de systèmes complexes. Intrinsèquement, ils sont aussi flexibles et peuvent être facilement et rapidement modifiés pour répondre aux besoins des différentes applications. Les FPGAs supportent aussi la réutilisation de blocs IP standards ou personnalisés, ce qui diminue encore plus les coûts et les temps de développement. Néanmoins, ces FPGAs sont généralement plus lents et consomment plus de puissance électrique que les circuits intégrés pour les applications spécifiques (ASICs). Ces deux inconvénients peuvent restreindre leur utilisation à un domaine d'application étroit. Cependant, les récentes avancées dans les architectures FPGA, telle que la reconfiguration partiellement dynamique (ou DPR pour *Dynamic Partial Reconfiguration*), aident à combler ce fossé. La DPR permet à une partie du système FPGA d'être reconfigurée en cours de l'exécution de l'application. Ceci permet d'avoir une plus grande adéquation entre les besoins des applications exécutées et l'architecture du système. La DPR autorise aussi les sous-systèmes mutuellement exclusifs à partager le même espace physique sur la puce. Cela réduit également la complexité et offre la possibilité de concevoir des circuits plus rapides et une consommation électrique modérée.

Le travail de cette thèse vise à exploiter les caractéristiques de la DPR pour les récents FPGAs pour supporter des applications de sécurité routière (ou DAS pour *Driver Assistant System*) et des applications multimédias où nous avons sélectionné l'encodeur H.264 comme exemple illustratif.

Pour l'application DAS, un filtre hardware et reconfigurable dynamiquement a été présenté. L'architecture du DPR-DAS que nous proposons ne provoque aucune surcharge de

reconfiguration. En effet, comme le système fonctionne avec la configuration d'origine jusqu'à ce que le système s'auto-reconfigure, il n'y a pas de temps morts entre les opérations de reconfiguration. Les régions reconfigurables libres peuvent être utilisées entant que blocs pour améliorer les performances des autres fonctionnalités du système DAS. Deux approches ont été utilisées pour concevoir le bloc de filtrage en fonction de la position du véhicule et de son environnement. La première approche est basée sur l'analyse du nombre d'obstacles au devant du véhicule pour choisir la meilleure architecture du filtre. Dans l'autre approche, la DPR utilise la distance du véhicule par rapport aux différents obstacles et le niveau de dangerosité des obstacles (quatre configurations) pour choisir la meilleure architecture du DAS. Cette approche pourra être facilement étendue pour prendre en compte d'autres informations sur l'environnement du véhicule pour choisir l'architecture du DAS la plus performante offrant ainsi un haut niveau de sécurité.

Concernant l'application H.264, nous avons proposé une nouvelle architecture de l'unité de mesure d'estimation du mouvement (ou ME pour *Motion Estimation*). Notre architecture est reconfigurable dynamiquement. Ce qui permet de répondre rapidement et automatiquement à des contraintes spécifiques d'énergie et de qualité d'image. Par l'utilisation de plusieurs régions reconfigurables du FPGA, nous avons été en mesure de mettre en œuvre plusieurs architectures offrant différents niveaux de qualité d'image et de consommation d'énergie. En fonction des besoins de l'utilisateur, le système choisi la meilleure configuration matérielle pour consommer le moins d'énergie électrique. En utilisant une heuristique de reconfiguration, notre système peut supporter jusqu'à 35 configurations architecturales différentes du ME avec une économie d'énergie électrique maximale de 51%. Le système peut supporter en outre toutes les tailles de bloc pour un encodeur H.264. Ces résultats ont été obtenus aussi grâce à la mise en ouvre d'une nouvelle architecture mémoire pour assurer un large bande passante des données mémoire.

Table of content

List of figures

List of tables

1

Introduction

Since the invention of the first electronic device, and humans were investigating the possibility of having more functionalities, less complexity, smaller device sizes and higher efficiency. All was done to provide speed, reliability and high performance. In this work, we explore two applications implemented in a state of the art electronic device technology. Speed, reliability and high performance for these two applications are well exploited and detailed. We firstly present a general introduction to the history of the technology evolution and our targeted technology. Following that, we present our two targeted applications and introduce how these applications can perform faster, efficiently and more reliable.

1.1. General introduction

In 1946, the first computer was invented in the University of Pennsylvania's Moore School of Electrical Engineering. This first general purpose electronic computer was called: the Electronic Numerical Integrator And Computer (ENIAC) with a cost of around 6 million dollars [ENIAC] [H.GOLDSTINE 1993]. The ENIAC contained tens of thousands of capacitors and resistors, weight around 27 tons, consumed around 150 KWatts in power and took a 167 m^2 in space area. Ever since this invention, the race to the most dense, fastest, smallest and lightest processor or on-chip electronic computational unit has begun. For example, the industry of processors has reached 100 of millions of transistors on a single chip until the mid 2000s. With a technology smaller than 40 nm and a power dissipation of less than 2 Watts, very high speed computations were achieved as can be shown in Figure 1.1.

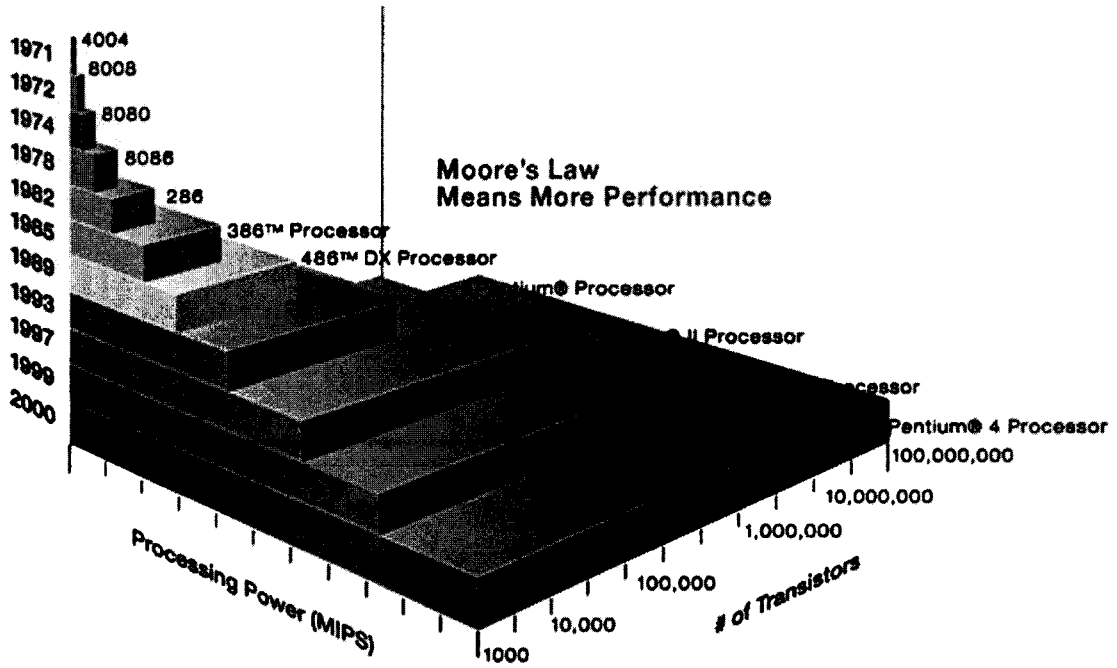

Figure 1.1 – The number of transistors in Intel microprocessors growth associated with Moore's Law [MOORE].

In addition to processors, researchers investigated more methods to accelerate the execution of applications, in other methods than the software programming of processors. One of the most interesting implementations that can accelerate the execution of a processor was the use of hardware accelerators aside of a processor system. The first hardware accelerators were implemented on Very Large Scale Integration (VLSI) fabric basis. Firstly emerging in the 1970's, these devices were mostly used in the implementation of digital signal processing, analog signal processing or both [E.VITTOZ 1994]. Although microprocessors are VLSI fabric based, in the 1970s, VLSI has been introduced as fabric for designing separate hardware computational system-on-chip units. Such devices were manufactured for specific purposes and were not possible to change their functionality once manufactured.

With the growing of technology, the integrated circuit hardware computational units have prospered as well. Following the VLSI fabric in the 1970's, Application Specific Integrated Circuit (ASIC) has been firstly introduced in the early 1980's. With more transistor density than in the VLSI fabrics, ASICs, in recent years, can now have full processors, memory blocks and large building blocks. The configuration of an ASIC is simpler to modify when compared to that of a VLSI. However, unlike VLSI, the fabric does not need to be refabricated but still can only be modified by the manufacturers [P.NAISH 1988].

Following the history and progress of on-chip computational hardware units, Field Programmable Gate Arrays (FPGAs) have been introduced in the late 1980's early 1990's. With more integrated transistors, these devices have reached hundreds of thousands of transistors in recent years. Our work is based on the use of FPGAs as a platform and fabric for our implementations. Hence, we will present a short summary about FPGAs as an introduction to our motivations and contributions.

FPGAs, similar to VLSI and ASIC fabrics, have been accumulating transistors and following the technology in transistor size reduction (Figure 1.2). One major advantage FPGAs have over ASICs is the ability to perform countless reconfiguration of the fabric without the need to go back to the manufacturer. Although FPGAs consume more power than ASICs, they overcome ASICs in terms of flexibility. In the early years also, ASICs outperformed FPGAs in speed and task execution. However, recently FPGAs have improved in size and performance to match ASIC and SoC tasks. Figure 1.2 summarizes a brief history of FPGAs showing the number of transistors and memory blocks growth over the past decade. In Figure 1.2 we also show the technology in size reduction progress over the past decade as well.

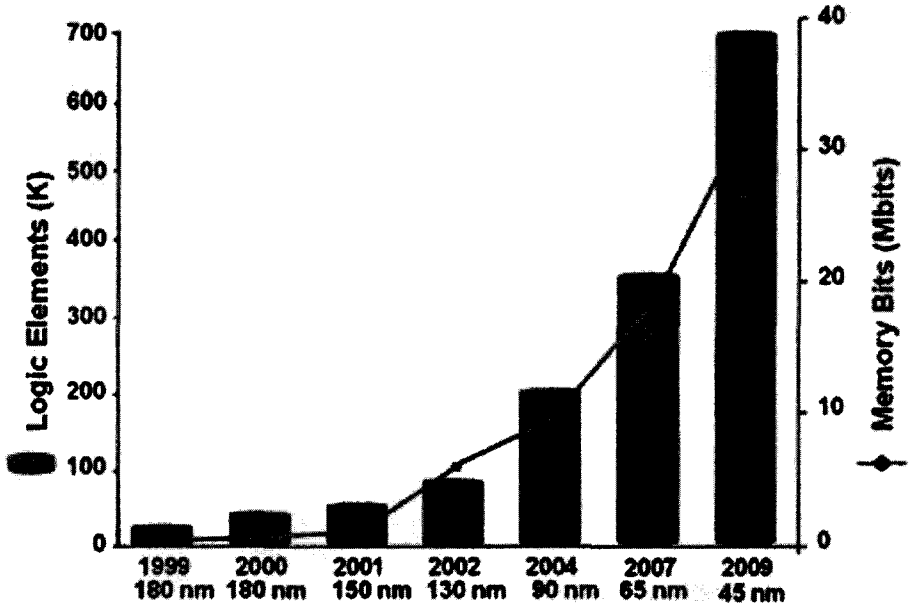

Figure 1.2 – FPGAs' last decade progress and improvements in terms of memory size, number of logic elements and the transistor technology size.

FPGAs can be used to implement any logical function that an ASIC could perform. FPGAs also contain programmable logic components called "logic blocks" or "configurable logic blocks" (CLB), and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together". Also, nowadays, FPGAs support the ability to partially reconfigure part(s) of its logic dynamically on the fly without the need to stop the main system's execution. While the system is running Dynamic Partial Reconfiguration (DPR) allows part of the FPGA to be replaced by another needed computational unit or hardware accelerator. The change into another hardware accelerator within a reconfigurable predefined region is based on application demands. This ability can better utilize hardware resources by enabling the possibility to have multiple configurations using the same hardware.

For example, a system can be running two tasks in parallel: Task A and Task B (Figure 1.3). At time t_2 the system is asked to perform Task A and Task C this time. Instead of spending time t_1-t_0 to make a full reconfiguration of the FPGA to have configurations A and C, the system can only partially reconfigure configuration B to become configuration C while configuration A is still running Task A. In this case, partial configuration allowed keeping the

system running on Task A while spending a smaller reconfiguration time to implement configuration C (t_3-t_2).

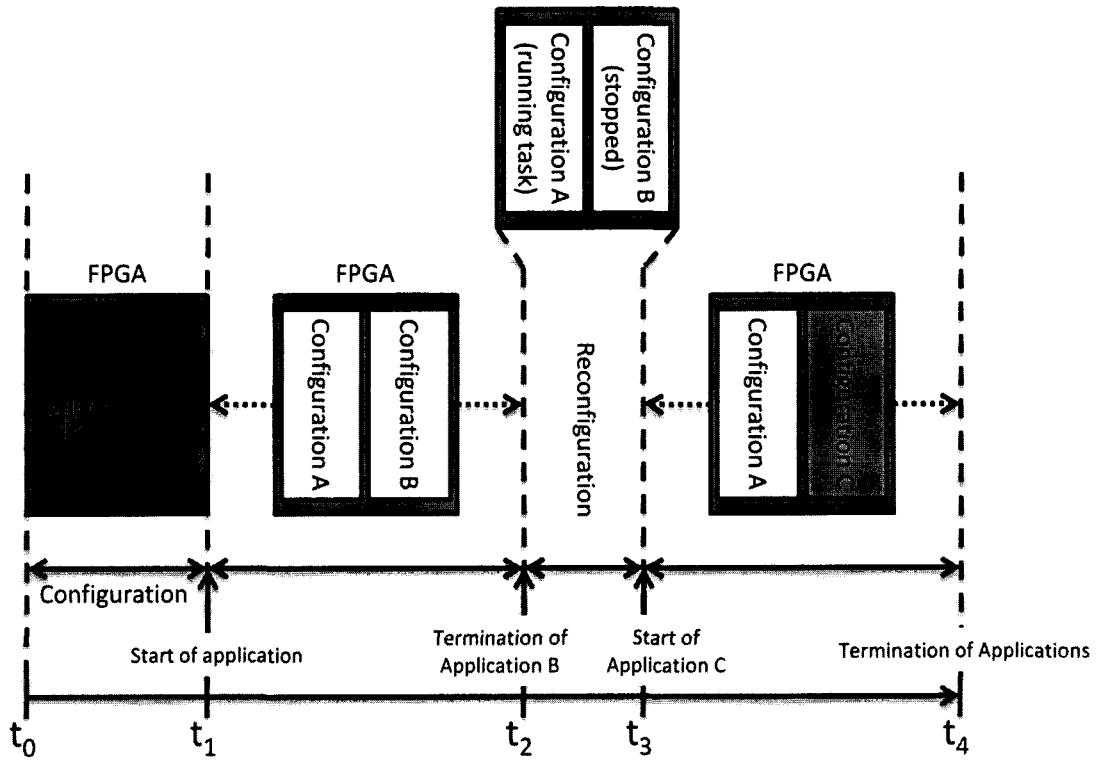

Figure 1.3 – A FPGA based system partially reconfiguring part of its fabric to support a new configuration while the other part is still running its mapped tasks.

FPGAs have been used for many different applications. Since FPGAs can support parallel hardware accelerators functionality, applications such as cryptographic algorithms and code breaking best exploited this massive parallelism feature on this fabric. Other applications such as Software Defined Radios (SDR), Driver Assistant Systems (DAS) and multimedia applications also made use of FPGAs implementations. However, only recently have such applications been explored on the basis of DPR and were found to be beneficial in a variety of matters. In this dissertation, we present the work exploration of DPR in both a DAS application and another on multimedia H.264 application systems. The reason for this choice is due to the fact that both DAS and multimedia application systems are part of the automotive application systems domain.

1.2. DPR in a DAS system exploration

Driver Assistance Systems (DASs) have become a widespread class of automotive applications in nowadays commercial vehicles. They lend even more confidence to driving and improve road safety in stressful driving conditions (e.g. at night or in bad weather). Adaptive cruise control, radar aided automatic proximity control, and navigation systems are examples of well-known range of high-tech DAS systems. Every DAS system is decomposed

of three stages: The *capturing* stage, the *treatment* stage and the *restitution* stage as can be seen in Figure 1.4. In the *capturing* stage, a sensor is mounted on the user's car to capture inputs to the DAS system. Such sensors can be a camera or radar. In the *treatment* stage, the data are taken from the capturing phase and applied to a set of algorithms that defines the functionality of the DAS system. Some algorithms are image processing based, like when using a camera sensor. In the *restitution* phase, the output from the treatment stage is treated before delivering them in voice, image, or mechanical action to the user.

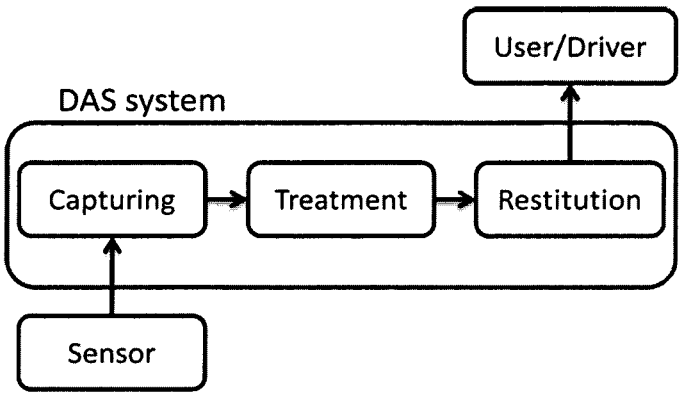

Figure 1.4 – Functional three stages of a typical DAS system.

Traditionally, automotive systems have been designed using 8- and 16-bit microcontrollers. However, increasing levels of complexity and computational demands in automotive applications are forcing a move to more powerful processors, DSPs, and even ASICs. In recent years, FPGAs also have been used to implement various automotive subsystems. With their inherent support for parallelism, high logic densities, and rich sets of embedded hardware components, FPGAs are very well suited candidates for implementing computationally demanding applications. Their flexibility, programmability, and fast design turnaround times also enable system designers to quickly introduce new features or update existing ones in response to changing requirements or new standards.

Our work is based and centered on the treatment phase of a radar based DAS system using FPGAs. We are working on a target tracking DAS feature which aims for an early warning and collision avoidance system onboard a vehicle. The purpose of target tracking is to collect data from the radar sensor field of view (FOV) containing one or more potential obstacles of interest and to partition the sensor data into sets of observations, or tracks. We use a radar sensor in our application because it has the advantages of longer range as compared to camera based systems. It performs better in bad visibility conditions and has lower computational requirements. Moreover, radar helps to detect obstacles at longer distances and hence ensures longer reaction time for vehicle drivers. Our contribution lies in the exploration of the use of DPR in FPGAs to implement a dynamically reconfigurable system for automotive target tracking DAS system and present results for two different possible implementations. Our work is novel in terms of being the first work that targets the use of DPR in a radar based DAS system.

1.3. DPR in a H.264 system exploration

The H.264 is a widely used video compression standard for recording, compression and distribution of high definition videos. The standard is used in a wide range of applications due to its high compression rate that can reach up to 50%. H.264 has also proven its efficiency and advantages in encoding and decoding video sequences and has therefore become a feature of mobile devices. H.264's superior compression rates have made it an attractive solution for mobile devices due to their relatively scarce storage space. However, mobile devices have also demands for real time video encoding/decoding of video sequences. This made researchers interested in the implementation of some, if not all, of the H.264 standard on hardware in order to meet real time constraints.

As an introduction to the H.264 standard, a very general overview of an H.264 multimedia system is shown in Figure 1.5. Like any other multimedia system, the H.264 is decomposed of an encoder and a decoder. The decoder phase performs exactly the opposite functions of that of the encoder. In the encoder, the first stage is the *prediction* stage. In this stage, the encoder predicts the value of pixels of smaller blocks in a frame based on previously coded data. The predictions from the *prediction* stage are then quantized in the *transform* stage using the Discrete Cosine Transform (DCT) integer transform. In the final stage, the video coding process produces a number of values that must be encoded to form the compressed bit stream.

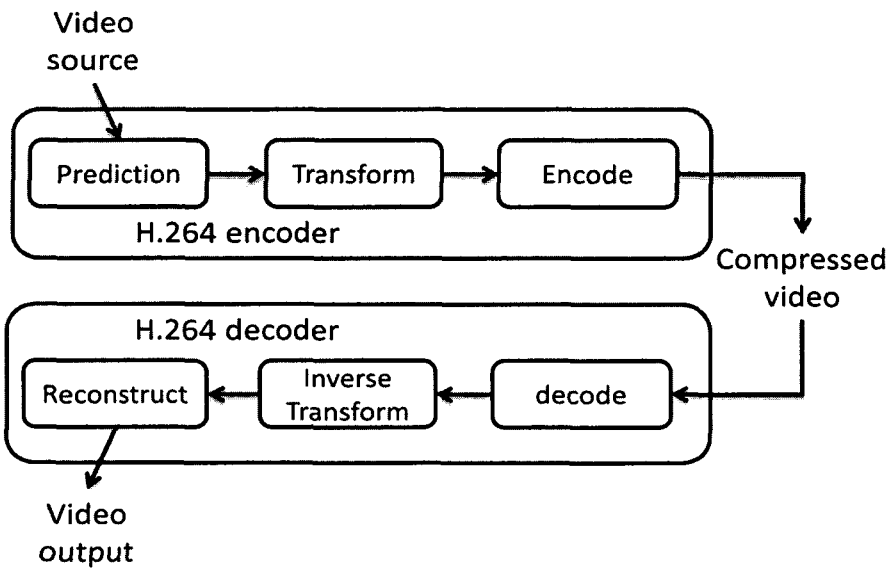

Figure 1.5 – H.264 encoder's and decoder's building blocks.

Our work targets, using the DPR feature in FPGAs, the implementation of a dynamically partial reconfigurable Motion Estimation unit, part of the encoder's *prediction* phase, inside an H.264 encoder. We present the computational complexity inside this unit and the implementation possibilities on an FPGA. Unlike other related work, we use the DPR feature not to replace our computational implemented units with other modules but rather to reduce

the energy consumption. The flexibility of our system in terms of image quality and power was the basis of our reconfiguration heuristics. We implemented our system on fabric and deducted results in terms of execution times, power measurements, reconfiguration times and acquired image qualities.

1.4. Plan of the document

This document is organized as follows: After the introduction, in the next chapter, chapter 2, we present the most recent related FPGA work in three sections. In the first field we cite DPR optimizations and research in literature. In the second section we cite the related work that targeted the implementation of a DAS system on FPGAs and specifically on DPR basis. In the third and last section of this chapter, we mention the most related work to our contribution regarding the implementation of H.264 multimedia block(s) on FPGAs using the DPR feature. In chapters 3 and 4, we present our contribution in exploring DPR for a DAS system. After presenting a short introduction of the problem, we discuss in details our targeted DAS system and the functions associated in it in chapter 3. In chapter 4, we show our contributions of using DPR in DAS safety systems by presenting two possible implementations showing the advantages and results gained in each implementation. In chapter 5, we present, like in chapter 4, our contribution but in exploring DPR for an H.264 multimedia encoder. After a short introduction of the base H.264 system, we discuss our DPR exploration inside this multimedia system. Our contribution is presented via showing results of our approach and implemented system. Finally, conclusions and perspectives are drawn in the last chapter, chapter 6.

2

Literature review

In the last few years, and following the introduction and progress of the DPR feature in FPGAs, researchers have proposed several improvements to this feature. These improvements were in terms of the way reconfiguration is performed and in speeding up this process. Here, we cite the most important work that has been done in this area firstly. Then, we present some recent most related work to the implementation of: first, a DAS system and second, an H.264 multimedia system on FPGAs using DPR. We cite important works in literature for each of these systems and discuss their merits and demerits. This presentation motivates our system's implementation and highlights our proposal in the research domain.

2.1. Dynamic Partial Reconfiguration related work

FPGAs, integrated circuits designed to be reconfigured by customer demands after manufacturing, has been firstly commercially introduced in 1985 by Xilinx co-founders Ross Freeman and Bernard Vonderschmitt [P.CLARKE 2009]. Starting up with mere 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs), the FPGA domain has reached millions of gates in the early 2000s [RECENT_FPGAS]. During the period between 1985 and mid 1990s, Xilinx [XILINX], the leader company in FPGA design and fabrication, was controlling the market unchallenged. However, Starting from early 1990s, new companies emerged and started to share the market with Xilinx such as Altera [ALTERA] and Actel [ACTEL].

Many improvements have been added to FPGAs ever since their introduction. One new feature is called: partial dynamic reconfiguration. Partial reconfiguration, as the name implies, is the ability to reconfigure a partial configuration space on an FPGA fabric. This feature allows the swapping of modules on the FPGA without the need to totally reconfigure the entire design area. Xilinx were the first to introduce the feature of partial reconfiguration

to their FPGA devices at around the mid 1990s. Their first devices to support DPR were the Spartan-III and Virtex-II Pro FPGA devices back then [SPARTAN-III] [VIRTEX-IIPRO]. The work done in this dissertation is based on the Xilinx FPGAs and hence, most of the related works cited are based on Xilinx hardware verification as well.

2.1.a. DPR design flows

Partial reconfiguration has two main design flows: The Modular Design Flow and The Early Access Design Flow [C.BOBDA 2007] [PR_FLOWS]. Initially proposed for design engineers, the Modular Design Flow was not initially intended to support partial reconfiguration. It was used by leader designers to divide the FPGA into several parts, each having the desired amount of hardware resources needed by that block's designer engineer. The communication to these regions was based on fixed communication channels called Bus Macros (BM). The Modular Design Flow is shown in Figure 2.1.

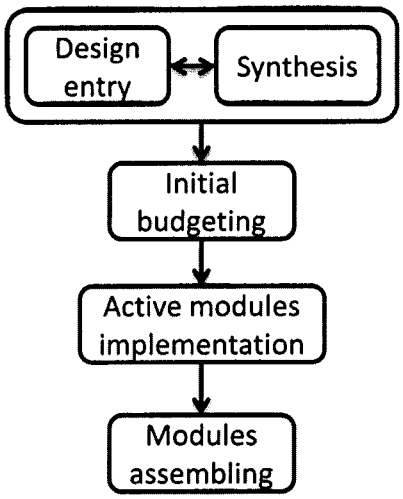

Figure 2.1 – Partial Reconfiguration's Modular Design Flow four steps.

The Modular Design Flow is divided into four steps [C.BOBDA 2007]. These steps are the following:

1. The *design entry* and *synthesis*. In this step a top level design is implemented using a Hardware Description Language (HDL). The top level includes all the global logic and I/Os. The modules that will be mapped as reconfigurable regions are assigned as black boxes in the top level design. Also, signals that connect these black boxes and I/Os are instantiated in this step.
2. The *initial budgeting*. In this step, initial design constraints are assigned to the project. These constraints contain: BM position, hardware modules space reservations, clocks, power, ground signals and timing constraints.
3. The *active modules implementation*. Here, the previously implemented modules are assembled into one top level design. The full design can be used to generate a full system bit stream that contains the initial configuration of the system.
4. The *module assembling*. After the top level initial design has been implemented, other modules can be designed for targeted initially implemented modules. Partial

reconfiguration is achieved by replacing modules within the same black box and during runtime.

The partial reconfigurable regions in this flow were instantiated on an FPGA column overtaking the whole FPGA height. Even without the need for all the reserved resources for each module, regions must span over the height of the FPGA.

The second partial reconfiguration design flow is the Early Access Design Flow [EARLY_ACCESS]. This new design flow is a Xilinx updated and enhanced Modular Design Flow. In this flow, scripts have been added to the design tools in order to better support the partial reconfiguration feature without the need to go in the design details. Graphical interfaces have been added to automatically trigger design scripts and insatiate different commands such as: new module instantiation, bus macros insertion, I/O direction... etc. In addition to the improvements over the original Modular Design Flow, Early Access Design Flow added more Bus Macros with synchronization and directional pointers as well. One other major enhancement was the partial reconfigurable region instantiation over the FPGA. The new flow allowed defined block wise partial reconfigurable regions that does not need to span over the full chip height. The Early Access Design Flow steps are the same as those in the Modular Design Flow but in a more user friendly manner.

One interesting work that summarizes the Xilinx tool flow, architecture and system integration was presented by [M.HUBNER 2006]. The work presents a tutorial to simplify for researchers how a partial reconfigurable system can be implemented. However, the targeted design is relatively old (Virtex-II Pro Xilinx FPGA board). Recent FPGAs, such as the Virtex-4 and Virtex-5, are different in design and fabric. Not only the fabric has changed, but also the design tools versions have changed and had switched from one design flow to another. There is merely any publication regarding a global partial reconfiguration tutorial due to the fact that such systems are directly related to the new software and tools versions in addition to the devices used. Hence, only basic knowledge can be acquired from the tutorials. This, in addition to an experience on the tools and HDL languages, can build a proper knowledge on how to design a partial reconfigurable system on any FPGA supporting device.

2.1.b. Architectural explorations of DPR systems

Partial reconfiguration is highly dependent on the targeted device FPGAs. For this reason, researchers tried to investigate enhancements to the reconfiguration process in both the theoretical level and the architectural level. The research was mainly aimed to target partial reconfigurable aspects that have the least dependency on the tools used or the target devices. Once an idea is established on hand, partial reconfigurable devices were used for verifications purposes. Most, if not all, of the research related work was verified on Xilinx boards and tools since they were the only devices supporting the partial reconfiguration ability. In this section, we present the most interesting work done in the field of architectural exploration of DPR systems.

Some researchers investigated the possible usage of reconfigurable regions to scale a hardware accelerator attached to a running system. By scaling and scalability, here we mean

implementing more processing elements executing in parallel. Hence, the system's enhancement lies in the reduction of its execution time but on the cost of more hardware resources consumption. Since scalability is one important advantage of the use of partial reconfiguration, a lot of work has been cited that made benefit of such usage for different applications. In this section however, we will mention few of the work that has been done on the reconfiguration exploration level.

In the work of [A.OTERO 2010], partial reconfiguration has been introduced as an enabling feature to have a scalable systolic coprocessor based system as shown in Figure 2.2. The system was mainly tested for multimedia applications on a System on Programmable Chip (SoPC) architecture. The work presented showed that a system's architecture can adapt the scalability of its hardware accelerators to match real time requirements or application requirements using a scalable systolic partial reconfigurable architecture. The system highlights the advantages of scalability when using partial reconfiguration. However, the work of [A.OTERO 2010] reserves a considerable amount of resources only for scaling a certain hardware accelerator. Their system does not make use of their free reconfigurable regions for other purposes such as enhancement blocks in other system's parts.

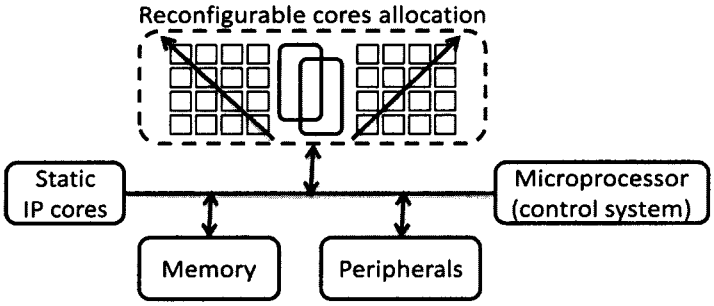

Figure 2.2 – Scalable systolic coprocessor system's general overview proposed by [A.OTERO 2010].

To the authors' knowledge, one of the most interesting work that has been done on the architectural level of partial reconfiguration has been proposed by Microsoft embedded and reconfigurable systems research group [MSR]. One of their earliest works was cited in [J.CARVER 2008]. In this work, an extensible processor was implemented with the ability to automatically floor-plan and relocate partial reconfigurable regions and hence, automatically instantiate partial reconfigurable regions. The process includes modifying a reconfigurable region generator according to the target FPGA and the reconfigurable region specs as shown in Figure 2.3.

Though the work sounds very interesting in its general idea, however the work targets only one specific FPGA device, the Virtex-4 Xilinx FPGA. Although the future work carries the idea of supporting more devices, however, such systems are highly dependent on the bit files formats for each FPGA that are only provided by the manufacturer of the FPGA devices. Since partial reconfiguration, from its introduction, is a highly device dependant feature, investigating the automatic instantiation of such systems could be only industrial beneficial on its entire.

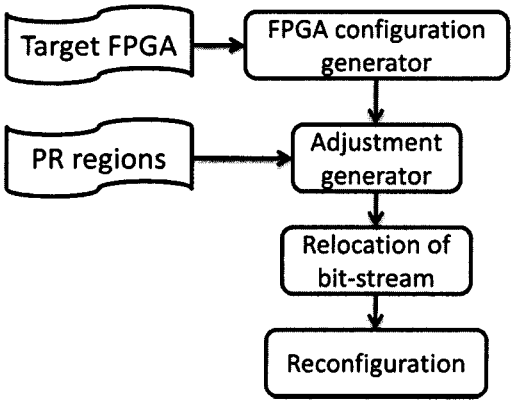

Figure 2.3 – Relocation flow of PR regions proposed by [J.CARVER 2008].

2.1.c. Hardware support for a fast reconfiguration process

One important aspect of partial reconfigurable system is their ability to be reconfigured on the fly and on demand. However, the reconfiguration process on its own incurs a considerable time and reconfiguration overhead. For some real time applications, this might introduce some failure in meeting the real time constraint. Hence, researchers were interested on the investigation of fast new methods to speed up the reconfiguration process, even to support the maximum reconfiguration access port's optimal configuration speed of 400 Mbytes/sec (T_{ICAP}).

One interesting work, in this field, is found in [M.LIU 2009]. In this work, the authors investigated various implementations of the Internal Configuration Access Port (ICAP) on a Xilinx FPGA. They found out that a Block RAM (BRAM) based ICAP can reach around 370 Mbytes/sec ($0.92 \times T_{ICAP}$). Their architecture design uses BRAMs to form a big buffering space inside the ICAP as shown in Figure 2.4. However this implementation requires 32 BRAMs (around 50% of the BRAMs 16 Kbits found in a Virtex-4 FX20 FPGA) to be implemented, which is considered as a big number and a high resource consuming implementation.

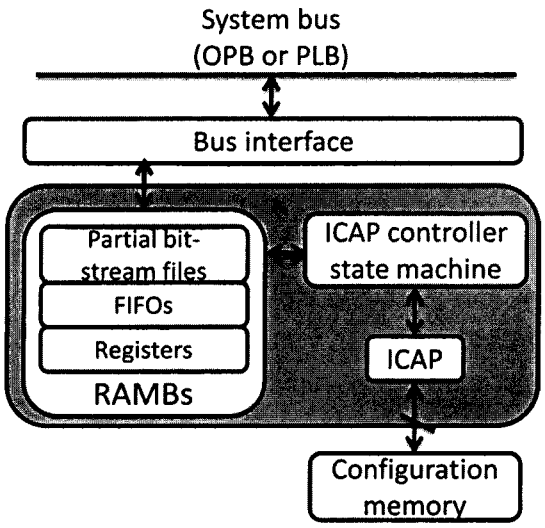

Figure 2.4 – BRAM based ICAP architecture as proposed by [M.LIU 2009].

In [M.HUBNER 2010] we cite another approach to speed up the reconfiguration process. Here, the authors targeted having a simpler implementation of the ICAP controller that can both speed up the classic ICAP speed and increase the flexibility of a FPGA based processor. The basic idea of this implementation is to place and connect the ICAP core as near as possible to the processor responsible of performing reconfiguration. The authors used the Fast Simplex Link (FSL) as the direct near interface between the ICAP and the processor as illustrated in Figure 2.5. In addition to simplicity, the authors achieved a speed of around 310 Mbytes/sec ($0.77 \times T_{ICAP}$) with no use of any BRAMs memory resources at all. However, although the implementation was simple and straightforward, the speed acquired is still not close to the optimal speed of an ICAP.

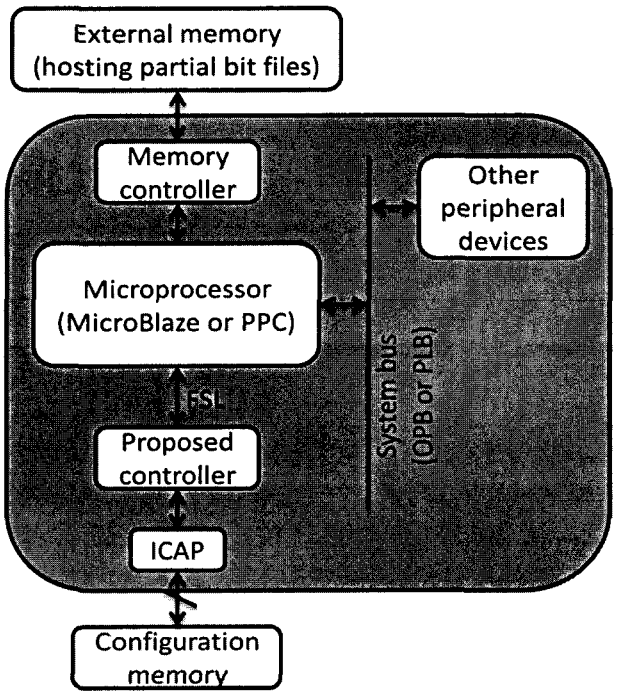

Figure 2.5 – An FSL based ICAP controller proposed by [M.HUBNER 2010].

To the author's knowledge, the work in [S.LIU 2010_1] is considered the most efficient implementation of an ICAP intelligent controller in terms of speed and resource utilizations. In [S.LIU 2010_1], the authors used a Direct Memory Access (DMA) based ICAP controller to achieve a throughput of more than 399 Mbytes/sec ($0.99 \times T_{ICAP}$) which is the closest to the optimal among different literature works. Their system aimed in maintaining a bridge between an external memory hosting the partial bit files and the ICAP. In the system they proposed, the external memory was a Static Random Access Memory (SRAM) connected to the ICAP via a master-slave DMA engine and thus gaining high reconfiguration speed while avoiding the busy system bus as shown in Figure 2.6. The only drawback of this work is that there was no interesting application that could benefit from such high reconfiguration speed presented.

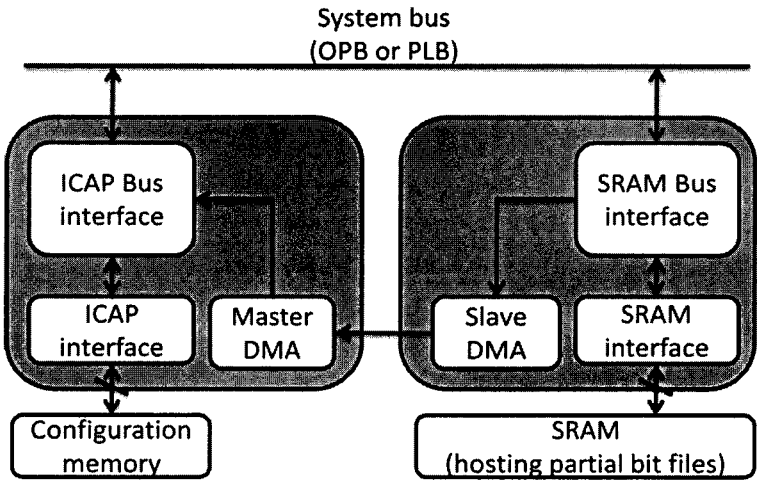

Figure 2.6 – A DMA-engine based reconfiguration accelerator as proposed by [S.LIU 2010_1].

2.1.d. Fast reconfiguration beneficial study

Researchers have fully investigated the enhancement of the ICAP throughput. In contrast to this investigation, [S.LIU 2010_1] have reached an almost optimal ICAP throughput which is around 400 Mbytes/sec. Although the authors did not present a real application that could benefit from such high reconfiguration speed, but [S.LIU 2010_2] found another advantage of the high speed reconfiguration process.

In [S.LIU 2010_2], the authors investigated the energy reduction that could be achieved with run-time partial reconfiguration. The authors proves, both in theory and in on-board measurements, that using a high speed reconfiguration component can make partial reconfiguration outperform clock gating in power and energy reduction. The authors used a 64-bit division component and compared the energy reduction by either using DPR or clock gating. The obtained results are shown in Table 2.1, where the first column represents the duration the hardware component was inactive shown in scientific notation.

t_inactive (sec)	DPR (J)	Clock gating (J)
5e-05	-0.028	0
5e-04	-0.026	0.001
5e-03	-0.012	0.01
5e-02	0.128	0.104
5e-01	1.530	1.038
5e+00	15.542	10.38

Table 2.1 – Energy reduction by using DPR and clock gating as cited in [S.LIU 2010_2].

As a conclusion of the work of Table 2.1 and the work of [S.LIU 2010_2], for longer inactive times of hardware accelerators, DPR can gain more energy reduction than clock gating. Although the work of [S.LIU 2010_2] is convincing, there was also no interesting application that could benefit from this energy reduction. The authors presented only a 64-bit division component and applied various tests to it.

The summary and the most important conclusion of the work of [S.LIU 2010_2] is summarized in equation (2.1). Where, energy can be reduced using DPR if the ICAP throughput T_{ICAP} is greater than the product of the reconfigurable region power (P_{pr}) with the size of the partial bit file (S_{bf}) divided by the product of the reconfiguration static components power dissipation (P_{static}) with the inactive time of the reconfigurable region ($t_{inactive}$). Hence if the ICAP throughput is 300 Mbps, even in very bad cases where the reconfigurable region consumed 1 Watts and was inactive for 1 msec, with a static power of 10 Watts and a bit file size of 1 Mbytes, still, 300 Mbps>100 Mbps. Our system where we used the work of [S.LIU 2010_2] has a lot less variables values than those proposed before.

$$T_{ICAP} > \frac{P_{pr} \times S_{bf}}{P_{static} \times t_{inactive}} \qquad (2.1)$$

Researching the architectural and enhancement level of partial reconfiguration is highly dependent on tools and fabric devices used. Hence, researchers have been focused on the following aspects of partial reconfiguration: the presentation of partial reconfiguration and how to implement a base system and the reconfiguration speed enhancements. In our work, we make use of what has been presented in [S.LIU 2010_1] and achieved by researchers and apply it to our base system architecture and show results.

After presenting the research work done on the improvements of the DPR process, we will present the research work related to using DPR for the benefits of certain applications. In the following two sections, two applications' research related works are presented to show how literature made use of the DPR feature in real system implementations.

2.2. Driver Assistant System related work

Different types of DAS systems have been proposed in the last few years. Most of the existing DAS systems have either limited functionalities or are too costly for a large-scale automotive utilization. Among the most popular DAS functionalities, we can cite: adaptive cruise control, lane keep assistance, parking assistance systems and obstacle detection and avoidance systems [PREVENT] [A.VAHIDI 2003]. In the past, most programmable platforms were based on 8-bit or 16-bit micro-controllers. These platforms are unable to efficiently support new processing intensive automotive applications.

Recent research activities concentrate on the use of DAS in complex environments and scenarios, such as detecting bikes, pedestrians and children under changing weather and lighting conditions. The proposed systems in these projects are implemented by different hardware and/or software architectures. From the hardware point of view, dedicated hardwired ASIC to pure programmable processors were used. To offer a good performance/flexibility/cost trade-offs, researchers designed multi-processor system-on-chips (MPSoC) and/or hardwired FPGA-based circuits that are more and more used in DAS systems.

2.2.a. The IMAPCAR and EYEQ2 projects

Two of the earliest and interesting works that implemented a DAS system on an FPGA basis and explored the benefits of such an implementation were proposed in the IMAPCAR [IMAPCAR] and EyeQ2 [EYEQ2] projects. Both the ImpaCAR and the EyeQ2 systems are examples of a fully programmable MPSoC implementation. The architectures are dedicated to automotive security applications using vision systems and are therefore camera-based implementations. The IMAPCAR uses SIMD architecture of 128 processing elements and a four-way Very Long Instruction Word (VLIW) control processor. On the other hand, the EyeQ2 uses two MIPS32 processors for scheduling and controlling the concurrent tasks, and eight programmable coprocessors for vision and vector processing (SIMD). These co-processors are dedicated to DAS specific tasks such as object classification, tracking, lane recognition and filter applications. One of the limitations of these systems is that unused co-processors may consume a lot of resources and energy even when not used. These two architectures provide support for a fixed set of real-time data intensive applications. For this reason, these systems are unable either to accommodate to new applications or to adapt the hardware to the different scenarios.

2.2.b. Radar based DAS systems

In another approach to implement a DAS system, we cite [S.LEBEUX 2006]. In this work, the authors also used a programmable MPSoC implementation in addition to some hardware accelerators attached to the main system. This security based application implementation was based on the use of a radar sensor. In addition to the radar and in order to enhance the system's prediction and obstacle tracking algorithm, an intelligent cruise control Global Positioning System (GPS) was used. Their results show a considerably huge amount of hardware resources usage for such an implementation. On the other hand, hardware resources consumed by processors and hardware accelerators may become obsolete when not used in certain situations. A general overview of the system proposed by [S.LEBEUX 2006] can be seen in Figure 2.7.

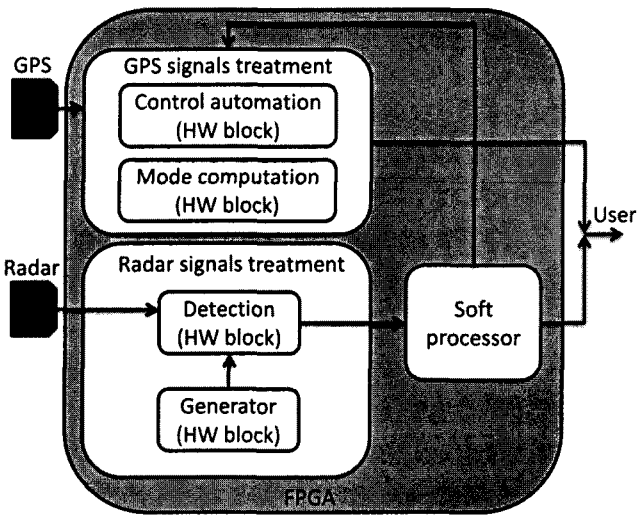

Figure 2.7 – A system overview of an early radar+GPS sensor based DAS system proposed by [S.LEBEUX 2006].

In the same context of a radar based DAS system implementation, we cite [J.SAAD 2009]. In the work of [J.SAAD 2009], a flexible FPGA based DAS implementation was proposed. The main work was done on the signal processing level where the proposed DAS system makes use of new particular radar waveform feature to enhance the capabilities of old adaptive cruise control radars. The system contains few processors in addition to some hardware accelerators, forming a hybrid MPSoC architecture (as shown in Figure 2.8). The proposed system proved its reliability and functionality via field studies and experimentations. The system showed very promising results both in terms of hardware resource utilization and fast execution times that made the system suitable for real time implementations. In contrast to the resource utilization and the fast execution times, the proposed system was mainly based on the treatment of the long or short range used radar. In addition to what have been mentioned, hardware resources consumed by either a not used processor or any hardware accelerator may become obsolete when not in use.

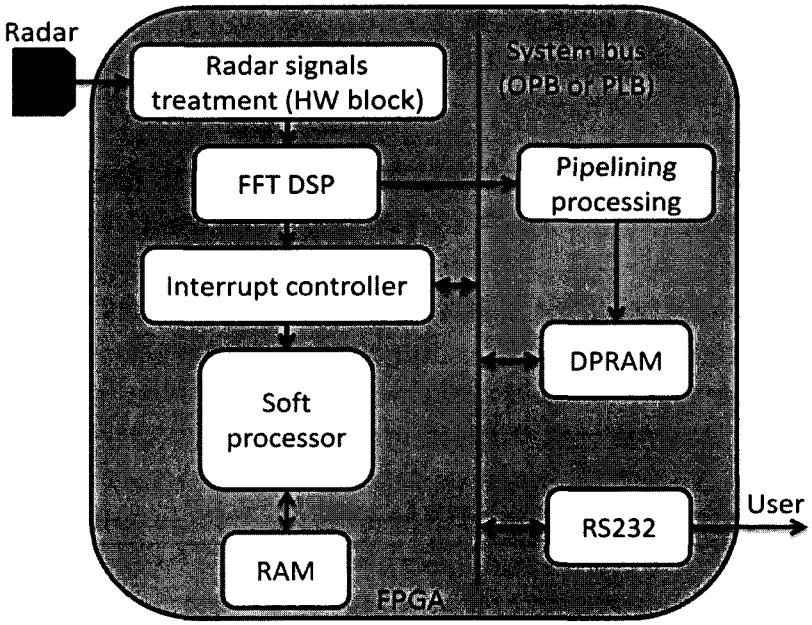

Figure 2.8 – A hybrid radar based DAS system architecture as proposed in [J.SAAD 2009].

2.2.c. Camera based DAS systems

The first work using partial dynamic reconfiguration for automotive applications has been proposed by [C.CLAUS 2007] and [C.CLAUS 2010]. To our knowledge, this is the closest work to ours. In this project, three different situations are considered: highway, tunnel entrance, and inside tunnel. For each situation, a given hardwired co-processor must be loaded and mapped on the FPGA. Two hard cores processors, in the FPGA's static part, were used. The first processor realizes high-level image processing while the second was set in charge of the control and partial reconfiguration management functions. Consequently, when this processor detects a modification of the driving environment, a new image-processing algorithm must be executed and thus a new co-processor is defined. The system overview is shown in Figure 2.9.

In contrast to the work mentioned above, our system uses a radar sensor instead of a video camera. As explained in the following chapter, the types of processing as well as the constraints are different. The use of radar as a sensor in our system has the advantage of longer range as compared to camera based systems. It also performs better in poor visibility conditions (e.g. foggy weather). Even if initially, radar based DAS was only possible at a significant cost, their cost is gradually decreasing making them a cost-effective solution.

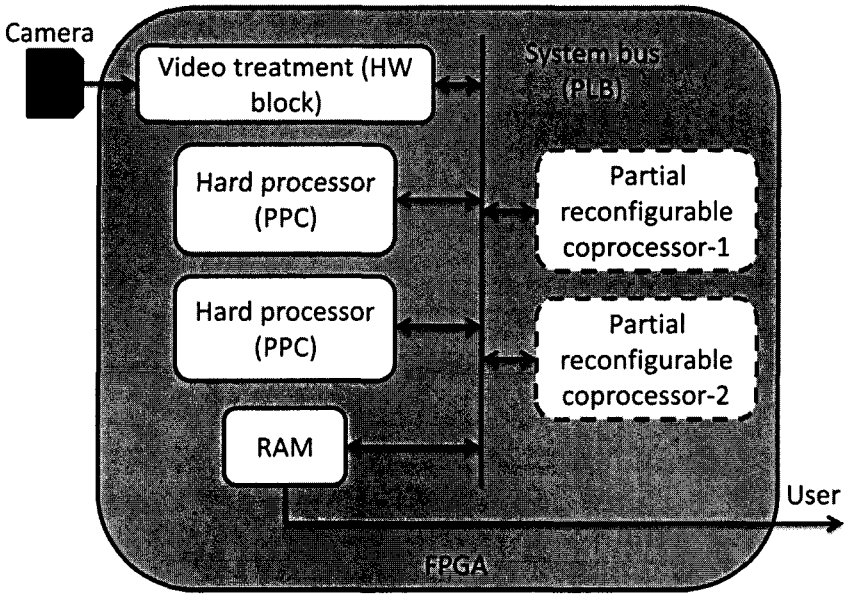

Figure 2.9 – The overview of a DPR DAS system using a camera as a sensor as proposed in [C.CLAUS 2007] and [C.CLAUS 2010].

2.2.d. Multiple Target Tracking radar-based DAS system

Our work is an extension and improvement of an earlier proposed DAS system in [J.KHAN 2008] and [J.KHAN 2009]. The base system was a utilization of an MPSoC architecture on FPGA to support DAS applications. The system was a radar based DAS sensor system. It used a Multiple Target Tracking (MTT) algorithm based on radar data input acquired from the driving environments. The tracking algorithm was divided into functional blocks mapped on 23 dedicated soft core processors as shown in Figure 2.10. The MPSoC system was connected via a mesh of buses in order to provide valid data transfer among different processors.

The proposed base architecture was validated via simulated data sets and proved to be reliable and functioning correctly. However, in contrast to the fully functional system, the proposed architecture consumes a large amount of logic gates and hence a high hardware resource utilization. Also, the system had to be pipelined in three stages in order to meet the real time constraints. In addition to that, this system does not offer DPR and thus cannot adapt the architecture to different driving scenarios or application requirements.

2.2.e. Summary

To summarize the work done in the past few years and to show what our proposed system will offer, we deducted Table 2.2. In Table 2.2 we show for each cited work, includ-

ing ours, the important features and we highlight the differences among them. In the table we present the sensor used in every DAS system. The used architecture is also mentioned in the table, whether it was based on a multiprocessor architecture or a hybrid combination of processors and hardware accelerators. We also show what the cited work was treating inside the DAS system: image processing, radar signal processing or a treatment algorithm such as the MTT algorithm. In addition to all the mentioned properties, we show which of the related work used the DPR feature.

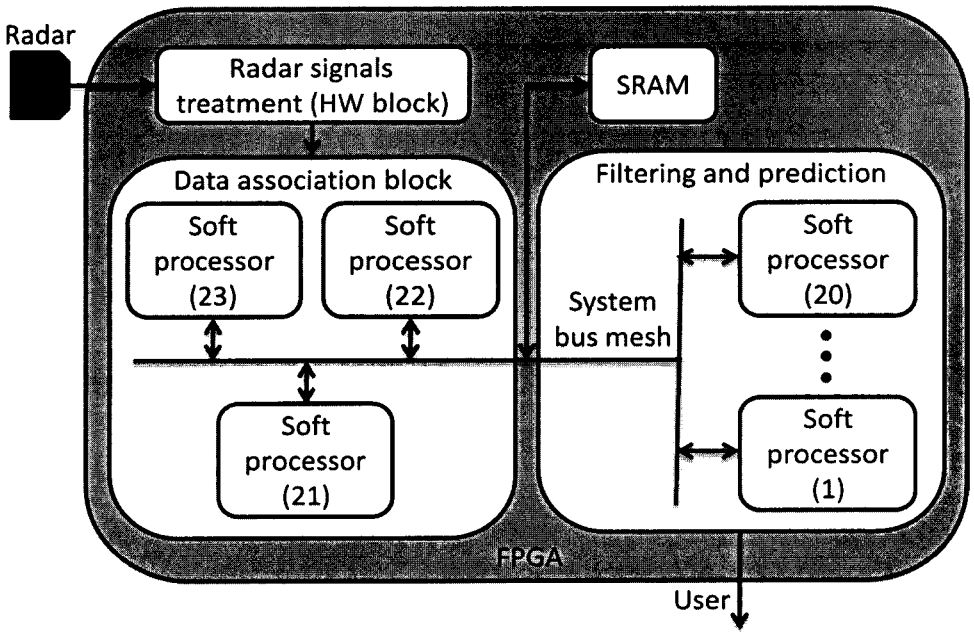

Figure 2.10 – The system architecture of an MPSoC MTT-based DAS system using a radar as a sensor as proposed in [J.KHAN 2009].

	Sensor	Architecture	Treatment	DPR
[IMPACAR]	Camera	MPSoC	Image proc.	No
[EYEQ2]	Camera	MPSoC	Image proc.	No
[S.LEBEUX 2006]	Radar	Hybrid	Signal proc.	No
[J.SAAD 2009]	Radar	Hybrid	Signal proc.	No
[C.CLAUS 2010]	Camera	MPSoC	Image proc.	Yes
[J.KHAN 2008]	Radar	MPSoC	MTT	No
Our Work	Radar	Hybrid	MTT	Yes

Table 2.2 – DAS related work main features and properties. The main aspects of the related work in comparison to our work are highlighted and cited in the table.

The main insight of Table 2.2, and to conclude this section, is to show the relationship between our work and the work in the literature citations. Most DAS systems are different, either with the used sensor, the architecture they are built on, which treatment method is being implemented or if they support multiple driving conditions using the DPR feature. In comparison to other work, some used a static radar based DAS system without the use of DPR. On the other hand, some work used a DPR based DAS system, but with the use of a

different sensor device. Our work is considered the first to use a radar based DAS system on the basis of hybrid DPR architecture.

2.3. H.264 multimedia system related work

We will now explore another application from literature that researchers implemented on the basis of DPR in FPGAs, the H.264 multimedia system. Similar to the previous section, we will cite the most important work related to ours and show where our contribution lies in the research community.

At first, when the H.264 multimedia standard has been introduced in 2003, a lot of research has been done to better optimize the functionality of the standard. In the same time, during the last decade, the H.264 migration into hardware has been a very attractive area. Most of which have been capable of implementing complex computational blocks onto ASICs or on FPGAs.

2.3.a. H.264 exploration on ASIC

The use of ASIC platforms has been proven to be very interesting due to their low power consumption and flexible architectures in terms of hardware resources and memory blocks. The early work done on the migration of the H.264 full or partial blocks into hardware design was done using VLSI architectures or ASIC fabrics.

In [S.YEOWYAP 2004], [Y.W. HUANG 2005] and [C.WEI 2007] for example, the authors investigated the implementation of the H.264 motion estimation onto a VLSI ASIC based architecture. Other researchers investigated the implementation of the motion estimation on ASIC fabrics as cited in [M.ELHAJJ 2009]. In this work we aim to explore the implementation of the motion estimation block on FPGA fabrics. This is due to the important role this block plays in the H.264 system's encoder and its direct relationship to the decoder.

Over the past few years, various techniques and methods have been proposed for partial or full implementations of the H.264 standard on FPGAs. The flexibility offered by these devices and the ease of reprogramming them with a variety of hardware resources has made FPGAs a good candidate platform to target. One of the most interesting works that targeted the implementation of the H.264 encoder and specifically the motion estimation is found in [M.SHAFIQUE 2008]. This work has become a reference of comparison. However, this work was based on a MIPS processor platform and not on the DPR basis or hardware accelerators. The work of [M.SHAFIQUE 2008] is different than ours in the following points:

- The cited work is based on MIPS processors while our work is a hardware accelerator unit.
- Our work, in concept, makes use of DPR for energy/performance trade-offs while the cited work is an optimized standard motion compensation unit.

- Our work is not an introduction to a novel motion compensation unit but rather a presentation of how even a simple motion estimation computational unit can trade energy with performance with the aid of DPR

Our work is made up of three major edges related to what has been done: H.264 encoder, FPGAs and DPR. For this reason, in this section we will mention few of the most recent related work to the usage of DPR in the H.264 system encoder/decoder.

2.3.b. Static scalable motion estimation on FPGAs

One of the early works that targeted the H.264's encoder implementation on FPGAs was the work done in [T.MOORTHY 2008]. The author's work was the first to allow the implementation of H.264 motion estimation on an FPGA that can support high definition image resolutions of 1920×1088 pixels. The author used a search range of 48×63 pixels in the implementation and the system can be scaled down to support lower image resolutions and, in turn, reduce hardware blocks usage. The work in [T.MOORTHY 2008] is scalable not by using partial reconfiguration, but rather by reconfiguring the data paths to the processing elements. Hence, by keeping the processing elements as is, re-routing the data sent to each element will result in a new output as will be shown in chapter 5. However, although high resolution image processing was achieved, the cost in hardware utilization was too high.

In [H.SAHLBACH 2010], the authors presented a motion estimation block implementation on FPGAs. Results show high performance achievement based on image resolution of 512×384 pixels and a search range of 32×64 pixels. The basic processing elements were scalable in order to provide a trade-off between the frame processing rate, image resolution or lower hardware resource utilization. Although the authors did not use the feature of partial reconfiguration in their work, the scalability of their system was based on the reconfiguration of data paths. The main processing elements of the system are statically implemented during all times, and depending on the system requirements; the data paths to the processing elements are reconfigured.

2.3.c. H.264 exploration on DPR basis

A handful of work has been targeting the implementation of the whole encoder on FPGAs. One example on this implementation can be cited in [NOVA]. Although a full FPGA implementation of the H.264 encoder can be attractive, however, a huge amount of hardware resources must be utilized. On the other hand, some hardware blocks might become obsolete since they are not functioning all the time. For this reason, in this work our most important point is to mention the usage of DPR in the encoder.

In [M.GUARISCO 2010] the authors used the DPR feature on the H.264 encoder to change from one standard of video adaptation to another. Starting with the Standard Video Coding standard (H.264/SVC) they were able to add few extensions on it in order to migrate the architecture from being a H.264/SVC to become a H.264/AVC (Advanced Video Coding) standard. The acknowledgment of this work relies on the utilization of hardware resources from the support of low quality videos using the SVC standard to the support of high quality videos using AVC standard. Since the two systems share a considerable amount of static

common features, partially reconfiguring the SVC can lead to an AVC system without the need to perform a full system reconfiguration. The AVC new system can support high definition image resolutions of 1920×1088 pixels.

One other work cited that made use of the DPR in the H.264 is the work cited in [R.KHRAISHA 2010]. In [R.KHRAISHA 2010], the authors used the methods of partial reconfiguration to implement the deblocking filter of the H.264 decoder on a DPR architecture. The system, with the use of DPR, became scalable, thus allowing the use of different taps depending on the video resolution feed to the system. The system proposed can adapt its four reconfigurable regions to perform the filtering of a frame in order to meet time constraints demands according to the video frame's resolution. When timing constraints are already met, the multiple tap filters can be used to enhance image quality.

The closest work to the authors' knowledge was found in [J.HUANG 2009]. The authors in [J.HUANG 2009] used the DPR ability in FPGAs to scale the DCT computation unit in an H.264 encoder. Eight reconfigurable regions were used to support a range scaling from 1×1 to 8×8 DCT computational units depending on the encoder's demands. However, our point of interest in this work was the introduction of the motion estimation computations in the proposed system. The Motion Estimation (ME) computational units were swapped in the proposed system whenever no DCT computational units were used in any reconfigurable region. One difference between our proposed system and the one cited in [J.HUANG 2009] is the purpose of each Reconfigurable Region (RR) in the system. In [J.HUANG 2009], ME computations were used as enhancement unit when available while in our case they are used as an essential building blocks in the system design. In the same context, the configuration of each RR in our system was to support a time or a block size constraint. Not only our ME computational units were explored as enhancement units, but also as a dynamic trade-off between accuracy or image quality and the system's energy consumption. One other difference in our work is the exploration of the use DPR to perform energy reduction rather than switching modules with other modules. It is a major advantage to swap modules in a reconfigurable region but blanking a region as well has the advantage of energy reduction as our work shows.

2.3.d. Summary

To conclude this section and to show what our proposed system will offer, compared to other related work in literature, we deducted Table 2.3. In Table 2.3 we show for each cited work, including ours, the important common features of each citation and we highlight the differences among them. The mentioned features are the following: the search range of systems targeting the implementation of the ME computational unit on an FPGA. The number of hardware logic cells occupied by the proposed system (fully or only the computation implemented hardware blocks). The targeted implemented block: The ME, SVC or the deblocking filter. And finally, whether or not any cited work explored the power savings/consumptions or the energy savings/consumptions.

Starting with the first feature in Table 2.3, we find out that our work is similar to that of [T.MOORTHY 2008] that is the best search range an H.264 encoder can have. Regarding

the hardware resources, our system consume the least hardware resources when compared to other cited work related to the ME implementation. One other insight from Table 2.3 is that our system is the only system among the related work that shows an energy analysis of a DPR implemented ME of an H.264 encoder.

	Search range	HW logic cells ~	DPR	Target block	Power analysis	Energy analysis
[T.MOORTHY 2008]	48×63	188.6K	No	ME	No	No
[H.SAHLBACH 2010]	32×64	45.87K*	No	ME	No	No
[M.GUARISCO 2010]	NA	40.68K	Yes	SVC	Yes	No
[R.KHRAISHA 2010]	NA	23.39K*	Yes	D-Filter	No	No
[J.HUANG 2009]	NA	73.57K	Yes	DCT+ME	Yes	No
Our Work	48×63	68.45K	Yes	ME	Yes	Yes

**: Hardware resources occupied by the computational implemented units only and not the whole system as cited by the respective reference*

Table 2.3 – H.264 related work main features and properties. The main aspects of the related work in comparison to our work are highlighted and cited in the table.

3

Hybrid Architecture for a Multiple Target Tracking Driver Assistant System

In this chapter, we introduce the targeted algorithm used in our DAS system, the MTT tracking algorithm. We present our base architecture used with analysis and description of each internal functional block. We follow by discussing the modifications made to the original proposed system and the ability to migrate it from a MPSoC architecture to a hybrid architecture. Results, analysis and discussions are mentioned as a proof to the validity of our new proposed system design.

3.1. Introduction

The past two decades have witnessed a proliferation of microelectronic devices in automotive systems. Today, such devices are commonly used in a wide range of automotive applications including engine and vehicle control, anti-lock brakes, navigation systems, and car entertainment units. While early automotive systems were designed using microcontrollers, DSPs, and ASICs, some of the newer systems use FPGAs [M.ULLMANN 2004]. In addition to providing high logic densities, integrated hardware components, and fast design turnaround times, FPGAs can be easily reconfigured to meet the needs of their operating environments. This makes FPGAs ideal platforms for new automotive applications.

An increasingly important class of automotive applications, particularly for commercial vehicles, is driver assistant systems. Such systems reduce a driver's workload and improve road safety in stressful driving conditions (e.g. at night or in bad weather). Driver assistant systems often require real-time monitoring of the driving environment and other vehicles on the road. A MTT algorithm, part of a DAS system, uses an on board radar to keep track of the speed, distance, and relative position of all vehicles within its field of view. Such information is crucial in applications such as collision avoidance or intelligent cruise control. The computational needs of a MTT system increase with the number of targets that must be tracked. It is therefore very difficult for a single processor to handle these computational requirements alone.

In this chapter we present a FPGA-based MPSoC architecture for implementing a MTT system. We describe the evolution of our system from a software-based implementation that distributes the computational workload among multiple soft processor cores to a hybrid implementation that combines soft processor cores with dedicated hardware blocks. We also discuss the system-level trade-offs we made and the possibility to now introduce the dynamic partial reconfiguration into our hybrid system.

This chapter is organized as follows: in section 3.2, we present a general overview of the DAS system. After that, we present a MTT collision avoidance system used in automotive applications as a DAS application. The MTT system is defined and is presented in detailed functional blocks. The base system architecture is then introduced and described. In section 3.3, we present our approach to migrate the MPSoC MTT DAS system into a hybrid system. We present details and analysis of our targeted block in the MTT system that can be changed from a software-based block into a hardware block. Advantages of this change are validated by performing tests and obtaining results. Finally, we conclude the chapter in section 3.4 and summarize the results.

3.2. Driver assistant systems and the base implementation

Driver assistant systems are an increasingly important class of automotive applications, particularly in commercial vehicles where they can greatly reduce a driver's workload and improve road safety in stressful driving conditions [F.KUCUKAY 2004]. Driver assistance systems commonly require real-time monitoring of the driving environment and other vehicles on the road.

DAS systems are usually composed of three stages: *capturing*, *treatment* and *restitution* stages (Figure 1.5). Such systems are built with the aid of one or multiple senor devices. These sensors could be mounted cameras, tracking radars, light sensors, speed acquiring devices and many more. For any sensor used, a *capturing* stage is needed to acquire information from such sensors (image, signals, light, etc...) and convert it to certain signaling formats to be used for treatment. The *treatment* stage, in conjunction with the *capturing* stage, is responsible for the decision making of a DAS system. In this stage, certain algorithms are invoked

in order to perform the decision making of the DAS system based on the information from the *capturing* stage. The decisions from the *treatment* stage are then delivered to the driver using the *restitution* stage.

In this section, we focus on the treatment stage of a DAS system. The work presented is part of the PRIMA-CARE project (Prevention of Road accidents through a combination of Intelligent radar Multi-sensors and dynamic management of sound Alerts according to the Risk Encountered) DAS system's treatment stage [PRIMA-CARE]. We present in this section the algorithm used inside the *treatment* stage (the MTT algorithm). We also highlight the base architecture of this system as an introduction and motivation to the DPR based proposed solution and implementation.

3.2.a. Multiple target tracking system

In most driving conditions and environments, multiple targets can be detected in the FOV of the sensor mounted. Hence, the use of highly accurate tracking algorithms must be used to ensure the proper tracking of multiple targets. One of the most common algorithm solutions is the MTT algorithm. First proposed in [N.WAX 1955], the MTT system is a track-while-scan based system. This system, meant for automotive driver assistance applications, must take into account the specific dynamics of the obstacles encountered on roads. Such systems uses radar based sensor devices, over a predetermined search volume, in which data is received at regular intervals as the radar and sent to the MTT system.

The basic elements of a MTT system are shown in Figure 3.1. Each MTT system is decomposed of three basic building blocks:

1. The *Measurement Data Processing* block responsible for the data processing before sending it into the MTT functional blocks. This block includes all signal processing blocks including the radar sensor device.
2. The *Data Association Block*, which plays an important role in the tracking of multiple obstacles and assigning each to a specific track.
3. The *Filtering&Prediction* block, which incorporates the assigned observations into a set of updated track estimates.

A MTT system functions, as shown in Figure 3.2, in the following manner: when an observation is received from the sensor device, specific signal processing is performed before sending the information into the *Data Association Block*. First, incoming observations are considered for existing tracks from previous scans. *Gate Computation* tests which of the possible observation-to-track assignment is more "reasonable" at the beginning before a more refined algorithm is used to determine the final pairing. In this case, certain observations without associated tracks can generate new tracks. A track is instantiated and confirmed only when the number and quality of observations satisfy a certain criteria. In a similar manner, low quality tracks, over time are deleted. Finally, a gate is set around each track and the cycle repeat itself [S.BLACKMAN 1999].

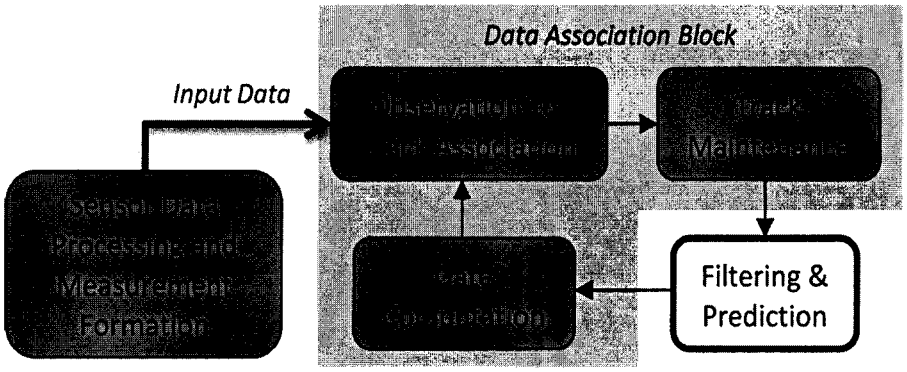

Figure 3.1 – MTT system's basic building blocks. The Data Association 3 Blocks are also shown in the figure.

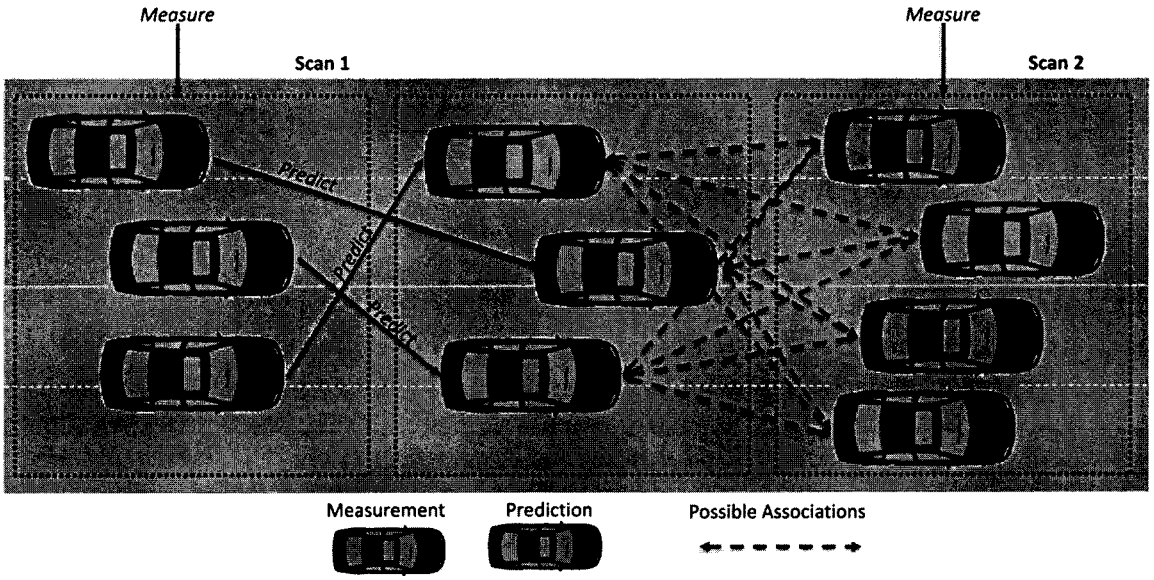

Figure 3.2 – MTT state prediction and estimation. [J.KHAN 2009]

The MTT application tracks targets by processing data measured by the radar. This data is processed in three iterative stages:

1. During the *observation* stage, the MTT application reads speed, distance, and azimuth angle data for each target.
2. During the *prediction* stage, an adaptive filter is used to predict the location of each target on the subsequent radar scan. This prediction is based on the actual location data measured during the observation stage and an estimate of the location data computed by the filter during the previous radar scan.
3. Finally, during the *estimation* stage, new target location estimates are computed for use in the subsequent prediction stage.

Figure 3.3 shows the structure of the MTT application. A more detailed description of the application and its mathematical formulation can be found in our earlier work [J.KHAN 2008] and [J.KHAN 2009].

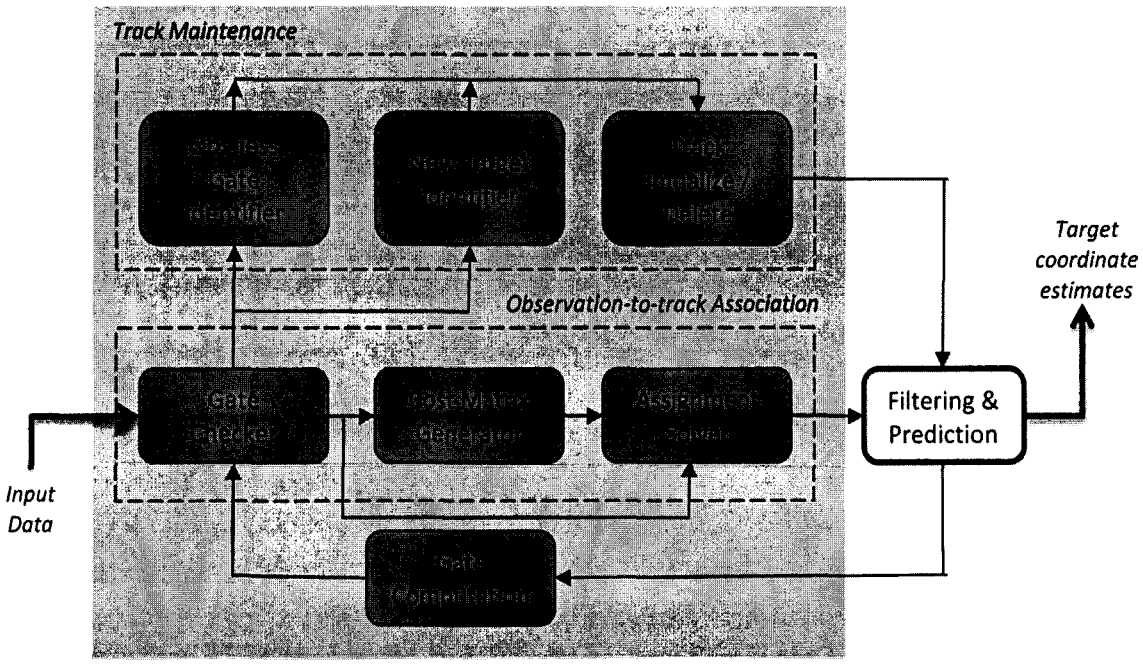

Figure 3.3 – MTT system's detailed building blocks. The Observation-to-Track Association and Track Maintenance blocks are shown as detailed inner blocks in the figure.

In Figure 3.3, we show a more detailed architecture of the MTT system and its inner blocks. Starting from the first stage of the input data, in our system, we use an AC20 TRW radar based sensor device mounted in front of our car, realizing a scan every 25 msec [RADAR]. The data sent to the MTT system includes: targets' distance measured away from the radar, positional angle, linear velocity and angular velocity.

The *Gate Computation* block is the first step in the data association process. This block receives targets' predicted states and predicted errors covariance from the *Filtering&Prediction* blocks. Using these two values, the predicted states and errors, the *Gate Computation* block defines the probability gates which are used to associate incoming observations to existing targets. The *Gate Checker* block carries inside of it the criteria in which an incoming observation is to be assigned with an existing target or the creation of a new detected target. In other words, this block is mainly responsible for the pairing of predictions to observations according to a certain criteria. The cost of every pairing inside a gate is sent to the *Cost Matrix Generator* block. This block, as the name implies, is responsible for the generation of a set of cost matrices including the cost of assigning certain measurements to existing targets. Based on these matrices, the *Assignment Solver* block is responsible for the final pairing between measurements and the existing targets. The output will be a one-to-one assignment of one measurement to one existing target.

The *Track Maintenance* block is made up of three sub-blocks: *Obs-less Gate Identifier, New Target Identifier* and *Track Initialize/Delete*. In real situations, some targets might leave the FOV of the radar. In other cases, a new target might enter the FOV of the radar. When a target leaves the FOV of the radar, the *Obs-less Gate Identifier* identifies this target and triggers the *Track Initialize/Delete* block to remove this specific target from the calculations and predictions since it left the detection area of the radar. However when a new target

is detected, the *New Target Identifier* identifies that a new target has entered the FOV of the radar and hence a new track must be initialized. This block triggers the *Track Initialize/Delete* block in order to establish a new track for the new detected target.

The *Filtering&Prediction* block is particularly important as the number of filters implemented inside this block is the same as the maximum number of targets to be tracked. In the earlier work of [J.KHAN 2008], 20 Kalman filters were implemented in order to support the tracking of a maximum number of 20 targets.

3.2.b. *Filtering&Prediction* block in tracking systems

The MTT application can be implemented using a variety of adaptive algorithms including α-β filter, mean-shift algorithm, and Kalman filter [S.BLACKMAN 1999]. In our system, we have chosen to implement both the Kalman and the α-β filters. The Kalman filter is used because it is an important block in the whole MTT system. It behaves properly and effectively with the other MTT blocks. A Kalman filter is designed to track a moving object having a constant velocity. The process and measurement models presented above for target dynamics can be classified as linear models with Additive White Gaussian Noise (AWGN). For this reason we use the Kalman filter because it is a recursive Least Square Estimator (LSE) considered to be the optimal estimator for linear systems with AWGN probability distribution [S.BLACKMAN 1999].

Due to the importance of the Kalman filter in our implementation, it is important to cite the mathematical computations inside this filter. The Kalman filter mathematical computations are shown in equations (3.1) to (3.8).

$$Z_k = Distance; Angle \tag{3.1}$$

$$Y_{k\,pred} = A * Y_{k-1} \tag{3.2}$$

$$P_{k\,pred} = A * P_{k-1} * A' + Q \tag{3.3}$$

$$K = P_{k\,pred} * H' * (H * P_{k\,pred} * H' + R)^{-1} \tag{3.4}$$

$$Y_{k\,estim} = Y_{k\,pred} + K * (Z_k - H * Y_{k\,pred}) \tag{3.5}$$

$$P_{k\,estim} = (I - K * H) * P_{k\,pred} \tag{3.6}$$

$$Y_{k-1} = Y_{k\,estim} \tag{3.7}$$

$$P_{k-1} = P_{k\,estim} \tag{3.8}$$

Z_k: is the measured input distance and angle.

A: is the assumed known state transition matrix.

$Y_{k\,pred}$: is the predicted state vector.

Y_{k-1}: is the previous estimated state vector.

$Y_{k\,estim}$: is the estimated state vector.

$P_{k\,pred}$: is the predicted error covariance matrix.

P_{k-1}: is the previous estimated error covariance matrix.

$P_{k\,estim}$: is the *estimation error covariance matrix.*

Q: is the AWGN assumed known covariance matrix.

H: an observation matrix that relates the current state to the Z_k.

K: is the Kalman *gain matrix.*

I: is an identity matrix.

R: is the measurement noise covariance matrix.

In summary, the Kalman filter takes as inputs: The measured distance and angle. These are assigned in the Z_k matrix as shown in equation (3.1). As for matrixes A, Q, H, R and I, they are considered input matrixes from other MTT functional blocks respectively. $Y_{k\,estim}$ is the filter's only output matrix, sent to the driver or restitution phase, that includes the predicted and filtered distance and angle.

However, real objects actually tend to move in variable accelerations and therefore different velocities. The choice of the α-β filter is due to its simple implementation and its consideration of target velocities in decision makings [S.BLACKMAN 1999]. In general, an α-β filter is a simplified estimation filter for data smoothing and control applications. When an α-β filter is applied to motion systems, it takes as inputs, the measured distance and velocity. Since we used this filter to enhance our results as will be shown in chapter 4, hence, we present the mathematical module of this filter in equations (3.9) to (3.15).

$$X_k = X_{k-1} + \Delta T * V_{k-1} \tag{3.9}$$

$$V_k = V \tag{3.10}$$

$$R_k = X - X_k \tag{3.11}$$

$$X_k = X_k + \alpha * R_k \tag{3.12}$$

$$V_k = V_k + (\frac{\beta}{\Delta T}) * R_k \tag{3.13}$$

$$X_{k-1} = X_k \tag{3.14}$$

$$V_{k-1} = V_k \tag{3.15}$$

X: is the input distance to the filter.

V: is the input velocity to the filter.

ΔT: is the time interval between measurements (e.g. the sensor's scans interval).

R_k: is the input distance difference between the current and the preceding distance.

X_{k-1}: is the previous predicted distance value.

V_{k-1}: is the previous predicted velocity value.

α and β: are constants correction gains of values between 0 and 1.

In conclusion, this straight forward filter takes selected α and β constants (adjusted experimentally), uses α times the deviation R_k to correct the position estimate, and uses β times the deviation R_k to correct the velocity estimate. The result of $\beta \times R_k$ is used in a consecutive iteration to further more enhance the target position.

3.2.c. Base system implementation

Figure 3.4 shows our baseline MTT system implementation, which uses multiple Nios-II soft processor cores [NIOS] implemented in an Altera Stratix-II FPGA [J.KHAN 2008]. It is important to distinguish that the work of [J.KHAN 2008] was Altera based that did not support DPR at that time. Hence, The first step in our work was the migration of the architecture from Altera to Xilinx while keeping the same functionality by replacing the Nios-II soft cores with MicroBlaze [MICROBLAZE] Xilinx soft cores.

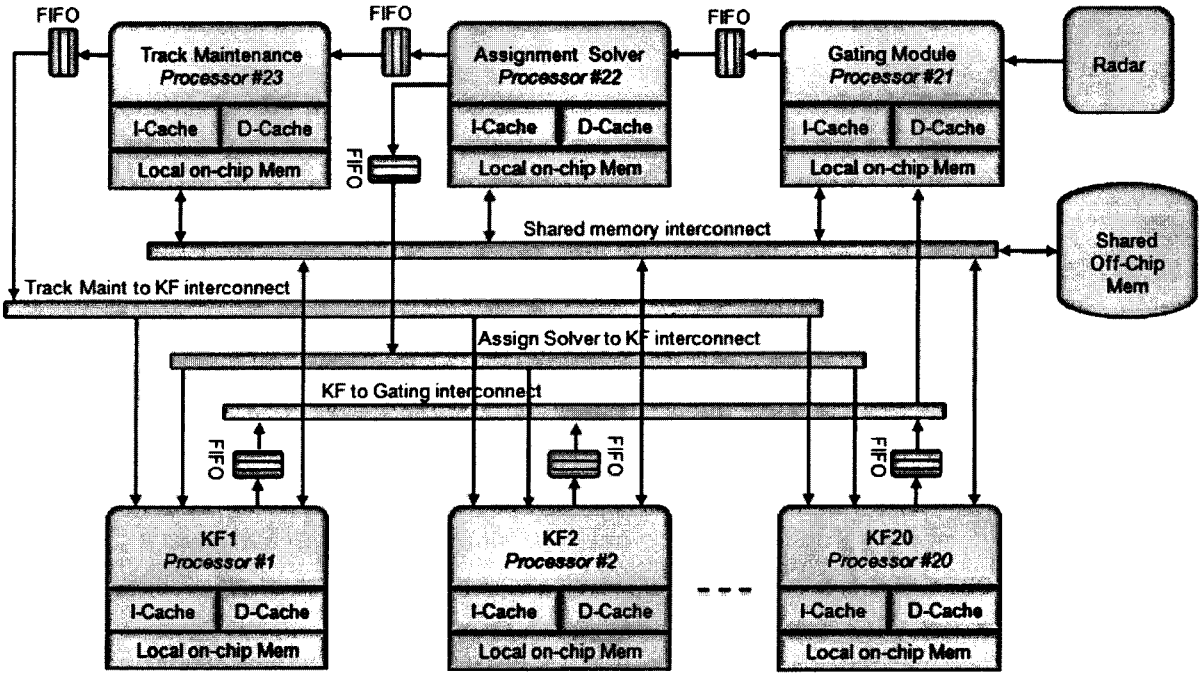

Figure 3.4 – The MTT system's base architecture. The base architecture is decomposed of 23 soft core processors connected via interconnections of a bus mesh. [J.KHAN 2008]

To map the MTT application onto multiple processors, the code was divided into functions that interact in a producer-consumer fashion. The functions are then distributed among the processors to meet the 25 msec real-time constraint imposed by the radar scan

window. To help us profile the run-time characteristics of our application and guide the allocation of functions to processors, we used dedicated hardware profiling counters to measure the latency and execution frequency of individual functions.

Our baseline system consists of 23 heterogeneous processor cores arranged in an MPSoC configuration. Due to the time-critical nature and floating-point requirements of the Kalman filter, and the need to track up to 20 targets simultaneously, our baseline system includes 20 dedicated processors configured with single-precision floating-point units that execute the Kalman filtering function. Moreover, the assignment solver, gating, and track maintenance functions [S.BLACKMAN 1999] are each allocated to a different processor to minimize the execution time and communication overhead. Respectively:

- Processors 1-20 host the functions of the *Filtering&Prediction* block (Klman filters).
- Processor 21 hosts the functions of the *Gating Module* including the *Gate Computation*, *Gate Checker* and *Cost Matrix Generator blocks*.
- Processor 22 includes the functions related to the Assignment Solver block from the *Observation-to-track Association.*
- Processor 23 contains all the function of the blocks inside the *Track Maintenance*: *Obs-less Gate Identifier*, *New Target Identifier* and *Track Initialize/Delete* blocks.

Where needed, processors are configured with appropriately sized instruction and data caches and local memories. Processors with interacting functions are also interconnected using appropriately sized FIFO buffers.

To meet the 25 ms real-time constraint, a number of optimizations were applied to reduce the execution times of different functions. These included:

- Sizing the on-chip instruction and data cache memories for different processors to reduce off-chip memory access times.
- Introducing on-chip Private Data Memory (PDM) banks to store the system stack and the heap to reduce the cost of dynamic memory allocation in various functions.
- Adding floating-point hardware support (Floating Point Units; FPUs) to some processors to execute floating-point operations more efficiently.
- Transforming some functions to only use integer data types and reduce overall execution time.

Table 3.1 shows the different optimizations applied to various functions along with their final, corresponding, execution times. The latency results in Table 3.1 shows that the total execution time of the MTT base system exceeds the real time constraints (25 msec). In order to tackle this problem, pipelining was inserted among different processing stages in order to process data in different stages. The use of pipelining stages allows the ability to still meet real time constraints.

Function	I cache	D cache	PDM	FPU	Int	Latency
Kalman	4 Kbytes	No	2 Kbytes	Yes	No	3 msec
Gating	16 Kbytes	2 Kbytes	3 Kbytes	Yes	No	23 msec
Solver	8 Kbytes	16 Kbytes	No	No	Yes	24 msec
Track	No	No	No	No	No	8 msec

Table 3.1 – Processor's optimizations applied in order to meet the highest performance of each respective function in the processor. Execution latency is also provided in msec.

3.3. Migration to hardware *Filtering&Prediction* block

The base MTT system demonstrated the high cost, in area and resource utilization, associated with a software-only implementation. This is particularly evident with the use of 20 soft core processors for the *Filtering&Prediction* stage of the MTT system. Even when multiple targets could be tracked by a single Kalman filtering block, the cost of implementing an entire processor in the logic may be too high. A more cost-effective solution would be to use a hybrid system that integrates dedicated Kalman filtering hardware blocks with software-programmable processor cores.

To better understand the functional and numerical characteristics of the Kalman filtering block, we developed a *MATLAB*™ model using single-precision floating-point arithmetic. Details of the mathematical model are described in section 3.2.b, [J.KHAN 2009] and [S.BLACKMAN 1999]. We then tested our model using two data sets:

1- A random set of target data from the radar's operational range (i.e. distances from 0 to 200 meters and azimuth angles from -6 to +6 degrees).
2- A more realistic data set that emulates a target being overtaken by the vehicle with the radar.

Gaussian noise was also added to both data sets to model the error introduced by the radar sensors.

3.3.a. Filter's coefficients stability

After running the tests using the two data sets, we observed Figure 3.5 and Figure 3.6. The figures show the output response curves of different Kalman filter implementations for the distance and angle measurements of the simulated data set. In the legend of the figures, "distance measured" is the simulated data input from the sensor, "*calculated* P_k" is the response curve of the Kalman filter without altering any of its internal calculations, "*stable K*" is the curve obtained when inserting fixed values of the *K* matrix (*Kalman gain matrix*). After a transient training interval, the filter output stabilizes and begins to track the input data (represented on the graph by the solid line).

When analyzing our results, we noticed that for all filter implementations, and for both data sets, the elements of the *estimation error covariance matrix* $P_{k\,estim}$ [J.KHAN 2009]

[S.BLACKMAN 1999] converge to optimal values and stabilize. This suggests that these elements, which are computed iteratively and depend on the characteristics of the radar, can be fixed at their optimal values.

By setting these elements to their optimal constant values, the elements of other matrices (e.g. the *Kalman gain matrix K* [J.KHAN 2009] [S.BLACKMAN 1999]) also become constants. This significantly reduces both the computational complexity and execution time of the filter, and leads to a more efficient implementation of the filter block. This efficiency is attained by utilizing less hardware resources for example, such as the reduction of the number of embedded multipliers. It also leads to a faster filter response time (c.f. the curves labeled *calculated Pk* and *stable K* in Figure 3.5 and Figure 3.6). Our hardware implementation of the Kalman filter block was therefore designed using the optimal constant values of various filter coefficients.

3.3.b. Fixed point data precision

The software implementation of the Kalman filter block uses single-precision floating-point arithmetic and requires a processor with a floating-point unit to execute. However, the dynamic range of the data obtained from the radar's sensors (200-meter distances and -6 to +6 degree angles, respectively) is significantly smaller than the range supported by a floating-point unit. Furthermore, an automotive target can still be tracked effectively even when the level of numerical precision is low. For example, the MTT system will likely behave the same way whether a target is determined to be at 29.8 or 29.7889993 meters away. This suggests that a fixed-point Kalman filter block that provides acceptable levels of numerical accuracy and precision is a more cost-effective alternative to a floating-point implementation.

Figures 3.7 and 3.8 shows a close-up view of the Kalman filter's response curves for the distance and angle simulated data after the output has stabilized. The solid line corresponds to the floating-point implementation while the other curves correspond to fixed-point implementations at different levels of precision. Since the radar has a range of 200 meters, only 9 bits are needed to represent the (signed) integer portion of the distance simulated. Similarly for the angle, since the radar range is of -6 to +6 degrees, only 4 bits are needed to represent the (signed) integer portion of the angle simulated. The curves in Figures 3.7 and 3.8 therefore correspond to the filter response as we increase the *fractional* portion of the distance and angle simulated data from 4 to 16 bits.

Figures 3.7 and 3.8 clearly show that the error between the floating-point reference and the various curves decreases as the fractional precision level increases. However, even when designing with the fast and area-efficient on-chip hardware multipliers found in most high-density FPGAs, increasing precision levels typically result in larger and slower hardware.

To optimize the utilization of 18×18 hardware multipliers, we implemented our filter block using an 18-bit fixed-point data format. For distance computations we used a 9-bit, signed, integer component and, hence, a 9-bit fractional component, and for angle computations, we used a 4-bit, signed, integer component and, hence, a 14-bit fractional component.

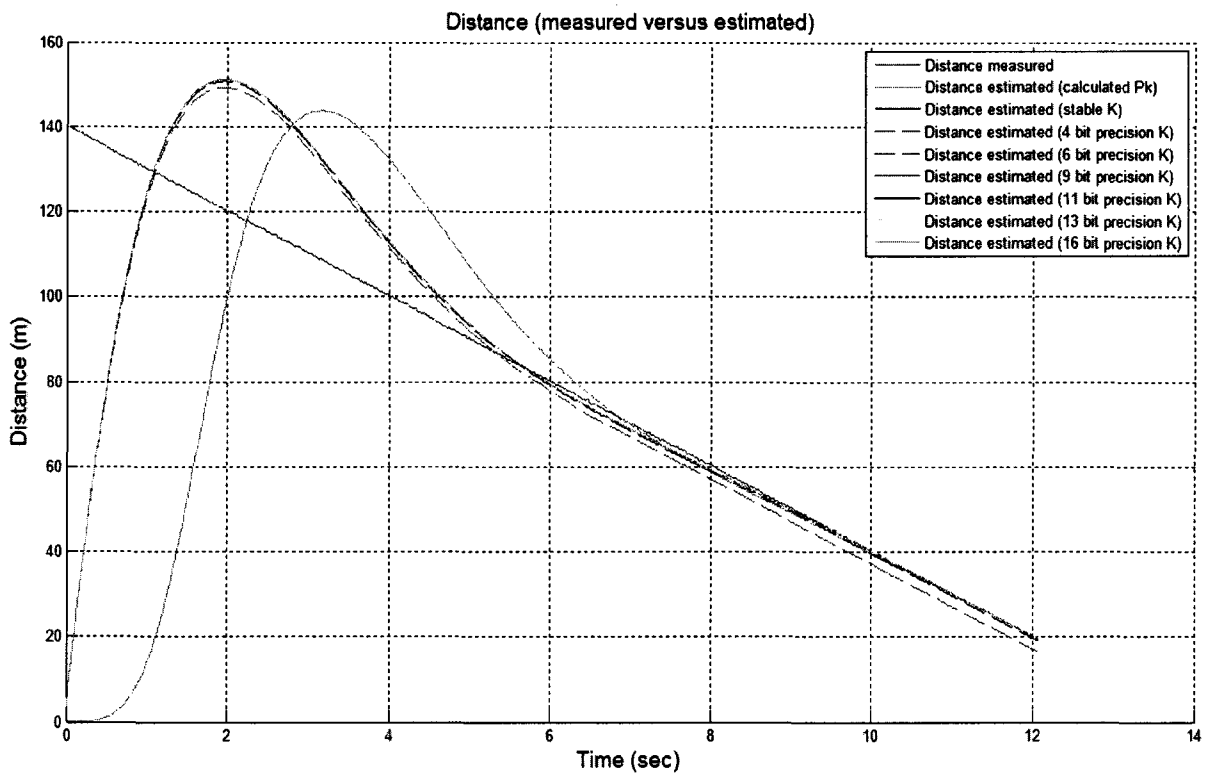

Figure 3.5 – Kalman filter response for distance simulated data.

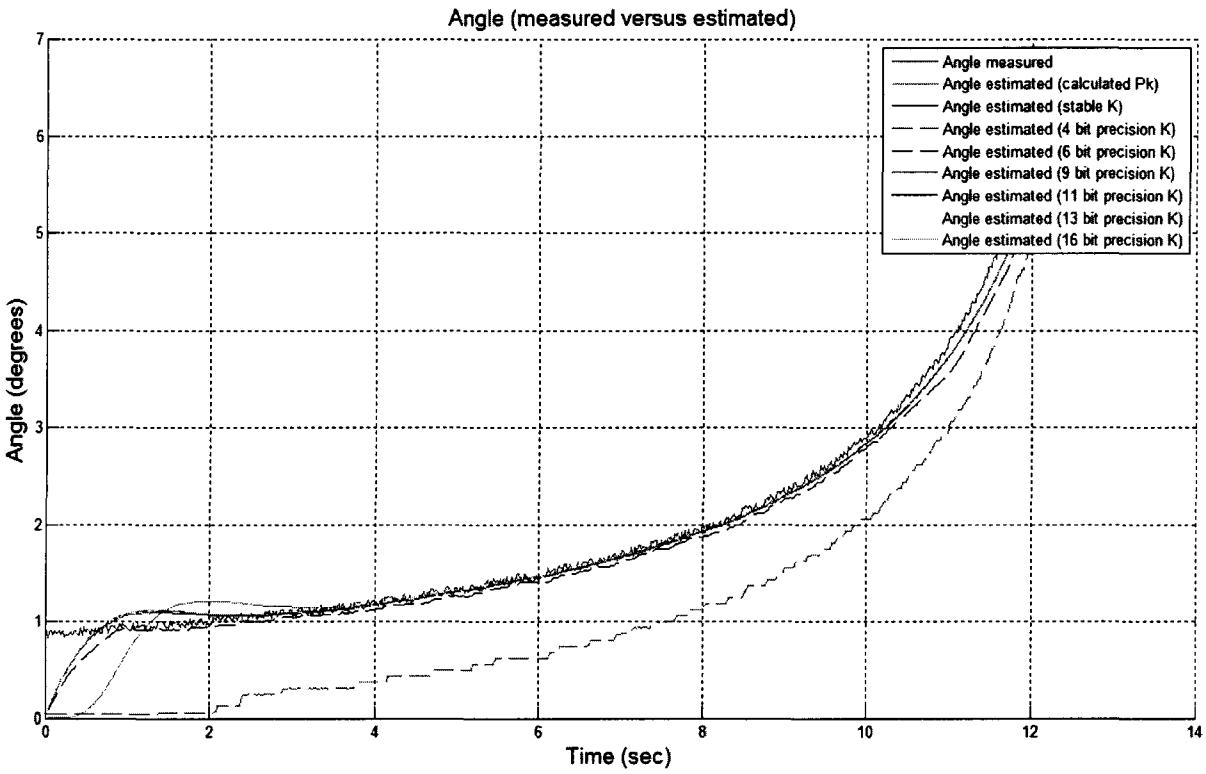

Figure 3.6 – Kalman filter response for angle simulated data.

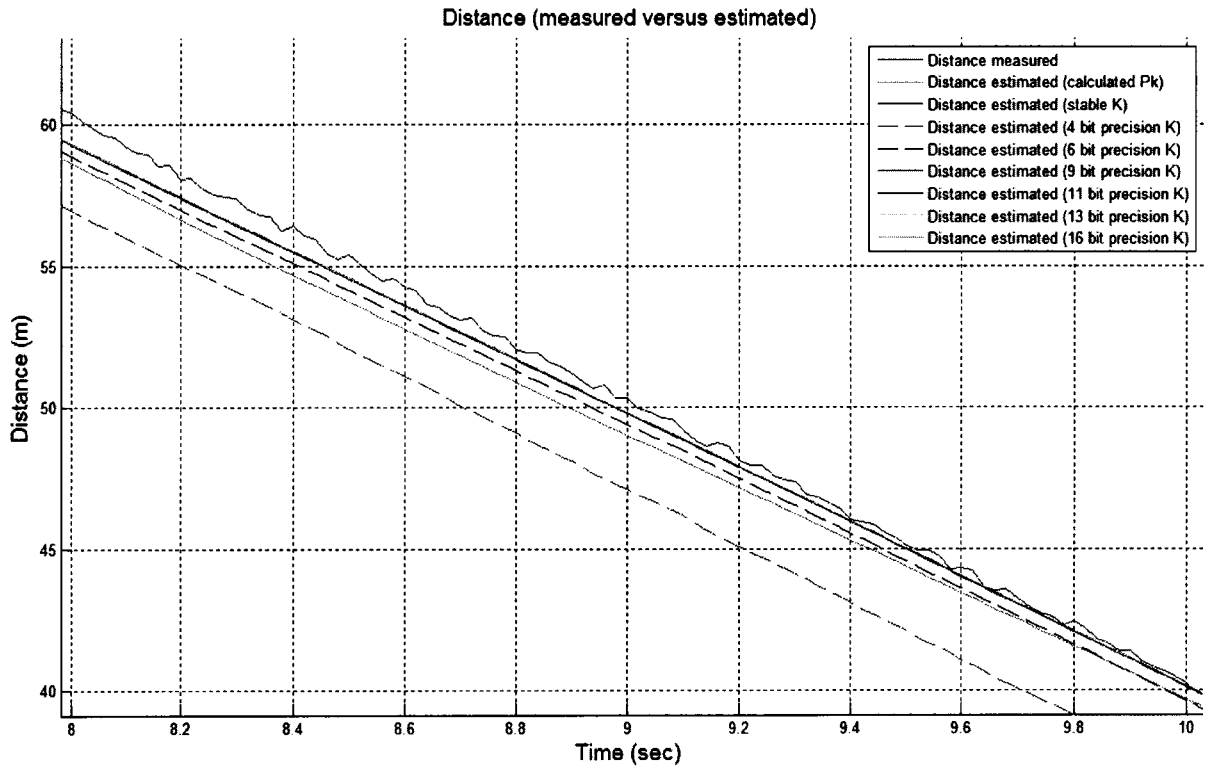

Figure 3.7 – Kalman filter distance response errors for different precision levels.

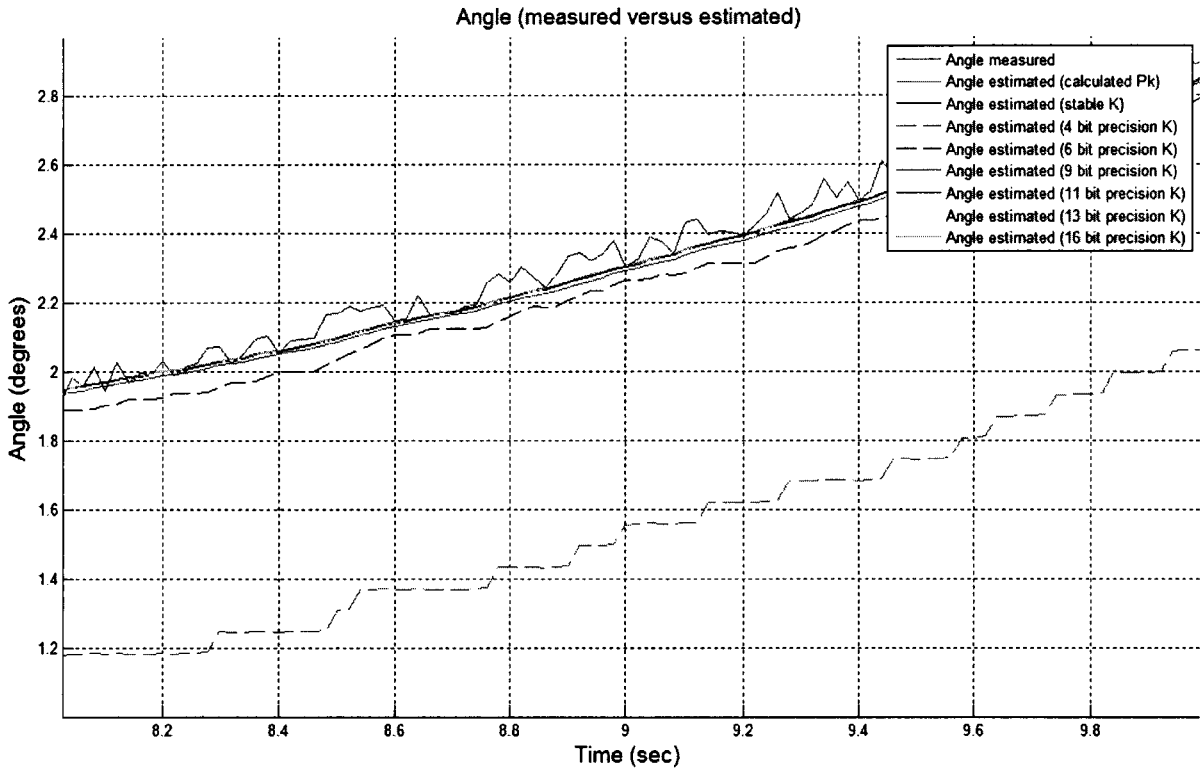

Figure 3.8 – Kalman filter angle response errors for different precision levels.

3.3.c. Software VS hardware implementation

In order to compare the software based system with the hardware based system, we first cite the hardware resources utilized by both implementations in Table 3.2. This tabel's results, unlike our baseline system, targeted the Xilinx XC4VFX12 Virtex-4 FPGA [XILINX] and Xilinx MicroBlaze soft processor core, respectively [MICROBLAZE]. Since we are interested in exploring dynamically reconfigurable MPSoC architectures, we are limited to using Xilinx devices and tools. However, this does not diminish the results of our baseline implementation, which demonstrated the feasibility of implementing a heterogeneous MPSoC architecture using customized soft processor cores.

FPGA resources	Available	Used (SW)**	Used (HW)*
Slices	5472	1265 (23%)	265 (4%)
Slice Flip Flop	10944	1347 (12%)	0 (0%)
4 input LUT	10944	1902 (17%)	403 (3%)
DSP48s	32	3 (9%)	12 (37%)
RAMBs	36	8 (22%)	0 (0%)

*: *For distance computations we used a 9-bit, signed, integer component and a 9-bit fractional component, and for angle computations, we used a 4-bit, signed, integer component and a 14-bit fractional component respectively.*
**: *Resources needed by one soft core processor.*

Table 3.2 – Hardware resources used by the software and hardware Kalman implementations on standard FPGA fabric XC4VFX12 Virtex-4.

Our results show that the hardware implementation uses almost 80% fewer logic resources (LUTs and slices) and no slice flip-flops or BRAMs compared to the software implementation. On the other hand, the hardware implementation uses four times as many DSP48 blocks as the software implementation. These results clearly demonstrate the architectural characteristics of the hardware implementation, which makes use of the constant filter coefficient and fixed-point arithmetic optimizations described earlier, in addition to making use of the fast and area-efficient DSP48 hardware blocks. Conversely, the software implementation requires enough logic and storage resources to implement a pipelined processor datapath.

We also compared the software and hardware implementations in terms of latency. For both implementations we used a 100 MHz system clock. To measure the latency of the Kalman software function we used a hardware timer to measure the number of CPU clock cycles spent inside the function. We then multiplied the clock cycle count by the period of the system clock, which showed the latency of the software implementation to be 268.0325 μsec. To measure the latency of the Kalman hardware block we used the results of the post-placement and routing timing report, which showed the latency to be 0.033 μsec. Our results therefore show that the hardware implementation is 8,122 times faster than our software implementation.

Thought the proposed new hybrid system consume less hardware resources and is faster, hardware blocks used in the system may become obsolete if the application or the algorithms around which they are based on, changes. For example, different types of filters

may be needed for tracking targets in different driving environments (e.g. urban, suburban, rural, etc...). One way around this problem is to implement a system with *all* the necessary hardware blocks, and to select and use the appropriate blocks at run-time. However, this is not a very area-efficient solution, and it does not safeguard against hardware block obsolescence. Another solution, particularly for systems implemented in FPGAs, is to reconfigure the entire FPGA to implement a new system with new hardware blocks. However, this also is not very efficient since only a small portion of the hardware implementation typically needs to be reconfigured. A better solution would be to build a *dynamically reconfigurable* system that enables an application to replace obsolete or inadequate hardware blocks with new ones on the fly. However, such a solution requires FPGA devices and tools capable of supporting partial dynamic reconfiguration. In the following chapter we explore two possibilities of the use of DPR on the hybrid MTT base system.

3.4. Conclusions

In this chapter, we presented the work we did to implement a hybrid MTT DAS based system. We started with an introduction of a general DAS system and highlight its usage and importance. Following the introduction, we highlight a base system implementation of a multiple target collisions avoidance tracking system. This system will further become our base system where DPR is explored. The base MTT system was originally designed on an MPSoC basis. This original base architecture showed a huge resource utilizations especially for its filtering and prediction inner block and hence was implemented on a hardware accelerator basis. The migration of the MTT MPSoC system into a hybrid system architecture was the first step to the exploration of the benefits of DPR in such a system. Our results showed that with the migration of a soft processor based filtering block into a hardware accelerator block reduced hardware resources by around 80% with an increase of processing speed that reached around 8000 times.

4

Exploring DPR in a Multiple Target Tracking System

In this chapter, we describe two implementations of a dynamically partial reconfigurable MTT module for our DAS system. Our modules leverage DPR to implement a dynamically reconfigurable filtering block that changes with changing driving conditions. We provide experimental results that demonstrate the feasibility of our systems and its resilience against reconfiguration overhead, which enhances reliability and driver safety.

4.1. Introduction

The baseline MTT system, proposed by [J.KHAN 2009], demonstrated the feasibility of using software-programmable processor cores to execute a complex application and still meet its real-time constraints. It also demonstrated the importance of optimizing the system's implementation to meet these performance constraints. However, it also demonstrated the high cost, in area and resource utilization, associated with a software-only implementation. This is particularly evident with the Kalman filtering block where dedicated processors, configured with floating-point units, cache memories, and private data memories, are used to execute the Kalman filtering code for each target. Also, one disadvantage of the current FPGA system is that hardware blocks may become obsolete if the application or the algorithms change.

A more cost-effective solution would be to use a hybrid system that integrates dedicated Kalman filtering hardware blocks with software-programmable processor cores.

This step can lead to a better hardware resource utilizations on an FPGA. The use of hardware filtering blocks can also be an introduction to the use of DPR in the MTT system. This can lead to the support of different driving conditions and environments on the fly during run time.

In this chapter, we present the work done on the MTT base system to migrate it from a hybrid based implementation to a DPR based implementation. We investigate two possible DPR systems that can be used in order to support multiple driving environments and obstacle behaviors. In the first possible implementation, we explore the use of DPR to modify the physical structure of the Kalman filtering subsystem in a MTT application in section 4.3. We also demonstrate how the accuracy of the filter block can be dynamically and automatically tuned to match the characteristics of the operational environment. This can be very useful when the environment changes from, say, open highway to narrow city street where higher levels of accuracy are needed to track multiple, potentially closer targets. Conversely, when the driving environment changes from dense to sparse, the accuracy of the filter can be reduced to minimize resource utilization and energy consumption. We provide experimental results that demonstrate the feasibility of this approach and its low overhead. We also demonstrate the ease with which we can switch between hardware implementations automatically using a simple heuristic. This contrasts with prevailing approaches to dynamic reconfiguration, which are mainly demand-driven.

As for the second possible implementation, we explore the use of DPR according to targets proximity to the radar sensor in section 4.4. As targets move closer to the radar, they should be tracked at higher levels of accuracy since they can potentially become more hazardous. On the other hand, as targets move further away, less accurate tracking can be used. In this section, section 4.4, we show how the functionality and accuracy of an MTT system can be automatically tuned to match the dynamics of moving obstacles on the road. Since lower levels of accuracy generally require fewer hardware resources, DPR can be leveraged to release hardware resources for other uses, such as tracking more obstacles, accelerating other computational functions, or reducing power consumption. In our work, free hardware resources are used to enhance the radar detection unit, which improves the detection of targets at different distances. Our design is based on using three modular filters that can be dynamically combined in three configurations to match different driving scenarios. Our design also includes an enhanced detection unit module for post-processing acquired radar signals and pre-filtering them before they are delivered to our MTT system. Our system is also designed to continue operating even when being reconfigured, and this enhances the system's reliability. Our experimental results demonstrate the feasibility and low overhead of our dynamically reconfigurable design.

This chapter is organized as follows, in section 4.2, we explore the use of DPR in the base MTT DAS system. We present two possible proposed DPR systems and we highlight the advantages of each by showing simulated and actual results of our proposed systems in sections 4.3 and 4.4. A comparison between an enhanced soft core processor based system and our two proposed DPR systems is presented in section 4.5. Finally, in section 4.6, we conclude and summarize the built systems.

4.2. Approach and design

Starting from the main objective of the system, i.e. providing a reliable DAS system, we divided the objective into a set of goals, problems and solutions. The main goal and objective of the exploration of DPR in a DAS system is designing a collision avoidance system using sensors that will support multiple scenarios on a single package. The main problems, derived from the final goal, were the following:

- Designing a proper and reliable collision avoidance system.
- Building the collision avoidance system and the use of a sensor that can support as much driving conditions as possible.
- Tackle the problem of the limitation in hardware resources when supporting multiple driving scenarios and environments. Hence, instead of having multiple hardware systems for each scenario, hardware reuse is considered of high importance and cost reducer.

For these three problems, three solutions were suggested:

1- The use a MTT based system to tackle the collision avoidance problem.
2- The use of radar as sensors in order to support multiple driving conditions when compared to other sensor devices such as camera sensors.
3- The use of hardware accelerators and in turn, FPGA's ability to dynamically reconfigure partial space on the fabric to tackle the problem of supporting multiple driving scenarios and environments.

Hence, we emphasized the goal to become: designing an FPGA based hybrid MTT system using radar sensors that will support, with the use of DPR, multiple scenarios on the same fabric. From the goal definition, two main steps must be implemented in order to achieve a valid goal.

1- Designing and introducing hardware accelerators in the MTT soft processor system and hence implementing a hybrid MTT system.
2- Using these hardware accelerators as reconfigurable regions and modules that can change their configurations according to changing environments and driving conditions.

Figure 4.1 shows a high-level view of our proposed system architecture. Like the hybrid MTT system, it includes a number of soft processor cores for executing the control-intensive portions of the MTT application. It also includes a number of slots that can be configured with pre-designed hardware blocks for accelerating the performance-critical portions of the application. The soft processors and hardware blocks would be able to exchange data through interconnected FIFO buffers. The system also includes an ICAP controller for managing the configuration of hardware blocks on the fly. The configuration bit streams could be stored on- or off-chip memories, while the loading, removal, and swapping of hardware

blocks would be controlled by the main MTT application software using various performance-enhancing heuristics.

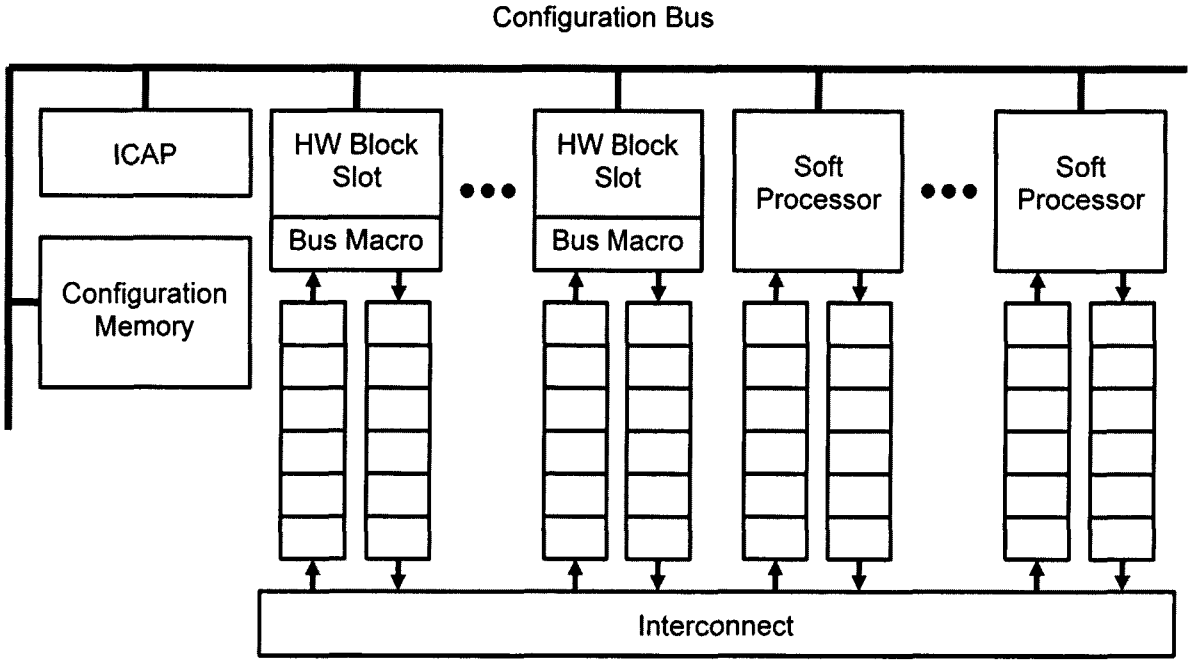

Figure 4.1 – The proposed dynamically partial reconfigurable MTT based system architecture.

4.3. DPR system based on targets density

In this part, we present one possibility to implement a DPR based MTT system. The important issue about this system is that partial reconfiguration is triggered based on the obstacles density. We present a new system architecture mostly used for the validation of the new DPR based Kalman *Filtering&Prediction* block. We then show the different filter implementations and their respective functional environments. We highlight the differences between them and show the impact of each configuration on the total system performance. Plots, results and hardware resource utilizations are presented as a comparison between the new DPR system of the hybrid architecture and software based MTT system. A reconfiguration heuristic is illustrated to show when each configuration is best fit according to detected driving conditions.

4.3.a. Validation system architecture

In order to implement a basic DPR system, first we need to implement a prototype design system. This system is important for the validation of the reconfigurable regions and the testing of their functionalities. In Figure 4.2 we show the architecture of our dynamically reconfigurable MTT system. The basic blocks and their respective functionalities of the prototype design are the following:

- PowerPC processor [POWERPC] used to implement all the blocks of the MTT application except the Kalman filtering block, which we implement as a dynamically reconfigurable hardware block.
- Processor Local Bus (PLB), used to provide fast communication between the processor and the external Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM).
- DDR SDRAM memory used to store the instruction and data files used by the MTT algorithm.
- On-Chip Peripheral Bus (OPB), used to connect various peripheral components and enable data transfers between the peripherals and the processor.
- "Timer" block, used to measure the latency of the hardware filter block and the time needed to transfer data and results between the processor (software) and the filter block.
- Advanced Configuration Environment (ACE) controller, used to load the configuration bit stream files of different filter block implementations from the compact flash drive. It is also used to configure the static portion of the system architecture and the MTT application code on system startup.
- HWICAP controller used for configuring reconfigurable regions in the FPGA [HWICAP]. It is initialized and operated under software control.

As for the Filter block, the Device Control Register (DCR) bus and its bridge, all are responsible for implementing the reconfigurable hardware. In other words, the partial reconfigurable region is inside the Filter box, which can communicate with the processor using the DCR bus.

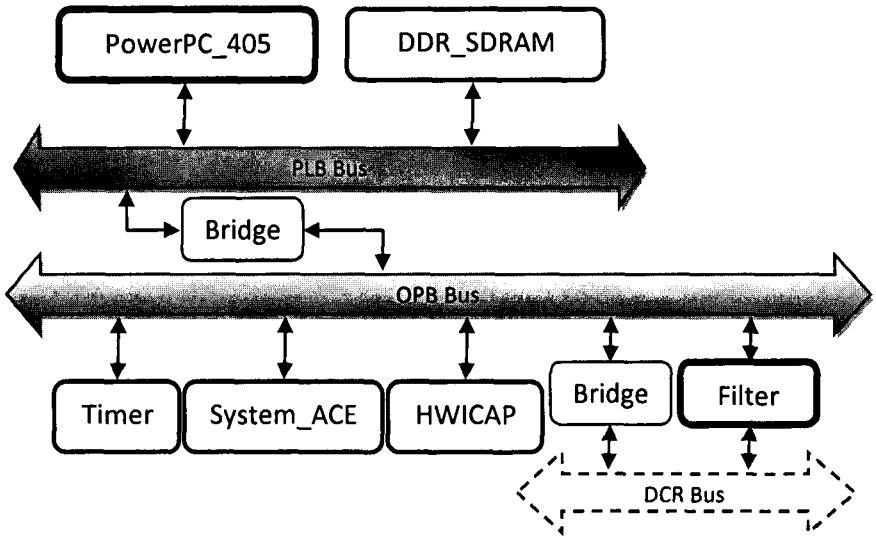

Figure 4.2 – Validation architecture basic IP cores. The MTT system runs on the PowerPC except the Filtering&Prediction Kalman block that runs on the "Filter" hardware block.

4.3.b. Different filter implementations

After implementing the Kalman filter as hardware block, chapter 3, the next step was to study the ability of implementing different types of filters in a reconfigurable region. The main advantage of such an implementation is its ability to implement different filter architectures having different characteristics in response to changing operating conditions. Such change is very common in driving scenarios where environments are constantly changing from urban to suburban to rural. To that end we have designed two Kalman filter implementations with different levels of tracking accuracy to match the needs of different road environments.

Our lower accuracy KFO filter is aimed at *open* road environments such as highways while our higher accuracy KFD filter is aimed at *dense* road environments such as those inside cities.

- *KFO Filter*: In addition to using constant, fixed-point filter coefficients, some coefficients are rounded to 2^{-x} while others are restricted to 18 bits. Such restrictions reduce the complexity of the hardware block and the number of 18×18 hardware multipliers needed to implement the filter. They also reduce the accuracy of the filter, which is an acceptable trade-off in open driving environments where fewer cars are typically within range of the radar.
- *KFD Filter*: This filter is based on the same architecture as the KFO filter but uses wider coefficients that result in a larger, more complex implementation. Wider coefficients are not rounded and hence need to be presented as 18 bits. This increases the number of 18×18 hardware multipliers and hence increase the filter's complexity. On the other hand, these also result in a more accurate filter that is better suited for dense driving environments (e.g. city streets, rush hour) where more cars are spotted within range of the radar.

4.3.c. Filters output and accuracy

Our first set of experiments was designed to measure the accuracy of different filter implementations. We used actual radar distance and angle measurements with different Kalman filter implementations including the original software implementation that uses floating-point operations (calculated P_k), a modified software implementation that uses constant floating-point coefficient values (stable K; *Kalamn Gain matrix*), and the reconfigurable KFO (KF1 in Figures 4.3 and 4.4) and KFD (KF2 in Figures 4.3 and 4.4) hardware blocks. Figures 4.3 and 4.4 show enlarged views of the noisy data fed into the Kalman filter block along with the various outputs observed both for distance and angle values. These plots are a zoomed in plots in order to highlight the different outputs of different filter implementations. It is important to note that the measured data contains high variations in it due to the inaccuracy of the radar sensor used or any radar sensor [M.RICHARDS 2005]. However, the use of an accurate filter can reduce the error and eliminate the sensor's noise.

After testing KFO with data sets from different environments, the average error in the distance and angle estimation relative to the original software implementation is 23 centime-

ters and 0.007 degrees, respectively. These results confirm the lower accuracy of the KFO filter, but also illustrate its suitability for open environments where targets are relatively far from each other.

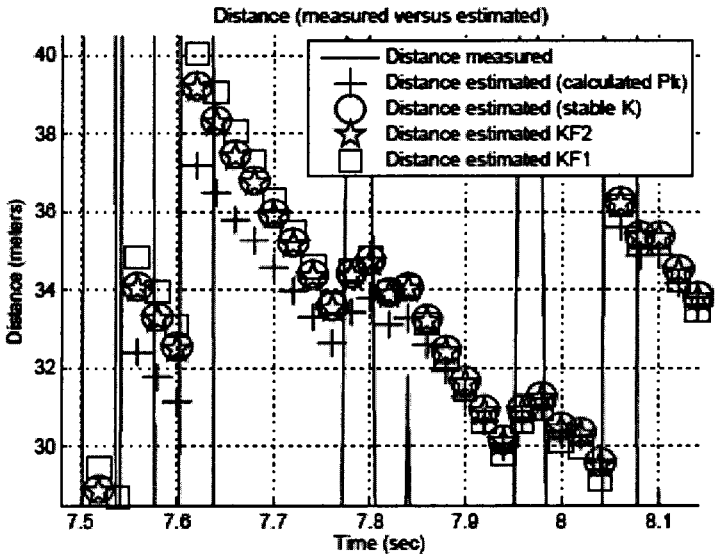

Figure 4.3 – Enlarged view of measured distance and output of different Kalman filter implementations.

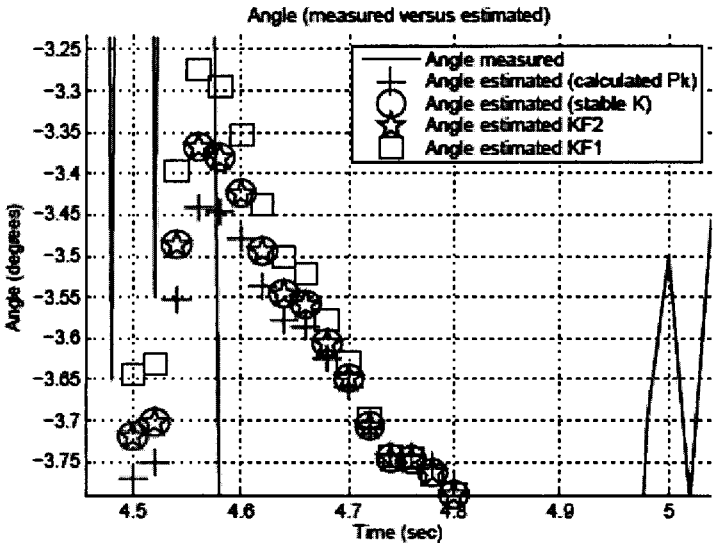

Figure 4.4 – Enlarged view of measured angle and output of different Kalman filter implementations.

Next we tested the KFD filters using the same data used with the KFO filter. Here we observed an average error in distance and angle estimates relative to the original software implementation of 5.2 centimeters and 4.6×10^{-4} degrees, respectively. These results illustrate the higher accuracy of the KFD filter and its suitability for tracking targets in dense environments.

4.3.d. Resource utilizations

Table 4.1 summarizes the resource utilization for both hardware filter implementations (KFO and KFD). It is also important to note that the resources consumed by either implementation are allocated as part of the reconfigurable region. This is due to the current technology for implementing reconfigurable regions. In general, a reconfigurable region can be specified as a rectangular area in the FPGA fabric. This region must contain all the hardware resources required by the most complex reconfigurable module implemented inside of it. As can be seen in Table 4.1, the dimensions of the RR region were set to accommodate the slices required by KFD. Although the overall utilization of slices for both filters with respect to the reconfigurable region shows high efficiency (97%), the utilization of other resources within the region (DSPs, RAMB16s, etc...) is not as high due to the rectangular design constrain of a reconfigurable region.

HW resources	KFO	KFD	RR	Utilization
Slices	2837	3555	3646	97%
Slice Flip Flop	306	568	8352	7%
4 input LUT	5285	6897	8352	80%
DSP48s	6	14	28	50%
RAMBs	0	0	42	0%

Table 4.1 – Hardware resources used by the different Kalman implementations. This table also shows the hardware resource utilization of an implemented filter with respect to a reconfigurable region RR.

Table 4.1 does not summarize all the resources required by the new implemented MTT system. In addition to the resources required by the HW block, we must also take into consideration the resources needed by the HWICAP block, CF System ACE, and the DCR socket and its bus. These components constitute the static part of our new system. Table 4.2 summarizes the resources used by the static region and the total resource utilization when added to those of the reconfigurable region.

HW resources	Static	RR	Total
Slices	2033	3646	5679
Slice Flip Flop	2752	8352	11104
4 input LUT	3331	8352	11683
DSP48s	0	28	28
DCM_ADVs	2	0	2
RAMBs	0	42	42

Table 4.2 – Hardware resources used by the reconfigurable region in addition to those consumed by the static IP cores.

Table 4.3 summarizes and compares both implementations in terms of hardware resources consumed and the resource reduction percentage of using the new MTT implementation. This table also shows the trade off in area of using a reconfigurable system compared to a software only implementation. As a conclusion of this table, a reduction of around 80% of hardware resources can be achieved by implementing the reconfigurable system. Note that

this table compares the resources utilized by only the *Filtering&Prediction* blocks in both the original MTT system and the new DPR system.

HW resources	Base SW Filter	DPR HW Filter	Resource reduction
Slices	37660	5679	85%
Slice Flip Flop	39080	11104	72%
4 input LUT	59780	11683	81%
DSP48s	140	28	80%
DCM_ADVs	0	2	-
RAMBs	160	42	74%

Table 4.3 – Base and new DPR Filtering&Prediction hardware resource utilization comparison.

4.3.e. Reconfiguration heuristics

We modified our original MTT application so it can reconfigure the Kalman filter block automatically. Our field studies had shown that, on average, in open environments the number of targets within range of the radar were fewer than six, while in dense environments this number increased to more than eight. Using these observations, we developed a simple heuristic that tracks the number of targets in range of the radar over the last five radar sweeps, and uses the appropriate filter implementation accordingly. The number of prior radar sweeps we track is arbitrary, but is done to ensure that the filter is not reconfigured unless the observed conditions favor one configuration over another. Figure 4.5 shows the pseudo code for the filter reconfiguration heuristic.

```
while (true) {
        //Track number of targets in last 5 sweeps
        for (i=0 ; i <= 3 ; i++) {
                targets [i+1] = targets [i];
        }

        //Check threshold and use appropriate configuration
        if (targets [0] <= 6 && targets [1] <= 6 &&
        targets [2] <= 6 && targets [3] <= 6 &&
        targets [4] <= 6) {
                Use KFO;
        }
        else if (targets [0] >= 8 && targets [1] >= 8 &&
        targets [2] >= 8 && targets [3] >= 8 &&
        targets [4] >= 8) {
                Use KFD;
        }
}
```

Figure 4.5 – Pseudo code for filter configuration heuristic.

4.3.f. Latencies and reconfiguration overheads

The latency analysis of such application is essential especially when this system is implemented as DAS system. In such systems, decision making should be fast in order to increase the reaction time of the driver.

When implemented as a software function, the Kalman filter consumed 4% of the MTT application's total execution time (Table 3.1). This time, 3 msec, is the same for

processing one or 20 targets since we have 20 processors implemented in parallel in the base system. Table 4.4 summarizes the execution times needed for the different Kalman filter implementations. It is important to note that for all implementations of the Kalman filter, the latency measurement includes not only the time spent processing data, but also the time spent transferring data to the filter module. We will therefore be comparing the time needed by all implementations to execute starting from sending the inputs, to receiving the final output. As shown in Table 4.4, although KFO and KFD have different architectures they still require the same processing time. This similarity is due to the data transfer time. As hardware blocks, KFO has a latency of 30 nsec, while KFD has a latency of 45 nsec, respectively. However, in our current implementation, the latency of the reconfigurable filter is determined by the largest and slowest of the two implementations. One other insight from Table 4.4 is that although for 20 targets, the HW filters consume more time than the software based implementation, it is important to remember that the HW filter is using 80% less hardware resources than the software based implementation. Also, the DPR based system can now support two driving conditions in the same small hardware utilized.

	Run time (1 Target)	**Run time (20 Targets)**
Kalman (SW)	3 msec	3 msec
KFO (HW)	181.3 μsec	3.6 msec
KFD (HW)	181.3 μsec	3.6 msec
Reconfiguration	192 msec	-

Table 4.4 – Latencies for different filter implementations and reconfiguration overhead.

Since the system uses a 100 MHz (10 nsec) clock, and KFD (the larger and slower block) has a latency of 45 nsec we used a 20 MHz (50 nsec) clock for the reconfigurable filter block. That is why both KFO and KFD will have a fixed latency of 50 nsec. Adding the data transfer time to this, results in a total latency of 181.3 μsec (Table 4.4).

The slower clock used for the reconfigurable hardware block will not affect the performance of the system for the following reason: If the block is still functioning on 100 MHz or at 20 MHz (100÷5), the time needed to fetch the first output, still covers both clock delays. In other words, the fetching command of a data from the socket consumes 1 μsec to execute. This means that whether the time needed by the hardware block to execute is 10 nsec (at 100 MHz) or 50 nsec (20 MHz), as long as the execution time is less than 1 μsec, the system will still consume 181.3 μsec.

Table 4.4 also shows the overhead of reconfiguring the HW block. During reconfiguration, the system will essentially be blind for eight radar sweeps. Although this time is not insignificant, it is nonetheless acceptable for this application for at least two reason:

1- First, the change in the environment is not likely to occur very frequently. Recall that the time needed to switch between filter implementations depends on the employed heuristics.
2- Second, the reconfiguration time does not adversely affect a driver's reaction time. The reaction time of a driver, as cited in [X.MA 2006], is around 2.7 seconds. In

the worst case, if a driver has to react while there was a change in the environment; the reaction time will be reduced to 2.5 seconds, which is still within the acceptable safety margin. Moreover, during this time, a target traveling at 120 km/hr would only move a few centimeters.

4.3.g. Conclusion

In this work section we described our implementation of a dynamically reconfigurable Kalman filtering hardware block for automotive multiple target tracking applications based on targets density. We also demonstrated that a simple heuristic can be used to modify the filter architecture in real time to match the tracking accuracy requirements of two driving environments. Our results show that we can achieve significantly lower latencies compared to a software implementation of the Kalman filter. Our results also show that the reconfiguration overhead is small enough to ensure safe and fast driver response times. Finally, our results show that using a reconfigurable region can significantly reduce FPGA resource utilization, but that this needs to be balanced with good resource utilization within the reconfigurable region.

4.4. DPR system based on targets position

In this part, we present another possibility to implement a DPR based MTT system. The important issue about this system is that partial reconfiguration is triggered based on the targets positions. Closer targets that have higher probability to be more dangerous will be tracked more accurately than further targets. We first show the blocks of an MTT system alongside extra blocks from the capturing and treatment phase. This step is important to show how free reconfigurable regions can be used as enhancement units as well. We then present a hybrid MTT system with new DPR based Kalman *Filtering&Prediction* block. We then show the different filter implementations and their respective functional environments. We highlight the differences between them and show the impact of each configuration on the total system performance. Plots, results and hardware resource utilizations are presented as a comparison between the new DPR system and the software based MTT system. A reconfiguration heuristic is illustrated to show when each configuration is best fit according to detected targets positions.

One major difference in this implementation is in the *Filtering&Prediction* block. The MTT application can be implemented using a variety of adaptive algorithms such as α-β filter, mean-shift algorithm, and Kalman filter, etc [S.BLACKMAN 1999]. As detailed in the following sections, each chosen filter has interesting features and could be used in specific situation. In our system, we have chosen to implement not only the Kalman filter but rather both the Kalman and the α-β filters. The Kalman filter is used because it is an important block in the whole MTT system. A Kalman filter is designed to track a moving object having a constant velocity. However, real objects actually tend to move in variable accelerations and

therefore different velocities. The choice of the α-β filter is due to its simple implementation and its consideration of target velocities in decision makings [S.BLACKMAN 1999].

4.4.a. Radar signal processing and enhancement block

Similar to Figure 3.1, we plot the same figure including the capturing stage blocks as shown in Figure 4.6. The radar signal processing blocks are responsible for analyzing captured signals and delivering them to the MTT system for decision making. The three main blocks can be seen in Figure 4.6.

- The Delay Estimation block is based on correlation and HOS (High Order Statistics) algorithms related to noisy radar signals or an FFT (Fast Fourier Transform) algorithm for continuous wave radars such as the one used in our system.
- The Detection Unit is an adaptive threshold based signal detector that is responsible for target identification and tracking [M.RICHARDS 2005]. The Detection Unit Enhancement block is used to enhance the detection unit and is responsible for amplifying and filtering weak signals detected from distant targets and obstacles. The motivation for implementing such a block is the fact that the side lobes of a signal recovered from close targets will hide the signals detected from targets farther away. More details will be given in a following subsection.
- The Sensor Data Processing block realizes data translation and interpretation in order to be delivered to the MTT MPSoC based system.

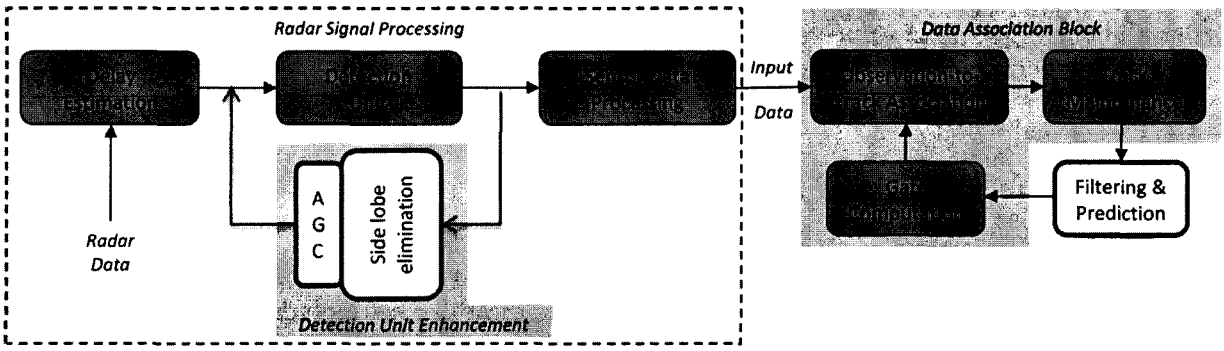

Figure 4.6 – MTT and radar signal processing blocks.

4.4.b. Validation system architecture

Similar to the previous Subsection, in order to implement a basic DPR system, first we need to implement a prototype like architecture. Figure 4.7 shows the architecture of our dynamically reconfigurable MTT system. Our system enables specific hardware blocks to be swapped on the fly. We implemented the system on a ML410 Xilinx board with an XC4VFX60 Virtex-4 FPGA. This system is used for validating the MTT's functionality and measuring data such as the reconfiguration overhead. Depending on the radar connected to the MTT and the number of obstacles to track, one or several hardware and software cores may be used. Here, as we focus on the dynamically reconfigurable hardware realization of the *Filtering&Prediction* block, we use only one PowerPC processor [POWERPC] to implement the data association block. Different other blocks in Figure 4.7 are of the same functionality as those discussed earlier in Figure 4.2.

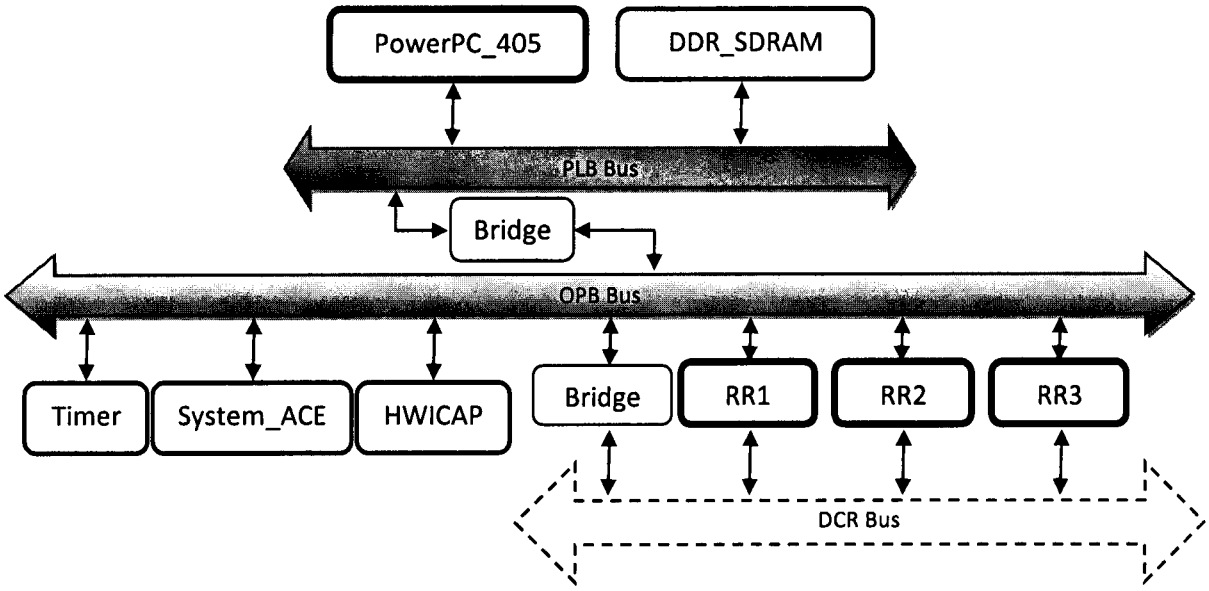

Figure 4.7 – Validation architecture basic IP cores. The MTT system runs on the PowerPC except the Filtering&Prediction Kalman block that runs on the "RR1, RR2 and RR3" reconfigurable hardware blocks.

4.4.c. Different filter implementations

Initially we implemented the Kalman filter as a dedicated hardware block, chapter 3, section 3.3. Our goal in this section is to study the ability of implementing different types of filters in reconfigurable regions. The main advantage of this implementation is its ability to implement different filter architectures having different characteristics in response to changing operating conditions.

We have designed two Kalman filter implementations having different tracking characteristics and an α-β extension filter. These three filters have different characteristics and can be used in different driving environments. They are used in three different configurations to support three different driving scenarios. Namely, the filter set includes a Kalman filter for angle estimation (KFA), a Kalman filter for distance estimation (KFD) and an α-β extension filter for both angle and distance estimations (ABF).

- *KFA Filter:* The Kalman filter for angle estimation (KFA) is a version of the original Kalman filter hardware block. This filter performs only the prediction of the angle.
- *KFD Filter:* The Kalman filter for distance estimation (KFD) is also a version of the original Kalman filter hardware block. KFD allows only the prediction of the distance.
- *ABF Filter:* The α-β filter is a simple filter mainly used for data smoothing and control applications. The ABF filter uses the same calculations as in an α-β filter not only for the distance estimation, but also for the angle estimation. This filter realizes the prediction of both the distance and angle with the help of linear and angular input velocities.

The two Kalman filters differ in their building blocks due to the different behavior of the input data. The tuning matrices, P_K and K matrices as mentioned in section 3.2.b, used to store different values for a distance input or an angle input are one of the major differences between these two filters. We also designed an extended α-β filter to provide even more accurate predictions for certain driving conditions. An α-β filter is used due to its smoothing characteristics of input data and ability to be mapped on a hardware block. Three reconfigurable regions, RR1, RR2 and RR3 are respectively mapped to three designed filters as shown in Figure 4.7.

4.4.d. *Filtering&prediction* configurations

The *Filtering&Prediction* Block configurations are directly related to a set of different combinations among the filters KFA, KFD and ABF. These configurations are associated with three regions: Zone 1 between 160 meters and the maximum radar range, Zone 2 between 100 meters and 160 meters and Zone 3 between 100 meters and 0 meters as shown in Figure 4.8.

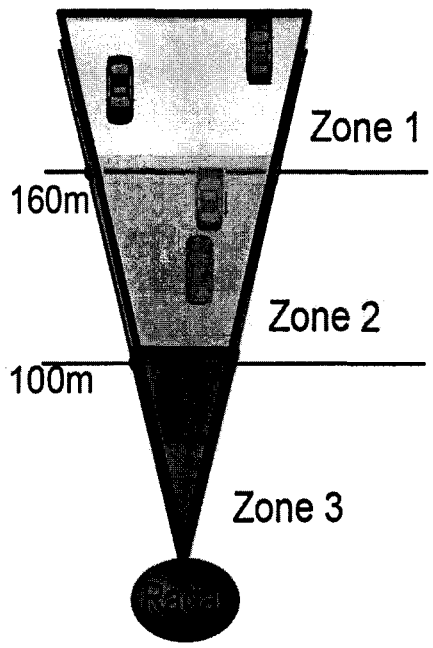

Figure 4.8 – Radar's FOV and Zone definitions. The Zones are identified by a radar mounted in front of the driver's car.

These regions are defined according to their proximity to the radar and are be mapped to 3 configurations simultaneously:

- <u>*Configuration 1 (C1):*</u> This configuration uses only the filter KFA. C1 is implemented when there are only targets tracked far away from the radar or those in Zone 1. These obstacles are considered to have a negligible hazard level. Also the distance estimation for such far away obstacles is considered to have a very low priority. Estimating just the angle for such obstacles is considered as an initiation step in the detection system. This means that when there are targets only in that region, then RR1 will be configured to KFA while RR2

and RR3 will just forward their inputs to the outputs. In addition to this, these regions (RR2 and RR3) will use different inputs for an enhancement block in the system. Only the signals related to the filtering block will be forwarded without any treatment.

- *Configuration 2 (C2):* This configuration uses both KFA and KFD. C2 will be used when targets are observed in Zone 2. Targets in that region can potentially be more hazardous than those in Zone 1. Thus, a better estimation will be computed for such targets by using estimated of both their distance and angle. Hence, RR1 and RR2 will be configured to KFA and KFD, respectively.
- *Configuration 3 (C3):* C3 includes the implementation of all KFA, KFD and ABF at the same time. If a target is spotted in Zone 3, it will be considered to have a high hazard level and a better prediction of its position is needed. This is why we consider ABF an enhancement over KFA and KFD that provides better distance and angle estimation of a target's data. In this configuration, RR1 will be configured as KFA, RR2 will be configured as KFD, and RR3 will be configured as ABF.

In configurations C1 and C2, the free reconfigurable regions are used as an enhancement unit related to the radar signal processing part as mentioned in the following subsection.

4.4.e. Detection Unit Enhancement block

The Detection Unit Enhancement (DUE) block is used to improve the detection of weak target signals captured from far away targets. As mentioned earlier, in section 4.4.a, closer target signals tend to hide farther target signals. In potentially hazardous situations where targets are spotted in Zone 3, the signals from these targets are the most important and are strong enough for analyzing and delivering to the MTT system. Other signals are not considered very important since they are not within the hazardous zone (Zone 3).

In situations where targets are only in Zone 2 and Zone 1, and with the free resources liberated from other reconfigurable regions, a better utilization of these resources can be achieved by implementing the DUE block in those regions. As an illustration, Figure 4.9 shows three detected signals, one from a target in Zone 2 and two other signals detected from targets in Zone 1. It is noticed that the signals detected from two targets in Zone 1 are hidden by the side lobes from the signal detected from the target in Zone 2.

Figure 4.9 shows the motivation for supporting a DUE hardware block whenever free reconfigurable regions are available. The DUE block consists of two basic functional blocks:

- Side lobe elimination unit and an automatic gain control (AGC) [M.RICHARDS 2005] for amplifying the weak signals as shown in Figure 4.14.
- After the side lobes are eliminated by the first unit, the AGC will amplify the signals after so that they will become ready for delivery to the MTT system.

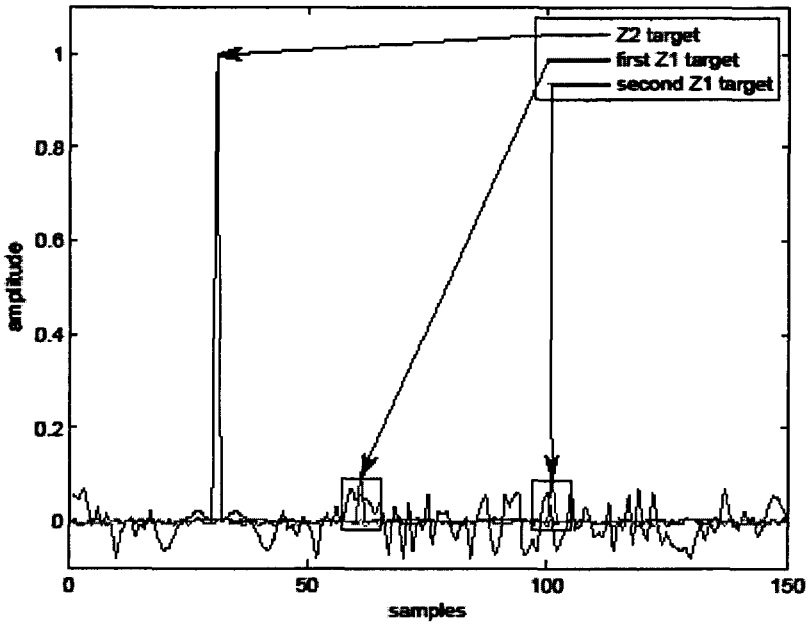

Figure 4.9 – Data corresponding to 3 detected targets in the Detection Unit. The 2 targets in Z1 are hidden by target in Z2 without the Enhancement Unit.

4.4.f. Filters output and accuracy

Our first set of experiments was designed to measure the accuracy of different filter implementations. We use actual radar distance, angle, and linear velocity and angular velocity measurements with different filtering block configurations. Here we present only the results for distance estimation with the 3 configurations C1, C2 and C3 in order to show the response of our system. In Figure 4.10 we show the measured distance of a target captured by the radar (system's input), its actual position, the output when implementing C2 (KFA+KFD output) and that when implementing C3 (KFA+KFD+ABF output).

When implementing C1, the output distance from the filtering block is identical to the measured distance so there is no error. On the other hand, our results show that the angle estimation error can reach a maximum of 1.8 degrees (15%), Table 4.5. This is similar to the angle estimation error achieved when implementing C2. However, since C2 filters the measured distance, it results in a distance estimation error that reaches a maximum of 6 meters (4%). Implementing C3, results in the lowest distance and angle estimation errors, which reach a maximum of 3.2 meters (2%) and 0.7 degrees (7%), respectively. These results show that the extended ABF reduces distance and angle estimation errors by 50%.

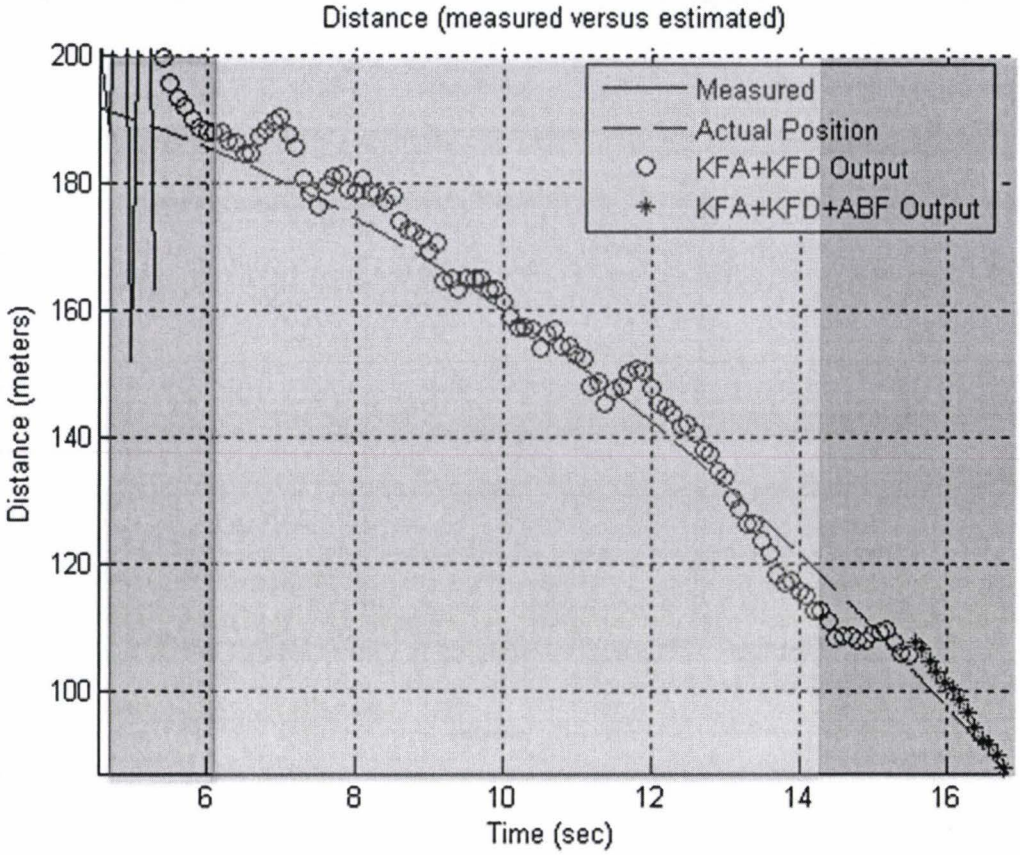

Figure 4.10 – Our DPR system distance output of a target moving across all different configurations.

Used filter	Name	Utilization	Zone/RR	Error
KF for angle	KFA	Used alone, D>160 m	Z1/RR1	Angle: 15% Distance: 20%
KF for distance	KFD	Used with KFA, 100 m <D< 160 m	Z2/RR1+RR2	Angle: 15% Distance: 4%
ABF for angle and distance	ABF	Used with KFA and KFD, D<100 m	Z3/RR1+RR2+RR3	Angle: 7% Distance: 2%

Table 4.5 – A summary table illustrating the filters used in each respective configuration. The error obtained from each configuration is also mentioned.

4.4.g. Resource utilizations

Table 4.6 summarizes the resource utilization for different hardware filter implementations (KFA, KFD and ABF). As can be seen in Table 4.6, the dimensions of the RR region were set to accommodate the slices required by each filter according to its respective reconfigurable region. Although the overall utilization of slices for both filters shows very high efficiency (99%), the utilization of other resources within the region (DSPs, etc...) is not as high.

HW resources	KFA	RR1	Util.	KFD	RR2	Util.	ABF	RR3	Util.
Slices	756	756	100%	936	931	99%	1319	1332	99%
4 input LUT	2477	3024	82%	3051	3744	82%	4323	5328	82%
DSP48s	7	10	70%	7	10	70%	2	18	12%

Table 4.6 – Hardware resources used by the different Kalman and α-β implementations. This table also shows the hardware resource utilization of an implemented filter with respect to the respective reconfigurable regions.

Table 4.6 summarizes the resources used by the reconfigurable regions RR1, RR2 and RR3. In addition to the resources required by the reconfigurable regions, we must also take into consideration the resources used by the HWICAP block, CF System ACE, and the DCR socket and its bus. These components constitute the static part of our new system and are also shown in Table 4.7. Table 4.7 also shows the system's total hardware resource usage.

HW resources	Static	RR1	RR2	RR3	Total
Slices	1415	756	931	1332	4434
4 input LUT	2611	3024	3744	5328	14707
DSP48s	0	10	10	18	38

Table 4.7 – Static and reconfigurable regions hardware resource utilization and total utilization by the new MTT system.

Table 4.8 summarizes and compares both implementations in terms of hardware resources consumed and the resource reduction percentage of using the new MTT implementation. As a conclusion of this table, a reduction of around 75% of hardware resources can be achieved by implementing the reconfigurable system. Note that this table compares the resources utilized by only the *Filtering&Prediction* blocks in both the original SW based MTT system and the new HW based DPR system.

HW resources	Base SW Filter	DPR HW Filter	Resource reduction
Slices	37660	4434	89%
4 input LUT	59780	14707	76%
DSP48s	140	38	73%
DCM_ADVs	0	2	-
RAMBs	160	48	70%

Table 4.8 – Base and new DPR Filtering&Prediction hardware resource utilization comparison.

One of the main contributions of the work presented in this section is the possibility of using some hardware resources in other blocks other than those in the MTT application. When the system is in C1 or C2, the reconfigurable regions not used in the filtering block can be exploited for different uses. Our experiments showed that the free resources can be used in the detection unit enhancement of the radar. This block is only an enhancement and is used to detect weak signals from farther detected targets.

4.4.h. Reconfiguration heuristics

The MTT application can reconfigure the filtering block automatically. The measured distance of any obstacle forms the basis for our reconfiguration heuristic. This information, along with the zone definitions will provide the decision to switch among the three configurations. For our experimental results we consider a real scenario where a target is traveling at 33.3 meters/sec (120 Km/hr) speed towards the radar [X.MA 2006]. Since the reaction time of the driver should be around 2.7 secs, a distance of around 90 meters is considered safe and thus defining Zone 3, section 4.4.d.

To verify the presence of an obstacle in a given zone, we rely on a window of 12, consecutive 25-msec-radar sweeps. This number is obtained through experimental testing. The value 12 is directly related to the user's relative linear velocity and the driver's reaction time and is therefore calculated frequently. 12 sweeps provide an additional margin of 10 meters around each zone threshold; 12 sweeps × 25 msec/sweep × 33.3 meters/sec = 9.99 meters.

For example if a target is initially spotted in Zone 1, 12 consecutive radar sweeps will be monitored to ensure that this target remains in Zone 1, upon which C1 will be instantiated. If the target moves closer and enters Zone 2, 12 consecutive radar sweeps will also be considered to ensure the target's position as well. These sweeps will be triggered for counting after the target crosses the 160 meters distance measured. Considering that the target is traveling at a speed of 33.3 meters/sec [X.MA 2006] and that each radar scan is 25 msec long, this will make our target ready to be tracked using C2 at a distance of 160-10=150 meters. The same applies if the target is entering or leaving Zone 1, Zone 2 or Zone 3. In summary, Figure 4.11 shows the reconfiguration triggering process based on measured distance.

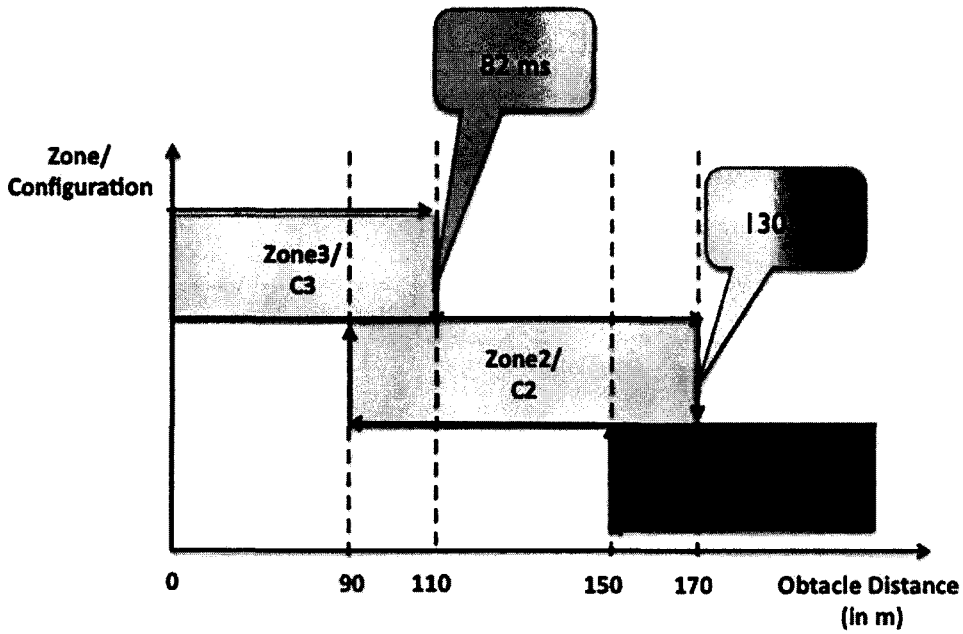

Figure 4.11 – Reconfiguration triggering distances supposing a target travelling at a speed of 120 km/hr.

Regarding the DUE unit, one region (RR3) out of the 2 free regions (RR2 and RR3) in configuration C1 will be implemented to as a DUE hardware unit. The same region will remain a DUE unit when the system changes into configuration C2 since RR3 will not change its configuration but only RR2. In configuration C3, the DUE unit will be replaced by the ABF filter.

4.4.i. Latencies and reconfiguration overheads

Latency analysis in driver assistant systems is very important to improve safety and ensure that drivers have adequate time to react to changing conditions. Starting from this point, we base our analysis on scenarios where targets travel towards the radar at 33.3 meters/sec (120 km/hr). Since the reaction time of the driver should be approximately 2.7 seconds [X.MA 2006], this results in a 33.3×2.7 = 90 meter safety buffer that we use to define Zone 3.

Regarding the implementation of the KFA, KFD and ABF filters, we note that they each have a latency of less than 50 nsec. Since these filters are in turn used to implement each of the three configurations, we note that the latency of configuration C1, which only uses KFA, is 50 nsec. Similarly, the latency of configuration C2, which uses filters KFA and KFD in parallel, is also 50 nsec. On the other hand, the latency of configuration C3, which uses the ABF to process the outputs of the KFA and KFD filters is 100 nsec.

Although pipelining can be used to maintain a 50 nsec clock, the additional hardware resources and system complexity are not necessary. Since the data transfer latency from a soft processor core to the filtering block is on the order of micro-seconds, the 100 nsec latency of the Filtering&Prediction block is not on the critical path.

The three reconfigurable regions can be reconfigured to KFA, KFD and ABF respectively as mentioned earlier. The time needed to reconfigure RR1 to KFA is 87.3 msec; RR2 to KFD is 82.8 msec; and RR3 to ABF is 130.1 msec. These are used in Table 4.9, which shows the reconfiguration time for the different scenarios that might occur.

	Reconfiguration time
0-C1 / C1-0	**87.3 msec**
C1-C2 / C2-C1	82.8 msec
C2-C3 / C3-C2	**130.1 msec**

Table 4.9 – Reconfiguration times needed for the switching among different configurations.

These results enable us to infer the number of radar sweeps that can be missed without filtering while reconfiguration is under way. The number of missed radar sweeps also enables us to validate the defined zones and the heuristic used to perform reconfiguration. For example, the AC20 radar has a maximum detection range of 200 meters.

When a target is detected in Zone 1 at a distance of 200 meters traveling at 33.3 meters/sec (120 Km/hr), four radar sweeps (100 msec) will be skipped due to the reconfiguration process. This still enables configuration C1 to be ready while the target is at 200-3.33=196.67

meters. When the target passes through the 150 meters boundary, configuration C2 will be triggered. In turn, four radar sweeps (100 msec) within Zone 2 will maintain configuration C1 until configuration C2 is ready to be used. Hence configuration C2 will be enabled while the target is still at a distance of 150-3.33=146.67 meters.

Finally, when the target passes through the 90 meter boundary, configuration C3 will be triggered. In turn, six radar sweeps (150 msec) within Zone 3 will maintain configuration C2 until configuration C3 is ready. Hence the target will be tracked using configuration C3 when it is at a safe distance of 90-3.33-1.665=85.005 meters. Note that even when being re-configured, our system is designed to continue operating and detecting obstacles using the previous configuration.

4.4.j. Conclusion

In this work a dynamically reconfigurable filtering hardware block for multiple target tracking applications in DAS based on targets positions was presented. Our system shows that there will be no reconfiguration overhead because the system will still be functioning with the original configuration until the system reconfigures itself. The free reconfigurable regions can be implemented as improvement blocks for other DAS system functionalities. For example all RRs can be KFAs for three radars attached to the MTT system.

4.5. DPR-MTT compared to enhanced soft core processors

Soft core processors come in different versions and configurations. This variety in choice, allows the enhancements in terms of latencies and hardware utilizations of any targeted and implemented system. However, this choice is directly related to the manufacturer and the utilized soft core processor. For example, the variety of soft core processor versions in Xilinx is not as wide as that in Altera based systems (MicorBlaze version 8.0 and NiosII versions 'e', 's' or 'f' soft cores) [MICROBLAZE] [NIOS].

In the work of [J.KHAN 2009], the author proposed enhancements on the MTT DAS system by utilizing enhanced soft core processors. Though an enhanced processor can execute in more time but utilize less hardware resources when compared to a simpler version of the same processor, the author's upper limit for the use of an enhanced soft core processor was its execution time. The author proposed the removal of the 20 Kalman processors from the base architecture, and replaces them with two processors instead as shown in Figure 4.12. One of the processors, Processor 1 *KFs 1 to 20*, was a NiosII/f processor with 16 Kbytes I cache. The second processor, Processor 2 *Housekeeper*, was a NiosII/e processor with no cache. These two processors represented the new enhanced *Filtering&Prediction* block. The author, as mentioned in chapter 3, pipelined his MTT system in three stages, one of them was after the *Filtering&Prediction* block. According to [J.KHAN 2009], the two processors based *Filtering&Prediction* block was sufficient to execute in around 25 msec.

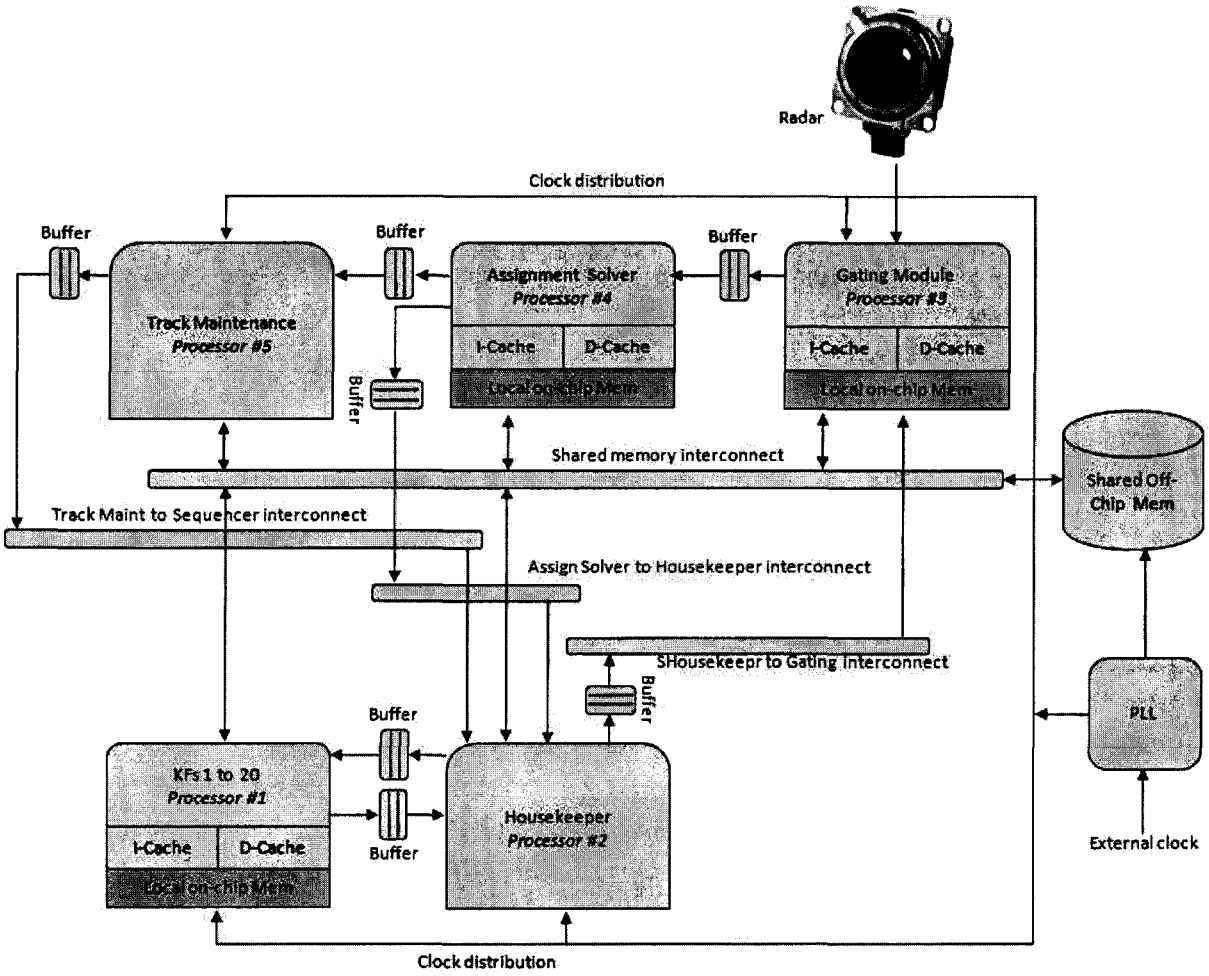

Figure 4.12 – Enhanced proposed architecture for an MTT based DAS system using an enhanced soft core processor instead of 20 simple processors for the Filtering&Prediction block. [J.KHAN 2009]

The rough comparison between our two proposed DPR systems and the new enhanced system in terms of hardware resources is presented in Table 4.10. The hardware resources utilized by the Base Filter are those needed for the implementation of five MicorBlaze soft core processors and a 16 Kbytes of on chip memory. The DPR Filter 4.3 and 4.4 represent the hardware resources utilized by our DPR implementation based on targets density and position respectively.

HW resources	Base SW Filter	DPR Filter 4.3	Resource reduction	DPR Filter 4.4	Resource reduction
Slices	7532	5679	25%	4434	42%
4 input LUT	11956	11683	3%	14707	-23%
DSP48s	28	28	0%	38	-36%
DCM_ADVs	0	2	-	2	-
RAMBs	40	42	-5%	48	-20%

Table 4.10 – Enhanced base and new DPR Filtering&Prediction hardware resource utilization rough comparison.

As shown in Table 4.10, the DPR based systems consumed little more hardware resources than the enhanced base architecture in one implementation (DPR Filter 4.4). However, the execution time is faster in our DPR than in the previous implementation since the new base system consume 25 msec and not 3 msec anymore (Table 3.1). A DPR system also dynamically supports different driving conditions without the need of any user feedback or input. Also, as shown in section 4.4, our DPR system can support enhancement blocks other than those of the MTT algorithm using the same hardware resources instead of leaving them to become absolute.

4.6. Conclusions

In this chapter, we proposed two scenarios where DPR can be beneficial in the MTT system. In the first system implementation, the filtering block was implemented as a reconfigurable hardware unit adapting to the number of observed targets. Results showed that our filtering block can adapt its accuracy and complexity according to the number of observed targets. As the number of targets increases, the system utilizes a more accurate filtering block in order to more accurately detect these targets. However, as the number of targets is reduced, the danger level of such targets is reduced and hence a lower accurate less complex filtering block is most fit for such an environment. Our results show that using a reconfigurable region can significantly reduce FPGA resource utilization, but that this needs to be balanced with good resource utilization within the reconfigurable region.

In the second proposed DPR system, the filtering block was implemented on multiple reconfigurable regions architecture. These regions can adapt their modules and architecture according to the proximity of detected targets. The detection region was divided into three zones where different filtering implementations were used for each zone. The furthest zone did not need a high accurate filtering block since targets in that zone are the least danger. However as targets get closer and enter the second and the third zones, the filtering block is adapted to provide higher accuracy levels of detections. This implementation however uses free reconfigurable regions in order to implement an enhancement detection unit block part of the detection phase. Final results show better hardware utilization when using a DPR architecture and the ability to use free resources for other enhancement blocks not necessary in the MTT system.

By also comparing to enhanced soft core processors, in conclusion, whether we used 20 or five soft core processors, the hardware may become absolute in certain cases. The DPR feature supports the ability to make use of absolute hardware for enhancement purposes (DUE unit), accuracy trade-offs (ABF filter) and the dynamicity to automatically support multiple driving conditions (DPR Filter sections 4.3 and 4.4).

5

Exploring DPR in an H.264 multimedia system

In this chapter, we present our second exploitation of DPR in embedded system design. This contribution concerns the utilization of DPR for making embedded systems more efficient while running multimedia applications namely the H.264 encoder. We first begin by introducing the H.264 multimedia system and its important functional blocks. We then explore the image quality and energy trade-offs of a dynamically reconfigurable H.264 motion estimation block. Our architecture exploits DPR to vary the granularity of the pixel masks used to compute the sum of absolute difference (SAD). The SAD calculation is needed to find the motion vectors between current and reference frames. This provides the ability to trade-off image quality for lower processing times and energy consumption. In addition to describing our architecture, we describe some of the optimizations we used to reduce the system's memory requirements. We also present a heuristic for automatically tuning the pixel mask size to meet image quality and energy consumption levels.

5.1. Introduction

During the last decade, the H.264 application's migration into hardware has been a very attractive area. Most of which have been capable of implementing complex computational blocks onto ASICs or on FPGAs. The use of ASIC platforms has proven to be very interesting platforms due to their low power consumptions and their flexible architectures in terms of hardware resources and memory blocks. FPGAs have been also explored for the H.264 implementation, partially and even entirely implemented on the same fabric [NOVA]. The use of FPGAs is due to their flexible programming and the variety of resources present on the fabric such as high speed memory blocks, Arithmetic Logic Units (ALUs) and embed

ded soft and hard core processors as well. Although FPGAs consume more power when compared to ASICs [A.AMARA 2006], however, the major advantage FPGAs have over ASICs is re-programmability. FPGAs can be reconfigured with a new design on the fly, while ASICs need both time and cost to implement a new design on the same fabric. Also, nowadays, FPGAs support the ability to partially reconfigure part(s) of its logic dynamically on the fly without the need to stop the system's execution. While the system is running, DPR allows part of the FPGA to be replaced by another needed computational unit or hardware accelerator.

In this work we target the Motion Estimation (ME) unit part of the H.264 encoder. As detailed in the following section, this unit is important due to its major role in the achievement of high compression rates. The ME unit encodes a motion vector of a set of pixels joined in a defined macro-block size. The lower the block size is, the better the quality of the video becomes but this also results in lesser compression rate. However, for certain video sequences, bigger macro-block sizes can be naturally used when there are not a lot of pixel changes between consecutive frames in the video. Hence, with larger block sizes, good quality is still achieved in this case and with higher compression ratios as well. The information from the ME unit, the motion vectors, is also used in the decoder as well to decode the video sequence.

The work in this chapter targets, using the DPR feature in FPGAs, the implementation of a dynamically partial reconfigurable ME inside an H.264 encoder. We present the computational complexity inside this unit and the implementation possibilities on an FPGA. Unlike other related work, we use the DPR feature not to replace our computational implemented units with other modules but rather to reduce the energy consumption. Our system is also capable to compute the ME of different supported block sizes by simply changing the data information sent to interconnected computational units. The flexibility of our system in terms of image quality and energy was the basis of our reconfiguration heuristics. We implemented our system on fabric and deducted results in terms of execution times, power measurements, and reconfiguration times and acquired image quality. Our system was tested to support mobile video services with resolutions such as Quarter Common Interface Format (QCIF 176×144).

The chapter is organized as follows: In section 5.2 we present a small introduction about the H.264 system in general. Following the H.264 introduction, we present an elaboration on the targeted computational unit inside the ME unit in section 5.3. A thorough analysis of this unit and the targeted computational blocks inside of it is studied and analyzed in the same section. After this study and analysis, section 5.4 presents the implementation details of these units on a DPR based functional system. In this section we give a more detailed analysis and description of the attached memory controllers to our proposed design. We also show the analysis and results achieved from the power and energy studies. Our system's control is described in section 5.4 via explaining the reconfiguration heuristics before concluding our work in the last section, section 5.5.

5.2. The H.264 encoder

In this section we introduce the basic building blocks and functionalities of the H.264 encoder. As shown earlier in Figure 1.5 and as shown in the next figure, Figure 5.1, the H.264 encoder is decomposed of three main stages: *Prediction, Transform and Quantization* and *Encoding* stages.

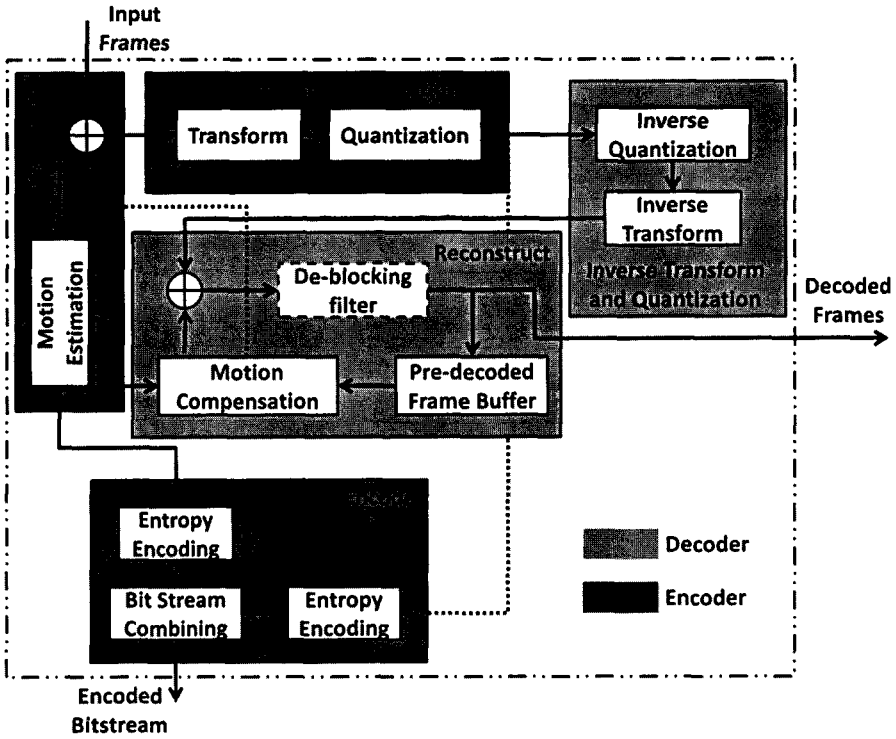

Figure 5.1 – Detailed figure showing the basic building blocks of an H.264 encoder and decoder. Also, the figure shows the placement of the ME unit and its relation among the encoder and the decoder of the multimedia system.

1. Starting with the *Prediction* stage of the encoder, it is made up basically of one unit, a Motion Estimation unit. *Prediction* exploits the redundancy of specific blocks of pixels in multiple frames in the video. This process is done using a motion estimation unit that predicts the placement of the pixels' blocks in consecutive frame(s) in the video. The block size of the pixels can vary between 4×4 to 16×16 pixels. Also, the number of consecutive frames, the motion of the pixels' block is predicted at, can vary from one to ten consecutive reference frames [J.W.CHEN 2006].
2. Another stage in the encoding process is the *Transformation and Quantization* stage. The main functionality of this stage is to reduce the coding information and scale down the transformed coefficients [J.W.CHEN 2006]. The most popular transformation and quantization block used in this stage is the DCT image and video compressor. The DCT also reduces the residual error data which is the difference between the actual and predicted data.

3. The final stage in the H.264 encoder is the *Encoding* stage. This stage is also known as the *Entropy Coding*. The *Entropy Coding* is responsible of converting all the syntax based elements, such as the motion vectors from the *Prediction* phase and the quantized coefficients from the *Transformation and Quantization* stage, to encoded bit streams of the video. The H.264 standard support two entropy coding methods: Context Adaptive Length Coding (CAVLC) and Context Based Adaptive Arithmetic Coding (CABAC) [J.W.CHEN 2006].

5.3. The H.264 encoder's Motion Estimation

Like most multimedia systems, the H.264 is composed of an encoder and a decoder. The main focus in this work is directly related to a functional block that is common between both the encoder and the decoder; The ME unit. This unit performs most of its computations to conduct the motion vectors in the encoder and the same information is used in the decoder. The ME estimation unit, as shown in Figure 5.1, shows its relation between the encoder and the decoder. Figure 5.1 also shows other functional blocks implemented on a DPR architecture such as the de-blocking filter in the decoder and the quantization (DCT) block in the encoder.

Our work is based on the implementation of the ME unit on an FPGA on the basis of using DPR. Hence, in this section we will give a detailed analysis of this unit, its major computational function block and the complexity of this block. This will provide a better understanding of the proposed implemented system and explore its efficiency when implementation and experimental results are discussed.

5.3.a. Motion Estimation

One of the main characteristics of an H.264 encoder is its high compression ratios, which are mainly achieved through the ME unit. The ME unit computes motion vectors for specific groups of pixels across multiple frames, and only encodes the motion vectors instead of encoding the pixels themselves. The accuracy of the compressed frames depends on the size of the pixel blocks used to compute the motion vectors. In general, higher levels of accuracy are achieved using smaller pixel blocks at the expense of higher computational requirements. The pixel block dimensions can vary between 4×4 and 16×16.

The most accurate method for computing motion vectors involves an exhaustive search, which is shown in Figure 5.2. For every pixel block in the Current Frame, a 63×48 pixel Search Window, centered on the same block's position in a Reference Frame, is used to find a best match. A Computation Window having the same size as the pixel block (16×16 pixels block) is used to scan the Search Window in one-pixel horizontal and vertical increments, and is used to compute the sum of absolute difference (SAD) with the pixel block. The best match corresponds to the Computation Window that results in the minimum SAD, which can then be used to compute the motion vector. Figure 5.2 shows a 16×16 pixels block, which

provides the best compromise between accuracy and computational complexity [P.KUHN 1999].

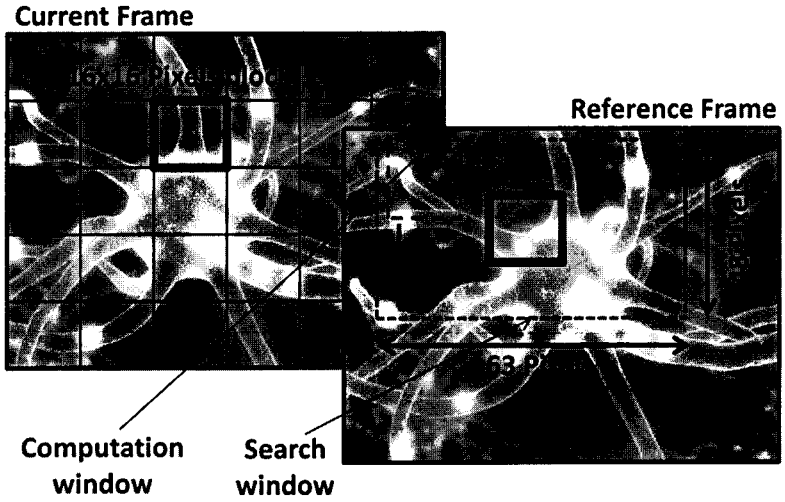

Figure 5.2 – Motion estimation of a group of 16×16 pixels in a Search Window among a Current and a Reference Frame.

Earlier studies have shown that the motion estimation corresponds to a significant proportion of the execution time of a H.264 encoder [Y.SHENGFA 2006]. To validate these results, we used the Joint Video Team (JVT) ISO/IEC JM codec [MPEG4] to profile the H.264 encoder and found that motion estimation accounts for an average of 40% of the encoder's run time. We tested some videos using the JVT simulator on an Intel Pentium IV 2GHz processor with 1 GBytes as RAMs to verify the average execution time of the ME unit. Profiling results of the ME unit are shown in Table 5.1.

As shown form the table, the ME unit consumes on average 42% of the H.264's encoder total execution time. Given the high frequency of the ME function and the significance of the SAD operation in computing it, we chose to implement the motion estimation function as a hardware accelerator block.

The SAD computations are used in the ME unit to select the best matching motion vector. The following equation, equation (5.1) shows the mathematical formulation for SAD computations. In this equation, "C" and "R" correspond to the dimensions of the pixel block (i.e. the block size). Each pixel is represented by Y, U and V values where Y corresponds to the luminance component and U and V correspond to the chrominance (color) components. Since the ME unit operates on Y values only, "cur" and "ref" correspond to the Y values for a given pixel in the current and reference frames, respectively. At the same time, "l" and "k" correspond to the coordinates of the Computation Window within the Search Window.

$$SAD_{C\times R} = \sum_{i=1}^{C}\sum_{j=1}^{R} |cur(\mathrm{i},\mathrm{j}) - ref(\mathrm{i}+\mathrm{l},\mathrm{j}+\mathrm{k})| \qquad (5.1)$$

Video	Execution time (%)
Forman	42
Akiyo	45
Bridge_Close	41
Bridge_Far	47
Carphone	43
Claire	48
Coastguard	39
Container	43
Hall	44

Table 5.1 – Profiling results of the ME unit for various video samples. The table shows the % of the ME unit execution time out of the total encoder's execution time.

5.3.b. SAD analysis and observations

The SAD computations depend on two main factors: pixel values and block size. The pixel values determine the magnitude of the difference between pixels in the current and reference frames, while the block size determines the group of pixels that are being considered. As the block size increases from 4×4 to 16×16, fewer $SAD_{C\times R}$ computations are needed. However, Figure 5.3 shows the total number of SAD computations for a 16×16 block using different C×R pixel block dimensions for a Computation Window to slide all across the Reference Frame's search window. This shows that the same computations can be made using Computation Windows with different dimensions. For example, to compute the SAD for a 16×16 block, we can use sixteen 4×4, four 8×8, or one 16×16 Computational Windows. As we will prove in section 5.4, the size of the Computation Window can be used to optimize energy consumption and image quality.

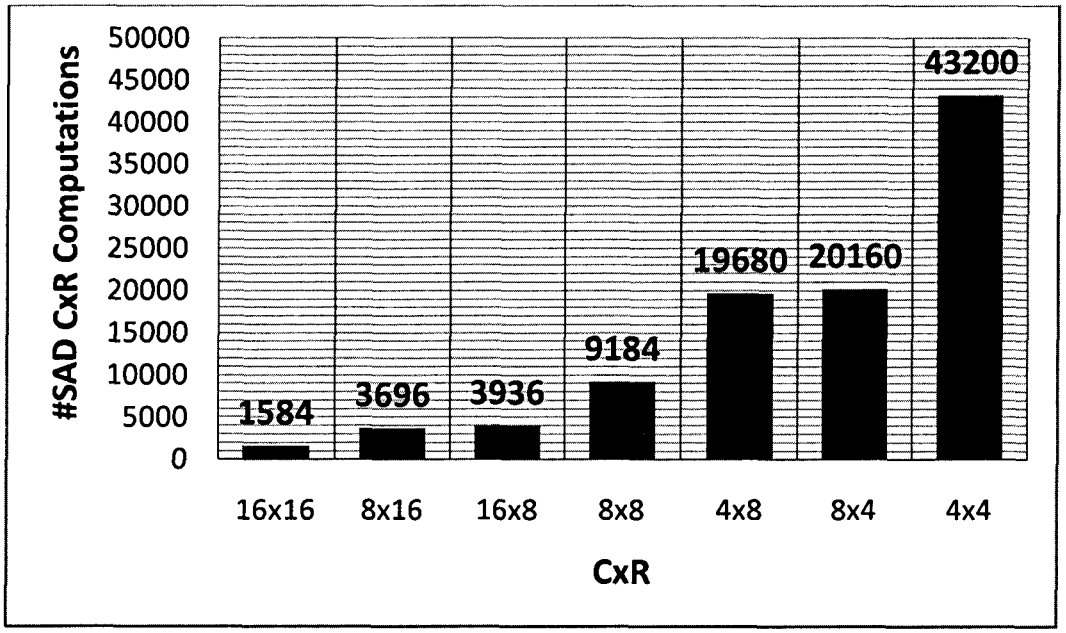

Figure 5.3 – Total number of SAD computations for different computational window sizes for a 16×16 pixels block.

We also show that a SAD computation for an arbitrary block size can be composed from SAD computations for the smallest block size. Figure 5.4 shows a 4×4 block divided into four 2×2 regions labeled a, b, c, and d, respectively. Replacing the values of C and R by four in equation (5.1) we can derive equation (5.2 a). If we now treat the block in Figure 5.4 as four blocks of size 2×2 pixels, we can reduce equation (5.2 a) to equation (5.2 b). This shows that a SAD4×4 can be computed as the sum of four SAD2×2 computations. It follows that an arbitrary $SAD_{C\times R}$ can be computed as the sum of appropriate terms of $SAD_{4\times4}$ computations.

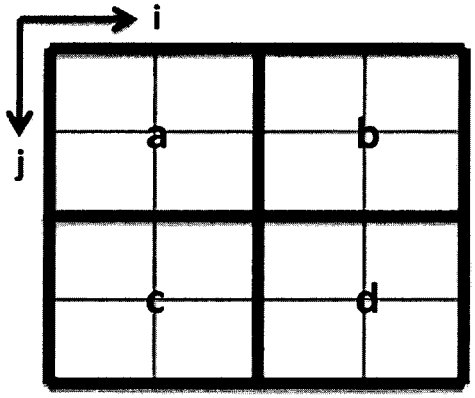

Figure 5.4 – A 4×4 pixels block composed of four 2×2 pixels blocks.

$$SAD_{4\times4} = \sum_{i=1}^{4}\sum_{j=1}^{4}|cur(\mathrm{i},\mathrm{j}) - ref(\mathrm{i}+\mathrm{l},\mathrm{j}+\mathrm{k})| \qquad (5.2\ a)$$

$$SAD_{4\times4} = SAD(a)_{2\times2} + SAD(b)_{2\times2}\, SAD(c)_{2\times2}\, SAD(d)_{2\times2} \qquad (5.2\ b)$$

As we show in section 5.4, our ME computational unit is designed around a reconfigurable array of $SAD_{4\times4}$ blocks. Figure 5.5 shows the resulting number of $SAD_{4\times4}$ computations needed to compute the motion vectors for a 16×16 pixel block using different sized C×R block sizes. Hence, by multiplying the number of SAD computations of block size 16×16 by 16, 8×16 and 16×8 by 8, 8×8 by 4, 4×8 and 8×4 by 2 and 4×4 by 1, we get the results in Figure 5.5. It noticeable that by even standardizing all computations for different block sizes C×R to one block size computation 4×4, bigger block sizes consume less computations.

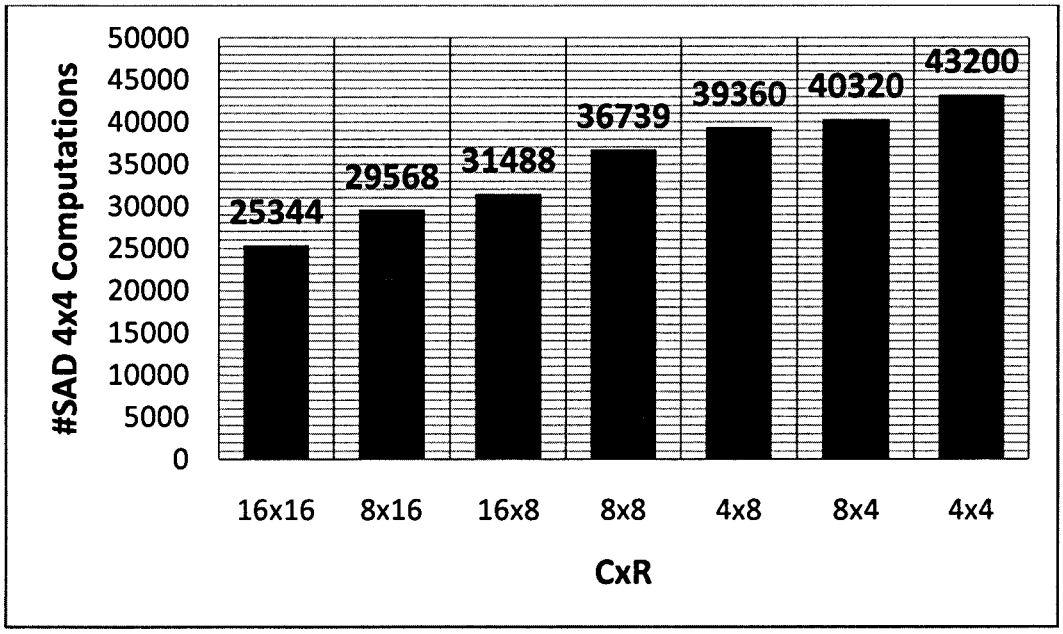

Figure 5.5 – Number of $SAD_{4\times4}$ computations needed for a 16×16 pixel image block using different sized search window blocks.

5.4. DPR exploration and implementation

In general, to study the ability of having the ME unit, or any suitable candidate, to be implemented on the DPR basis, few forecasted points must be addressed and studied. One point is to find out where and in which part of the designated system can DPR be implemented and found most beneficial. Our designated system is the H.264 encoder, and it was found out that the SAD computation block in the Motion Estimation unit was a suitable candidate to be implemented on the basis of DPR for the following reasons:

1. This block can be implemented as a hardware accelerator block. The SAD function does not need a large amount of hardware resources to be instantiated for it does not include complex functions such as division or exponential computations.
2. The SAD computational block of different block sizes can be implemented based on one basic building block. This means that any configuration can be implemented by scaling and replicating the basic building block. This makes the system a scalable system, which is one of the major advantages of the use of DPR when best explored.
3. The SAD computational block in the implemented system benefits from the hardware reuse feature of DPR. The system can reuse hardware, instantiate new hardware or erase some hardware blocks in order to achieve dynamic power/energy consumption.

Before describing the system implementation level details, it is important to mention the following: The DPR system is implemented to be a black box performing the SAD com-

putations of any block size. This implementation takes into considerations the sensitivity issues, which are addressed to be the available power and the image quality provided or required. The system is connected as a peripheral to a memory buffering architecture holding all necessary data to be processed. The DPR system presented is implemented in a manner where it can be connected to a simple memory buffer and add no extra complexity to the memory control unit. Following this note, along with the computational unit and the reconfiguration heuristic, we explored one DPR implementation possibility and design in the H.264 multimedia encoder.

For our system to achieve all previously mentioned points, it was theoretically studied, designed and then physically implemented. After the implementation phase, the system was fully verified and tested. Based on the theoretical design and actual implementation, we designed our reconfiguration heuristic based on power available and image quality demands. The advantages of the use of DPR in the H.264 encoder were validated by highlighting the energy and performance trade-offs in the overall system. Discussions and results were followed and conclusions were drawn.

5.4.a. Implementation and design

In order to build a basic DPR system, the essential components of the system must be studied, designed and implemented. We follow the theoretical approach that can best explore the benefits and advantages of the DPR feature in our system design. At first, and based on the theoretical analysis of the SAD computational block, we designed a hardware SAD4×4 component. This component is used as our basic building block for all other block sizes the H.264 encoder can support (4×4, 4×8, 8×4, 8×8, 8×16, 16×8 and 16×16). A tree-like hierarchical structure was designed to support all block sizes based on the basic SAD4×4 building block as can be shown in Figure 5.6.

Sixteen reconfigurable regions have been instantiated as shown in Figure 5.6 marked from RR1 to RR16. The regions are placed in a way to make the architecture clearer and simpler to understand. The placement is done in this manner in order to minimize the total system complexity and reconfiguration heuristics as will be shown in a subsequent section. In our system, a reconfigurable region can either be implemented as SAD4×4 hardware block or be a blank region.

The proposed architecture also includes some static blocks and control logic. The static blocks are adders and are marked by a sign "+" in Figure 5.6. There are 15 summation blocks in the static part of the architecture. The architecture also has some control logics which are basically enable signals of the summation blocks. In Figure 5.6, five levels of outputs can be found: Level 1, 2, 3, 4 and Level 5. When the system is configured on Level 1, all the summation blocks are enabled and one output is drawn from Level 1. In the case the system is configured on Level 2, all summation blocks are enabled except the one at Level 1 (summation block beyond Level 2). Level 2 drives two outputs as marked in Figure 5.6. When the system is using Level 3, all summation blocks beyond Level 3 are disabled for they are not used at the time and this level derives four outputs as shown in the figure. In the case of Level 4, the same idea follows. All summation blocks beyond this level are disabled and

the level drives eight outputs. Finally, Level 5 drives 16 outputs and no summation blocks are enabled when this configuration is being used.

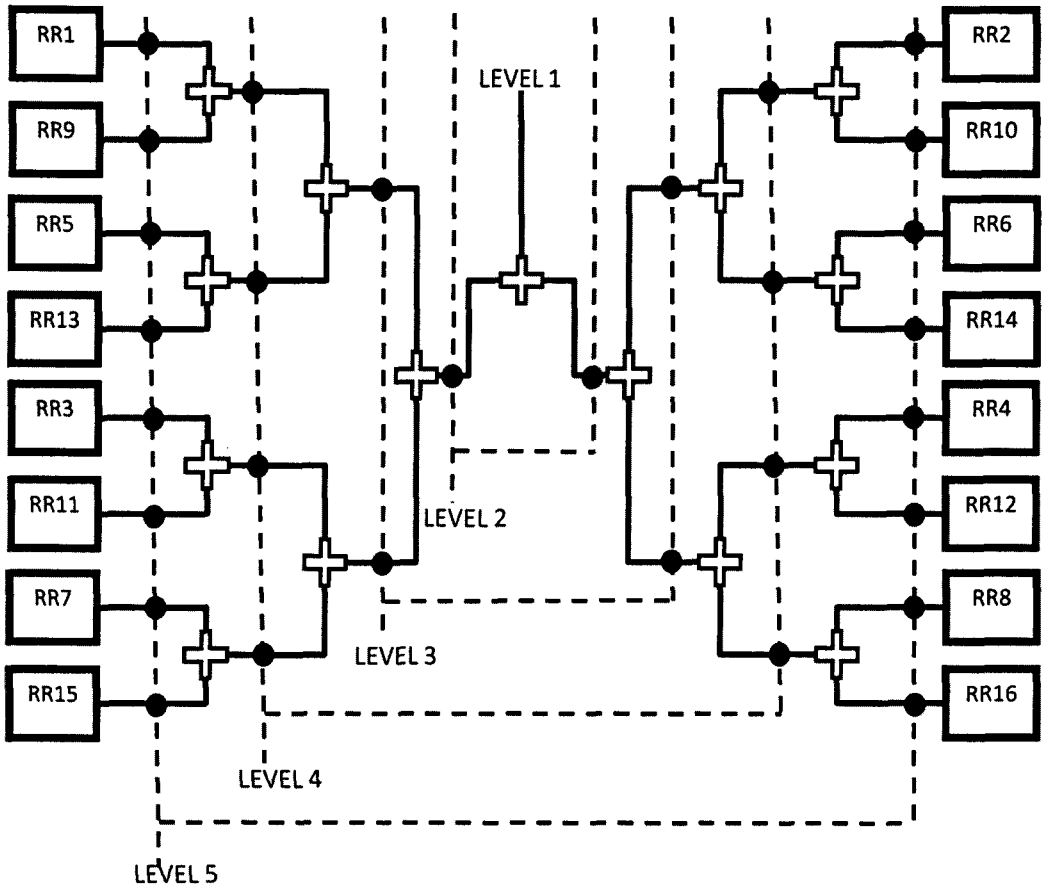

Figure 5.6- The design of our proposed architecture of multiple reconfigurable regions including static and fixed control blocks.

Based on the mathematical theory provided in the previous section 5.3, the proposed architecture follows the same manner and was implemented on the same basis. The summation of two regions means the addition of two SAD4×4 outputs. With the proper data sent as input of those regions, two SAD4×4 can function as the output of one SAD4×8 or one SAD8×4. The same applies to four SAD4×4, which can result in computing one SAD8×8. Also, eight SAD4×4 performs one SAD8×16 or one SAD16×8 and 16 SAD4×4 to perform the computations of one SAD16×16.

The regions placement over the summation mesh and the instantiations of different output levels is implemented in order to minimize the overall system complexity. The following example illustrates the complexity reduction. Note that in the following example, the block sizes and the allowed instantiated blocks are considered after the reconfiguration heuristics stage.

5.4.b. Illustration of example of the DPR computational unit

In this example we see the architectural form when the system can support the block size of 4×4 and is capable of instantiating four regions. Two steps are required for the process of reconfiguring and constructing the architecture available for this system.

The first step is to find out how many outputs the system should deliver. Since the block size is 4×4 and four regions can be supported, hence four outputs can be driven from such a system. Four outputs reflect that the selection of the respective level should be Level 3.

After that, since four regions can be supported, the regions RR1, 2, 3 and RR4 are configured to hold SAD4×4 respectively as can be shown in Figure 5.7. The output level selection is easy, when it is supposed to be 4, in this case since Level 3 can support exactly this number of outputs. This reduces certain output selection procedures and allows a faster level selection without the use of a higher selection method. In the case of this example, the level selection can be issued by enabling Level 3.

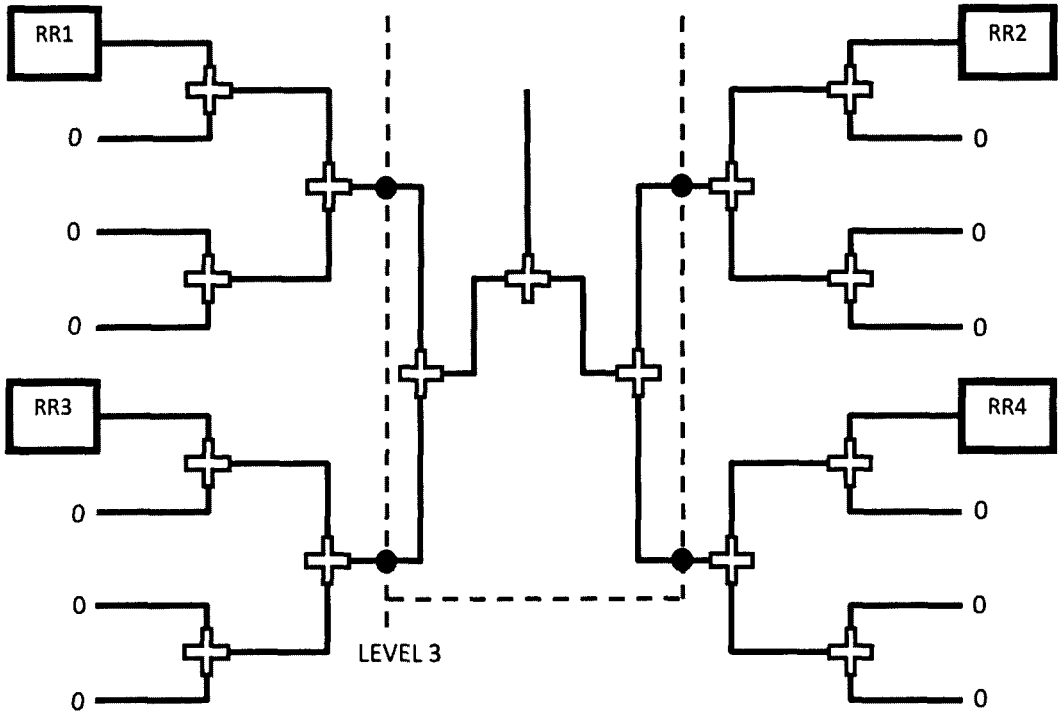

Figure 5.7- The architecture instantiated for an example. Four regions are configured in this system and the output selection level is Level 3.

The numbering and placement of the reconfigurable regions also reduces the reconfiguration control and heuristics. If the four nearby regions are to be configured, then the output level selection will change, in this case to Level 5. 16 outputs will be delivered but only the first four will be considered and the rest will not be. In addition to that, when the system changes its configuration, the less the control and reconfiguration performed the better and more optimized the system became. The system is more likely to change into a configuration that does not include high changes on the previously implemented architecture. This change

is to achieve either lower/higher power consumption, or due to a lower/higher image quality demand or both as will be explained in proceeding sections.

After foreseeing the system's architectural overview, and after identifying the functionality and manner such a system should function on, certain procedures were followed to verify the stated design. At first, the basic computational block, which is the SAD4×4, was implemented and studied. Physical implementation of the hardware basic block was implemented to validate the block and verify its functionality.

5.4.c. Hardware aspects

After implementing and designing a proper architecture for our system to be based on, we focused on building the basic component of the architecture, the SAD4×4 basic building block. The SAD4×4 block built takes as an input 32 8-bit wide data operands. These 32 internal buses pass the data needed by a SAD function to compute, using two 4×4 blocks (16 bytes). The first 16 bytes (d1_0 to d1_15) driven, are the pixels values of the block in the Reference Frame while the second 16 bytes (d2_0 to d2_15) driven are the pixels values of the block in the Current Frame. The reason for choosing one byte data bus width is because a pixel value can range between 0 and 255, thus eight bits are enough to represent the input data. The block derives one output which is the SAD computed value "SAD" and is 16 bits wide. A clock is inserted into the block for synchronization "CLK" and an enable signal for enabling or disabling the output "ENA". The processing done inside the SAD4×4 involves the change of input data signals into integers, then perform the absolute difference and after, change into signals and sum all differences to form the output "SAD". Figure 5.8 shows an illustration of the SAD4×4 block with the corresponding input buses, internal computations, output and the control signals.

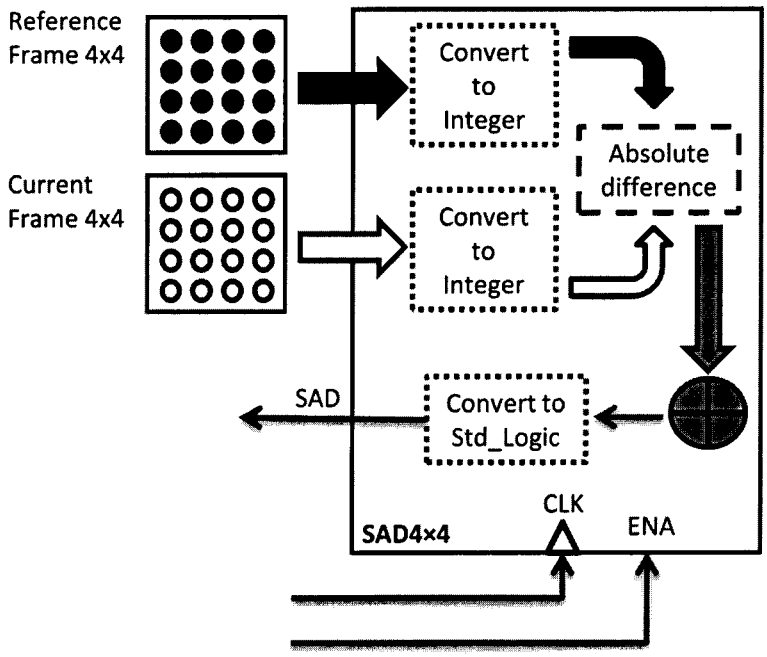

Figure 5.8 – Internal design of the SAD4×4 block. This plot contains the internal operations of the inputs in addition to the control signals as well.

The block is coded using the VHSIC (Very High Speed Integrated Circuit) hardware Description Language (VHDL) programming language [VHDL]. The design was fully compiled and synthesized using the Xilinx ISE synthesizing tool version 12.4 [ISE12]. The code was implemented, synthesized and tested via simulations before the implementation. To test the block, we used simulated data and verify the output. This was done to insure the functionality of the block and validate its functionality before the actual board implementation. The following example demonstrates the procedures of the validation process. The data driven to the component after feeding a clock "CLK" and enabling the block by raising to '1' the enable signal "ENA" are found in Table 5.2.

Data bus	Integer value	Binary value	Data bus	Integer value	Binary value
b1_0	1	00000001	b2_0	1	00000001
b1_1	5	00000101	b2_1	1	00000001
b1_2	7	00000111	b2_2	1	00000001
b1_3	8	00001000	b2_3	1	00000001
b1_4	9	00001001	b2_4	3	00000011
b1_5	26	00011010	b2_5	2	00000010
b1_6	10	00001010	b2_6	1	00000001
b1_7	10	00001010	b2_7	5	00000101
b1_8	10	00001010	b2_8	7	00000111
b1_9	10	00001010	b2_9	1	00000001
b1_10	11	00001011	b2_10	15	00001111
b1_11	12	00001100	b2_11	20	00010100
b1_12	13	00001101	b2_12	20	00010100
b1_13	1	00000001	b2_13	21	00010101
b1_14	5	00000101	b2_14	21	00010101
b1_15	7	00000111	b2_15	5	00000101

Table 5.2 – Simulated fed in data for testing the SAD4×4 block.

If we calculate the sum of absolute differences of the data found in Table 5.2, the result will be 130, which is '0000000010000010' in binary. Figure 5.9 shows the simulated signals of the data mentioned in Table 5.2. In the left side of the figure, we can see the signals declarations and the respective bit bus widths. The bottom left side of the figure shows also the clock signal "CLK", the enable signal "ENA" and the output "SAD" signal with its respective bit bus width of 16 bits. The right side of the figure shows the simulated signals with their binary representations. In the bottom of the figure, we can notice that the value of "SAD" when the data of Table 5.2 are fed is '0000000010000010' which is 130 in its integer format. This validates the SAD4×4 block and that the block is properly functioning.

After coding the computations of the component and forming the SAD4×4 block, we instantiated the block on hardware and implemented it too. For prototyping the block we implemented it on a Virtex 4 'XC4VFX60-11FFG1152C' FPGA [V4_FPGA]. In order to optimize the block and make it consume the only needed hardware resources, we had to update the timing constraints file associated with the instantiation of the SAD4×4 block. The higher the timing constraint is, the more the flexibility of the synthesis and routing tool has become and in turn the tool is more relaxed in the placement of the hardware resources of the block. By decreasing the timing constraints, the synthesis and routing tool is forced to try as much

as possible to place the hardware resources of the block as close as possible in order to meet this timing constraint. When the tool fails to meet the constraint, it will signal a 'timing constraint not met' error message. In that case, the timing constraint is set to the best case achieved where the tool was able to synthesize and route the hardware of the block. Our hardware SAD4×4 block, in summary, consumes 282 slices, 459 LUTs and can function on a maximum of 500 MHz clock frequency.

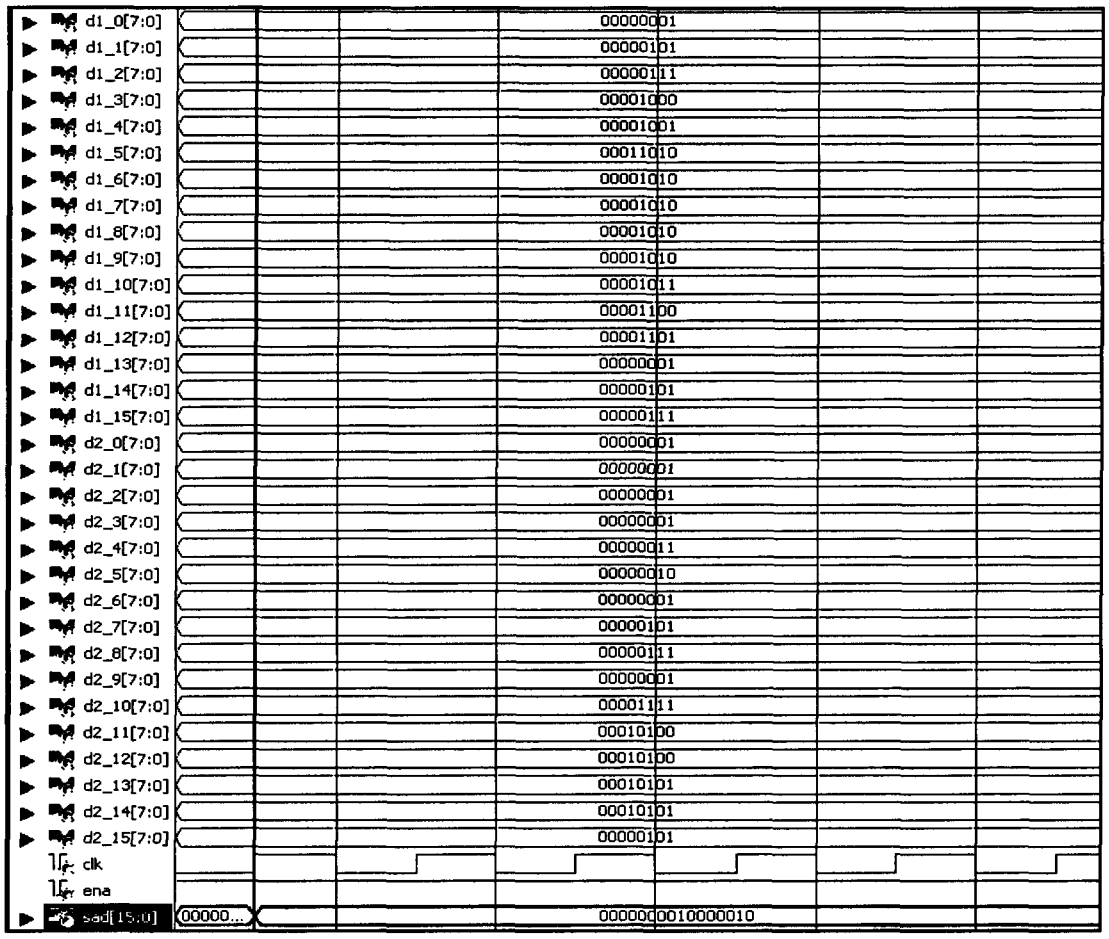

Figure 5.9 – Simulation output of the SAD4×4 block when processing the data in Table 5.2.

The very small execution time of the SAD4×4 block is due to two reasons:

1. The throughput of the block and the computations involved inside the SAD4×4. The SAD4×4 blocks takes all its input at the same clock edge. As we have already shown and discussed in Figure 5.8, 32 8-bit data buses are driven in parallel to the block. This makes our system a fully parallel implementation and with no pipelining stages.
2. For this short time constraint is the computations performed on the input data. The two inputs are converted to integers in parallel and then the absolute difference of the outputs is commutated in parallel as well, after that, the summation of the absolute differences is executed. This process takes very short time and the parallelism of the inputs makes the execution time even shorter and hence faster to ex-

ecute. Figure 5.10 shows the density of the routed signals and hardware the synthesis and routing tool had to implement in order to meet the 2 nsec timing constraint.

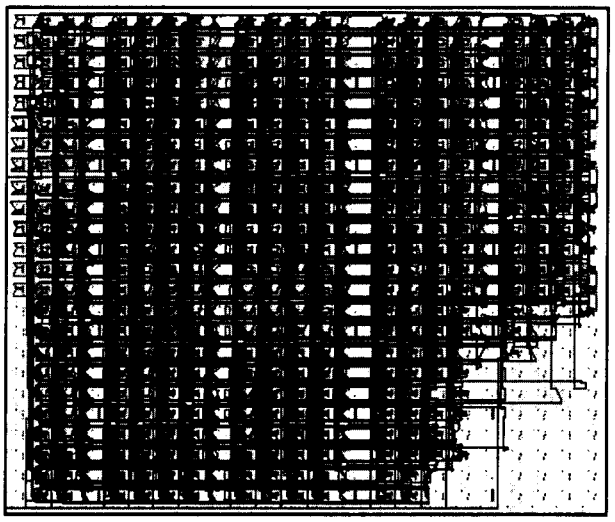

Figure 5.10 – Routed signals and hardware connections inside the SAD4×4 block.

After explaining the hardware aspects of our block, we will analyze this block when mapped into a reconfigurable region. One important aspect of performing reconfiguration is the time needed for this process to finish. This step is considered important due to the impact this process can have on the overall system.

5.4.d. Reconfiguration time analysis

In every dynamic partial reconfigurable base system implementation, the reconfiguration time plays a major factor in the design process. In general, one advantage of the use of the DPR feature is that the FPGA will continue working when part of it is being reconfigured. But the system should also know when the reconfiguration process will end in order to adapt to the new configuration.

Reconfiguration time is important for the reconfiguration heuristics as well. The reconfiguration heuristic, when knowing how long the reconfiguration process can take, can predict how much time the system can continue functioning in the current configuration until the new configuration is available. This provides prior knowledge of the system's functionality and ensures proper configuration at the right time.

One other important point to mention regarding the reconfiguration time is that when power is related to the system's performance, it is better to try to minimize this time as much as possible. The reason for doing so is to reduce the energy consumption of the total system. Since the system is spending more time to perform reconfiguration, this will add on its energy consumption. The more the reconfiguration time is reduced, the less the energy loss of the system has become. This point will be mentioned in the following section (section 5.3.e)

In order to acquire the reconfiguration time, we used build-in hardware timers to acquire this time. The hardware timer is a build-in IP core that could be attached directly to a

processor. When the timer is triggered, it sends the number of clock cycles from the command was executed till the timer was triggered. When the timer is stopped, it sends the number of clock cycles spent when it was initially triggered. By subtracting these two values, we can get the total number of clock cycles spent between the stopping and triggering of the hardware timer. When this number is multiplied by the frequency of the clock the processor is running on, we get the time the counter read. This time is basically the execution time needed by the commands executed between the starting and the stopping of the hardware timer. To measure the reconfiguration time, the reconfiguration command lines are put between the start and stop commands of the timer.

In general, when a system includes reconfigurable regions, the bit files that host the configurations in these regions are stored in a Compact Flash (CF) memory card. Since the CF is the easiest way to transfer files to the board including the FPGA, it is used as a default memory hosting the configuration files. Taking this into account, few steps are followed to perform a partial reconfiguration process. First, the HWICAP must be initialized and tested if any errors were in the hardware of the component. After that, a function is called to read the desired bit file from the CF and sent to the HWICAP controller. At that stage, the reconfiguration process starts after isolating the targeted region.

We designed a sample self-reconfigurable system with HWICAP in order to physically measure the reconfiguration time. We used the Xilinx version 9.1.02_PR10 design tools in order to build our system.

As shown in Figure 5.11, the system consists of multiple components which are mandatory in order to perform partial reconfiguration. A HWICAP IP core is included with its inner controller and cache BRAM responsible for performing partial reconfiguration. A hardware timer IP core that will be used to measure the elapsed time needed to perform the reconfiguration process. A processor is used to host the software commands and communications with the peripheral components. This processor could either be a PowerPC v.405 processor [POWERPC], or a MicorBlaze v.5 [MICROBLAZE]. A backbone OPB is used to connect the processor and all the other peripherals. A flash memory controller peripheral is added to communicate with the CF memory.

In our system, since all 16 regions are identical, we are interested in the measuring of two reconfiguration times: one is the reconfiguration time spent to configure a SAD4×4 module in a reconfigurable region. The other is the reconfiguration time spent to configure a blank module in a reconfigurable region, or in other words, the blanking of a region. After performing the reconfiguration process multiple times for both the SAD4×4 and the blank module, and on a system's clock of 100 MHz, Table 5.3 was concluded.

Module	.bit File Size (Kb)	Reconfiguration Time (sec)
SAD4X4	127	1.4824
Blank	105	1.0972

Table 5.3 – bit file sizes and different reconfiguration times for the SAD4×4 and the Blank modules.

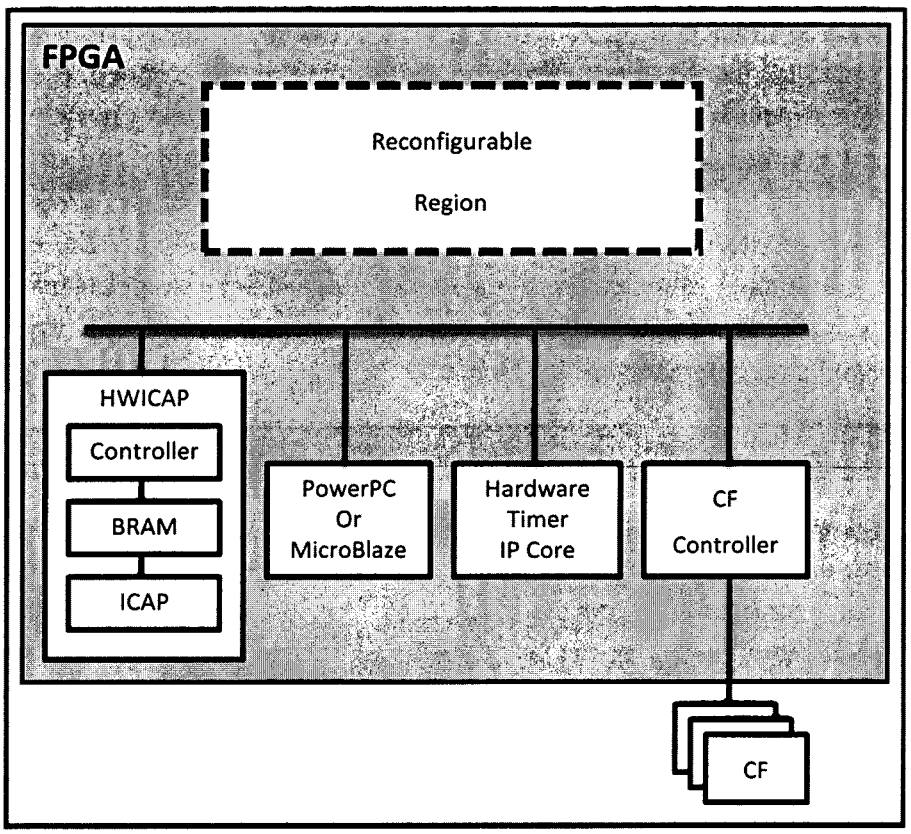

Figure 5.11 – Example of a basic partial self reconfigurable system including all necessary IP peripherals to perform a partial reconfiguration and measure the time needed to finalize the operation.

Table 5.3 includes the bit file sizes in addition to the reconfiguration times needed for each of the SAD4×4 and the Blank modules to be reconfigured in a single reconfigurable region. The bit file includes information about the module's logic to be configured inside a reconfigurable region. We designed our reconfigurable regions to be as dense in logic as possible. This justifies the almost similar bit file sizes among the two modules. The bit file contains location addresses of logic elements alongside an activate/deactivate value for each logic element (CLB, BRAM, Slice, LUT, FlipFlops...) [V4_FPGA]. The bit file sizes are close in size and this is due to the bit file format and repetition. In other words, the component used inside a reconfigurable region utilizes most, but not all, of the reconfigurable region's resources and hence the bit file mostly contains 'ones'. The same applies for the blank bit file; it contains 'zeros' for deactivating logic blocks in the region. Since the same disk storage is needed for storing '1' or '0', the bit file sizes are very close in size [V4_FPGA]. Our system uses 16 reconfigurable regions in its final implementation. This requires the reservation of 3712 Kbytes of memory space to store all partial bit files since we need 16 SAD4×4 and 16 Blank bit files.

From previous results, the reconfiguration time is considered very long especially if the system is to be implemented for real time applications. In the worst case we have to reconfigure 16 regions at the same time with a SAD4×4, thus resulting in a reconfiguration time of 16×1.4824=23.7184 secs. Since an H.264 encoder receives a new frame every 60

milliseconds, hence losing around 390 frames due to the reconfiguration process is considered unacceptable. In order to reduce the reconfiguration time, certain techniques have been proposed in literature in order to minimize this time. The approach we used in our system is based on the system proposed in [S.LIU 2010_1]. The authors used a DMA based streaming engine of partial bit files to an intelligent ICAP controller. Their technique has proven to reach up to 395-400 Mbytes/sec which is almost the optimal ICAP throughput (400 Mbytes/sec). We tested this system with a variety of reconfigurable regions with different bit file sizes. The results are presented in Figure 5.12 that shows the reconfiguration time according to different bit file sizes.

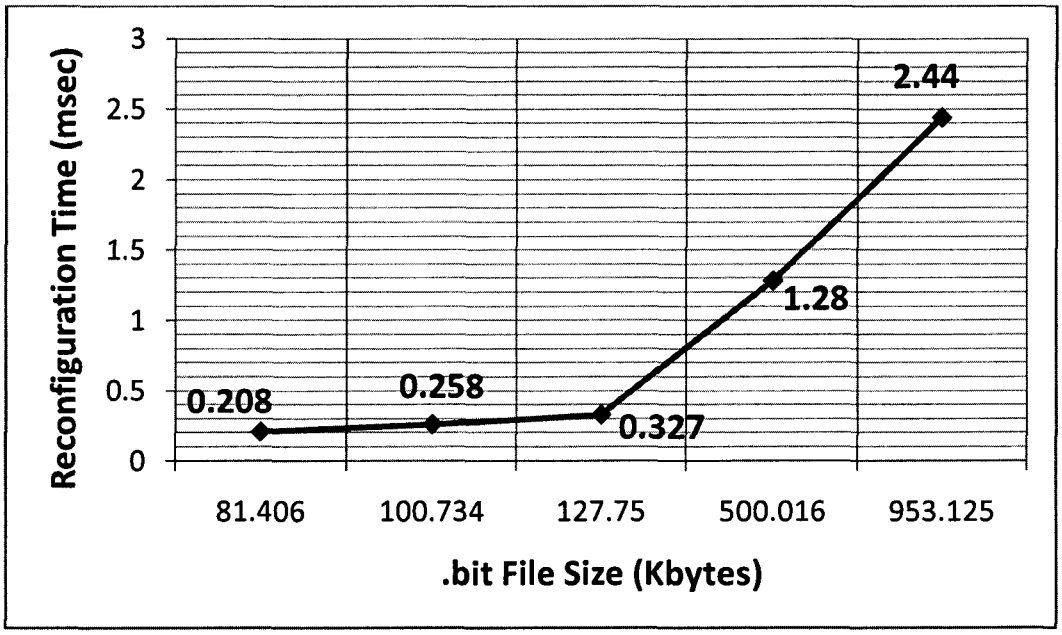

Figure 5.12 – The different reconfiguration times measured when using the component in [S.LIU 2010_1] when a system is reconfigured with a variety of bit file sizes.

Based on the results in Figure 5.12, and with the use of the same technique in [S.LIU 2010_1] we were able to reduce the reconfiguration times of our bit files to those in Table 5.4. As noticed, the reconfiguration times are highly reduced and hence can be used for real time applications. In the worst case we have to reconfigure 16 regions at the same time with a SAD4×4, thus resulting in a reconfiguration time of 16×0.000327=0.005232 secs << 60 msecs.

Module	.bit File Size (Kb)	Reconfiguration Time (sec)
SAD4X4	127	0.000327
Blank	105	0.00028

Table 5.4 – bit file sizes and different reconfiguration times for the SAD4×4 and the Blank modules using the technique in [S.LIU 2010_1].

5.4.e. Memory interface design and architecture

In order to fully exploit the benefits and the performance of our computational system we need to design a memory buffering system attached to it. This system is composed of memory hardware blocks as well as a controller that manages data from the memory blocks to the computational units. As mentioned in section 5.3.a, the ME unit uses data transfers from current and reference frames to perform its internal computations. Data from each frame is sent to the computational units to compute the SAD values. Thus, every RR should be able to access data from both frames to deliver a proper and valid output. For this reason, we implemented two memory units consisting of multiple memory blocks each to store the pixels of the current and reference frames, respectively.

i. System exploration

In our design we use an exhaustive search algorithm to compute the motion vectors. This requires scanning a large Search Window using a sliding computational window that is moved one pixel at a time in the horizontal and vertical directions. However, other search algorithms, with lower computational requirements, can be used without degrading accuracy significantly. In video multimedia systems, accuracy is measured by the Peak Signal to Noise Ratio (PSNR) of the video sequence.

According to [C.TING 2003], many algorithms search for a best match in specific regions within smaller Search Windows than the 63×48 window used in the exhaustive search algorithm. These include New Three Step Search (NTSS), Diamond Search (DS), and Hexagonal Search (HS). The NTSS algorithm has been proven to achieve the same average PSNR as an exhaustive search algorithm [D.CHUN 2008]. The NTSS algorithm shifts the Computation Window 2-8 pixels within a smaller Search Window. This indicates that we can modify the exhaustive search algorithm by shifting the Computational Window in increments of two pixels without affecting accuracy significantly [C.TING 2003] [D.CHUN 2008]. As will be shown later in this section, the two pixels shift will allow us to store, more efficiently, pixels of frames in our memory system.

To better illustrate the computations demands on real systems, using two pixels shift, we considered the number of the SAD operations on a variety of video sequences. Figure 5.13 shows the number of SAD4×4 computations needed to compute the motion vectors, using C×R block sizes, in one video frame for five different video sequences (Quarter Common Intermediate Format (QCIF), Video Graphics Array (VGA), Super VGA (SVGA), Extreme VGA (EVGA) and High Definition (HD) video sequences). The exact values of the number of SAD4×4 computations for every video are found in Table 5.5.

The results in Figure 5.13 are based on those in Figure 5.5 but not only for a 16×16 pixels block using a C×R block size, but rather for a whole frame according to specific video format. The values are obtained according to the mathematical model in equation (5.3). Hence, by reading the #SAD4×4 value of a specific block size C×R from Figure 5.5, dividing it by two (2 pixels shifts and not one as in Figure 5.5), multiply it by the number of respective blocks in a certain video and then dividing it by the number of C×R blocks in a 16×16 pixels block, we get the results both shown in Table 5.5 and Figure 5.13.

$$SAD_{C\times R}(5.13) = \frac{SAD_{C\times R}(5.5)}{2} \times \frac{Video\ C}{C} \times \frac{Video\ R}{R} \div \frac{16\times16}{C\times R} \qquad (5.3)$$

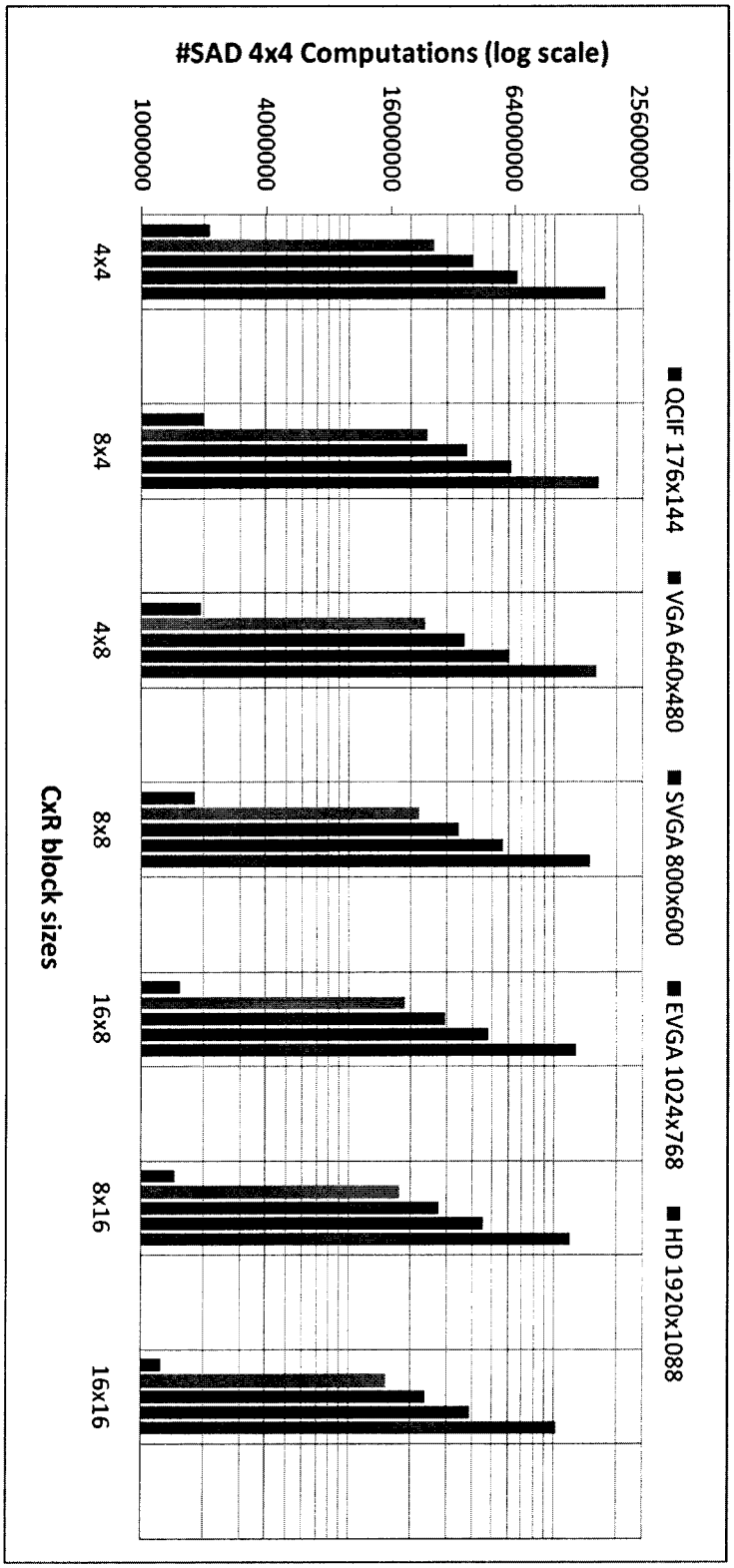

Figure 5.13 – Number of SAD computations based on respective block sizes, using two pixels shift, needed per frame for a variety of video sequences (the y-axis is in logarithmic scale).

Video	4×4	8×4	4×8	8×8	16×8	8×16	16×16
QCIF	2138400	1995840	1948320	1818581	1558656	1463616	1254528
VGA	25920000	24192000	23616000	22043400	18892800	17740800	15206400
SVAG	40500000	37800000	36900000	34442813	29520000	27720000	23760000
EVGA	66355200	61931520	60456960	56431104	48365568	45416448	38928384
HD	176256000	164505600	160588800	149895120	128471040	120637440	103403520

Table 5.5 – The exact numerical values of the SAD computations from Figure 5.13.

The SAD computations depend on the size of the Computational Window, which is assumed to scan the Search Window in two-pixel increments. These results show that the number of SAD4×4 computations decreases as the size of the computational block increases. For example, in QCIF videos, using 8×4 block sizes uses 7% less SAD4×4 computations than using 4×4 block sizes; 100-(2138400÷1995840)×100. This has implications on both the frame storage requirements as well as energy consumption.

ii. Memory organization

When designing our memory architecture, our goal was to develop a simple and fast memory unit that consumes the least amount of resources without compromising accuracy. Given the availability of embedded memory blocks in contemporary FPGAs, we decided to build our memory architecture around the random access memory blocks (RAMB36) found in Xilinx Virtex 5 and Virtex 6 FPGAs.

The RAMB36 is a dual-ported memory block with a capacity of 36 Kbits. These blocks can take two addresses and generate two 32 bit wide outputs. However, since a RAM block should be capable of storing and loading data, one port is used for input while the other is used for output. This restricts the bandwidth of each RAMB36 to 32 bits (four bytes) on each port. Each RR in our ME computational unit requires two 16-byte inputs to perform a SAD4×4 computation: one from the Reference Frame and another from the Current Frame. Since the SAD computation operates on 8-bit luminance data, each pixel requires one byte of storage.

To compute the motion vectors, and depending on their sizes, pixel blocks in the Current Frame are shifted in fixed increments of four-16 pixels in both the horizontal and vertical directions. The memory unit used to store the Current Frame is therefore organized to provide efficient access.

Starting with the Current Frame, in this frame the window shift is fixed and is at least a four pixel shift, hence a 4×4 block shift which is the lowest block size. The reason for four pixels shift is due to the fact that the blocks in the Current Frame are the ones we need to compare with in the Reference Frame and find their corresponding motion vectors. Since the smallest block size that the H.264 encoder can support is 4×4, and this block size is used for the block definition in the Current Window, hence the worst smallest shift the system should support is four pixels shift. Other cases should shift the block in the Current Frame by eight (8×16 for example) or 16 (16×8 for example) depending on the block size. If a memory sys-

tem can shift four pixels the block in the Current Frame, it can easily shift 8 or 16 pixels as well.

Using the previous explanation and the throughput of a RAMB36, in a Current Frame memory controller, the smallest memory block can have four RAMB36; 4×32=16×8 bits. Each single RR needs four RAMB36 (one current memory block) which means we need 64 RAMB36 in order to feed all regions if all were active in the worst case. And since we are using an incremental address controller for both Current and Reference frames controllers. It is easy to slide across the Current Frame with any desirable block size with the proper increments of designated current memory block(s).

Regarding the Reference Frame, in this frame the Computation Window (Figure 5.2) shift is at least one pixel. In that case and using the incremental address controller, each pixel will be stored in one RAMB36. This way when we increment 16 RAMB36 aligned on the same column we get a Computation Window shifted one pixel to the right (every vertical 16 pixels in the Reference Frame are in 16 RAMB36). This means that for each RR we need 16 RAMB36 (one reference memory block) in order to have a 4×4 block from the Reference Frame. In order to support 16 RRs, in the worst case, this means we need to use 256 RAMB36 which is considered a huge number of memory blocks and resources. For this reason we considered two pixel shifts which reduce the memory blocks used without changing the accuracy of the system. Two pixels shift means that the smallest memory unit should have four pixels data. This means that we can use one RAMB36 to host this block of four pixels next to each others. From this point we can achieve a reference memory block of size four RAMB36 for each single RR. In this case, we need only 64 RAMB36 memory blocks rather than 256. When shifting more than two pixels, two RAMB36 for each smallest window shift (3 pixels, 4 pixels, 8...) is used. Since the RAMB36 cannot send more than four bytes in a cycle, thus more than two pixels shift will still end up using one RAMB36. Also more pixels shift might hinder the accuracy of the system. The main insight from this explanation is that more than two pixels shift will use less reference memory blocks but with more RAMB36 in each.

Table 5.6 shows the number of RAMB36 usage per number of pixel shifts for both Reference and Current memory controllers. Notice that the Current Frame memory uses 64 RAMB36 blocks all the time and hence this number is constantly added whatever the pixel shifts were. By using two pixels shift we reduced the number of RAMB36 blocks used by 60% when compared to the one pixel shift.

Pixel shift	**1**	**2**	**4**	**8**	**16**
#reference memory blocks	256	64	16	4	1
#RAMB36 / block	1	1	4	16	64
#current memory blocks			16		
#RAMB36 / block			4		
#RAMB36 total	320	128	128	128	128

Table 5.6 – Total number of RAMB36 memory blocks used for different pixel shifts.

iii. Current frame memory controller

In the proposed DPR based computational unit architecture, not all regions are active all the time. Also the same block size is not used all the time. For these reasons, certain controllers have to be implemented to manage the correct data sent to the active regions taking into consideration the block size and the number of active regions selection.

We designed four units attached to the current frame memory controller controlled by the number of active regions and the block size selection where the block size can be one of seven possibilities each informing the controller which of the seven block sizes is being used. And the number of active regions can be one of five possibilities each informing the controller how many regions are active (1, 2, 4, 8 or 16). Figure 5.14 shows the current frame memory controller and a sample frame to demonstrate how the pixels in the frame are stored.

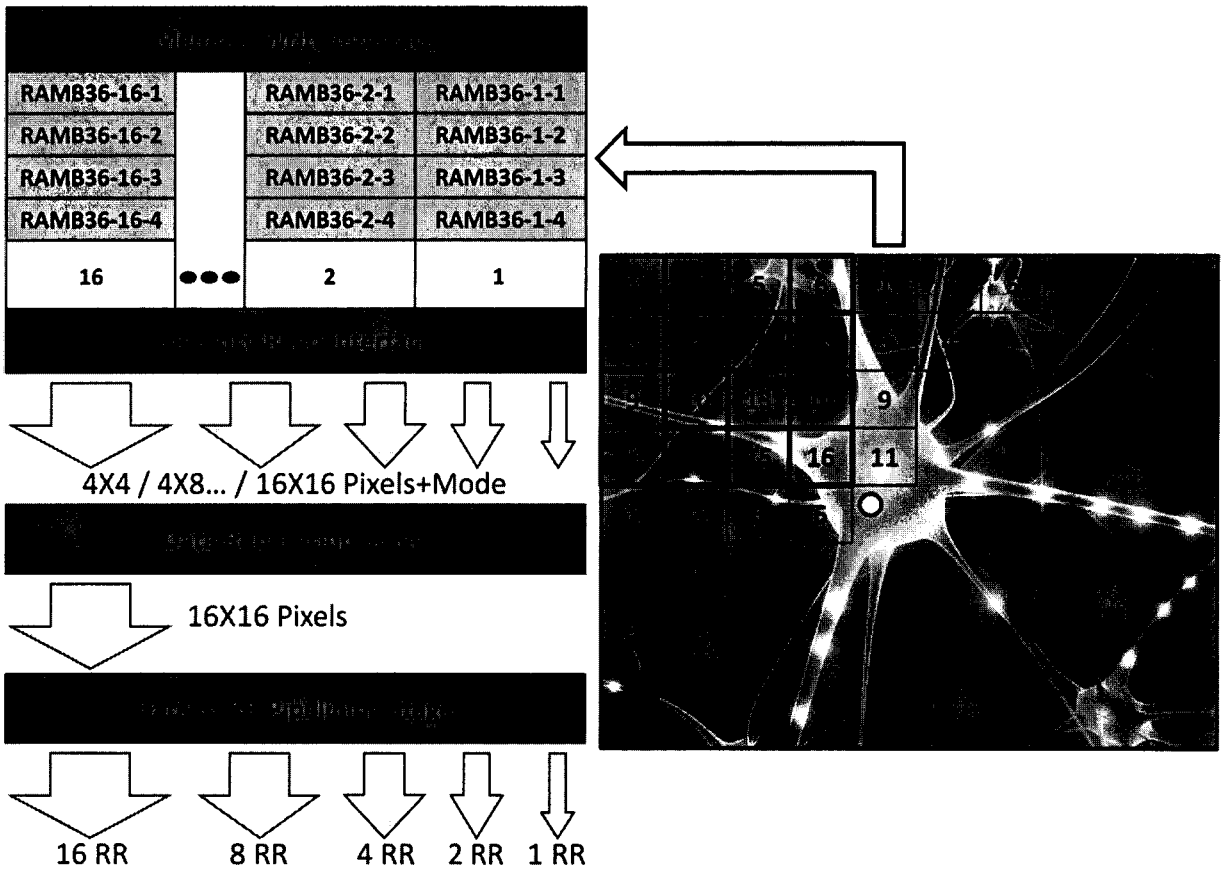

Figure 5.14 – Current Frame pixels' mapping on memory and the respective frame controller's building blocks.

In Figure 5.14 we show a frame divided into blocks each of size 4×4 pixels numbered from 1 to 16. We need four RAMB36 to store 16 pixels, where each group of four RAMB36 is joined to be one memory block. In order to deliver data to all 16 regions when active, we need 16 memory blocks, marked 1 to 16 in Figure 5.14. The storing of pixels in this controller is based on a combination of row and column-major order. The *Memory Write Interface* manages the mapping of 4×4 pixels from a Current Frame into one memory block. Each memory block stores each 4×4 pixels into four RAMB36 numbered x-1 to x-4. Whenever the

Current Frame is changed, this unit manages the re-writing of this frame in its proposed manner as show in the figure. Only one Current Frame is needed to be stored in order to compute the block of pixels motion vectors with respect to another Reference Frame.

The *Memory Read Interface* is the next unit attached to the memory blocks. It uses the selected block size to control which memory block(s) to read and which to increment their addresses for the next reading if needed. For example if all regions are active and we are using a 16×16 block size, the memory blocks are all read to deliver 16×16 block to the 16 computational units. When another 16×16 block is needed for processing, after finishing the computations over a Search Window, the addresses are incremented for all the memory blocks and hence, as shown in Figure 5.14, a new 16×16 block is used. This unit also controls data selection from each memory block if a smaller block size is used. For example if we are using an 8×8 block size, the memory blocks 1 to 4 are read and sent to the active computational units. When a new 8×8 block is needed, the controller sends data in memory blocks 5 to 8 and so on till the data in all 16 memory blocks are sent.

The current frame memory controller does not perform address increments of its memory blocks addresses as frequently as the reference frame memory controller. In other words, if we want to find the ME of a 4×4 block size, the same block is sent to all active regions in parallel in order to finish searching for its motion vector across the Search Window. For this reason, we designed the *Data Replication Stage* to take any data from the *Memory Read Interface* and transfer it to 16 16 bytes. Hence, if this unit receives 16 bytes (4×4 pixels block), it outputs 16 similar 16 bytes (16 4×4 similar pixels blocks). This controller functions independently of the block size or the number of active regions and its output is always 16 16 pixels.

After this stage, we designed a *Data to RR Pipelining Stage* that manages data according to the number of active regions. This unit takes 16 16 pixels and sends them in parallel or pipelined to the active regions. If 16 regions are active, 16 16 pixels are sent in parallel to these regions, each region receiving 16 pixels. However if for example only four regions are active, the 16 16 pixels are pipelined on four stages in order to send in each stage four 16 pixels to the four active regions. Hence, the only output from the current frame memory controller to the computational DPR unit is one bundle of 16 smaller buses each of 16 bytes width.

iv. <u>Reference frame memory controller</u>

The reference frame memory controller follows the same flow and design as the current frame memory controller. However, there are a few differences in control and organization between them. The first difference is the mapping of Reference Frame's pixels on the memory blocks. Each memory block in the reference frame memory controller is made up of one RAMB36. Each memory block stores 2×2 pixels and hence in order to store the whole 16×16 pixels we need 64 memory blocks as shown in Figure 5.15. The second difference is that this controller does not use a *Data Replication Stage* unit as will be proven later in this section.

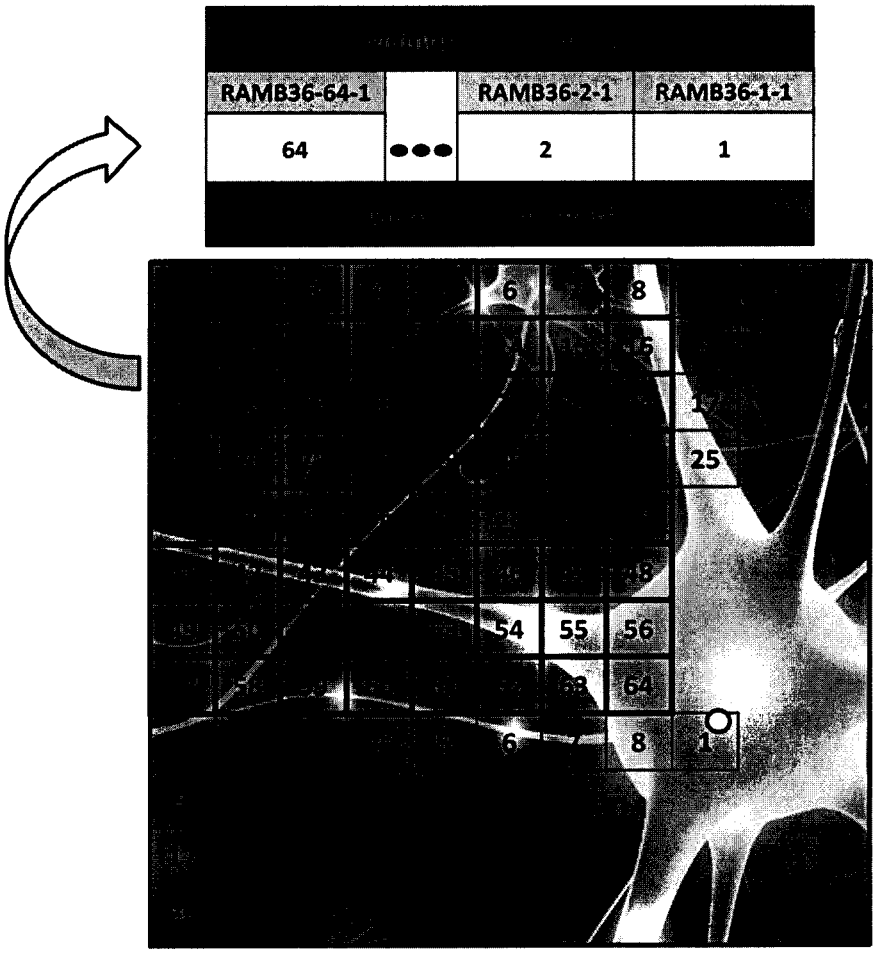

Figure 5.15 – Reference Frame pixels' mapping on memory and the respective memory controller's different building blocks.

In Figure 5.15 we show a frame divided into blocks each of size 2×2 pixels numbered from 1 to 64. In order to deliver data to all 16 regions when active we need 64 memory blocks marked 1 to 64 in Figure 5.15. The storing of pixels in this memory is based on a row-major order. We show in Figure 5.15 the *Memory Write Interface* and the *Memory Read Interface* only since the rest of the units are the same as those in the Current Frame memory.

The *Memory Write Interface* manages the mapping of 2×2 pixels from a Reference Frame into memory blocks. Each memory block stores each 2×2 pixels into a single RAMB36 labeled x-1. When the Reference Frame is changed, this unit manages the re-writing of the new frame in its proposed manner as in Figure 5.15. Multiple Reference Frames can be stored in this memory that will be used to compute motion vectors with respect to one Current Frame. The number of Reference Frames can range from 1 to 10 [I.M.JUAHIR 2006]. In the following section we will present how many frames can fit in both the current and reference frames memory units.

The *Memory Read Interface* is attached to the memory blocks that use the selected block size to control which memory block(s) to read and which to increment its address for the next reading if needed. If 16 regions are active and we are using a 16×16 block size, all the memory blocks 1 to 64 are read to deliver a 16×16 block to the 16 computational units.

When a new 16×16 block is needed for processing, only memory blocks 1-9-17-25-33-41-49-57 addresses are incremented and hence, as shown in Figure 5.16.a, a new 16×16 block is used. The new 16×16 block is shifted in the Search Window two pixels to the right. This unit also controls data selection from each memory block if a smaller block size is used. For example if we are using an 8×8 memory blocks with four or less number of active regions, the memory blocks 1-2-3-4-9-10-11-12-17-18-19-20-25-26-27-28 are read and sent only, Figure 5.16.b. When a new block is needed, the controller sends data in 2-3-4-5-10-11-12-13-18-19-20-21-26-27-28-29, Figure 5.16.b, and so on. However, when eight regions are active for example, the Memory Read Interface sends data from memory blocks: 1-2-3-4-9-10-11-12-17-18-19-20-25-26-27-28 (1st 8×8 block), and 5-6-7-8-13-14-15-16-21-22-23-24-29-30-31-32 (2nd 8×8 block); Figure 5.16.c. In the next cycle we send data from memory blocks representing the bottom two 8×8 blocks. However in the 3rd cycle, we send data from memory blocks: 2-3-4-5-10-11-12-13-18-19-20-21-26-27-28-29 (1st 8×8 block), and 6-7-8-1+1-14-15-16-9+1-22-23-24-17+1-30-31-32-25+1 (2nd 8×8 block, where '+1' represents an increment in the address of respective memory block); Figure 5.16.c, and so on.

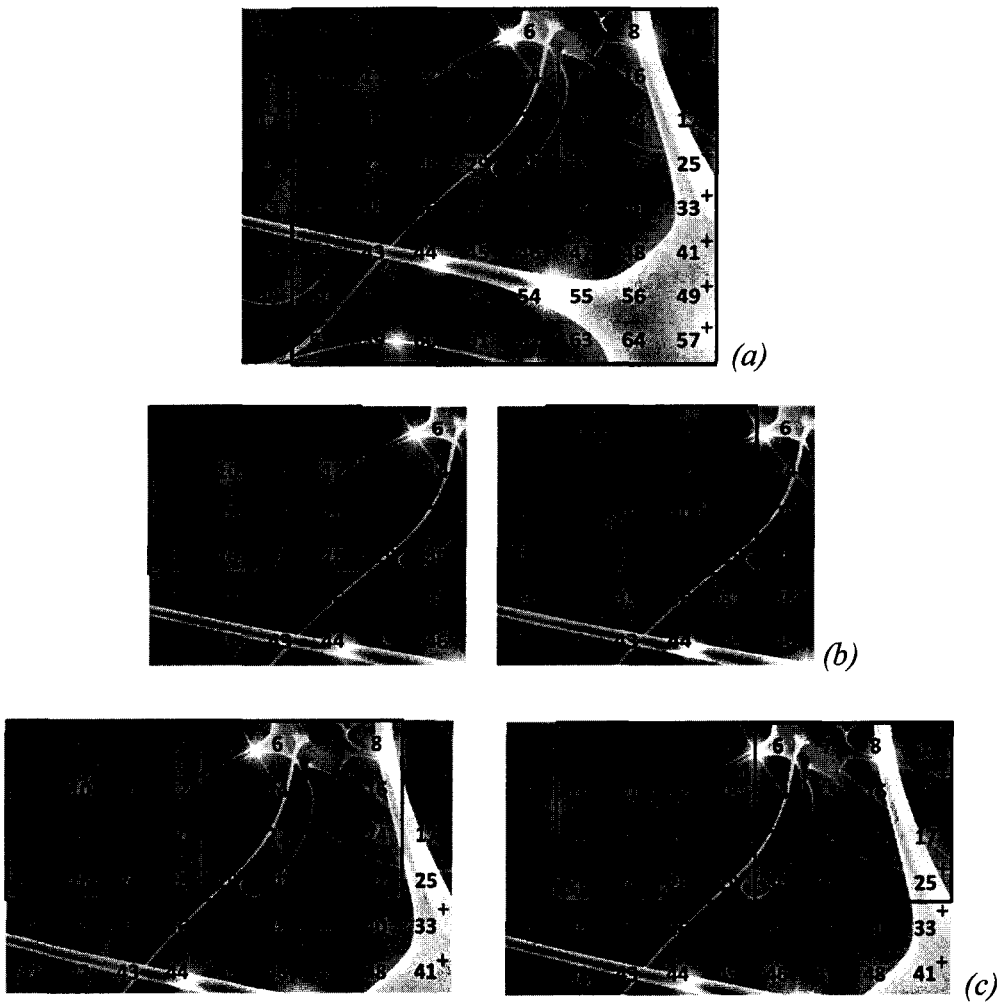

Figure 5.16 – Pixels reading example from respective memory blocks according to block size and number of active computational units.

It is important to know that in our system data sent from the current frame memory is the same while that from the reference frame memory is being changed until the Search Window finishes the Computational Window. For this reason we stated that the addresses change is more frequent here than in the current frame memory. For the same reason, this memory does not need a Data Replication Stage since only sufficient data are sent to the corresponding number of active regions.

It is noticeable that the proposed memory system will have no memory conflicts in addressing specific memory blocks for any data requested. For any block size functioning at any number of active regions, the memory blocks are read in parallel in only one cycle to support that computation. This justifies the amount of memory blocks used in order to support the full parallelism of our computational units.

v. Memory capacity

The RAMB36 memory hardware unit is a dual port memory block with a capacity of 36 Kbits. These blocks can take two addresses and generate two 32 bits wide outputs. However, 4 Kbits of this capacity is for parity bits and the remaining 32 Kbits are data. We kept the design based on 32 Kbits in order not to add more logic related to making use of the parity bits as data bits.

In the proposed architecture each RAMB36 has 32 bits per location. In the current frame memory, a 4×4 pixels block is stored in one memory block of four RAMB36 each and having 2×2 pixels in a location. 16 4×4 pixels are stored in 16 memory blocks, hence require 64 RAMB36. In the reference frame memory, a 4×4 pixels block is stored in four memory blocks of one RAMB36 each and having 2×2 pixels in an address. 16 4×4 pixels are stored in 64 memory blocks, hence 64 RAMB36. In both controllers, memory blocks are 64 in each regardless how they are aligned (4 or 1 per memory block). For real time applications, such as live transmission or video conferences, to function normally, one Current Frame and at least one Reference Frame should be stored in memory. Each 16×16 pixels block from a Current Frame can be stored in one location in the respective controller and the same applies for 16×16 pixels block from a Reference Frame in its respective controller. By taking each video type resolution and dividing it by 16×16 we get how many locations we need to store the pixels from both frames. Our system's memory layer is made of 2×64 RAMB36, 128 RAMB36, with a capacity of 2×1024 16×16 pixels block. In Figure 5.17 we show the number of locations required for two frames of each video and the necessary number of memory layer(s). Video types are of resolutions QCIF 176×144, CIF 352×288, DCIF 528×384, VGA 640×480, SVGA 800×600, EVGA 1024×768 and HD 1920×1088.

The left vertical axis in Figure 5.17 shows the number of memory locations both needed to store two frames of various video types and locations available for every memory layer(s). The right vertical axis shows the number memory layers; which when multiplied by 128 gives the total number of RAMB36 blocks used. We implemented our system mainly for mobile devices that uses video resolutions such as that of QCIF, CIF and DCIF. However, by simply adding extra layers to our controllers, the system can support higher pixels resolutions such as VGA and SVGA used in tablets devices (+1 layer) or even high definition HD TVs

(+7 layers). This is done without the need to change the algorithm in the controller architecture but only the change of memory hardware components.

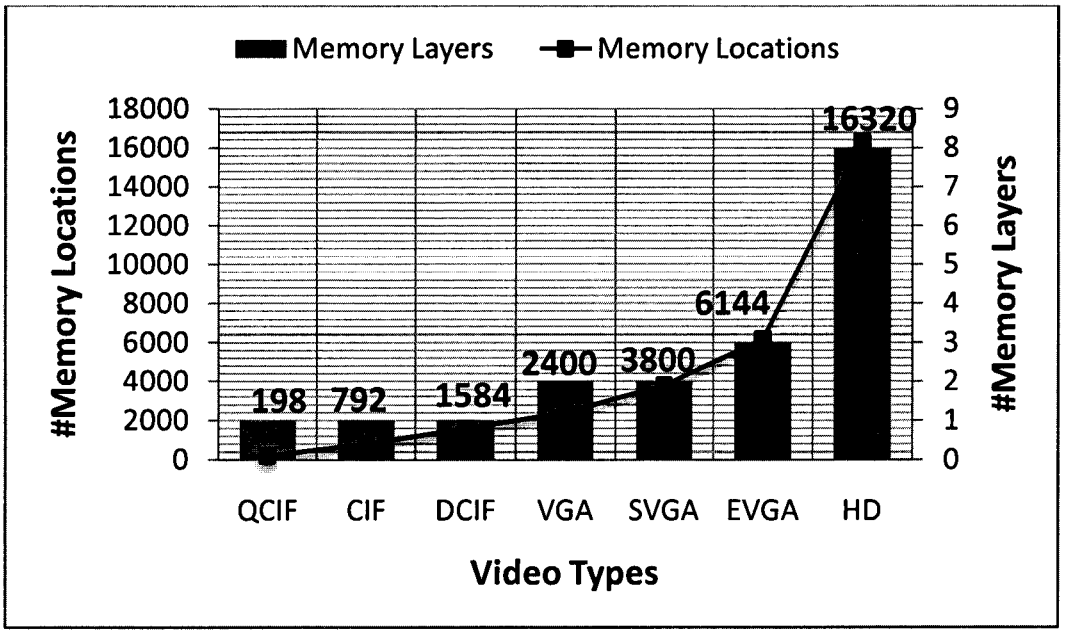

Figure 5.17 – Required memory locations for various video types' resolutions and necessary memory layers.

5.4.f. Power and energy analysis

In this section we analyze the power and energy consumed by the ME computational unit. We present measurements that demonstrate how energy can be reduced when bigger block sizes are used. This provides the flexibility to leverage reconfigurability to trade-off accuracy for energy savings based on application demands.

One objective of our work is to reduce energy consumption. And there are two ways to achieve this: first, to control the computational load by selecting an appropriate block size, and second, to blank reconfigurable regions that are not being used. This latter approach was proposed in [S.LIU 2010_2], where it was shown that partial reconfiguration can lead to more energy reduction than clock gating as long as the reconfiguration component throughput is high enough to overcome the extra energy required to perform reconfiguration. In our case we are using the ICAP to perform reconfiguration, and the Direct Memory Access (DMA) technique proposed in [S.LIU 2010_1] to accelerate the reconfiguration process. Using the high speed DMA engine, our system requires 327 μsec to configure a RR with a SAD4×4 hardware block, and 280 μsec to blank a RR. We present our results to validate the conclusions presented in [S.LIU 2010_2].

Next we analyzed our system's energy consumption for different computational modes. Since energy is the product to power and time, we measured both quantities. We used a Xilinx ML605 FPGA board to measure the power consumed in the static and reconfigurable regions. Figure 5.17 shows the different power measurements corresponding to 1, 2, 4, 8 or 16 active regions.

This plot shows that power dissipation is not linear. Due to the architecture of the FPGA fabric, when a module is configured in one region, the corresponding column in the FPGA is enabled and clock regions and access ports to this region are powered on as well. Since multiple regions can be configured within the same FPGA column, additional power is consumed by the additional logic resources, but no additional power is consumed by clock resources. This explains the non linearity in Figure 5.18, which also affects energy consumption in the reconfigurable ME computational unit.

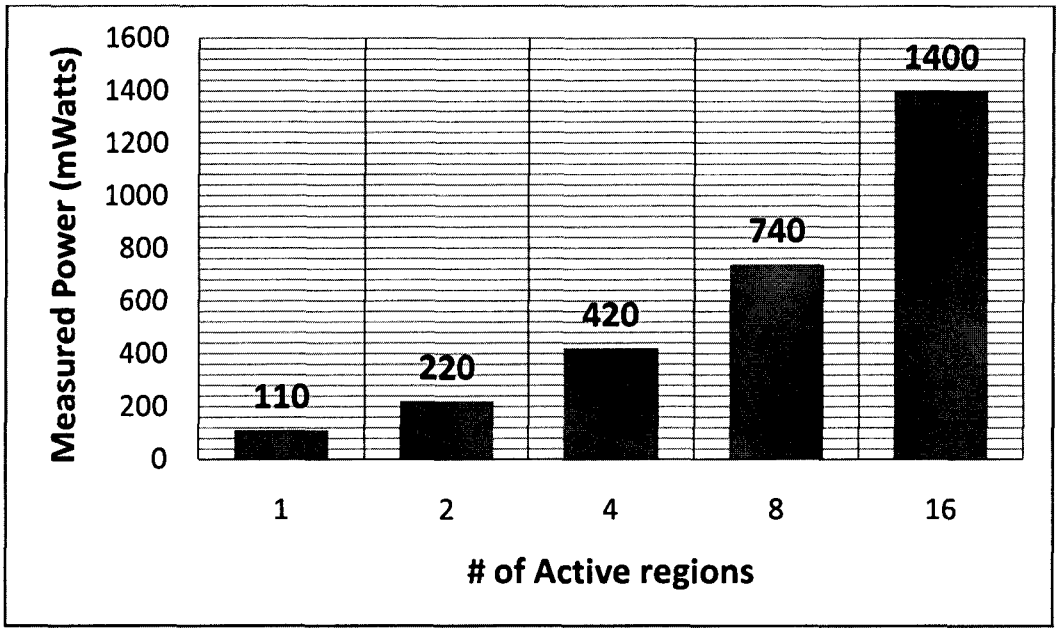

Figure 5.18 – Power measurements acquired for different configuration implementations using variable number of active regions.

Note that the reconfiguration process is achieved via implementing static components responsible for performing the reconfiguration control. These components consume power during all time since they are static components (720 mWatts). According to [S.LIU 2010_2], it is important to mention that the static power is overtaken and eliminated from the energy model by using high reconfiguration speed as the one we used and proposed by [S.LIU 2010_1] (T_{ICAP}=399 Mbps). Verifying that, we recall equation (2.1) and by replacing 720 mWatts in P_{static}, 127 Kbits×16 regions in S_{bf},, 1400 mWatts in P_{pr} (power for a maximum of 16 regions) and 60 msec as a maximum inactive time (1 frame) we get (399 Mbps>66 Mbps).

After measuring the power in our reconfigurable ME computational unit, according to active regions, we measured and studied the time used to perform various SAD computations. Recall that our system can complete a SAD computation every clock cycle.

Using a system clock frequency of 100 MHz and using the power measurements from different active regions, we obtained the results shown in Figure 5.19. The y-axis shows the energy in nJ and the x-axis shows different block sizes implemented in 16, 8, 4, 2 or 1 RR(s). The energy shown in this figure corresponds to that consumed by a Computation Window to finish scanning a Search Window using a certain block size implemented in the specified number of active RR(s).

In Figure 5.19, it is important to note that when the system is running on a number of active regions that do not support the full parallelism of a certain block size, repetitive usage of these regions is performed.

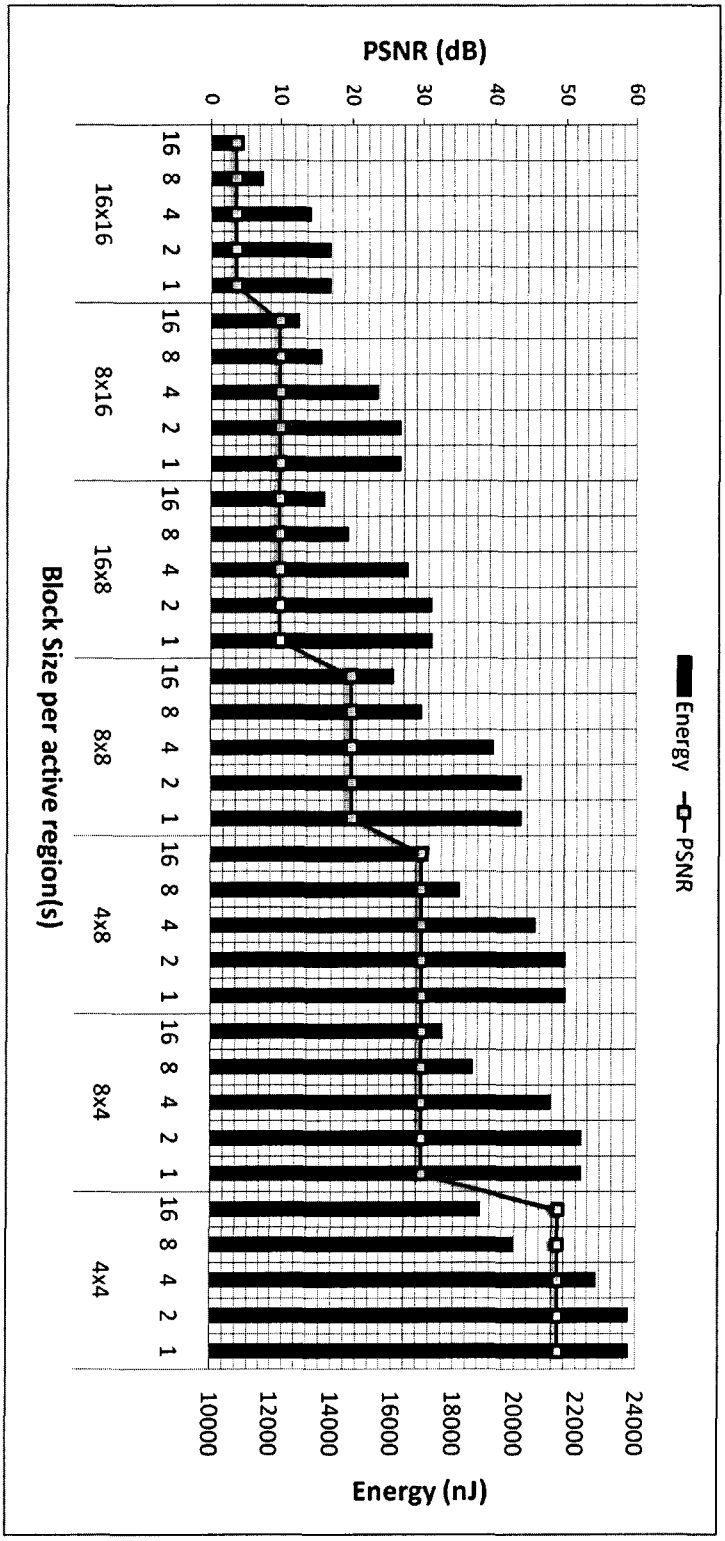

Figure 5.19 – Energy and PSNR plot based on different block sizes and various number of active regions. This plot is sorted in an ascending order of the accuracy PSNR.

For example computing the SAD16×16 using one RR is performed by running the same region 16 times with different data each time. Figure 5.19 shows that using larger block sizes can reduce energy consumption by up to 51% when comparing the average energy spent using a block size of 16×16 to a block size of 4×4. Figure 5.19 also shows the PSNR achieved using different block sizes and presents how energy consumption can be traded for accuracy.

5.4.g. Image quality and compression ratios

The PSNR has a direct impact on the quality of the image as can be shown in Figure 5.20. Video images with higher PSNR (50 dB) are clearer and smoother for the vision. However, as the PSNR, and hence image quality, decreases, the video images become more *noisy* (<10 dB).

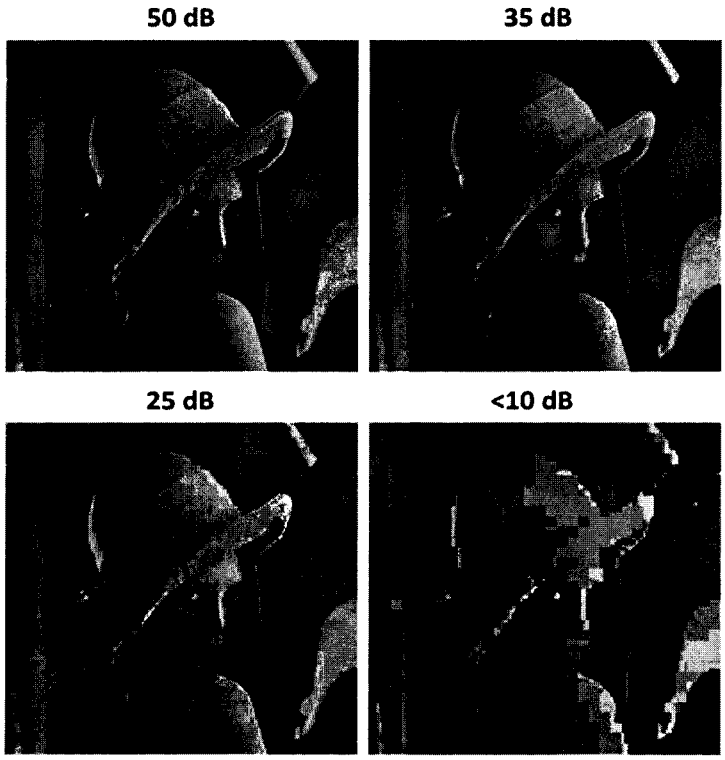

Figure 5.20 – The effect of PSNR on image quality. The higher the PSNR, the clearer the video image has become.

As presented in section 5.4.f, the energy consumption can be traded with the image quality or the PSNR. However, less image quality in videos means that the videos have been encoded using bigger block sizes. The use of bigger block sizes also means that fewer bits were used to encode the video. In conclusion, bigger block sizes tend to compress, encode, videos more than smaller block sizes while trading that with image quality. The compression ratio for a QCIF video image size using 16×16 block sizes is around 50%. However, using lower block sizes such as 4×4 decreases this ratio to around 4%.

In the next section, we present a heuristic for controlling the reconfiguration of the ME computational block based on this trade-off.

5.4.h. Reconfiguration heuristics

To exploit the trade-off between system energy consumption and image accuracy, we developed a reconfiguration heuristic that manages different configurations according to system demands. To better explain the heuristic, we plot in Figure 5.19 the energy consumption per Search Window for different block sizes and active regions versus the PSNR in dB for a sample QCIF, 256 Kbps, and video sequence using different block sizes.

Figure 5.19 shows that the PSNR is constant for different active regions due to its being derived from the image quality, which only depends on the block size used. We also observe that 8×16 and 16×8 blocks, and 4×8 and 8×4 blocks, achieve similar PSNR levels. This is due to the fact that these blocks have the same size but different orientations, which results in images having similar quality.

Our heuristic is controlled as shown in the flow chart in Figure 5.21. The system starts with a specified accuracy level, and selects an appropriate block size corresponding to the requested accuracy level. For example a mobile user can start a real time video conference on his QCIF dimension mobile screen. A configuration point is selected based on the energy level in the mobile device's battery. The system continues to monitor energy and accuracy levels to determine if reconfiguration is needed. When a reconfiguration is needed, an appropriate configuration and memory interface are invoked based on an appropriate block size. The system continues to monitor energy and accuracy levels until a change becomes necessary upon which a new reconfiguration cycle is invoked.

To evaluate our system, we compare its energy consumption and accuracy under the proposed reconfiguration heuristic to two static cases.

1. A fixed block size of 4×4 running on 16 computational units.
2. A fixed block size of 16×16 running on 16 computational units.

These static cases correspond to two extremes: One, the highest accuracy and energy consumption and Two, the lowest accuracy and energy consumption, respectively (refer to point 4×4 on 16 regions and 16×16 on 16 regions in Figure 5.19). The basic test involves decoding a QCIF video on a mobile device with a battery capacity of 1350 mAh. All three systems were run until the battery is completely discharged. Figure 5.22 shows the time used by the three systems to drain the battery. Also, in Figure 5.23 we show the accuracy degradation while battery draining of the three systems.

Figure 5.22 and Figure 5.23 show that the static system with a block size of 4×4 achieves the highest accuracy (49 dB) with the battery lasting 2500 minutes. The static system with a block size of 16×16 achieves the lowest accuracy (6 dB) with the battery lasting 4500 minutes. On the other hand, our dynamically reconfigurable system achieves an accuracy of 25 dB with the battery lasting 3000 minutes. This shows that our heuristics succeed in achieving 20% more battery life that the highest accuracy system while also achieving 5.2 times the accuracy of the system with the longest battery life.

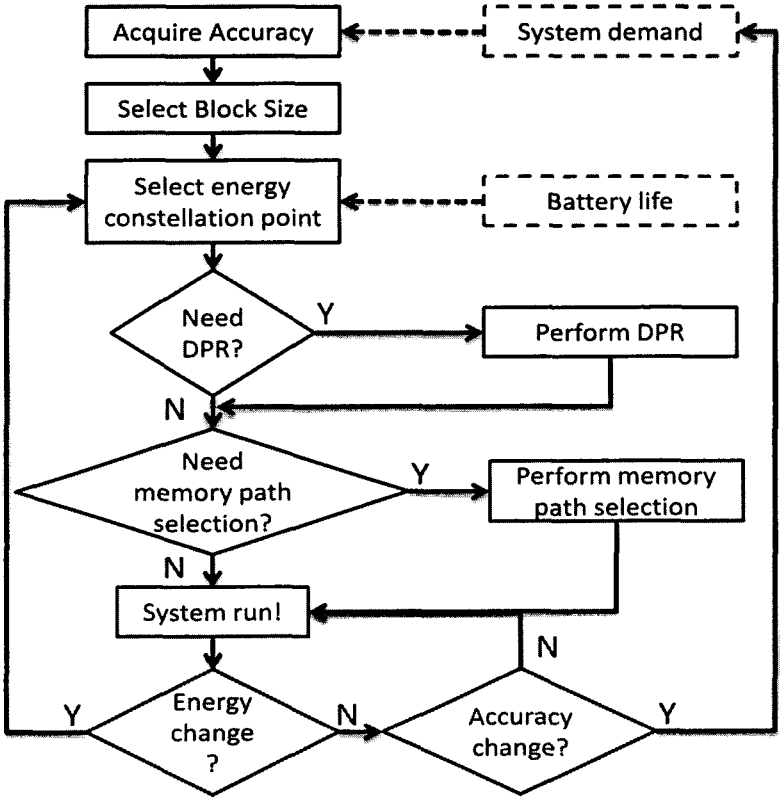

Figure 5.21 – Our proposed reconfiguration heuristics flow diagram.

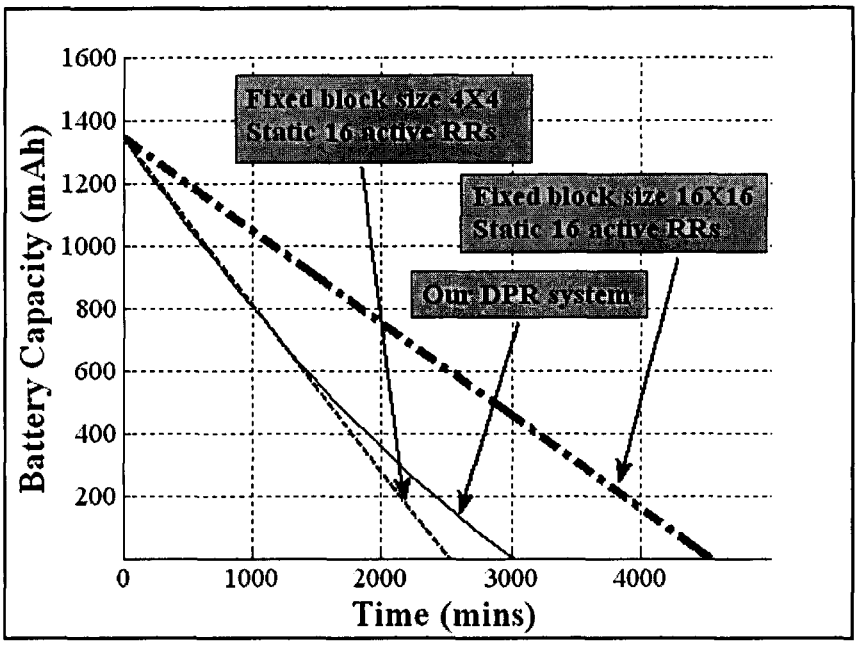

Figure 5.22 – Battery draining in two static systems and our DPR system over time.

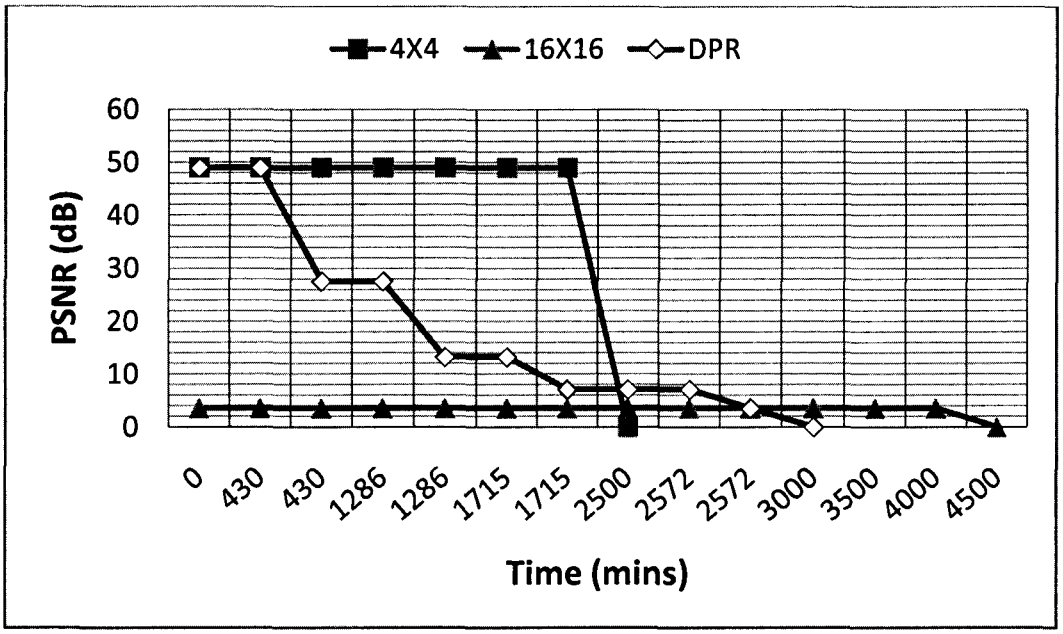

Figure 5.23 – Accuracy degradation in two static systems and our DPR system over time.

5.5. Conclusions

In this chapter, we presented the work we did to implement a dynamically reconfigurable H.264 ME computational unit. We started with an introduction of the whole multimedia system, the H.264. Following the introduction, we highlighted the ME unit in the H.264 encoder system and its relevant importance to the whole H.264 multimedia system. The ME unit was analyzed afterwards in terms of computational demands and execution times compared to the total H.264 encoder execution times. We found out that this block consumed around 40% of execution times out of the total encoder execution time. This percentage was caused by one major computational block in the ME unit, the SAD block. For this reason, the SAD block was analyzed and studied for different pixels block sizes and for different video resolutions. The change in the block size was an important factor in the reduction of the energy consumption of the system. However, we showed that the energy consumption can be further more reduced when implemented on a DPR basis.

We proposed a tree-like architecture of 16 partially reconfigurable computational blocks performing the SAD computations. High speed memory controllers were connected to the computational units in order to provide high throughput and speed to the computational blocks. Execution times were acquired and in addition to the power measurements of different configurations of our system, we were able to compute the energy consumption for any configuration. Our proposed system's architecture can be modified to meet specific energy and image quality constraints. By implementing 16 reconfigurable regions, we were able to support five configurations for each of the seven block sizes each with different levels of ac-

curacy and energy consumption. Image accuracy levels were controlled via application demands, user demands or support demands. Using a reconfiguration heuristic, our system can support up to 35 different configurations. With a maximum saving of up to 51% in energy consumption, the system can support all block sizes in an H.264 encoder. The data delivery was optimized to support a fully parallel architecture using a memory architecture that has negligible effect on quality.

6

Conclusions and perspectives

By definition, the term "Technology" stands for the making usage and knowledge of tools, techniques, crafts, systems or methods of organization in order to solve a problem or serve some purposes. On the other hand, the term PhD is used in a broader sense in accordance with its original Greek meaning, which is "love of wisdom". Combining both, results in what has been disserted in this book that is a combination of PhD and technology. This thesis is aimed in using wisdom and research in making usage of knowledge of software tools for DPR techniques in FPGA crafts and systems for the purpose of enhancing a DAS and H.264 systems. The conclusions of my three years work are summarized in this chapter, followed by some proposals for possible extensions of this work.

6.1. Conclusions

In this thesis, an exploration of the DPR feature in FPGAs is explored for the benefit of a DAS system as well as a multimedia system. We presented these two system applications mapped on FPGA fabrics. Using DPR, we explore the benefits that DPR provides for each of the presented application. For each system implementation, we present steps to approach the final implementation and validate the approach by showing obtained results. The benefits achieved with the use of DPR for each of the implemented applications are concluded from the obtained results. A detailed summary of the motivation to this work, implementation steps, applications explored, results obtained and benefits has been fully addressed and documented in the following manner:

In order to present the details of our work, an introduction to the field and motivation had to be presented first. At the beginning of this thesis, we introduced fabric technologies and highlighted the advancements done in the field of electronic computational devices and fabrics. Starting with VLSI designs and ending with the recent technology of FPGAs. These

devices proved their promising future and capabilities in the research and industrial domains by being useful for many applications and introducing new features such as the DPR feature. This feature, in which in particular, has been proposed recently and many research started targeting it since the last decade. Hence, being in this era of interest in the DPR feature in FPGAs, we were motivated to investigate this and show how it can be beneficial in our own thinking and research methodology.

In order to better investigate the DPR feature, certain applications were targeted in order to verify the approach we propose and sell our ideas to the research community. Being in the *North of France*, a region famous in automotive industry and research, motivated the investigation of automotive related applications. We therefore chose two applications that, with the proper advising and guiding, were interesting both in terms of fabric investigation (DPR and FPGAs) and field investigation (automotive). We therefore started the investigation of DPR on FPGAs in DAS safety driving systems and multimedia video processing systems that are automotive related as well.

After a motivational introduction for both the targeted fabric and its feature and the targeted applications, thoughtful studies and analyses of recent researches and works were inevitable.

- The first literature review was related to the FPGA fabric and its DPR feature. A selection of the most interesting work was done in this review, showing what has been proposed as architectural improvements in the recent few years since the introduction of the DPR. Since our aim was not introducing a new architectural improvement, a review of the literature in that field made benefits for our goal and our systems designs. One of which we picked to benefit from was the making use of high partial reconfiguration speeds IP cores in order to sustain energy [S.LIU 2010_1] [S.LIU 2010_2].
- The second literature review was related to the DAS safety systems and the work done targeting its implementations on FPGAs and using DPR. This review of the latest research was important in order for us to target some work that was either not targeted or to enhance another work. Seeing that most of the DAS systems were based on camera sensors motivated us to choose a different sensor that is better in longer ranges and weather conditions such as a radar sensor. And because from our literature review, no work has targeted a DPR-radar based DAS system, made us pioneers in investigating this system and added more to our motivation.
- A final review was made related to our second targeted application, the multimedia application. The choice landed on choosing the H.264 standard due to its importance and wide applicability in the multimedia video processing field. Like in the DAS systems review, we reviewed the work done related to the H.264 and its implementation on FPGAs and using the DPR feature. We found out that most of the standard's decoder blocks were investigated thorough fully on the basis of DPR and hence, we targeted the encoder block instead. Literature cited many blocks in the encoder as well but one, the ME unit. It was statically implemented and ameliorated on FPGAs and only added as an enhancement block on the basis

of DPR. This motivated us, by using the work of [S.LIU 2010_1] and [S.LIU 2010_2] to present a considerably important application, the ME unit in the H.264, that can make use of energy savings especially due to the fact that the H.264 is a mobile application as well.

After motivating the work and setting the main milestones to tackle, we went in describing our approaches, systems propositions and implementations. Starting with building up a DPR-radar based DAS system, we first defined DAS systems and their functionalities. We based our system proposal on a static MPSoC radar based DAS system [J.KHAN 2009] and hence, we described the base system. With properly analyzing the base system for a good DPR candidate block, we found that the filtering block of this DAS system was the most suitable. Being based on multiple soft processors motivated its migration into a hardware block instead of based on soft processors. The filtering block was hence analyzed and optimized in way to meet timing constraints and accuracy levels demanded by driving safety standards before implementing it as a hardware accelerator block. This accelerator showed to be 8000 times faster and 80% less hardware resource consumption compared to the soft processor. In addition to that, the migration of the filtering block from soft processors to a single hardware block made the system a hybrid architectural system and introduced the possibility to have a DPR based system.

We then proposed our first DPR-radar based DAS system that explores the use of DPR to modify the physical structure of the filtering block in the DAS system. We demonstrated in this approach how the accuracy of the filtering block can be dynamically and automatically tuned to match the characteristics of the driving environment. This was very useful when the environment changes like from rush hours on highways or narrow city streets with more obstacles to track to calm open roads and less obstacles to track. A high accurate filter was used in order to track more targets in dense environments and hence, ease some stress off the driver and another less accurate, less resource utilized filter, used in relaxing environments. We demonstrate, based on real data sets and experimental studies that the system can function normally with low overhead and can be automatically controlled by a simple heuristic.

In a second contribution based on the same system, we explored yet another possibility to benefit from DPR. We investigated, unlike the previous target density driven reconfigurable system, the possibility to automatically tune the accuracy of our filter to match the dynamics of moving obstacles on the road. Since lower levels of accuracy generally require fewer hardware resources, DPR can be leveraged to release hardware resources for other uses, such as tracking more obstacles, accelerating other computational functions, or reducing power consumption. Our design was based on using three modular filters that can be dynamically combined in three configurations to match different driving scenarios. Our design also included an enhanced detection unit module for post-processing acquired radar signals and pre-filtering them before they are delivered to our system. This enhancement block was implemented whenever a free reconfigurable region was available. Our system was also designed to continue operating even when being reconfigured, and this enhances the system's

reliability. Our experimental results demonstrated the feasibility and low overhead of our dynamically reconfigurable design.

On the other hand, and by using DPR, we contributed to the exploration of DPR for an H.264 multimedia system. We targeted the Motion Estimation (ME) unit part of the H.264 encoder, a unit that was not very investigated on the basis of DPR in literature. We targeted using the DPR, the implementation of a dynamically partial reconfigurable ME inside an H.264 encoder. We presented the computational complexity inside this unit and the implementation possibilities on an FPGA. Following that, we presented the proposed architecture for this computational unit and showed its hardware aspects and functionality. Our system was capable of computing the ME of different supported block sizes by simply changing the data sent to interconnected computational units. Hence, this made our system functioning similar to a fully static implementation but with a better hardware resource usage due to the use of DPR. In order to support the high hardware computation speed, a memory buffering architecture was also designed and connected to the DPR computational unit in order to provide fast data throughput. By using the work of [S.LIU 2010_1] and [S.LIU 2010_2], we were able to use DPR to adapt the energy levels according to 35 different configurations. Hence, the flexibility of our system in terms of image quality and energy was the basis of our reconfiguration heuristics. We implemented our system on fabric and deducted results in terms of execution times, power measurements, and reconfiguration times and acquired image quality. Our system was tested to support mobile video services.

6.2. Perspectives

While the work accomplished the envisaged objectives set at the start, several extensions are possible for future continuation. The future work and perspectives can be divided into three main targets: First perspective deals with architectural improvements and investigations of enhancing the DPR process. The second perspective is related to DPR in DAS systems and multiple enhancements levels investigations. The third and final perspective is related to enhancing and better exploring the DPR feature in the H.264 multimedia encoder system.

The perspectives related to the DPR architectural improvements and enhancement investigations can be summarized in the following points:

- The DPR process mostly involves the reading of bit files from an external compact flash memory. Enhancements on the DPR process were done by buffering these bit files in other faster accessible memories such as BRAMs or SRAMs. One room for enhancement is to investigate the implementation of an IP core that, using specific user's inputs, that can automatically read and write those bit files from the compact flash memory to other selected memories such as BRAMs, SRAMs, DDR, or DDR2 memories. While showing the trade-offs between capacity and speed of the DPR process for each available memory,

the system's intelligence can take these trade-offs and can automatically select the most suitable DPR memory bit files storing architecture it wants according to application demands.

- Recent Reconfigurable Regions (RRs) are rectangular in shape. In some cases, a design mapped to a reconfigurable region consumes extra resources due to the shape restrictions. Sometimes a region might contain unnecessarily resources such as BRAMs or embedded multipliers that are not utilized. However, in order to support the required number of logic cells for a module in a RR, these unnecessarily resources have been added due to fabrications restrictions and design. One room for improvements is to investigate the possibility to have an n-edges polygon based RR region design. This way, with automatic or manual tuning of the RR, resources that are not needed can be excluded from a reconfigurable region. Hence, better inner RR region hardware resource utilizations can be achieved. By acquiring bit files formats and enhancing the placement constraints of a RR, the n-edges polygon is now possible to be defined.

As for DPR enhancements and investigations in DAS systems, the following perspectives could be interesting for future investigations:

- We used in our approach only one radar sensor and applied DPR to the decision making system using this sensor. One room for investigation is to apply the same DAS system to support multiple radars (one in the front, one in the rear and two side radars for example). Since we proved the reduction of resources and the gain of high processing speed with the migration to hardware accelerators, we can make benefit of supporting multiple sensors. In this case, instead of adapting the RR to modify only the accuracy of a filter based on one radar sensor, the regions can function, using DPR, as separate sensors to other car mounted radars.
- In our DAS DPR based system implementation, an enhancement block was implemented when RRs are free and not needed. One room for enhancements is to investigate more units from not only the capturing phase but also from the restitution phase. Such units can be sound amplification for sound based restitution DAS systems, or image rendering enhancements for image base restitution DAS systems.
- Although our work was mainly focused on the treatment phase in a DAS system, other phases can be investigated for their implementations on a DPR basis separately as well. Once this is achieved, a whole system can be built with joined RRs that can support computational units from different DAS phases. A more detailed investigation can include hardware reuse whenever one phase is in no need for its hardware unit. Hence, regions reserved for this unit can be built as RRs and can be reconfigured to become hardware accelerators used in other DAS phases.

For the final targeted perspective related to enhance and better explore the DPR feature in the H.264 multimedia encoder system, the following can be investigated:

- Our base system includes a buffering memory system that can support fast throughput to the reconfigurable ME computational units. One room for enhancement is investigating the implementation of an IP core that is capable of reading video data from an external memory and be able to store this data in its proposed alignment in the buffering memory architecture.
- We showed using a simple heuristic how energy can be reduced by blanking some computational regions using DPR. A detailed study on the video types viewed by mobile users adds a realistic criterion to the proposed reconfiguration heuristics. Using such criteria can improve the validity of our proposed architecture based on real life demands. This can be achieved by taking into consideration a proper study of the mobile video scenarios and applying it to our system. Such study is like that proposed in [J.HAMERS 2008]. By testing it on our system, this adds more validity and reliability on our system.
- Since the H.264 in addition to the DAS systems have been both investigated on the basis of DPR. And since both applications, DAS and video processing, are part of the automotive industry domain. Implementing one common system, using both radar sensors and camera video recording sensor can be one interesting project to target. The investigations we did for both applications and the three DPR system implementations represent the first hardest step in achieving this goal. What is next is to design a system joining both, sharing a RR of hardware accelerators used by a DAS and a video H.264 multimedia system.

References

[A.AMARA 2006] –A Amara, F Amiel, and T Ea, "FPGA vs. ASIC for low power applications," *Microelectronics Journal* 37, no. 8 (August 2006): 669-677.

[A.OTERO 2010] – Andres Otero *et al.*, "Run-Time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs," in *2010 International Conference on Field Programmable Logic and Applications* (presented at the 2010 International Conference on Field Programmable Logic and Applications (FPL), Milan, Italy, 2010), 70-76, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5694223.

[A.VAHIDI 2003] – A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems* 4, no. 3 (September 2003): 143-153.

[ACTEL] – Actel Inc., http://www.actel.com/.

[ALTERA] – Altera Inc., http://www.altera.com/.

[C.BOBDA 2007] – Christophe Bobda, "Introduction to reconfigurable computing: architectures, algorithms, and applications" (Dordrecht: Springer-Verlag, 2007).

[C.CLAUS 2007] – Christopher Claus, Johannes Zeppenfeld, Florian Müller and Walter Stechele, "Using Partial-Run-Time Reconfigurable Hardware to Accelerate Video Processing in Driver Assistance System," in *2007 Design, Automation & Test in Europe Conference & Exhibition: Nice, France, 16-20 April 2007.* (Piscataway NJ: IEEE, 2007).

[C.CLAUS 2010] – Claus *et al.*, "Towards Rapid Dynamic Partial Reconfiguration in Video-Based Driver Assistance Systems," in *Reconfigurable Computing: Architectures, Tools and Applications*, ed. Phaophak Sirisuk *et al.*, vol. 5992 (Berlin, Heidelberg: Springer Berlin Heidelberg, 2010), 55-67, http://www.springerlink.com/index/10.1007/978-3-642-12133-3_8.

[C.TING 2003] – Chi-Wang Ting, Lai-Man Po, and Chun-Ho Cheung, "Center-biased frame selection algorithms for fast multi-frame motion estimation in H.264," in *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003* (presented at the 2003 International Conference on Neural Networks and Signal Processing, Nanjing, China), 1258-1261.

[C.WEI 2007] – Cao Wei *et al.*, "A Novel reconfigurable VLSI architecture for motion estimation," in *2007 7th International Conference on ASIC* (presented at the 2007 7th International Conference on ASIC, Guilin, China, 2007), 774-777, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4415745.

[D.CHUN 2008] – Dongyeob Chun *et al.*, "Design of high-performance unified motion estimation IP for H.264/MPEG-4 video CODEC," in *2008 International SoC Design Conference* (presented at the 2008 International SoC Design Conference (ISOCC), Busan, Korea (South), 2008), I-156-I-159.

[E.VITTOZ 1994] – Eric A. Vittoz, "Analog VLSI signal processing: Why, where, and how?," *Journal of VLSI Signal Processing* 8, no. 1 (February 1994): 27-44.

[EARLY_ACCESS] – Xilinx Inc., "Early Access Partial Reconfiguration User Guide," *Xilinx user guide UG208 (v1.1)* March 6, 2006. Available: http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf

[ENIAC] – Electronic Numerical Integrator And Computer (ENIAC), http://en.wikipedia.org/wiki/ENIAC.

[EYEQ2] – The MobilEye safety project. Available: http://www.mobileye.com/node/69

[F.KUCUKAY 2004] – F. Kucukay and J. Bergholz. "Driver Assistant Systems". *Proc. Int. Conf. on Automotive Technologies*, 2004.

[H.GOLDSTINE 1993] – Herman Goldstine and American Council of Learned Societies., *The computer from Pascal to von Neumann* (Princeton, N.J.:: Princeton University Press,, 1993).

[H.SAHLBACH 2010] – Henning Sahlbach *et al.*, "A Scalable, High-Performance Motion Estimation Application for a Weakly-Programmable FPGA Architecture," in *2010 International Conference on Field Programmable Logic and Applications* (presented at the 2010 International Conference on Field Programmable Logic and Applications (FPL), Milan, Italy, 2010), 15-18, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5694213.

[HWICAP] – OPB HWICAP Data Sheet. DS 280. http://www.xilinx.com.

[I.M.JUAHIR 2006] – Izyan Musfirah Juahir and Nasreen Badruddin, "Block Mode Decision Based on Motion Vectors for H.264/AVC," in *TENCON 2006 - 2006 IEEE Region 10 Conference* (presented at the TENCON 2006 - 2006 IEEE Region 10 Conference, Hong Kong, China, 2006), 1-4.

[IMAPCAR] – Nec Electronics 2006,"Nec introduces Imapcar image processor with advanced parallel processing capabilities". Available: http://www.nec.co.jp/press/en/0608/2501.html

[ISE12] – Xilinx ISE In-Depth Tutorial UG695 (v 12.3). September 21, 2010.

[J.CARVER 2008] – Jeff Carver, Richard Neil Pittman, and Alessandro Forin, "Relocation and Automatic Floor-planning of FPGA Partial Configuration Bit-Streams," no. MSR-TR-2008-111, August 2008.

[J.HAMERS 2008] – Juan Hamers and Lieven Eeckhout, "Automated hardware-independent scenario identification," in *Proceedings of the 45th annual conference on Design automation - DAC '08* (presented at the the 45th annual conference, Anaheim, California, 2008), 954, http://portal.acm.org/citation.cfm?doid=1391469.1391710.

[J.HUANG 2009] – Jian Huang *et al.*, "Scalable FPGA-based architecture for DCT computation using dynamic partial reconfiguration," *ACM Transactions on Embedded Computing Systems* 9, no. 1 (October 2009): 1-18.

[J.KHAN 2008] – Jehangir Khan *et al.*, "An MPSoC architecture for the Multiple Target Tracking application in driver assistant system," in *2008 International Conference on Application-Specific Systems, Architectures and Processors* (presented at the 2008 International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Leuven, Belgium, 2008), 126-131, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4580166.

[J.KHAN 2009] – Jehangir Khan, "Embedded Multiprocessor Architectures for Automotive Driver Assistance Systems," 2009. *Thesis and dissertations*. Available (hard copy): http://www.univ-valenciennes.fr/SCD/bibliotheque-universitaire-du-mont-houy.

[J.SAAD 2009] – Jean Saad, Amer Baghdadi, and Frantz Bodereau, "FPGA-based Radar Signal Processing for Automotive Driver Assistance System," in *2009 IEEE/IFIP International Symposium on Rapid System Prototyping* (presented at the 2009 IEEE/IFIP International Symposium on Rapid System Prototyping (RSP), Paris, France, 2009), 196-199, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5158519.

[J.W.CHEN 2006] – Jian-Wen Chen, Chao-Yang Kao, and Youn-Long Lin, "Introduction to h.264 advanced video coding," in *Asia and South Pacific Conference on Design Automation, 2006.* (presented at the Asia and South Pacific Conference on Design Automation, 2006., Yokohama, Japan, n.d.), 736-741, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1594774.

[JVSG] – JVSG.com. H.264 joint video surveillance group compression research data: 2008. May 2010.

[M.ELHAJJ 2009] – Majdi Elhajj *et al.*, "A Low Power ASIC Design of a FSBM Motion Estimator for H.264/AVC," *ICGST-PDCS Journal* 9, no. 1 (October 2009): 53-58.

[M.GUARISCO 2010] – Michael Guarisco, Hassan Rabah, and Abbes Amira, "Dynamically reconfigurable architecture for real time adaptation of H264/AVC-SVC video streams," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops* (presented at the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops), San Francisco, CA, USA, 2010), 39-44, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5543764.

[M.HUBNER 2006] – M. Hübner and J. Becker, "Exploiting dynamic and partial reconfiguration for FPGAs," in *Proceedings of the 19th annual symposium on Integrated circuits and*

systems design - SBCCI '06 (presented at the the 19th annual symposium, Ouro Preto, MG, Brazil, 2006), 1, http://portal.acm.org/citation.cfm?doid=1150343.1150350.

[M.HUBNER 2010] – Michael Hübner *et al.*, "Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)* (presented at the Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 2010), 1-8, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5470736.

[M.LIU 2009] – Ming Liu *et al.*, "Run-time Partial Reconfiguration speed investigation and architectural design space exploration," in *2009 International Conference on Field Programmable Logic and Applications* (presented at the 2009 International Conference on Field Programmable Logic and Applications (FPL), Prague, Czech Republic, 2009), 498-502, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5272463.

[M.RICHARDS 2005] –M Richards, *Fundamentals of radar signal processing* (New York: McGraw-Hill, 2005).

[M.SHAFIQUE 2008] – Muhammad Shafique, Lars Bauer, and Jörg Henkel, "Optimizing the H.264/AVC Video Encoder Application Structure for Reconfigurable and Application-Specific Platforms," *Journal of Signal Processing Systems* 60, no. 2 (November 2008): 183-210.

[M.ULLMANN 2004] – M. Ullmann *et al.* "On-Demand FPGA Run-Time System for Dynamic Reconfiguration with Adaptive Priorities". *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 454-463, Springer-Verlag LNCS Vol. 0302, 2004.

[MICROBLAZE] – Xilinx MicroBlaze Processor Reference Guide UG081 (v9.0). January 17, 2008.

[MOORE] – Moore's law Intel Software Network, http://www.developers.net/intelisnshowcase/view/127.

[MPEG4] – hhi.fraunhofer.de. H.264/mpeg4-avc. Jan. 2011.

[MSR] – Microsoft Inc., Embedded and Reconfigurable Systems Research Group. Available: http://research.microsoft.com/en-us/groups/embeddedsystems

[N.WAX 1955] –Nelson Wax, "Signal-to-Noise Improvement and the Statistics of Track Populations," *Journal of Applied Physics* 26, no. 5 (1955): 586.

[NIOS] – NIOS-II Processor Reference Handbook. http://www.altera.com.

[NOVA] – opencores.org. Project nova. Oct. 2009.

[P.CLARKE 2009] – Peter Clarke, EE Times, "Xilinx, ASIC Vendors Talk Licensing." June 22, 2001. Retrieved February 10, 2009.

[P.KUHN 1999] –Peter Kuhn, *Algorithms, complexity analysis, and VLSI architectures for MPEG-4 motion estimation* (Dordrecht;Boston: Kluwer, 1999).

[P.NAISH 1988] – Paul Naish, *Designing Asics* (Chichester West Sussex England; New York: Ellis Horwood;; Halsted Press, 1988).

[POWERPC] – Xilinx PowerPC 405 Processor Block Reference Guide. January 16, 2004.

[PR_FLOWS] – Xilinx Inc., "Two Flows for Partial Reconfiguration: Module Based or Difference Based," September 9, 2004. Available: http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf

[PREVENT] – "The european fp7, prevent-intersafe project," Available: http://www.preventip.org/en/prevent subprojects/intersection safety/intersafe

[PRIMA-CARE] – PRIMA-CARE Project. 3 May 2011. http://www.primacare-project.com/.

[R.KHRAISHA 2010] – Rakan Khraisha and Jooheung Lee, "A scalable H.264/AVC deblocking filter architecture using dynamic partial reconfiguration," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (presented at the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing, Dallas, TX, USA, 2010), 1566-1569, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5495525.

[RADAR] – Cognitive Safty Radar Systems, www.trw.com.

[RECENT_FPGA] – Clive Maxfield, "The design warrior's guide to FPGAs: devices, tools and flows" (Boston: Newnes; Elsevier, 2004).

[S.BLACKMAN 1999] – Samuel Blackman, *Design and analysis of modern tracking systems* (Boston: Artech House, 1999).

[S.LEBEUX 2006] – Sébastien LE BEUX *et al.*, "FPGA Implementation of Embedded Cruise Control and Anti-Collision Radar". *9th Euromicro conference on Digital System Design : Architectures, Methods and Tools* (DSD 2006), Dubrovnik, Croatia, August 2006.

[S.LIU 2010_1] – Shaoshan Liu, Richard Neil Pittman, and Alessandro Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller (abstract only)," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '10* (presented at the the 18th annual ACM/SIGDA international symposium, Monterey, California, USA, 2010), 292.

[S.LIU 2010_2] – Shaoshan Liu, Richard Neil Pittman, and Alessandro Forin, "Energy reduction with run-time partial reconfiguration (abstract only)," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '10* (pre-

sented at the the 18th annual ACM/SIGDA international symposium, Monterey, California, USA, 2010), 292.

[S.YEOWYAP 2004] – S.Y. Yap and J.V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs* 51, no. 7 (July 2004): 384-389.

[SPARTAN-III] – Xilinx Inc., "Spartan-3 Generation FPGA User Guide Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families," *Xilinx user guide UG331 (v1.7)* August 19, 2010. Available: http://www.xilinx.com/support/documentation/user_guides/ug332.pdf

[T.MOORTHY 2008] – Theepan Moorthy, "Scalable FPGA Hardware Acceleration for H.264 Motion Estimation," 2008. *Thesis and dissertations*. Paper 121. Available: http://digitalcommons.ryerson.ca/dissertations/121.

[V4_FPGA] – Xilinx Virtex-4 FPGA User Guide UG070. December 1, 2008.

[VHDL] – Wikipedia. 7 February 2011. "http://en.wikipedia.org/wiki/VHDL" (last accessed at 7 February 2011).

[VIRTEX-IIPRO] – Xilinx Inc., "Virtex-II Pro and Virtex-II Pro X FPGA User Guide," *Xilinx user guide UG012 (v4.2)* 5 November 2007. Available: http://www.xilinx.com/support/documentation/user_guides/ug012.pdf

[X.MA 2006] – X. Ma and I. Andrasson. "Driver reaction time estimation from real car following data and application in GM-type model evaluation". *In Proceedings of the 85th TRB annual meeting*, Washington D.C., 2006.

[XILINX] – Xilinx Inc., http://www.xilinx.com/.

[Y.SHENGFA 2006] –Yu Shengfa, Chen Zhenping, and Zhuang Zhaowen, "Instruction-Level Optimization of H.264 Encoder Using SIMD Instructions," in *2006 International Conference on Communications, Circuits and Systems* (presented at the 2006 International Conference on Communications, Circuits and Systems, Guilin, Guangzi, China, 2006), 126-129.

[Y.W.HUANG 2005] – Yu-Wen Huang *et al.*, "Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder," *IEEE Transactions on Circuits and Systems for Video Technology* 15, no. 3 (March 2005): 378-401.

Scientific output of this work

- Journal:
  - HARB N., NIAR S., KHAN J., SAGHIR M. (2009). A Reconfigurable Platform Architecture for an Automotive Multiple-Target Tracking System. ACM SIGBED Review, vol6, num3, ISSN 1551-3688.
  - HARB N., SAGHIR M., NIAR S., BEN ATITALLAH R., KURDAHI F., Performance/Energy Trade-offs in a Dynamically Reconfigurable H.264 Motion Estimation Block, *ACM* Transactions on Architecture and Code Optimization TACO (submitted).

- Conferences
  - HARB N., NIAR S., KHAN J., SAGHIR M. (2009). A Reconfigurable Platform Architecture for an Automotive Multiple-Target Tracking System, 2nd Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'2009), in conjunction with Esweek (CASES'09, CODES+ISSS'09, EMSOFT'09), Grenoble, France, October.
  - LANGE T., HARB N., NIAR S., LIU H., BEN ATITALLAH R. (2010). An Improved Automotive Multiple Target Tracking System Design. 13th EUROMICRO Conference on Digital System Design DSD'2010, Lille France, September.
  - HARB N., NIAR S., SAGHIR M., EL HILLALI Y., BEN ATITALLAH R. (2011). A Dynamically Reconfigurable Filtering Block in a Driving Assistance System, IEEE Symposium on Application Specific processors SASP 2011 (part of DAC 2011) San Diego CA, USA, June.
- Posters
  - HARB N., NIAR S., SAGHIR M., A Reconfigurable Heterogeneous Multi-Core Architecture for an Automotive Driver Assistant System. ACES 2009, Edegem, Belgium, June 2009.
  - HARB N., NIAR S., SAGHIR M., A Dynamically Reconfigurable Filtering Block in a Driver Assistance System. Colloque du GDR SOC SIP, Paris, France, June 2010.
  - HARB N., NIAR S., SAGHIR M., A Reconfigurable Filtering Architecture in Driver Assistance System. 1st PROGram for Research on Embedded Systems & Software (PROGRESS, part of STW.ICT conference), Veldhoven, Netherlands, November 2010.