1-1-2013

A Cost Efficiency Analysis and Mechanism for Dynamic Partially Reconfigurable Computing Systems

David Diaz
Ryerson University

A COST EFFICIENCY ANALYSIS AND MECHANISM FOR DYNAMIC PARTIALLY RECONFIGURABLE COMPUTING SYSTEMS

by

David Diaz

Bachelor of Engineering, Ryerson, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2013

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions of individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopy or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

A COST EFFICIENCY ANALYSIS AND MECHANISM FOR DYNAMIC PARTIALLY RECONFIGURABLE COMPUTING SYSTEMS

Master of Applied Science 2013

David Diaz

Electrical and Computer Engineering

Ryerson University

Abstract

Dynamic Partially Reconfigurable Computing Systems have proven to be useful in environments where a multiplicity of tasks is required. These systems used Dynamic Virtual Components (DVCs) for reconfiguration. However, the computing architecture (dedicated, software, hybrid) of the DVC must be selected during the design stage. In this thesis, a mechanism in which we evaluate and analyze the cost efficiency of a DVC based on a cost efficiency factor (CEF) is proposed. Data centric and stream centric experimental tests were performed and the CEF of a 3D stereo-panoramic augmented reality hybrid DVC was determined. The results show that development costs and number of units to be produced influence the cost efficiency of a DVC. From the results it is concluded that the CEF can be a useful tool for selecting the computing architecture of a DVC, particularly when only a few units are to be deployed.

Acknowledgements

Completing my Master of Applied Science studies would never have been possible without the guidance of my professors, help from friends, and support from my family.

I would like to express my deepest gratitude to Dr. Lev Kirischian, my supervisor, for his guidance, patience, motivation, enthusiasm and immense knowledge. I could have not chosen a better supervisor for my thesis and I am glad to have chosen his lab for my graduate studies in pursue of my graduate degree.

I would also like to give my sincere thanks to my fellow colleagues Victor Dumitriu and Artur Saakov for their help in the lab. Their advice and sharing of their knowledge throughout my graduate studies is greatly appreciated.

I also would like to thank the Department of Electrical and Computer Engineering at Ryerson University for giving me this educational opportunity.

And lastly, I would like to thank my parents and my brother who have given me their support in pursuing my educational goals.

Dedication

This thesis is dedicated to my mother Gilda, my father Roberto, and my brother Roberto J. for their love throughout my life and their never ending support and encouragement in achieving my education goals in life.

Contents

List of Tables

List of Figures

List of Abbreviations

3D-SPADS 3D Stereo-Panoramic Acquisition and Display System

ASIC Application Specific Integrated Circuit

BRAM Block RAM (Xilinx)

FPGA Field Programble Gate Array

CPR Cost-Performance Ratio

CEF Cost Efficiency Factor

CLB Configurable logic block

CPU Central processing unit

DPRCS Dynamic Partially Reconfigurable Computing System

DRCA Dynamic reconfigurable Computing system

DVC Dynamic Virtual Component

FPS Frames per second

HW Hardware

I/O Input/Output

LED Light Emitting Diode

LUT Look-up table

MB Megabyte

NUI-ARDS Natural User Interface Augmented Reality Display System

SoC System on chip

VHDL Very-high-Speed-integrated-circuit hardware description language

VVL virtual video layers

XVGA Extended Video Graphics Array

1 Thesis Introduction

1.1 Motivation Statement

The motivation behind this thesis is to provide the embedded reconfigurable engineering community with an approach that will help guide hardware design engineers in determining the correct and most appropriate computing architecture design for a particular reconfigurable region allocated for a specific application in a Dynamic Partially Reconfigurable Computing System(DPRCS). Currently the most widely explored approaches involve the use of performance [1], power [2], area [3], and resources [4] for analysis of various computing systems. These types of analyses are useful when trying to develop a computing system that has strict performance requirements, low power, less chip space allocation and use particular resources more than others on chip. These constraint based analyses are useful when a computing system is being designed to meet particular constraint needs. However, in practice, during the design stage there are other factors such as number of units to be deployed and development costs. In the early years of FPGAs reconfigurable resources available on chip for a particular design may have been limited and, therefore, resource constraint analyses were useful to fit the design on the chip. Today, FPGA chips such as Xilinx's Virtex 7 are huge in terms of configurable resources [5] ; therefore, the limitation of not having enough resources or space is less of an issue. Also, the same VHDL architecture design on a newer FPGA may take up less power due to advancements in manufacturing technology that reduces static and dynamic power and increases performance [6]. These constraints become less critical as new design factors appear. Also, there are situations where the constraint variable is not strict

and instead of having a specific constraint value there is a range for the constraint. Let's assume we are to design a computing architecture for a configurable region on an FPGA. Now let us suppose there are three types of architectures considered for production. Each of the three designs have varying degree of performance, power, resources, and area but all of them are within the specified constraint range. Presented with this situation how does a design engineer select the most appropriate architecture for the given task? The design engineer may use his previous experience or intuition regarding each computing architecture option (qualitative analysis) and select one for production. However, this approach has a few drawbacks. It assumes the engineer has lots of experience behind him but he would not have the quantitative analysis to support his decision. To avoid this situation and provide a quantitative analysis to support more subjective qualitative analysis, some form of analytic mechanism needs to be introduced. A quantitative analysis previously introduced as a solution to this problem is the cost performance ratio (CPR). This CPR analysis takes into consideration the performance and cost of the system and is currently used to arrive at a decision. However, this earlier CPR analysis is based on the assumption that development costs are relatively small because large numbers of units will be produced. This assumption does limit its use for the design of Dynamic Virctual Components (DVCs) for partially reconfigurable systems because only a few DVCs may be deployed and more of the resources are directed to the development cost of a DVC; therefore, a design engineer should also take into account development cost and the number of units to be produced. For this reason a more refined version of the CPR is proposed to address the quantity and development costs in the cost efficiency analysis of a DVC.

1.2 Objectives

In order to address our motivation and achieve the goal of proposing a valid mechanism to determine the cost efficiency of a DVC the following steps need to be taken:

1. Research & analyze currently available methods that may be applicable for the efficiency analysis of partial reconfigurable systems.

2. Propose an alternative approach based on the concept of a Cost-Efficiency Factor (CEF).

3. Test the proposed approach by implementation of dedicated hardware components, hybrid components and software based components.

4. Analyze the results from the two test experiments.

5. Implement the proposed approach during the computing architecture design phase of an industry specific application involving live 3D stereo panoramic video streaming with a remote robotic control system.

6. Identify any foreseeable limitations or future improvements to the proposed approach as observed during test experiments and specific industry application implementation.

1.3 Original Contributions

The contributed research work to the objectives stated previously are:

1. Analysis of currently available methods.

2. Concept and approach of using a CEF as the mechanism of determining the correct computing architecture design for a particular application for dynamic partially reconfigurable computing systems.

3. Software and Hybrid computing architecture implementation for two test applications.

4. Computing Architecture design and Implementation of the Natural User Interface Augmented Reality Display System (NUI-ARDS).

5. Experimental results and discussion of computing architecture design using a CEF approach.

1.4 Thesis Organization

The remaining organization of this thesis is as follows:

1. Chapter 2 is dedicated to the collection and analysis of current related work in the field and to highlight the need for the proposed approach.

2. Chapter 3 proposes the concept of using a Cost-Efficiency Factor (CEF). It will discuss how this CEF can be used in determining the correct computing architecture design of a given programmable logic given that a dynamic partially reconfigurable computing system is to be used.

3. In chapter 4, two experimental test will be examined and the results presented. The CEF for a complex design and implementation of an industrial application involving 3D stereo panoramic video streaming with augmented reality and remote robotic agent control system is determined.

4. Chapter 5 discusses the results and the implications CEF can have in the design of dynamic partially reconfigurable computing systems.

2 Related Works

2.1 Introduction

Over the years embedded digital systems development has become more prominent as devices become smaller in size and specialized in function. To date embedded systems have become an integral part of every computer systems; however, the engineering design aspect of computer systems still remains to be fully explored. This chapter will overview the current available approaches for the design of embedded partially reconfigurable systems.

2.2 Related Work Observation & Analysis

Computer systems are commonly categorized into one of the following three main computing architecture: Dedicated, Software, or Hybrid computing systems. Dedicated computing architectures generally consists of dedicated hardware that is specifically designed for a particular task and does not contain any sequential processing elements such as a microprocessor. A diagram for the general organization of a dedicated computing architecture is depicted in Figure 2.1. In this type of computing architecture there exists only inputs and outputs, data paths, and control units for data synchronization. Dedicated computer architectures are usually suited for computational intensive tasks. On the other hand, a CPU-based computing architecture, the cheapest and most general purpose architecture to utilize, may not be optimized for the intended task. This type of computing architecture, depicted in Figure 2.2, contains a sequential processing element such as a microprocessor and no dedicated hardware components. This CPU-based computing architectures are suitable

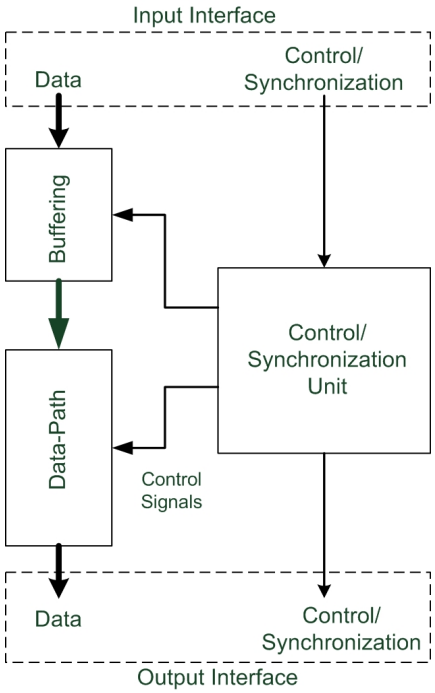

Figure 2.1: Diagram of a general dedicated computing architecture [7]

for algorithmic intensive or are very sequential by nature tasks. A hybrid computing architecture, depicted in Figure 2.3, tries to take advantage of both worlds by containing both a sequential processing element (sequential instruction processor) and a dedicated hardware (HW accelerator). A communication infrastructure is then used to exchange data between both systems. In order to take advantage of all three computing architectures one can use

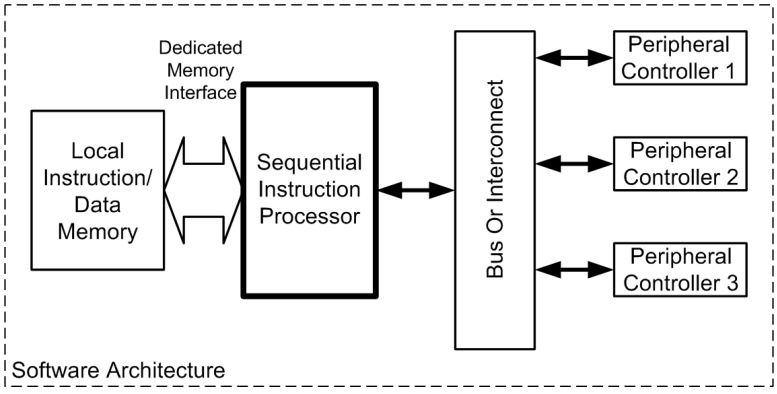

Figure 2.2: Diagram of a general software computing architecture [7]

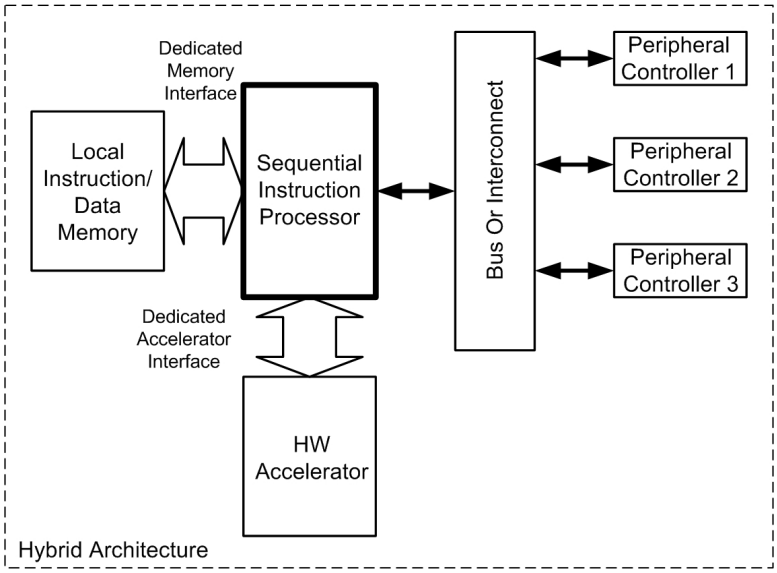

Figure 2.3: Diagram of a general hybrid computing architecture [7]

a dynamic reconfigurable computing architecture (DRCA) depicted in Figure 2.4. This type of computing architecture usually consists of two sections: a static section and a dynamic (Reconfigurable) section. The static section usually consists of a microprocessor and the dynamic section consists of reconfigurable logic which can be reconfigured by the microprocessor whenever a different architecture(dedicated, software or hybrid) is needed for a given task. For example, this approach is effective for small dynamic sections in which the time in between tasks is greater than the reconfiguration time. The reconfiguration time is the time it takes for the programmable logic area to be reconfigured with another computing architecture.

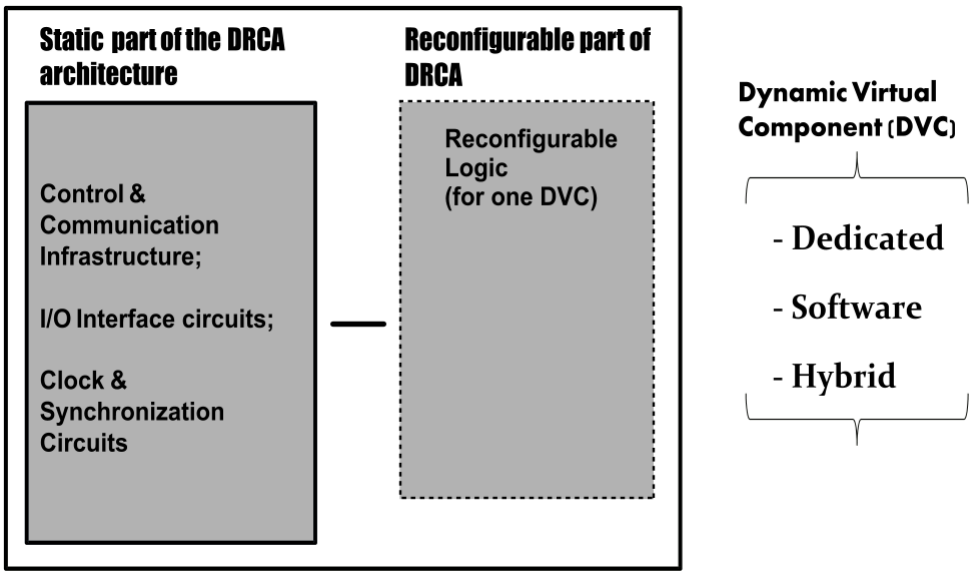

Figure 2.4: Diagram of a DRCA

A potential problem may arise when other tasks are required to continue executing while at the same time another programmable computing architecture is needed for a new task execution. In this situation a Dynamic Partially Reconfigurable Computing Architecture(DPRCA), depicted in Figure 2.5, has been proposed as a possible solution. A DPRCA

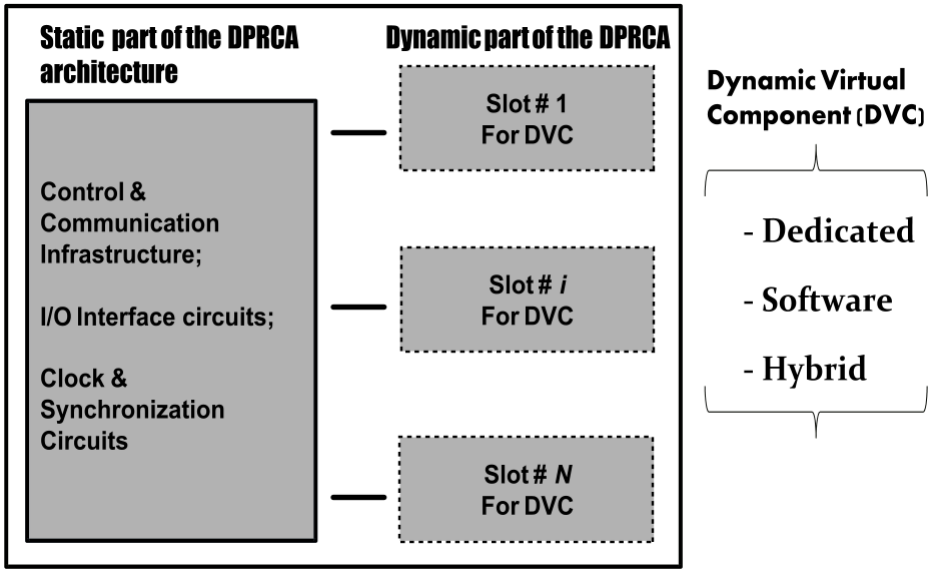

Figure 2.5: General diagram of a DPRCA

[8, 9] is similar to a DRCA except that the programmable logic section of DPRCA consists of

multiple slots. Each slot can be reconfigured with any Dynamic Virtual Component(DVC) which can be either a Dedicated, Software, or Hybrid computing architecture design. One advantage of a DPRCA is that the hardware can adapt depending on the task at hand. Traditionally, a task is usually constructed or adapted to fit a particular computing architecture but in a DPRCA one can think of adapting the computing architecture to fit the task instead, which is achieved by loading reconfigurable slots with a specific DVC depending on the task that needs to be performed. However, in order to adapt the computing architecture to the given task one must have a way to determine which computing architecture type is to be implemented in a DVC for a particular task, the main focus of this thesis. Previous related work done in this area revolves around the idea of simulating computer architecture performance, effectively mapping a given task onto whatever processing elements are available or how to create effective architectures for a particular task. In [10] and [11] the emphasis is on computer architecture modeling of hybrid architectures so that one can determine if a hybrid architecture is suitable for the task. However, the results are very related to how well the simulators are built. Even well built simulators have flaws, such as the inability to consider new technology resources, the assumption that all resources available are the same and unlimited, and that all solutions are possible to develop in terms of cost. To build a simulator one must have a particular method to solve the problem, however, to our knowledge, there is no method or simulators capable of making the decision as to which computing architecture should be implemented in a DVC given a particular task. Much work has also been done in using algorithms for finding optimum hardware software partitioning methods to address constraints such as configuration area, configuration blocks (CLBs), available resources, and power consumption. In [12] a Particle Swarm Optimization

inspired approach is used to find a range of valid hardware-software partitioning solutions. However, a method for selecting which of the implementations should actually be used in a DVC design for a particular task is not provided. Other works such as [13] try to find ways in which to effectively schedule tasks between dedicated hardware and software architectures in order to improve efficiency in terms of performance and area. This would result in having many possible solutions but not in providing a quantitative analysis to help decide as to which computing architecture should be implemented into the DVC. From a hardware computing architecture point of view there has been work done such as [14] and [15] that show the benefits and limitations of both heterogeneous and homogeneous computing systems. In [14] the system cost is discussed in terms of the cost of the processors selected and the links between them but a DVC could also potentially be a dedicated or hybrid architecture. [15] Looks at how heterogeneous architectures can provide a more linear performance increase when compared to homogeneous architectures under certain task conditions by having more optimized processing components (such as multipliers, LUTs, Dividers, etc) catered towards specific functions. This however would also lead to multiple architecture solutions for a DVC but without being able to know which of them should actually be implemented as a DVC. However, all the preceding approaches are constraint driven. On the other hand there is a need to have an evaluation based on a cost-efficiency of the system. There are few works [16, 17, 18, 19] which consider ratio between performance and cost; however, these works focused on mass production, thus the number of units as well as the cost of research and development plus the cost of design were implicitly embedded in the cost of each unit, and this is not explicitly analyzed.

2.3 Summary

In summary, there has been much work done as to how to design proper dedicated, software and hybrid architectures. Many techniques have been developed to properly construct computing architectures and the advantages and disadvantages of each architecture has been explored. However, there has not been any method or technique, to our knowledge, that will aid in the selection of the correct computing architecture to be implemented into a DVC given multiple possible solutions which meet all task requirements.

3 Proposed Approach to Evaluate the Cost Effectiveness of a DVC

3.1 Introduction

Given all the previous research work done into how to design proper dedicated, software, and hybrid computing architectures our approach will focus on how to determine the correct computing architecture implementation for a DVC given a particular task to perform. This decision will require certain efficiency measurement mechanism and in this work the use of a CEF is proposed as such mechanism. It will be explained in detail in this chapter and used as a mean to arrive at a decision for a FPGA-based computer architecture. Before proposing our approach, we first analyze and discuss the various factors that affect the cost of a given on-chip computing architecture.

3.2 Concepts and Theory Analysis

When hardware design engineers build computing architectures many factors that affect the cost of a computing architecture are considered. In most published literature in the field, the term "cost" for the FPGA-based SoC usually refers to how many configurable logic blocks or how much area, power, or memory is required for a particular computing architecture [20]. For the purpose of this thesis the cost of a particular DVC architecture will refer to the monetary quantity required to produce the DVC.

In general, determining the cost of an entire computing architecture can be very complex and may change over time. However, as we will show in this study, it is a critical aspect to

consider when designing a DVC. There are many trends and factors that are well known to affect the design of a computing architecture and they may be associated with manufacturing technology, quantity of production, market target and development costs [21]. The manufacturing of a specific computing architecture (e.g. ASIC) decreases as time passes because the manufacturing techniques improve over time which leads to a higher yield. Yield is the number of computing architecture components that pass the testing stage. For integrated circuits, the manufacturing cost consists of many variable costs such as the cost of die, testing die, die yield, cost of packaging, etc. Without going into copious details regarding integrated circuit costs, it is important to note that die size and yield have an important effect on its cost. The more die area required by the computing architecture leads to an increase in the cost of the integrated circuit. This is important to take into account if there are various DVC candidates that require different die areas. Manufacturing affects cost because it is always evolving over time. Improvements in the manufacturing process over the years contributes to cost reduction as die yield improves and contributes to the reduction of die cost which designers don't have control over. Both of these cost reductions translate into a reduced integrate circuit cost which is important, particularly if there are multiple competitors.

When manufacturing any type of integrated chip there is usually the question of quantity. Determining how many chips are to be manufactured with a given computing architecture is important when considering costs. The three major common computing design markets are: low cost design, cost-performance design and high-performance design market areas [21]. In each design area the design objective is different. In a low cost design market the goal of the computing architecture design is to arrive at the least cost solution for the computing architecture with performance being acceptable and not a priority in the decision process.

Usually, in a low cost design market the demand for the computing architecture is high hence large quantity orders are expected. In a cost-performance computing architecture market the goal is to design a computing architecture which is neither cheap nor expensive but rather has the best performance for its cost. For the high-performance design market the computing architecture goal is purely performance driven and the cost is not a primary concern, usually these types of computing architectures are not produced in large numbers due to the large cost associated with them. The key concept to understand here is that the relationship between price and quantity can get intricate.

We will now discuss some important cost implications for a product containing an on-chip computing architecture design. Two main cost categories that affect the development of a computing architecture product are direct and indirect costs. Direct costs are directly associated with the making of a product and are usually 10% to 30% of component costs [21]. Direct component cost refers to the raw components available for production which do not need to be acquired externally from the company through purchase. Some examples of direct costs are: purchasing of components(which are not internally available), labour, and warranty. Indirect costs result from the architecture design and cannot be associated to the production of a single product. Examples of indirect costs are: research and development (R&D), manufacturing equipment maintenance, marketing, building rental, cost of financing and sales. Figure 3.1 shows how the cost of a product evolves in each step of its production. Indirect costs range usually from 10% to 45% of the average selling price [21]. The average selling price of the product is defined as the summation of components costs, direct costs and indirect costs. On top of all these costs it is possible to add an additional average discount cost resulting in the list price of the product. The list price is the price most likely seen by

consumers and includes costs if the product is to be sold at a discount price. In markets where there is high competition the list price may not exist and to further compete companies have the option of reducing direct, indirect and component costs. Since at every production stage the preceding cost affects the subsequent costs it is very crucial to select the appropriate computing architecture in order to gain a cost advantage over a rival competitor in a cost-performance or low-performance design market. These cost implications also highlight the need of having some form of quantitative mechanism that will help design engineers to determine the correct computing architecture approach with some form of computational or logical analysis to support their decision instead of relying solely on past experiences and intuition (qualitative analysis).

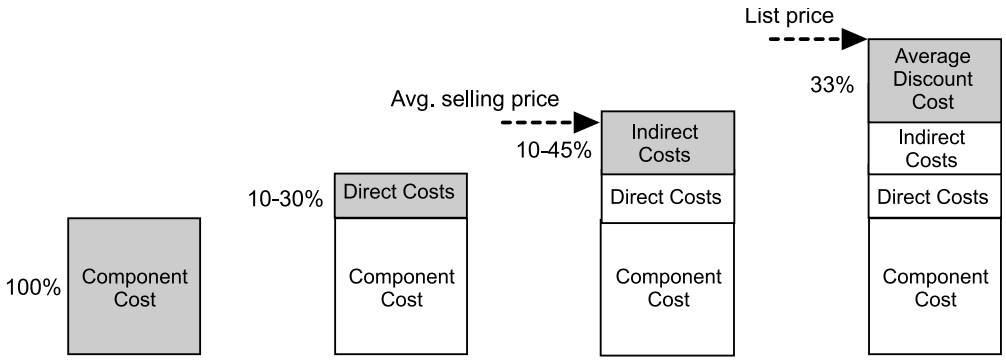

Figure 3.1: The cost component of a computing device as the price evolves with the addition of direct and indirect cost [21].

After discussing the many factors that may affect the production of computing architecture we will specifically identify and discuss those factors that are applicable in the decision process for the development of a DVC. Firstly, since a DVC is to be loaded into a slot located in a DPRCS the cost of the integrated circuit for a particular DVC can be considered as part of the overall cost (both static and dynamic sections of the DPRCS) of the integrated circuit that would house the entire DPRCS. This means that the integrated circuit design

cost may be affected by the DVC architecture. If a DPRCS is to be used for a particular application with three possible DVC variations (Dedicated, software, or hybrid) and each requiring a different size of integrated circuit area resulting in three DPRCS variations with different integrated circuit area needs. Figure 3.2 illustrates such a situation when a Field-Programmable Gate Array (FPGA) logic device is used for implementation. In Figure 3.2B the Hybrid variant of the DPRCS requires 1600 configurable logic blocks(CLBs) which requires a larger and more costly FPGA chip than the dedicated and software DVC architectures in Figure 3.2A and Figure 3.2C, both of which can fit on a smaller sized FPGA chip.

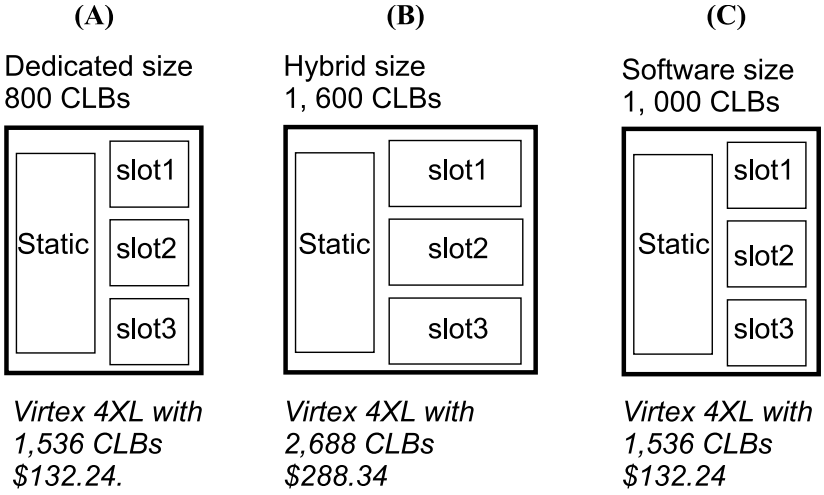

Figure 3.2: DPRCS with different sized slots and DVC

This observation suggests that any method used in determining the nature of a DVC must take into account the integrated circuit cost of the static and dynamic area of the DPRCS in which the DVC will be placed in. When dealing with FPGA's as the reconfigurable platform the integrated circuit cost of the DPRCS can be considered as the cost incurred by the resources required by the static and dynamic area. The incurred resources can be viewed

as fine grain such as LUTs, flip flops,and gates or viewed as course grain such as divider, adders, and multipliers. For the purpose of this study we will simplify the resources to the amount of CLBs required by both the static and the dynamic sections. The cost of each section is proportional to the platform cost (an FPGA in this case) where the fraction is the percentage of CLB's used by each section. In equation 3.1, SSC represents the cost of the static section of the DPRCS, SS_R represents the resource count (CLB count) required by the dynamic section. The term T_R represents the total resource count available in the reconfigurable platform and RPC refers to the reconfigurable platform cost which is the cost of the FPGA chip in this case. In equation 3.2, DSC refers to the cost of the dynamic section of the DPRCS and DS_R refers to the resource count required by the dynamic section which in our case is the resource count required by the DVC.

$$SSC = \frac{SS_R}{T_R} \times RPC \tag{3.1}$$

$$DSC = \frac{DS_R}{T_R} \times RPC \tag{3.2}$$

The second direct cost required by DVC are its labour development cost. Depending on which architecture is deployed different knowledge and expertise with different cost values is required. For example, if the choice was made that a particular DVC is to be deployed using a dedicated hardware then one must employ skilled labour with hardware knowledge. However, if the DVC is to be deployed on a software based DVC then one can simply acquire skilled labour with software programming knowledge. Table 3.1 shows how different engineering skill set levels may vary in cost, having an effect on labour cost which is an important factor in the development of a computing architectures. Table 3.1 is the average Canadian labour

cost for electrical, computer, and software engineers for 2010 obtained from Living In Canada [7] data. Since these labour skills have different labour cost any methodology that will help determine the nature of the DVC to be produced should probably take into consideration this variable.

Table 3.1: Engineering Avg. Wage per Hour (Toronto)

Electrical Engineer	Software Engineer	Computer Engineer
$34.53	$36.60	$31.78

A third factor that may influence the nature of the DVC is its market area. If the DVC is to be used in thousands of units rather than just tens of units then perhaps a higher DVC cost may or may not be worth it in some cases. Therefore the number of units to be produced will have to be considered in some way as well.

After discussing some costs related to embedded digital systems we will look at how these costs influence architecture design decisions. Currently when designing reconfigurable systems designers have three main general architecture design approaches to choose from. One design approach may consist of using an Application Specific Integrated Chip (ASIC), another option is to use a Field programmable Gate Array (FPGA), and lastly a Microprocessor could be used as well. In order to compare the three types of implementation solutions and select the most appropriate one, development designers commonly use a cost-performance ratio (CPR) for comparison. This CPR is calculated using Equation 3.3.

$$\text{Cost-Performance Ratio (CPR)} = \frac{\text{Performance}}{\text{Cost}} \tag{3.3}$$

Where CPR is equal to the performance metric of the computing architecture divided by

its price [22]. Simply using the CPR computing systems with an FPGA approach can be seen as inefficient. ASIC based computing systems give higher performance over FPGAs, have lower power consumption, and are less expensive compared to FPGA chips that can cost more than $1,000. Therefore when the performance of a microprocessor architecture is not enough the CPR clearly indicates that an ASIC design is a better design solution because of its superior performance gains and price per chip compared to its FPGA counterpart. However, in reality this appears not to be the case and the industry shows quite a different picture. FPGAs architecture designs today are vast and FPGA manufacturers like Xilix and Altera continue to produce and sell millions of FPGAs chips yearly. Figure 3.3 shows how FPGA designs over the resent years and into the future will continue to rise while ASIC designs continue to fall. If the CPR as it is used currently is correct then there should have been an increase in ASIC designs and a steady fall in FPGA instead. This contradiction implies we must take a look and reassess the way in which the CPR is used for the design of computing architectures.

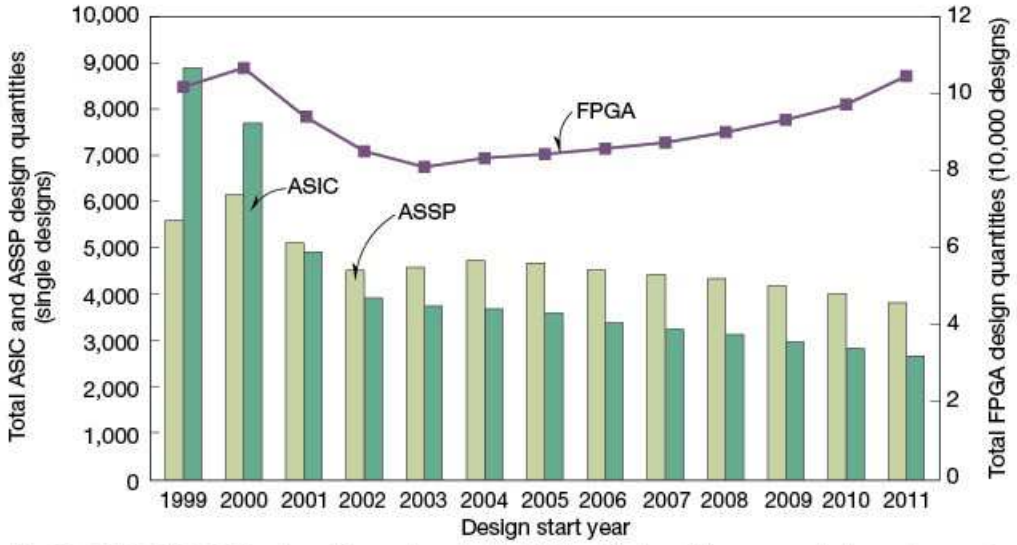

Figure 3.3: FPGA Vs. ASIC Startups [23]

Today, embedded systems can be found in many markets such as consumer electronics, aviation and aerospace, medicine, machinery and other custom electronics. Furthermore, the computing designs in these areas are multi-functional. In the past, it was fairly common to find an electronic device designed for a single function. Today, electronic devices are increasingly becoming multi-functional and at the same time they must continue to be reliable, low power, low cost and fast enough to perform all the functions required. One example of such a device is mobile phones. They are not just used to make phone calls but also for taking pictures, editing documents, listening to music, watching video, and sending or receiving data using multiple wireless standards. Cell phones are a good example of a consumer market electronic where the production of units is very large. However, the design market for consumer electronics is just a small part of the overall embedded systems market and the number of units produced or manufactured in other sectors range from a hundred units to just tens of unit only. This means that systems are now being designed to be multi-functional and limited units being produced. For this reason many designs requiring multiple functions and limit units to be produced have turned to FPGAs. FPGAs provide the reconfigurable ability to allow the computing architecture to be multi-functional without the need to have large ASIC development costs or long development cycles for limited production.

Over the last few years, FPGA configurable logic gates have gone from a few millions to hundreds of million. The large size FPGAs allows embedded reconfigurable design engineers to develop a System on Chip (SoC) on a single FPGA. Although larger FPGAs provide the ability to have a SoC on a large FPGA changes or modifications to the design result in large compilation times and larger configuration bit streams. For this reason partial reconfigurable systems have become a viable solution to this problem. This is because the

particular component (which we address in this study as a DVC) of the design that needs to be changed or modified can be recompiled with the three common architectures discussed earlier. Although this solution leads to lower compilation times and smaller bit streams it presents a question as to how a designer would now select the proper computing architecture for a particular DVC given that they have three main options to choose from. At first glance it appears that the CPR could help in this decision making. However, the current CPR already fails in explaining why configurable computing systems have become more abundant in the industry and perhaps the CPR should be modified in order to more accurately address and explain the current architectural design issue faced by engineers when determining the computing architecture of a DVC in partially reconfigurable computing systems. The biggest disconnect between how the CPR is obtained and the development of DVC is that CPR does not explicitly take into account the number of units to be produced. When designers use CPR most inherently assume that the number of units to be produced is extremely high; however, as we explained before, that is only the case for the consumer market but not for the entire engineering marker as a whole. Therefore, by using the CPR as it is currently used for partially reconfigurable computing systems we are incorrectly assuming that the DVC to be designed will be deployed in thousands of units when in reality the DVC might only be deployed in tens of custom units. We also have to acknowledge that since the quantity of computing architectures can be few, we must take into account the development cost portion of direct cost which can substantially affect the final cost of a computing system. Unfortunately the CPR fails to address the development cost because of its inherit assumption that development cost becomes insignificant by the fact that millions of units are to be produced. To address these two main issues, we propose a modification to the way the CPR

is calculated by making sure that development cost and quantity of computing units to be sold are taken into consideration when determining the correct computing architecture of a DVC. Lastly the design of a DVC must also take into consideration the cost of reconfigurable resources required by the DVC since the price of an FPGA chip increases with size.

Taking into consideration all the aspects that affect the cost of a computing architecture and examining how the design of DPRCS has evolved we propose the use of a Cost-Efficiency Factor (CEF), Equation 3.4, to help guide the decision process as to whether a particular DVC should be implemented as a dedicated , software, or hybrid computing architecture. As one can see in Equation 3.4 the CEF, unlike the CPR, takes into account the number of DVC units to be produced and their development cost. The cost of the reconfigurable resources required, and the performance obtained from the computing architecture solution are the same as in Equation 3.

$$CEF(A_i) = \frac{P(A_i)}{UC + \frac{DC(A_i)}{N_{units}}} \tag{3.4}$$

Where:

A_i = Architecture A_i

$CEF(A_i)$ = Cost Efficiency Factor of Architecture A_i

$P(A_i)$ = Performance of Architecture A_i

$DC(A_i) = DT(A_i) \times AC(A_i)$ = Development cost of Architecture A_i

$DT(A_i)$ = Development Time for Architecture A_i

$AC(A_i)$ = Average Development Cost for Architecture A_i

N_{units} = Number of units to be produced

UC = Unit Cost: SSC + DSC + manufacturing + etc

The Unit Cost term, UC, consists of the platform cost (static + dynamic) and indirect costs such as manufacturing, packaging, etc. The $P(A_i)$ term is the desired performance metric result for the proposed DVC architecture. For example, in a video application the $P(A_i)$ variable can be frames per second(fps) or bits per second(bps) in a data execution. The Development Cost term, $DC(A_i)$, is calculated based on the number of hours needed to develop the architecture ($DT(A_i)$) and the average national wage for such particular work $AC(A_i)$. The average national wage may or may not be the same for all possible architecture solutions and varies depending on location. Lastly the N_{units} term represents the total number of computing units anticipated for production. This term is important because it is important to notice that a particular DVC may or may not be used in millions of of DPRCS instead it may just be used in tens of units. Since the quantity of DPRCS with the specified DVC unit to be produced has an effect on costs and varies depending on application one cannot eliminate the N_{units} term. Assuming the DVC will be used in millions of units would be incorrect since it is not always the case, especially in areas concerning R&D or specialized custom systems where production is few.

The units for the CEF factor is always performance metric per dollar. Therefore, if we are designing a system for video processing then one performance metric could be frames per second (FPS), then the CEF would be FPS per dollar. With this information the designer can see how much performance metric is attained per every dollar that is put into developing the architecture. The higher the CEF value the more cost effective the design is since one is getting a higher performance value per every dollar spent on developing the

design. This type of computing architecture analysis can be beneficial in the design of a DVC architecture tailored towards the cost-efficiency computing market or application such as multitasking smart-phones, and high performance video streaming. It is important to note that performance metric calculations and analysis for a given computing architecture may differ from one application to another application; therefore, during the CEF analysis one must decide on one particular performance metric. For example, if the performance metric for the dedicated DVC is FPS then the performance metric for the software and hybrid DVCs should also be FPS.

Although there may be many ways to analyze and measure performance there are two widely used performance metrics which are execution time and throughput. Execution time is the time elapsed from the start of the task to the end of the task. Execution time includes everything the computing system needs to do to accomplish the task such as memory access, I/O bus service routines and Operating System (OS) overheads. Throughput is the amount of work done within a specified time. One way to increase throughput is to reduce the task execution time but one can also take advantage of parallelism in tasks as well. For the purpose of determining the CEF in this study, we analyze performance in terms of execution time.

In order to determine what kind of an impact our proposed CEF approach may have on the architecture design decision of a DVC we need to consider multiple experiments that: 1) Contain commonly used architecture features such as architectures that are mostly stream based where data is mostly coming in, processed and sent out quickly without the need to store large amounts of data; and, 2) Architectures that are memory centric in which data that is being processed and its results need to be stored in memory. For this reason we

designed three experiments that will help us to see how useful the CEF can be in the design of DVC architectures.

The first experiment involves RGB to YC_BC_R colour conversion which is a more of a streaming application since the RGB colours are convert to YC_BC_R colours and do not have to be necessarily stored. The second experiment deals with Sobel mask computation for the purpose edge detection techniques. This experiment is more memory centric because the inputs and outputs to the Sobel mask calculations need to be stored so that they can be further processed. The third experiment is more complex and incorporates both streaming and memory centric features of the first two experiments.

3.2.1 Summary

By using equation 3.4 along with performance requirements and costs we will demonstrate how a CEF value can be used to correctly select the proper DVC computing architecture for a particular application for development in situations where cost-performance is of importance without ignoring the development costs and without the assumption that production will always be extremely high. In this study, we evaluate the use of a CEF on three experimental implementations: RGB to YCBCR colour conversion, Sobel Mask computations and a natural user interface with augmented reality display system(NUI-ARDS). The detailed experimental procedure and observations for these experiments are described in the following chapter 4.

4 Approach Using Experimental Applications

4.1 Introduction

In this thesis we use three video processing applications as mentioned in chapter 3 to demonstrate how the CEF can help guide a design engineer or an automatic tool select the most appropriate computing architectural structure of a DVC to be developed. The applications used for this study are in the area of edge detection, colour conversions, and augmented reality. We will look at all three applications and evaluate the usefulness of a CEF when designing a DVC for a DPRCS hardware platform. The following is a list of experiments selected to examine our proposed use of a CEF for determining the architectural nature of a DVC which were introduced in chapter 3:

1. Sobel Mask Computation
2. RGB to YCBCR Colour Conversion
3. Natural User Interface with Augmented Reality (NUI-ARDS)

The reason these three experiments were chosen was that they deal with streaming and memory centric architecture applications which are very commonly found in computing designs and, therefore, will give us a good measure as to how this CEF can address the design concern of DVC in partially reconfigurable computing systems.

4.2 DPRS Experimental Setup

In order to perform the three listed experiments, an appropriate platform is required. For the purpose of this thesis an existing Multi-Application Reconfigurable System (MARS) platform was selected. MARS is a custom platform consisting of a Virtex 4 FPGA with a 1024 x 768 projector display, 3D stereo panoramic capture, memory for video buffering, and a stereo camera for motion tracking. The embedded DPRS inside the Virtex 4 FPGA in which the DVC would be placed is depicted in Figure 4.1.

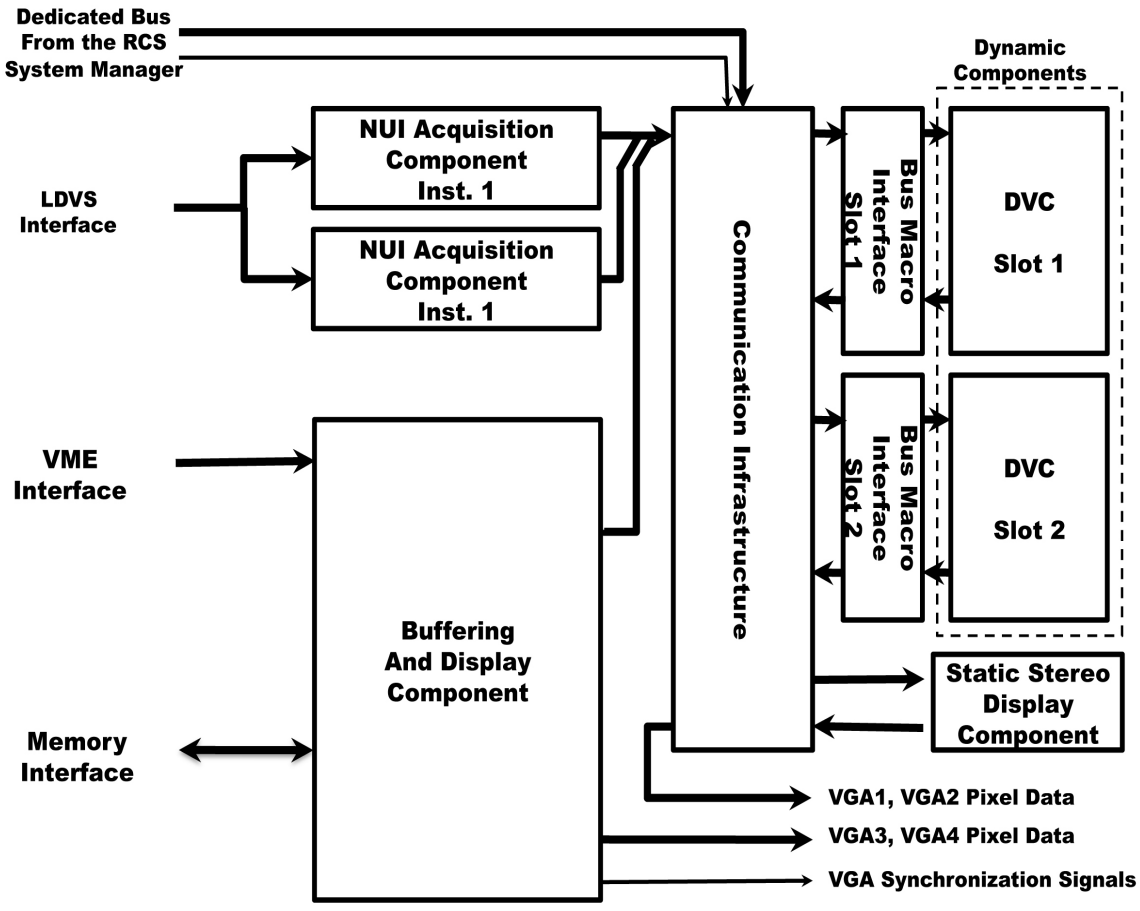

Figure 4.1: MARS Experimental Embedded Platform On-Chip Architecture Organization

4.3 Data Centric Experiment Test

4.3.1 Sobel Mask Computation Description

In image processing a common task performed that is computationally intensive and requires memory storage for the inputs and output results is edge detection. There are many edge detection methods and algorithms but in this particular experiment we specifically implement Sobel mask computation DVC for the purpose of an Edge detection application.

4.3.2 Sobel Mask Computation Principles

Edge detection is defined as an approach for segmenting an image based on abrupt local changes and this process is explained in [24]. When performing edge detection on an image there are three main tasks to perform:

1. Take the image and apply some method or process to reduce noise. One example of image noise reduction is image smoothing.

2. Extract all possible edge points from the given image using local operations such as various masks (for example a Sobel mask).

3. Select and determine the true edge points from the set of points obtained in step 2.

To extract all possible edge points in an image one can take advantage of the image gradient. The strength and direction of the gradient at each pixel of an image can be determined using equation 4.1 where the gradient is denoted by ∇f. The original image is

denoted as f . The term g_x is the gradient vector component in the x-direction and the term g_y is the component of the gradient vector in the y-direction.

$$\nabla \equiv grad(f) \equiv \begin{bmatrix} g_x \\ \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{4.1}$$

In order to calculate the partial derivatives to obtain the gradient of an image one can use Sobel partial derivatives using Equations 4.2 and 4.3. Where the variables Z_1 to Z_9 represent the colour(red, blue, or green) values for the image pixels under the Sobel mask as depicted in figure 4.3.

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{4.2}$$

$$\tag{4.3}$$

The main purpose of this experiment is to determine the appropriate computing architecture for the computation of the gradient in an application that would be performing the three main steps in edge detection. Our goal is to determine if the DVC component that will be responsible for gradient computation in step 2 should be implemented as a dedicated, hybrid, or software DVC.

4.3.3 Sobel Mask Experimental Implementation

In our experimental setup, we use a Sobel mask to obtain the partial derivatives of an image that is of 32x32 pixels in size. The Sobel equations 4.2 and 4.3 used for partial derivatives are implemented using a mask as depicted in Figure 4.2(A). The variables $M_{0,0}$ to $M_{2,2}$ represent

the value for each element in a mask, thus different masks have different element values.

The Sobel mask in Figure 4.2(B) was applied on the image to get the gradient in the x-direction, g_x. The mask is implemented by aligning the mask starting at the top left corner of the image and sliding it along the image with each pixel of the image centred under the mask centre element. At each pixel the partial derivative is calculated using the mask. since the image pixel has to be centred under the mask the gradient for the first and last row, and the first and last columns cannot be calculated. Figure 4.3 illustrates how this calculation process was performed. In Figure 4.3, (A) shows the starting pixel position in the image, (B) shows how the mask is shifted and centered for the next pixel in the image and (C) shows the last pixel position in the image from which the gradient can be calculated using the Sobel mask.

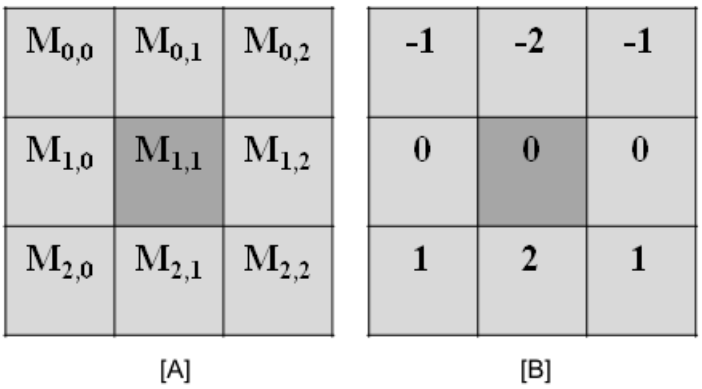

Figure 4.2: [A]General 3x3 mask structure [B]Sobel Mask

The gradient for pixels along the first and last rows, and for pixels along the first and last columns can not be computed because the mask cannot be centred on these pixels which results in the final image being two rows and two columns smaller than the original image. For the purpose of this experiment a random set of values were used for pixel image values.

In order to determine performance we used a system cpu time approach because it pro-

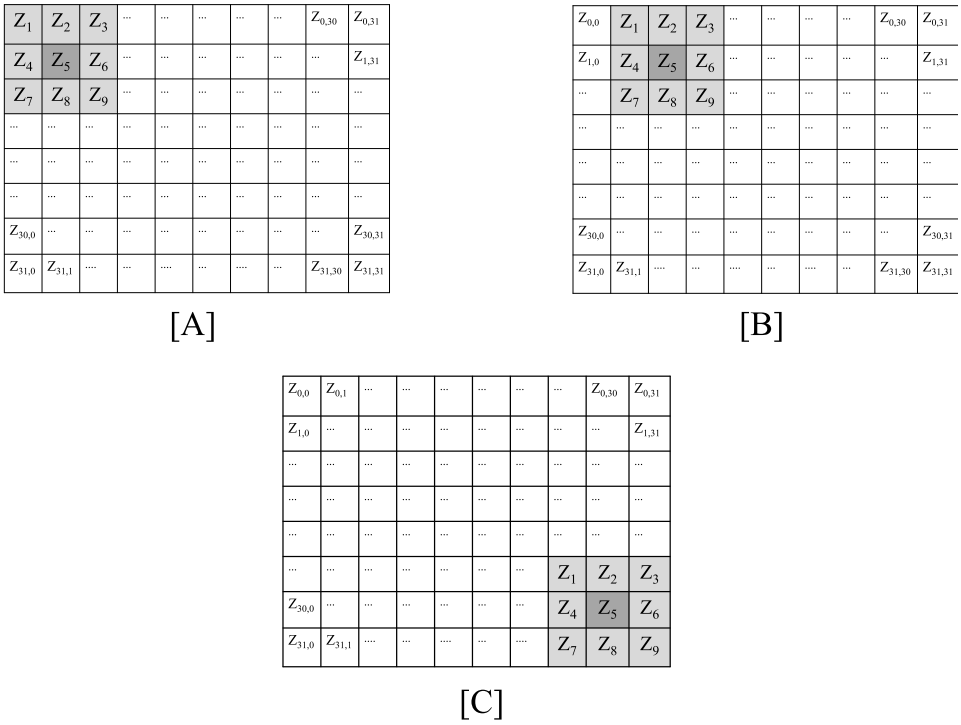

Figure 4.3: Sequence of images depicting implementation of Sobel mask on a 32x332 pixel image

vided the best comparable measurements between the three types of systems. Each DVC asserts a signal to indicate the start of the gradient computation and deasserts the signal when the gradient computation has been completed. The start and end signal of the gradient task were captured using a digital analyzer connected to an output pin on the MARS platform. The execution time for the task was taken to be the time between the rise and fall of the signal as depicted in Figure 4.4. The hybrid and software DVC have no operating system overhead and all low level software is written in C to ensure that the execution time measured consisted only of system CPU time.

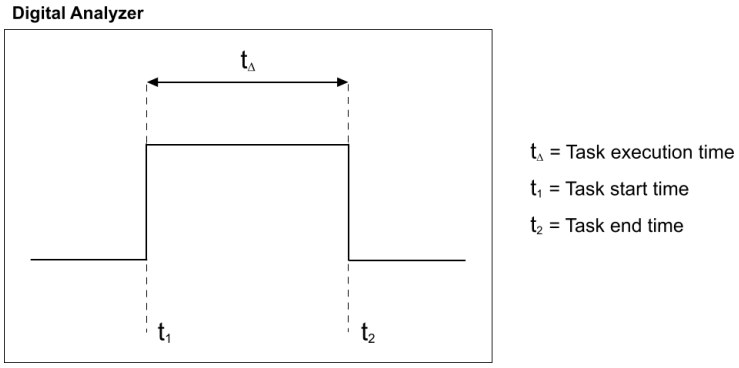

Figure 4.4: Measurement of task execution

4.3.4 Sobel Mask DVC Symbol

Figure 4.5 show a block diagram of the Sobel Mask DVC.

4.3.5 Sobel Mask DVC I/O Signals

Table 4.1 explains the input and output signals of the Sobel Mask DVC.

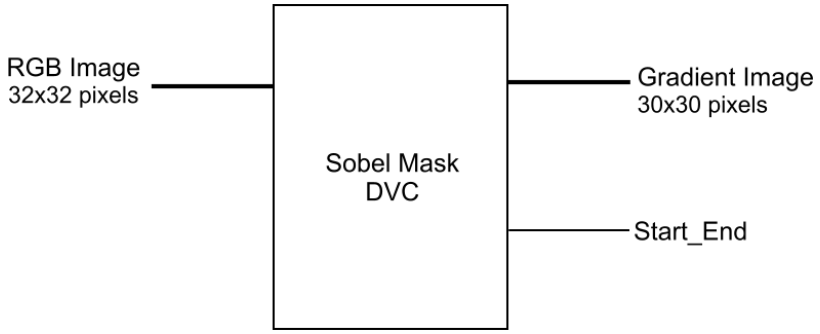

Figure 4.5: Sobel mask DVC symbol

Table 4.1: Sobel DVC input signals

Name	**Size**	**Description**
RGB Image	32x32 pixels	32x32 pixel image to which the Sobel Mask will be applied to.

Table 4.2: Sobel DVC output signals

Name	**Size**	**Description**
RGB Image	30x30 pixels	This is a 30x30 pixel image resulting from the applied Sobel Mask computation
Star_End	1 bit	This signal is used to determine the start and end of the Sobel Mask Computation on a 32x32 image

4.3.6 Sobel Mask DVC Organization

The organization of components within the dedicated Sobel Mask DVC were treated as a black box hardware accelerator. Figure 4.6 shows the dedicated computing accelerator and its interface signals. The input and output signals for the dedicated DVC are defined in tables 4.3 and 4.4.

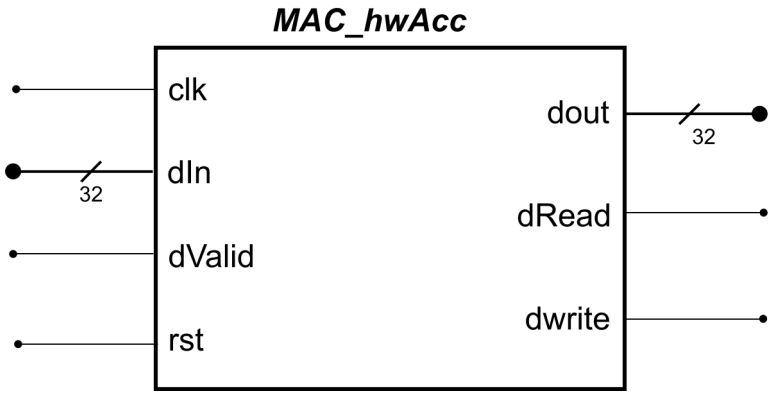

Figure 4.6: Dedicated circuit accelerator for Sobel and YC_BC_R computation

Table 4.3: MAC_hwacc input signals

Name	Size	Description
clk	1 bit	Clock input signal
dIn	32 bits	Input data for computation
dValid	1 bit	Signals if data on dIn is valid (active high)
rst	1 bit	Component reset signal (active high)

Table 4.4: MAC_hwacc output signals

Name	Size	Description
dout	32 bits	Output data signal of computed results
dRead	1 bit	Determines if dIn has be latched (active high)
dwrite	1 bit	Determines if data on dout is valid (ative high)

The organization of internal components for the hybrid Sobel mask DVC is depicted in Figure 4.7 and a more detailed Microblaze structure is found in appendix A.

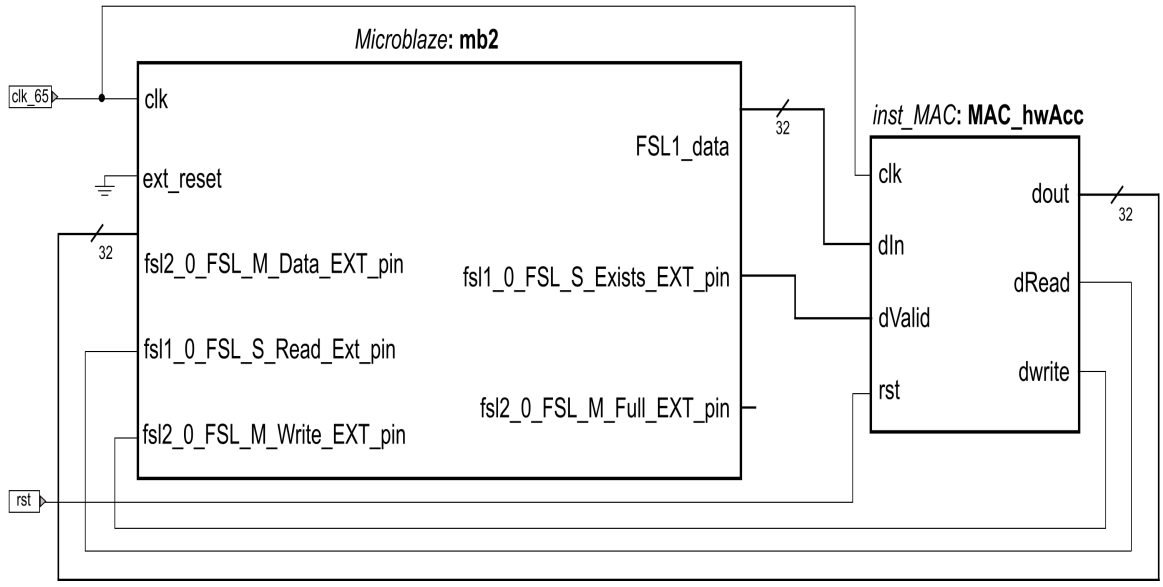

Figure 4.7: Sobel mask hybrid DVC internal organization

The organization of internal components for the Software Sobel mask DVC is depicted in Figure 4.8 and a more detailed Microblaze structure is found in appendix A.

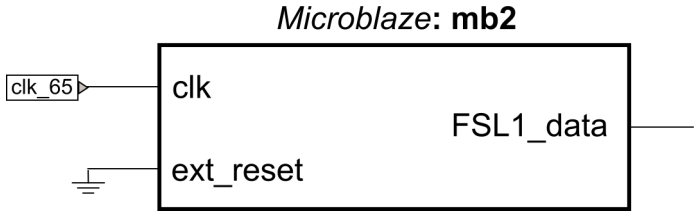

Figure 4.8: Sobel mask software DVC internal organization

4.3.7 Collected Experimental Data From Sobel Mask DVCs

Table 4.5 shows the observed experimental execution times and CEF calculation for the three types of DVC considered.

Table 4.5: Sobel mask experimental data

DVC Type	Execution Time	Development Time	Development Cost
Dedicated	20.67 μs	80 hr	$2762.40
Hybrid	556 μs	30 hr	$1045.32
Software	484 μs	6 hr	$216.60

4.4 Stream Based Experiment Test

4.4.1 RGB to YC_BC_R Conversion Description

In image processing there are many colour schemes to work with and many times it is necessary to convert from one colour scheme to another depending on the application. In this experiment our goal was to determine the proper computing architecture for an RGB to YC_BC_R converter DVC.

4.4.2 RGB to YC_BC_R Principles

Today, there are many hardware and software colour imaging systems and there are millions of colours to manage and organize in some way so that systems can exchange image data correctly. Colour images are made up of colour pixels where each pixel colour is determined by the system using a particular colour model such as the RGB(Red, Green, Blue) colour model [24]. The RGB colour model assumes that an image is composed of three distinct images; one red, one green and one blue image. When combined these three images produce the resulting RGB colour image.

In the YC_BC_R encoding model the colour image is composed of three image components: the luma, blue-difference and red difference components. The luma component, Y$'$, carries the (Achromatic) brightness information of the image. The blue difference (C_B) and red

difference (C_R) components carry the colour information of the image. To convert an RGB image into its YC_BC_R components Equations 4.4, 4.5, and 4.6 [25] can be used for an 8 bit pixel depth digital image.

$$Y' = 16 + \frac{65.738 \times R'_D}{256} + \frac{129.057 \times G'_D}{256} + \frac{129.057 \times B'_D}{256} \tag{4.4}$$

$$C_B = 128 + \frac{37.945 \times R'_D}{256} - \frac{74.494 \times G'_D}{256} + \frac{112.439 * B'_D}{256} \tag{4.5}$$

$$C_R = 128 + \frac{112.439 \times R'_D}{256} - \frac{94.154 \times G'_D}{256} + \frac{18.285 \times B'_D}{256} \tag{4.6}$$

The terms R'_D, G'_D and B'_D refer to the red, green, and blue pixel values of the image respectively.

4.4.3 RGB to YC_BC_R Experimental Implementation

A randomly generated 32x32 RGB matrix was initially generated. Then the matrix was converted to its YC_BC_R component values using equations 4.4, 4.5, 4.6. The execution time is taken as system time and obtained in the same manner as in section 4.3.3.

4.4.4 RGB to YC_BC_R DVC Symbol

Figure 4.9 shows a block diagram of the RGB to YC_BC_R DVC.

4.4.5 RGB to YC_BC_R DVC I/O Signals

Tables 4.6 and 4.7 explains the input and output signals of the Sobel Mask DVC.

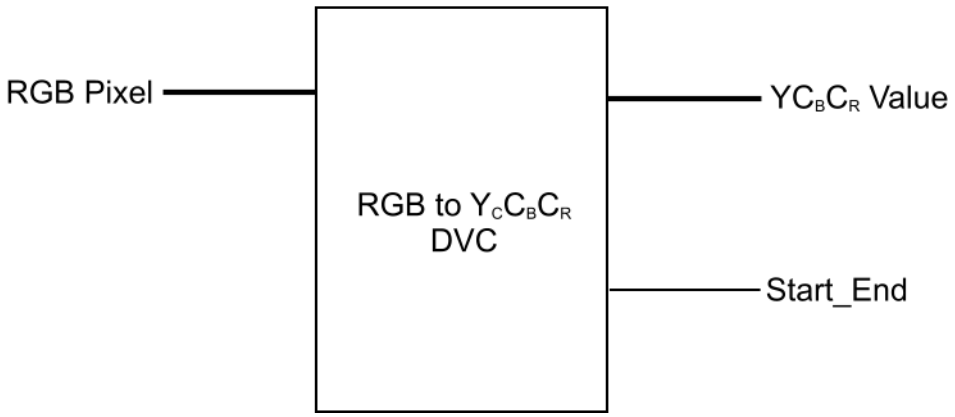

Figure 4.9: Sobel Mask DVC symbol

Table 4.6: Input Signals

Name	**Size**	**Description**
RGB Image	32x32 pixels	32x32 pixel RGB image to be convert into YC_BC_R values

Table 4.7: Output Signals

Name	**Size**	**Description**
YC_BC_R Image	32x32 pixels	The converted 32x32 pixel YC_BC_R image.
Start_End	1 bit	This signal is used to determine the start and end of the RGB to YC_BC_R conversion task on a 32x32 pixel image

4.4.6 RGB to YC_BC_R DVC Organization

The organization of components within the RGB to YC_BC_R dedicated DVC was previously designed in the lab and depicted in Figure 4.6. The YC_BC_R accelerator is the same component however configured differently before sending the data for computation. For example, the accelerator is configured to expect six or three sets of values depending if the task is a Sobel or YC_BC_R operation, respectively.

The organization of internal components for the RGB to YC_BC_R hybrid DVC is depicted in Figure 4.7 with the accelerator component configured for YC_BC_R operations. Similarly, the organization of internal components for the software RGB to YC_BC_R DVC is depicted in Figure 4.8 configured for YC_BC_R operations.

4.4.7 Collected Experimental Data from RGB to YC_BC_R DVCs

Table 4.8 shows the observed experimental execution times, development time and development cost for the three types of DVCs considered for the RGB to YC_BC_R task.

Table 4.8: RGB to YC_BC_R Experimental Data

DVC Type	Execution Time	Development Time	Development Cost
Dedicated	21.66 μs	40 hr	$1381.20
Hybrid	800 μs	30 hr	$1045.32
Software	122 ms	6 hr	$216.60

4.5 Data and Stream Based Experiment

4.5.1 NUI-ARDS DVC

In our third experimental test we apply our methodology to a more complex application to determine the CEF of a DVC that would extend the capabilities of an existing 3D stereo-panoramic vision system depicted earlier in Figure 4.1.

4.5.2 NUI-ARDS DVC Description

The 3D Stereo-Panoramic Acquisition and Display System (3D-SPADS) [26] developed in the Embedded Reconfigurable Systems Lab at Ryerson University displays captured video on three screens. A portion of the captured video displayed in the center screen can be seen in 3D by the human eye. Our goal is to provide a DVC computing architecture for a Natural User Interface Augmented reality Display System (NUI_ARDS) which could extend the systems capability by allowing 3D virtual objects to be superimposed on the center screen. Our main objective is to determine the cost effectiveness of the computing architecture using our proposed analysis. The added capability would allow the 3D-SPADS to be used in applications such as telepresence or telemedicine by allowing a human operator to remotely control, plan, and manipulate a robotic agents movement or action. Figure 4.10 illustrates an example of how the NUI_ARDS could work.

The architecture design selected needs to also meet the following functional and technical specifications.

Functional Specifications

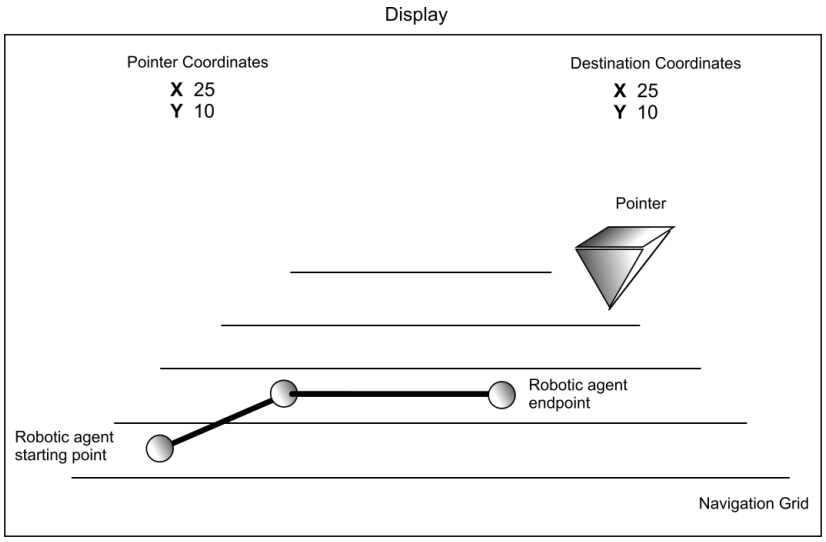

Figure 4.10: Example of NUI_ARDS visual display capability

1. System should be able to inject virtual objects into the captured video data.

2. Virtual objects can be 3D or 2D such as pop-up menus, and 3D pointer.

3. Virtual objects follow the movement of the human operators interface device.

4. The size and shape of virtual objects should react to movement of the human operator interface device.

5. Allow human operator to select robotic agents motion path.

Technical Specifications

1. Virtual objects should be superimposed on captured stereo video data from XVGA camera 1 and VGA camera 2.

2. The entire display is divided into 3 screens, left screen, center screen, and right screen. The left screen displays the captured left peripheral video data, the right screen displays the right peripheral video data. The center screen displays both the left and right video

data simultaneously for the left and right eye respectively to provide a stereoscopic display.

3. Each virtual object displayed is no more than 64x64 pixels in size.

4. System should be a loadable and unloadable DVC on the MARS platform.

5. The position and tracking of the human operator interface device is with respect to the xyz Cartesian co-ordinate system in Figure 4.11:

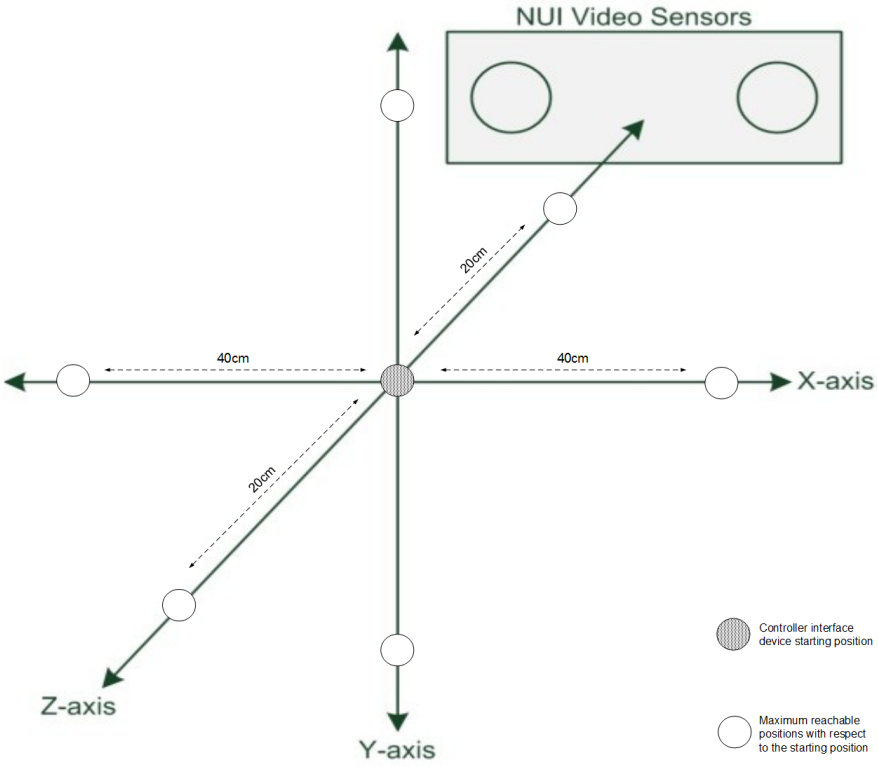

Figure 4.11: Controller interface motion capture design

In this design situation a dedicated DVC was eliminated from consideration because the human-computer interaction with the system is vastly slower than the processing speed of the 3D-SPADS system. For example, the required 60 FPS with a screen resolution of 1024x768 means that the system has 9.82 frames to process a single character input by

a human operator if a keyboard control device was used at a speed of 367 characters per second. With this system the user uses hand motion and selection of points which is even slower. With a user randomly selecting any 10 points at 0.45 point selections per second the system has even more time for processing. However, this system needs to draw 2D and 3D images and then output the appropriate pixel. Adding special dedicated hardware for the drawing of images is possible but can also be done by the processor at a slower rate without affecting the user interaction with the system. In this situation, a hybrid DVC is a good computing architecture candidate.

4.5.3 NUI-ARDS DVC Principles

Graphical display systems such as computer graphics cards usually have a large frame buffers to store entire display image where the system can make any change before output to the screen. This type of approach requires large amounts of memory and in our case a 1024x768 display resolution with a 24 bit RGB colour depth would require 2.25 MB of memory. For an embedded system with minimal memory available 2.25 MB would take up almost all the memory available on the Virtex 4 [27] which would also be required by other components. In this case, we developed a different approach for displaying virtual graphical images superimposed on live video stream which lead us to use a bitmap approach. To superimpose virtual objects onto the live video stream the display screen is divided into 32x32 pixel sections which we refer to as grid_blocks. In each grid_block a virtual object can be injected into the display. Figure 4.12 depicts how the display screen is divided into grid_blocks and how the virtual image to be superimposed onto the live video stream is stored in memory.

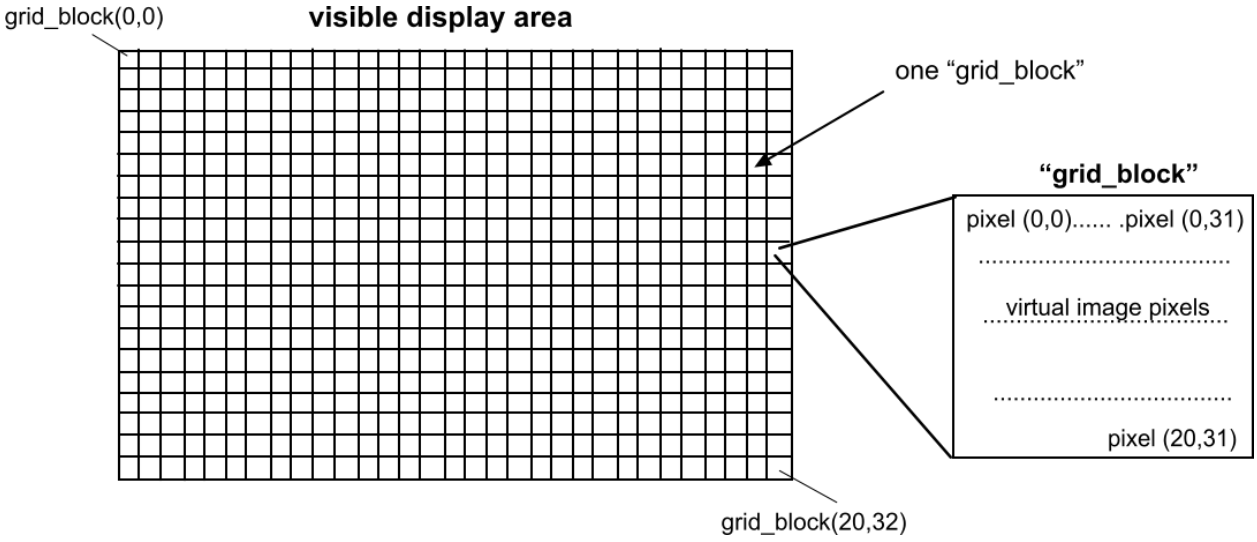

Figure 4.12: Visual display divided into grid_blocks

Each virtual image pixel colour information is encoded into six bits. One bit is used for transparency, two bits for red channel, three bits for green channel, and two bits for blue channel. The six bits are decoded using the method in tables 4.13 to 4.16 in order to get the actual eight bit RGB virtual image colour channel data output on the display.

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	8bit Red channel value
0	0	x	x	x	0	0
0	1	x	x	x	0	42
1	0	x	x	x	0	84
1	1	x	x	x	0	126
0	0	x	x	x	1	0 (transparency)
0	1	x	x	x	1	171
1	0	x	x	x	1	213
1	1	x	x	x	1	255

Figure 4.13: Six bit encoding scheme for (8 bit) red channel

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	8bit Green channel value
x	x	0	0	x	0	0
x	x	0	1	x	0	42
x	x	1	0	x	0	84
x	x	1	1	x	0	126
x	x	0	0	x	1	0 (transparency)
x	x	0	1	x	1	171
x	x	1	0	x	1	213
x	x	1	1	x	1	255

Figure 4.14: Six bit encoding scheme for (8 bit) green channel

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	8bit Blue channel value
x	x	x	x	0	0	0
x	x	x	x	1	0	64
x	x	x	x	0	1	0 (transparency)
x	x	x	x	1	1	255

Figure 4.15: Six bit encoding scheme for (8 bit) blue channel

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RGB values
0	0	0	0	0	1	RGB from cameras used

Figure 4.16: Six bit encoding scheme for pixel transparency

In order to superimpose 3D virtual images there must be an offset between the left and right eye virtual images presented to the human operator to account for disparity. Disparity is the slight difference between two images when captured by image capturing devices such as the human eyes or in our case the front two cameras of the 3D-SPADS. Without any disparity virtual images will be visible in 2D. In order to provide such disparity we consider having two video display layers. One layer contains the visible video stream display area and the other layer contains the virtual video layer area which consists of the virtual images. Figure 4.17 depicts how the virtual video layer can be moved vertically or horizontally in order to calibrate the disparity for displaying 3D virtual images properly.

In order to superimpose virtual images onto the live video stream there needs to be a component that will determine whether or not the pixel to be displayed should come from a virtual image object or from the live video stream. This component can also be either dedicated, software, or hybrid based. Mixing and combining the three types of computing architectures for each component leads to the design of a dedicated, software or hybrid computing system. For example, if the drawing of virtual images into memory is performed by software and the display part is done by dedicated hardware then the computing system is a hybrid DVC. If the drawing of virtual images were to be dedicated hardware as well then the result would be a dedicated DVC.

Complex virtual 3D images that are to be superimposed onto the live video stream can

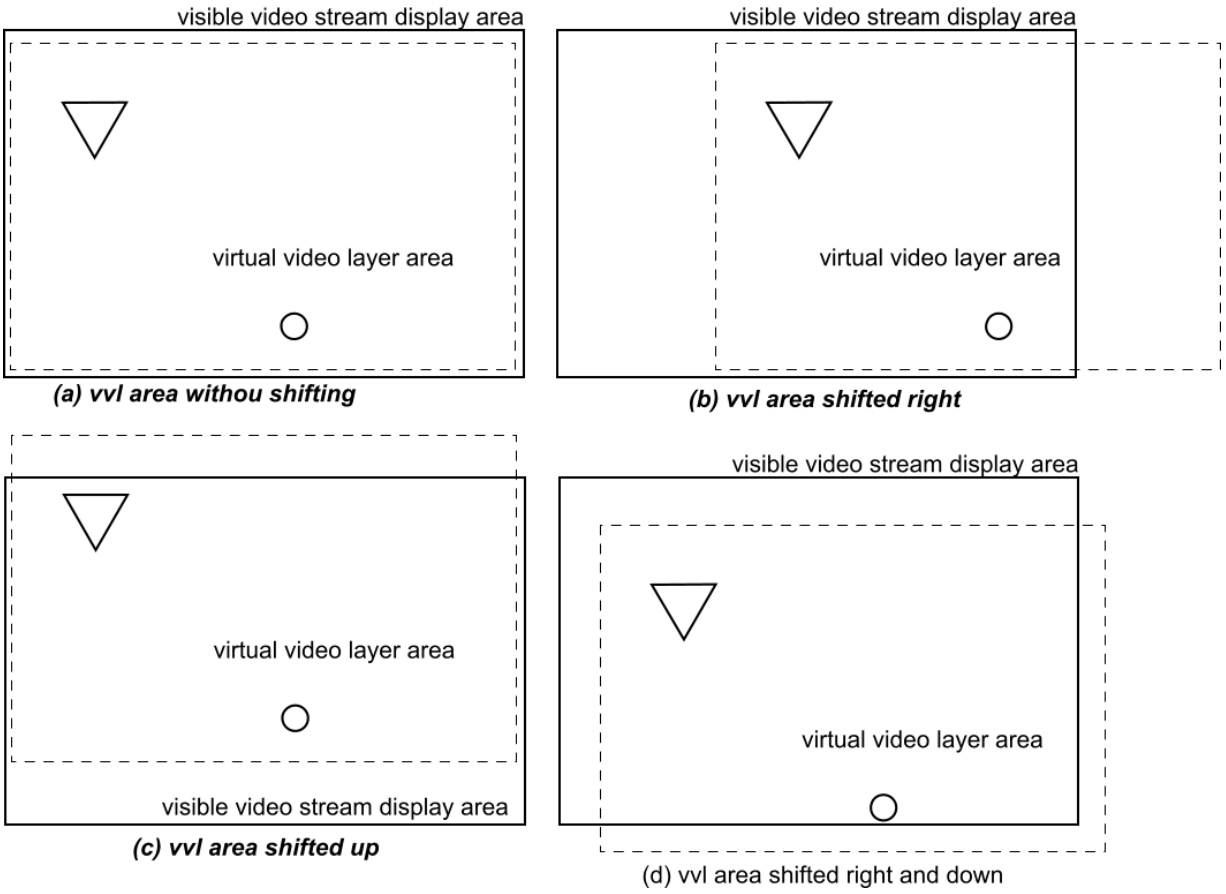

Figure 4.17: Virtual video layer (vvl) shifting

be broken down into smaller components such as point, lines, and colour gradients. The construction of virtual 3D images falls into three categories pertaining to the field of 3D computer graphics; 3D modeling, layout & animation, and rendering [28, 29]. 3D modeling deals with the method or process involved in the creation of the 3D image. For example, a 3D pyramid can be built by drawing a point at the corners of the pyramid and then drawing lines between the points in order create the pyramid shape. Layout & animation deal with the mechanism as to how the objects are positioned in the scene, how they interact with each other, and how their size and shape may change over time. Rendering is when the 3D object model is displayed on the screen with simulated lighting that affects the visibility aspect of the 3D object displayed. There are many complex mechanics to achieve rending such as ray tracing [30], ambient occlusion [31], and index [32]. For the purpose of this study we will us a more simplified approach by rendering 3D objects using our understanding of visual 3D effects and shading as depicted in Figure 4.18.

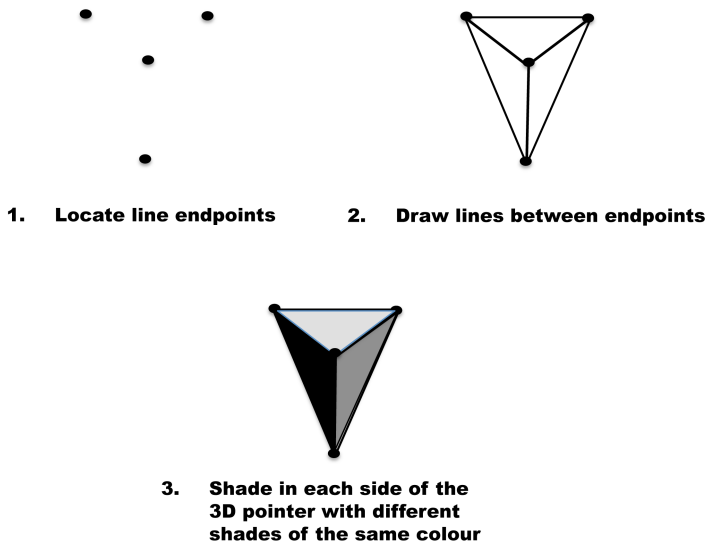

Figure 4.18: Construction of a virtual 3D pointer using primitive shapes (points, lines) and shading

4.5.4 NUI-ARDS DVC Implementation

The design path taken was to divide the tasks into a sequential system and into a Parallel system in order to design a hybrid DVC. Since it is not necessary to draw images very fast and drawing various virtual image shapes was necessary, the component responsible for drawing virtual objects into memory can be a microprocessor or a processor with an accelerator that can specifically draw a particular object. However, from our Sobel mask execution results that show that software architecture is more appropriate for data centric tasks we chose to have the software processor perform data centric tasks such as drawing images into memory. For the software portion of the design a Xilinx's Microblaze soft-core processor can be used. A microprocessor is a sequential processing system and many different images can be produced by using programming algorithms such as Bresenham's line and circle algorithms [33] [31].

On the other hand, the YC_BC_R experiment showed it is reasonable to have dedicated hardware for streaming tasks such as displaying streaming video and virtual images simultaneously. In our design the injection of virtual images and the video controller component are responsible for superimposing virtual images onto the video data stream maintaining the 60 FPS specification of the system. Therefore, the components that perform the display aspect of our task were implemented as the dedicated hardware portion of the NUI-ARDS hybrid design.

The NUI-ARDS system was divided into components so that only the components needed for the current active mode can be loaded and unloaded into reconfigurable slots. The MARS platform contains a sub component called the RCS Video Processor which may contain the NUI-ARDS DVC as an on-chip plug-gable device by means of partial reconfiguration. Within

the NUI-ARDS DVC lies the NUI_AR_COMPONENT sub component responsible for the creation of virtual objects which can then be superimposed on the video stream. Figure 4.19 depicts how the NUI-ARDS is a loadable DVC of the RCS Video Processor running on the MARS platform or as loadable DVC on its own. Figure 4.20 shows how the NUI-ARDS forms part of the RCS Processor.

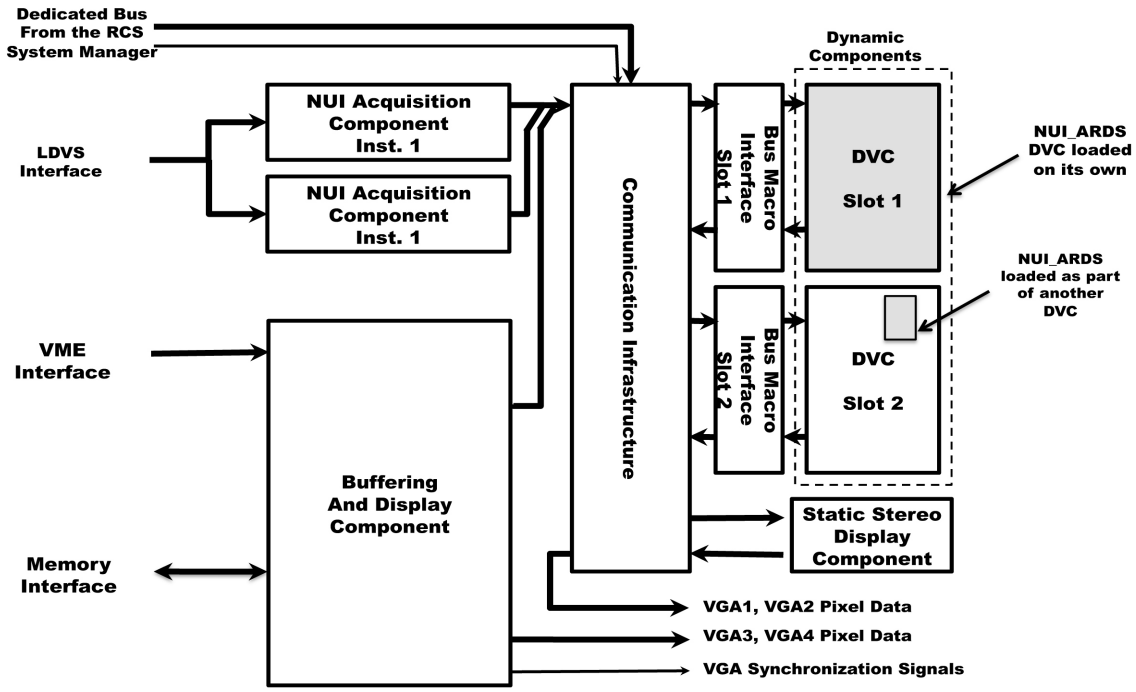

Figure 4.19: MARS Platform with NUI-ARDS and RCS PRocessor

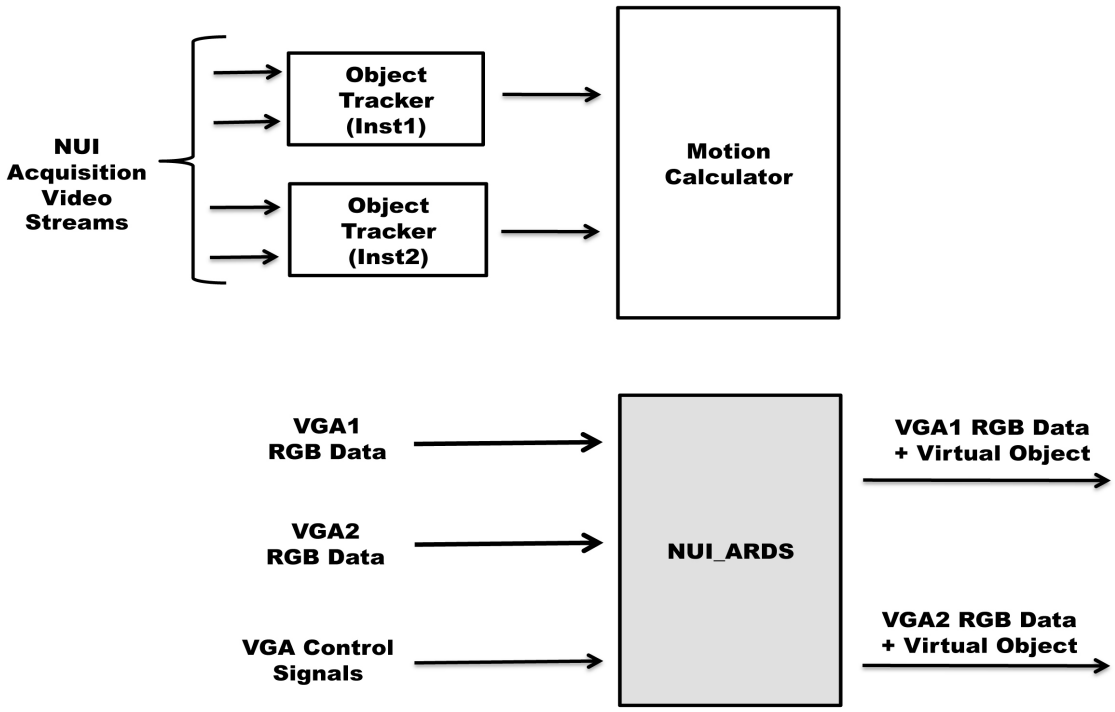

Figure 4.20: NUI-ARDS as part of the RCS Processor Internal Components

4.5.5 NUI-ARDS DVC Symbol

Figure 4.21 shows a block diagram of the NUI-ARDS DVC.

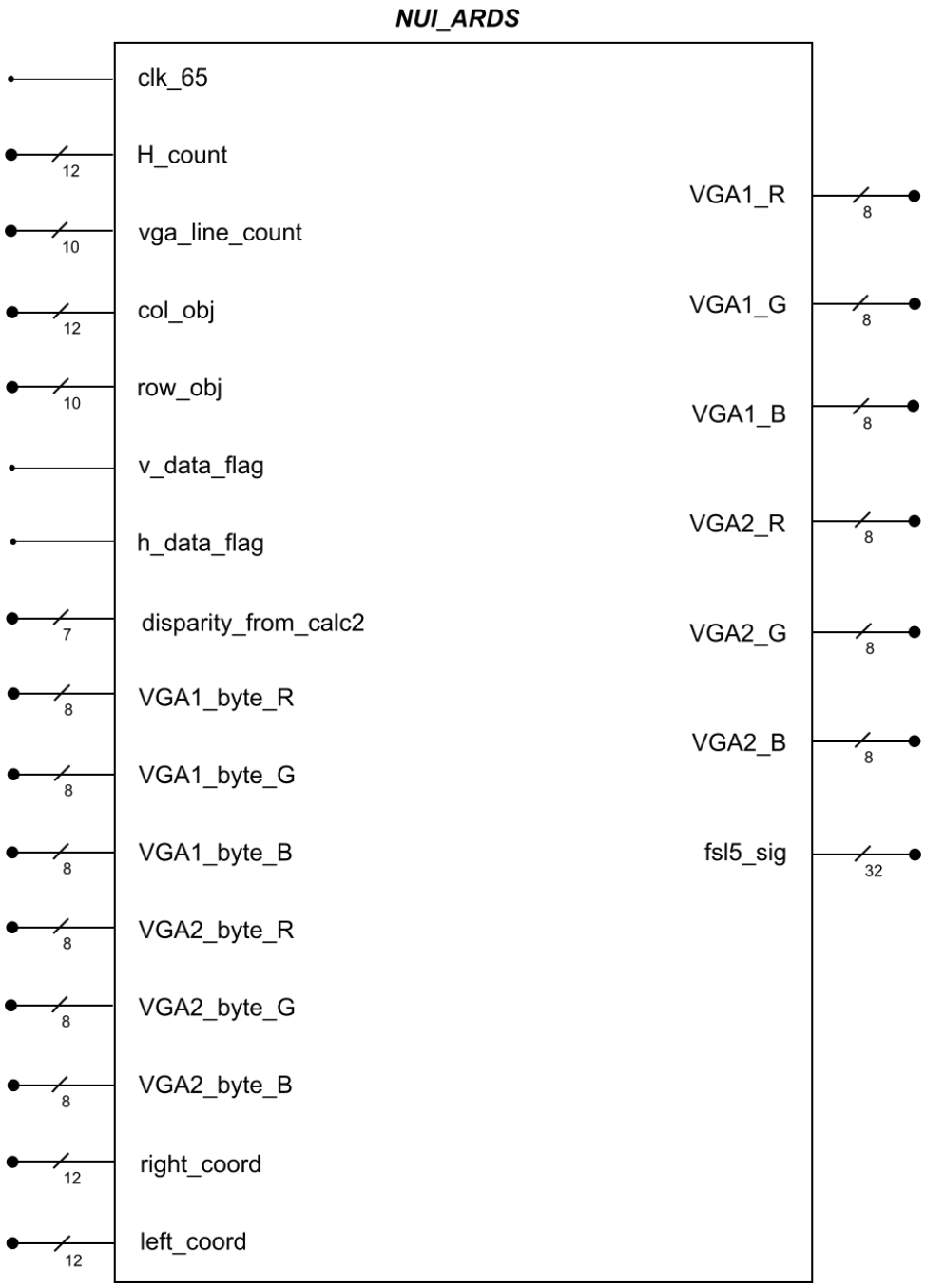

Figure 4.21: The NUI-ARDS DVC

4.5.6 NUI-ARDS DVC I/O Signals

The description of the input and output signals for the NUI-ARDS can be found in Appendix B.

4.5.7 NUI-ARDS DVC Organization

The NUI-ARDS DVC component is responsible for manipulating the video stream data obtained from stereo-cameras. A 65 MHz clock frequency is used for the projector device horizontal and vertical sync signals and is also used by the NUI-ARDS DVC. In order to display and manipulate the video stream data correctly the NUI-ARDS DVC requires information about display signals and human operator input signals. The human operator input signals is provided by an infrared LED controller with a button so that the human operator can point and select the desired position using the button. The display signals are provide by the buffering and display component and the human operator signals are provided by the motion calculator [26]. Within the NUI-ARDS there are memory modules based on BRAMs which the soft-core processor (Microblaze) can use to perform its drawing task. The images stored in memory by Microblaze are displayed when the horizontal and vertical counts of the projector device and pointer device are equal or if that area of the screen has been selected by the user as a valid destination for the robotic agent. At the end of each frame the image memories are checked for any new content which will need to be displayed next.

The Internal organization schematic of NUI-ARDS can be found in documentation [34]. The NUI-ARDS is composed of the following components: NUI_AR_COMPONENT, Two

VO_Injector2, Two VGA_Controller, Two vv_counters, two vvl_shifter, Two vo_maps, Thirty-one img32x32, One img32x32memctlr, Two mux4to1, and Eight mux8to1.

The NUI_AR_COMPONENT is responsible for providing the left and right image of a virtual object (stored in BRAMs) to a pair of VO_Injector components. The VO_Injector component is responsible for determining whether the pixel data from BRAM or the pixel data from the VGA camera should be passed on to the VGA_Controller. The VGA_Controller component is responsible for combining the video data obtained from a camera and the virtual object stored in BRAM. For stereoscopic vision, two VGA_Controllers are used; one for the left eye projector display device and the other for the right eye projector display device. The vvl_shifter component is responsible for shifting the virtual video layer left and right or up and down to provide the proper video disparity in order to correctly produce virtual 3D objects. The vo_map component is memory reserved for storing the type of virtual object that needs to be displayed at the current grid_block. The img32x32 component is block memory used for storing the pixel data of a particular virtual image. The mux4to1 component is used to select which memory bank of img32x32 components is to be used for display. The mux8to1 component is used to select which of the eight img32x32 components in the selected memory bank is to be used for display. The Microblaze component, mb1, is found inside the NUI_AR_COMPONENT component and it's internal organization can be found in documentation [34]. The flow chart in Figure 4.22 shows the program execution flow on the mb1 processor component.

For the purpose of this thesis only a few parts of the design have been explained, however a full component description and their principles of operation can be found in the documentation report [34].

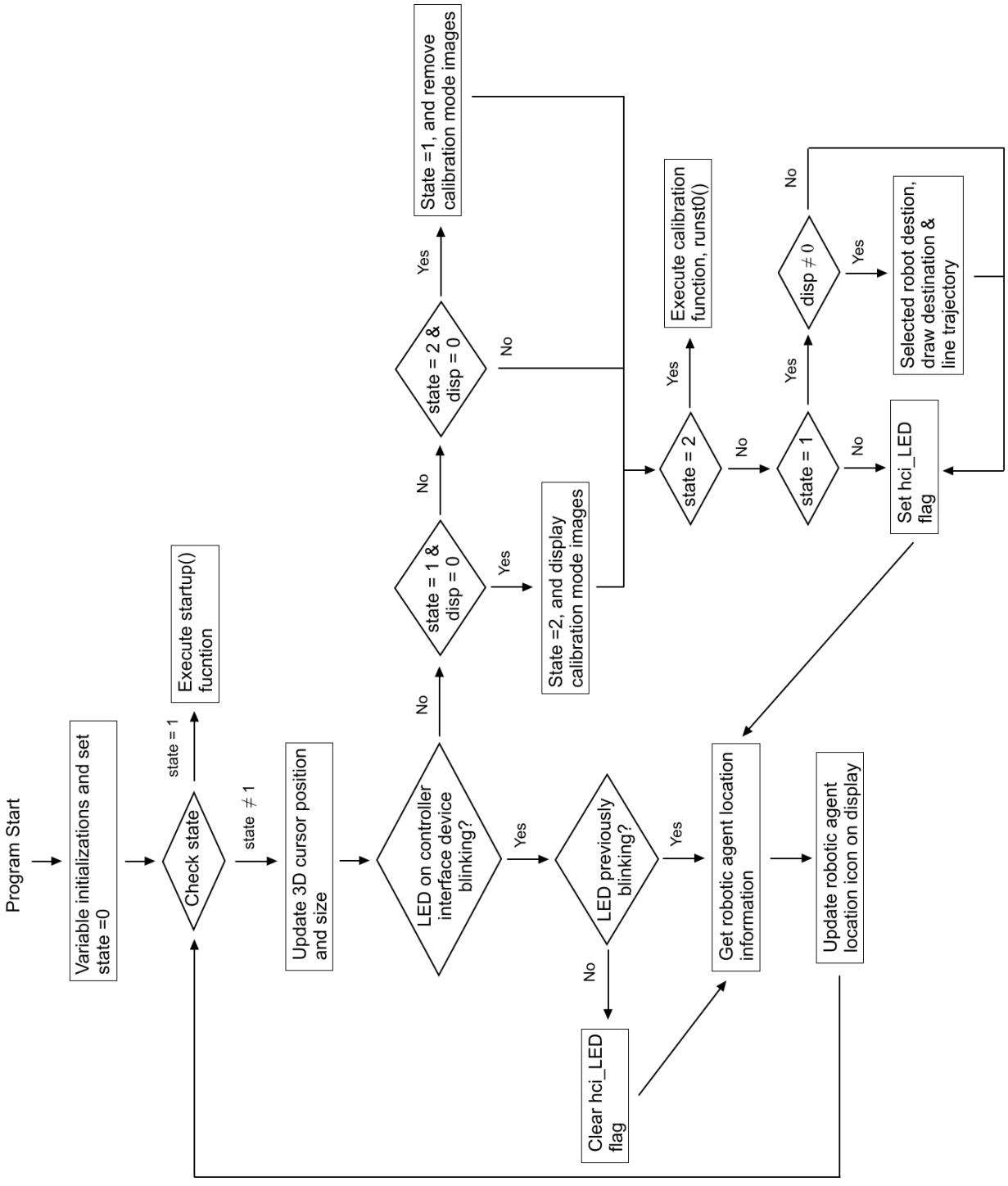

Figure 4.22: Software execution flow chart for the mb1 Microblaze component

4.6 Summary

In this chapter we focused on the implementation of the NUI-ARDS Hybrid DVC and the implementation and gathering of experimental data for three Sobel mask and and three RGB to YC_BC_R DVCs. In the following chapter 5 we focus on the experimental data analysis from the experiments performed in this chapter and discuss our findings.

5 CEF Analysis and Discussion

5.1 Introduction

In this chapter we analyze the experimental data presented in chapter 4 and discuss our findings. The calculated CEF is used to determine the most appropriate DVC computing architecture design for the two experiment applications presented in chapter 4

5.2 CEF Analysis Explanation

In our analysis we take into consideration that the DVC components were designed by an averaged skilled engineer with knowledge of both software and hardware for SoC design. From this assumption we used an average wage of $34.19 per hour for the average cost (AC) value and calculation of development cost (DC) for the given DVC. The average wage was determined by taking the average wage of software and hardware engineers from Table 3.1 in Chapter 3. To determine the cost of the silicon required by a given DVC can be a complicated task since the cost depends the resource granularity level you are looking at. The cost of silicon area or resource taken up by a DVC can be quantified in many ways such as determining the number of flip-flop, number of configurable logic blocks, number of Ram Bit, or even number of look-up-tables(LUTs). Each of these resources also have a different cost associated with them. For example, a RAM16 memory resource may be more expensive than using a LUT. Therefore, a DVC with large amounts of RAM16 compared to a DVC with few RAM16 resources, and if all other resources are equal, would have a higher cost. To simplify this calculation, we chose to assign unit cost (UC) to the price of an FPGA device

with equivalent resources to accommodate the given DVC. Table 5.1 shows the Spartan 3E FPGA prices selected for the use of UC for the Sobel mask and YC_BC_R experiment analysis.

Table 5.1: Selected UC equivalent resource Spartan FPGA chip cost

FPGA Chip	CLB Count	Price
XC3S1200E-4FGG320C	3688	$67.56
XC3S250E-4VQG100C	612	$14.50
XC3S100E-4VQG100C	240	$10.51

5.3 Analysis of Data Centric Experiment Test

5.3.1 Sobel Mask Experiment Analysis

In section 4.3.7 we obtained the execution time for the computation of a Sobel Mask task and its resulting silicon area required. Since the dedicated DVC requires 65 CLBs we took the UC for the dedicated implementation to be $10.51. Similarly, we took the $67.56 as the UC for both the Software and Hybrid implementation. Although the Software DVC takes up about half the resources when compared to the Hybrid DVC there was no other FPGA pricing within the same family that had a similar amount of CLBs at the time of analysis. Using this data we were then able to calculate the CEF for each Sobel mask DVC. Figure 5.1 shows the CEF results in a graph format while Tables C.1, C.2 and C.3 in appendix C.1 show the detailed calculations performed.

The calculated development cost (DC) for each Sobel Mask DVC is listed in Table 5.2.

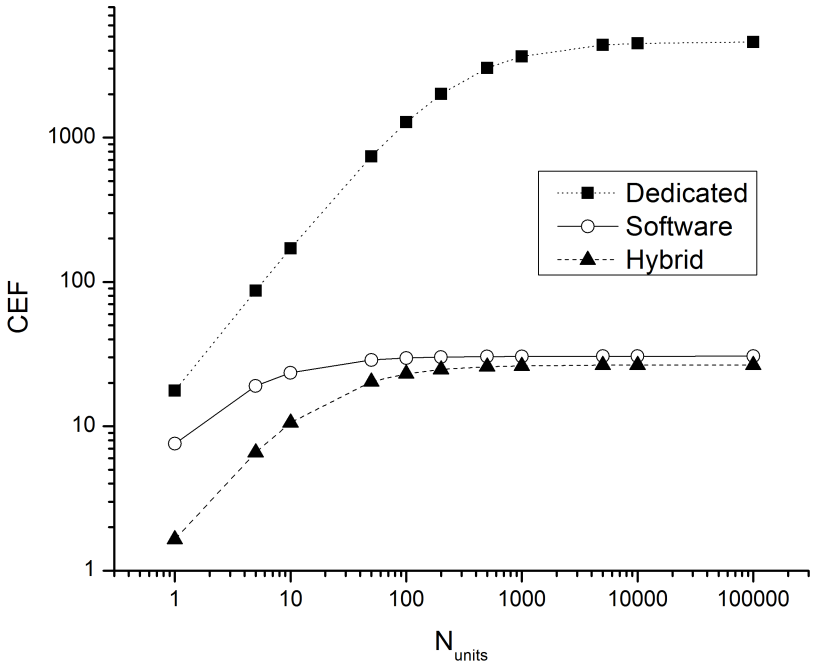

Figure 5.1: CEF for Sobel mask DVCs

Table 5.2: Sobel mask DVC development cost

DVC Architecture	**Development Cost (DC)**
Dedicated	$2735.20
Software	$205.14
Hybrid	$1025.70

5.3.2 Sobel Mask Experiment Discussion

From Figure 5.1 we can see that the most cost efficient design is the dedicated DVC followed by the software DVC and then the hybrid DVC. Ideally, when an (dedicated circuit) accelerator is added to a design it is expected that the execution time for the same task is decreased; thus increasing performance. However, it is noticed in Table 4.5 that the performance for the hybrid DVC is lower than the software DVC. In order to determine the reason for this observation, we went back to analyze the design and noticed that although the software DVC and hybrid DVC had similar software code when it comes the time to optimize the code for both the software and hybrid DVC, that the optimized software DVC code had fewer instructions compared to that of the Hybrid. The extra instructions present in the optimized code of the Hybrid DVC could not be removed because they are configuration signals needed to control the Sobel mask computing accelerator. This means that the performance increase obtained by the accelerator for the purpose of computing the gradient was overshadowed by the extra time required for proper control signals between the processor and the dedicated circuit. For example, in order to send data out of the processor a 32 bit Fast Simplix Link (FSL) was used. From the processors point of view these are control and data signals bunched up for the dedicated circuit. In order to send out the control signals and bit shifting to align the data and control signals in the proper order was necessary. The extra time required by these instructions appears to be greater than the gain in performance given by the accelerator. In addition to a lower performance, the hybrid DVC also had 5 times the development cost which further contributed to its lower CEF value compared to that of the Software DVC for this particular task. The dedicated DVC development cost was twice expensive than the

hybrid; however, it gave a performance increase 10 times greater which reduced the effect of having a large development cost even when only small quantities were to be produced.

5.4 Analysis of Stream Centric Experiment Test

5.4.1 YC_BC_R Converter Experiment Analysis

In section 4.4.7 we obtained the execution time for the RGB to YC_BC_R task and its resulting silicon area required. Since the dedicated DVC requires 281 CLBs we took the UC for the dedicated implementation to be $14.50. Similarly, we took the $67.56 as the UC for both the Software and Hybrid implementation. As in the Sobel Mask experiment, although the Software DVC took up about half the resources when compared to the Hybrid DVC there was no other FPGA pricing within the same family that had a similar amount of CLBs at the time of analysis. Using this data we were then able to calculate the CEF for each YC_BC_R DVC. Figure 5.2 shows the CEF results in a graph format while Tables C.4, C.5, and C.6 in appendix C.1 shows the detailed calculations performed.

The calculated development cost (DC) for each RGB to YC_BC_R DVC is listed in Table 5.3.

Table 5.3: RGB to YC_BC_R DVC Development Cost

YC_BC_R DVC	**Development Cost (DC)**
Dedicated	$1367.60
Software	$205.14
Hybrid	$1025.70

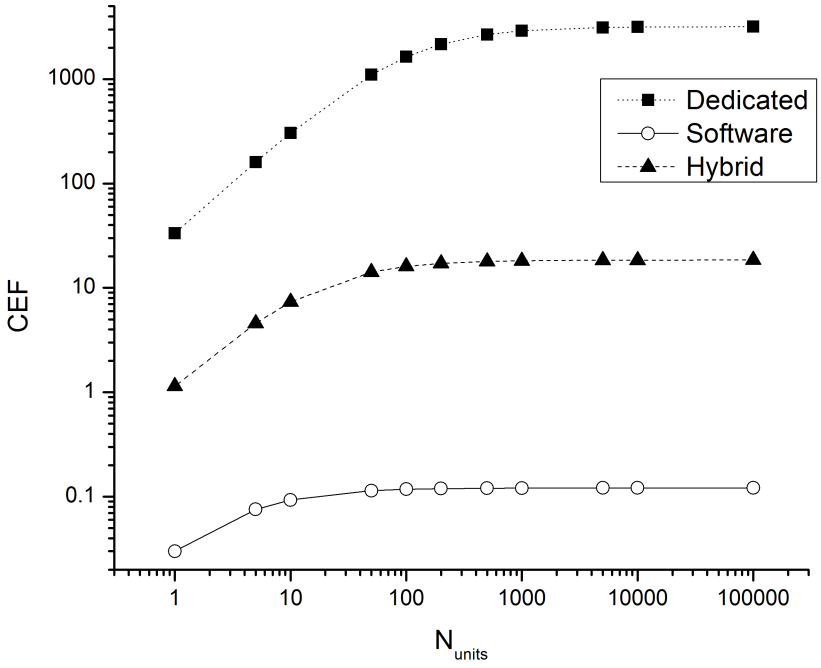

Figure 5.2: CEF for YC_BC_R DVCs

5.4.2 YC_BC_R Experiment Discussion

From Figure 5.2 we can see that the most cost efficient design is the dedicated DVC followed by the hybrid and then software DVC respectively. The execution time for the hybrid in this situation was faster than the software by 10 times; however, its development cost was 5 times higher. This means that the performance increase obtained by the accelerator was more influential than the development cost associated with the increased performance in the determination of the CEF. The dedicated hardware, once again, had the best CEF because its performance was far greater than its development cost. However, that does not mean the only choice to consider should be the dedicated DVC. The dedicated DVC is clearly the most cost efficient but sometime its large speed increase may not be necessary and the

hybrid or software DVC performance could more than enough to meet the performance specification. Another reason not to select a dedicated DVC is simply because of its time to market. The development time to market is important to consider, especially in competitive markets where cost efficiency is also important. A longer development time extends the time to market which means other competitors that may have gone with the option of a hybrid or software DVC (with shorter development time) instead but still meet the performance requirements would have an advantage by receiving extra revenue from hitting the market first.

5.5 Hybrid Software Code Optimization CEF

Since companies are always looking to be the first to release their product to the market, it is possible for them to produce more than one design that may or may not be optimized either software or hardware wise. However, the results show that the code optimization can play a role and affect the performance of a hybrid and software DVC. This lack of optimization is also commonly seen when companies are developing prototypes or when companies just want to test the market. Sometimes a product may be released to market knowing that its software may not be optimized; however, getting it out into the market is better and once in the market slowly optimize components such as software can be accomplished by releasing new versions or new models of the product. For this reason we decided to take a look and see how a lack of optimization in software may affect our decision to select a DVC. Tables 5.4 and 5.5 show the execution time of each DVC for both the Sobel mask computation and RGB to YC_BC_R tasks.

Table 5.4: Un-Optimized code execution time for Sobel mask DVC

Sobel Mask DVC	**Execution time**
Hardware	20.67 μs
Software	6.29 ms
Hybrid	4.8 ms

Table 5.5: Un-Optimized code execution time for RGB to YC_BC_R DVC

YC_BC_R DVC	**Execution time**
Hardware	21.66 μs
Software	149.0 ms
Hybrid	3.4 ms

From the Table 5.4 we observed that the execution time for the Sobel mask DVC is now faster for a dedicated DVC than for a software DVC which is the effect of both designs not being optimized to their full potential. On the other hand, Table 5.5 shows that the DVC optimization did not affect which DVC was faster than the other. To see what effect this had on the CEF results we recalculated the CEF for both Sobel mask and the RGB to YC_BC_R task for analysis. Tables C.7, C.8 and C.9 in appendix C.2 shows the detailed CEF calculations for the Sobel mask task using non-optimized software code. Figure 5.3 shows the resulting CEF graph.

Tables C.10, C.11 and C.12 in appendix C.2 show the detailed CEF calculations for the RGB to YC_BC_R task using non optimized software code. Figure 5.4 shows the resulting CEF graph.

The software DVC for the Sobel mask task was 1.15 times faster then the hybrid DVC for the optimized software CEF results shown in Figure 5.4. Figure 5.4 also depicts that the software Sobel mask DVC maintains its cost efficient above the hybrid DVC no matter how

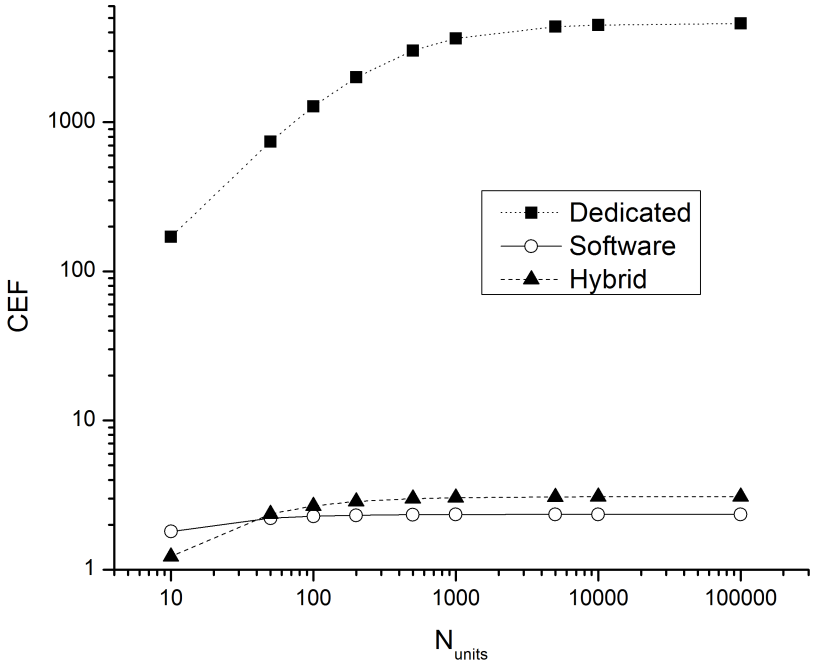

Figure 5.3: CEF for YC_BC_R DVCs

many units are produced. From this result one might expect a similar outcome with the CEF calculations for non optimized software DVCs; however, this is not the case. Figure 5.3 shows that even though the Sobel mask hybrid DVC is 1.31 times faster than the software Sobel mask DVC it did not keep its CEF value higher than that of the Sobel mask software DVC throughout all product production levels. From Table C.8 we see that for production quantities of 50 units and up the CEF for the hybrid DVC is higher then that of the software DVC, hence a hybrid DVC approach is the most cost effective. However, for quantities lower than 50 units the CEF is less than that of software DVC which means that a software DVC would be more cost effective in this case. This analysis is important because it shows how the development cost is linked to cost efficiency and that it should not be left out of consideration. By comparing the DC(Ai)/Nunits values from Table C.8 and Table C.9 we noticed that

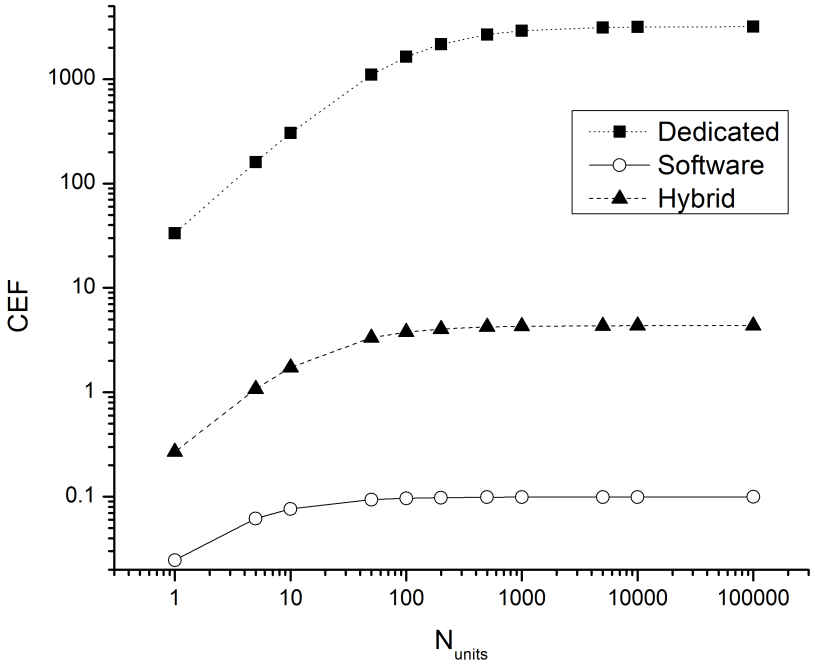

Figure 5.4: CEF for YC_BC_R DVCs

although the Sobel mask hybrid DVC has a higher CEF when it reaches its maximum CEF. The rate at which the maximum CEF is reached is related to the rate at which the cost per unit term DC(Ai)/Nunits reaches zero. Since the Hybrid DVC development cost was 5 times higher than that of the software Sobel mask DVC, the rate at which the cost per unit reached zero was slower for the Sobel hybrid DVC compared to that of the Sobel software DVC.

Figure 5.5 shows how in the case of the Sobel mask the CEF curves for the hybrid and software DVCs intersect at some point while the CPR curves never intersect even though the number of units is low. It also demonstrates how the CPR fails to correctly select the most cost effeicent computing architecture when the number of units to be produced is low.

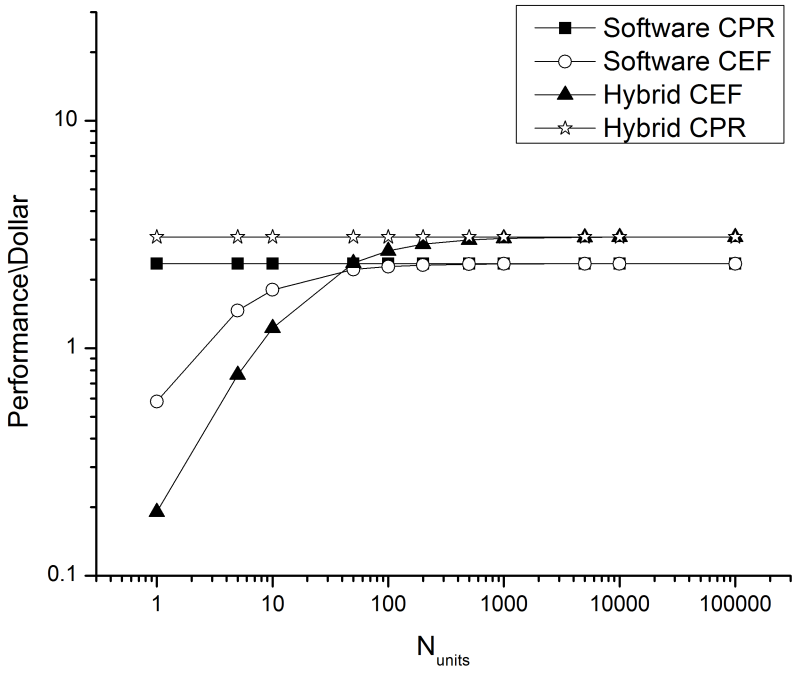

Figure 5.5: A CEF and CPR comparison of Sobel DVCs

5.6 NUI_ARDs CEF Vs. CPR Analysis

In section 4.5 we implemented a NUI-ARDS hybrid DVC. The main purpose of this experiment is to go through the steps one would take in order to calculate a CEF for a more complex hybrid DVC. By Using the development cost data from our implementation process the CEF for the particular NUI-ARDS hybrid DVC was determined. Figure 5.6 shows the CEF analysis for the NUI-ARDS hybrid DVC compared to a CPR analysis.

When comparing both the CEF and CPR curves in Figure 5.6 both lines eventually level off at the same value but their rising rate level is different. The CPR reaches the maximum cost performance ratio from the very beginning, however, the CEF levels off gradually as number of units to be produced increases. When the number of units to be produced is

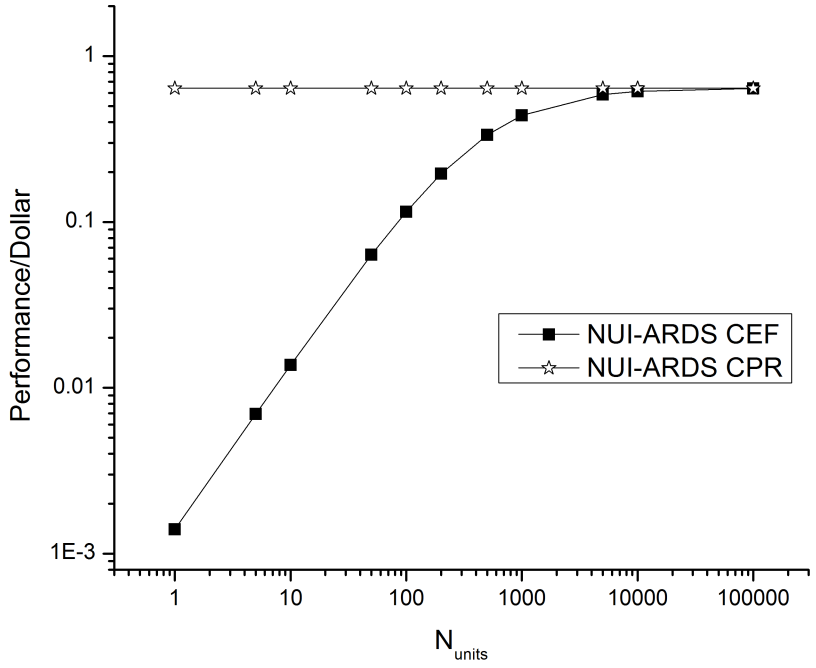

Figure 5.6: A CEF and CPR comparison of a Hybrid NUI-ARDS DVC

significantly large the CPR becomes a special case of the CEF where development cost per unit is zero. This demonstrates how the CPR is appropriate to use only in cases where large number of DVCs are to be produced. However, when only a few DVCs are to be produced a CEF should be used to properly correct for development costs.

5.7 Summary

After analyzing and determining the CEF of various DVCs we determined the most cost effective DVC for the applications of Sobel mask computation and RGB to YC_BC_R conversion. Under optimized software code conditions the software Sobel mask DVC had the best CEF for any number of units produced. Similarly, the hybrid RGB to YC_BC_R DVC has the best

CEF as well. Under non-optimized code conditions however the best CEF DVC for Sobel mask depended on the number of units to be produced. At lower production units the Sobel mask software DVC was the better choice. However, at higher levels of unit productions the Sobel Hybrid DVC was the better choice. For NUI-ARDS hybrid component design the CEF was calculated for various production levels. As production units of the NUI-ARDS hybrid DVC incremented its CEF increased by 55% from 50 and 100 units in production as per Table C.13.

6 Conclusion and Future Work

In Chapter 2 we looked into previous work analyses involving computing architectures. Specifically, we looked for cost effective analysis and found that previous work did not research much about the cost efficiency of a dynamic reconfigurable system in terms of monetary cost. Various cost analyses pertaining to the cost of silicon area, cost of power, cost of performance, and cost of specific resources were encountered. However, not much attention was paid to the monetary cost of developing a computing system. The current way of analyzing the cost effectiveness of a DVC computer architecture is determined in terms of monetary value for the Cost Performance Ratio (CPR) where performance is divided by price. This CPR evaluation mechanism is a valid and valuable tool if the assumption that large quantities of units will be produced is true. However, if this assumption is not correct and low production quantities are desired, such as in custom electronic design markets, then the CPR may fail to give a more realistic solution. In Chapter 1 we expressed our motivation of finding an adequate method in determining the cost effectiveness of a DVC for partially reconfigurable computing systems. Then in chapter 3 a potential analysis method based on a cost efficiency factor that took into account the development cost and production quantity of a computing system was proposed. Three Experimental tests listed in section 4 were then introduced to explore the usefulness of the CEF. The first experiment involved a data centric task in the form of Sobel mask computation application. The second experiment involved a stream based application in the form of an RGB to YC_BC_R converter. The last experiment involved the design of a complex system containing both data and stream based application characteristics and determined the resulting cost efficiency of the design. Because such a

design is quite large only the CEF for the hybrid component design was calculated as an example for the application of the method in more complex systems. The data from all the experiments was then recorded and analyzed. The results from the Sobel mask and YC_BC_R experiments showed that the development cost of a computing architecture indeed affects its cost efficiency. However, the effect it has on the CEF at different levels of unit production depends on various factors such as how much the performance difference is between two architectures, or how much performance increase is achieved versus the extra hours put into the design. Without some form of mechanism to compare the changes in these factors it becomes difficult to properly select the computing architecture for a DVC. However, with the introduction of the CEF all these factors are taken into account giving us a tool for determining the most cost efficient computing architecture type of a DVC for any quantity of units to be produced. In order to systematically select the computing architecture of a DVC for a DPRCS the following steps can be followed:

1. Determine task to be performed by DVC.

2. Determine or estimate number of DVC to be deployed.

3. Determine or estimate development time in hours, cost per hour, unit cost, and performance for dedicated DVC.

4. Determine or estimate development time in hours, cost per hour, unit cost, and performance for Software DVC.

5. Determine or estimate development time in hours, cost per hour, unit cost, and performance for Hybrid DVC.

6. Calculate the CEF for each DVC.

7. Select the DVC with the highest CEF for the determined task in step 1.

Moving forward these steps could potentially be used by a component which automatically selects which DVC to load into a reconfigurable slot on a DPRCS based on their CEF. Future work can also involve determining the most cost effective computer graphics generating DVC. A dedicated and software version of the NUI-ARDS component could be implemented and then compared to the CEF of the hybrid NUI-ARDS. The possibility of refining and determining the UC term in the CEF factor analysis can be further explored as well. FPGA chips have many different configurable resources each with potentially different costs associated with it. Determining the most appropriate level of granularity for the calculation of the UC would be useful. Further more, since a Hybrid DVC has both a software and dedicated circuit portions the CEF could benefit from splitting the CEF calculations into two portions reflecting a more refine way of calculating the CEF for a hybrid DVC.

A Microblaze mb2 Component Internal Organization

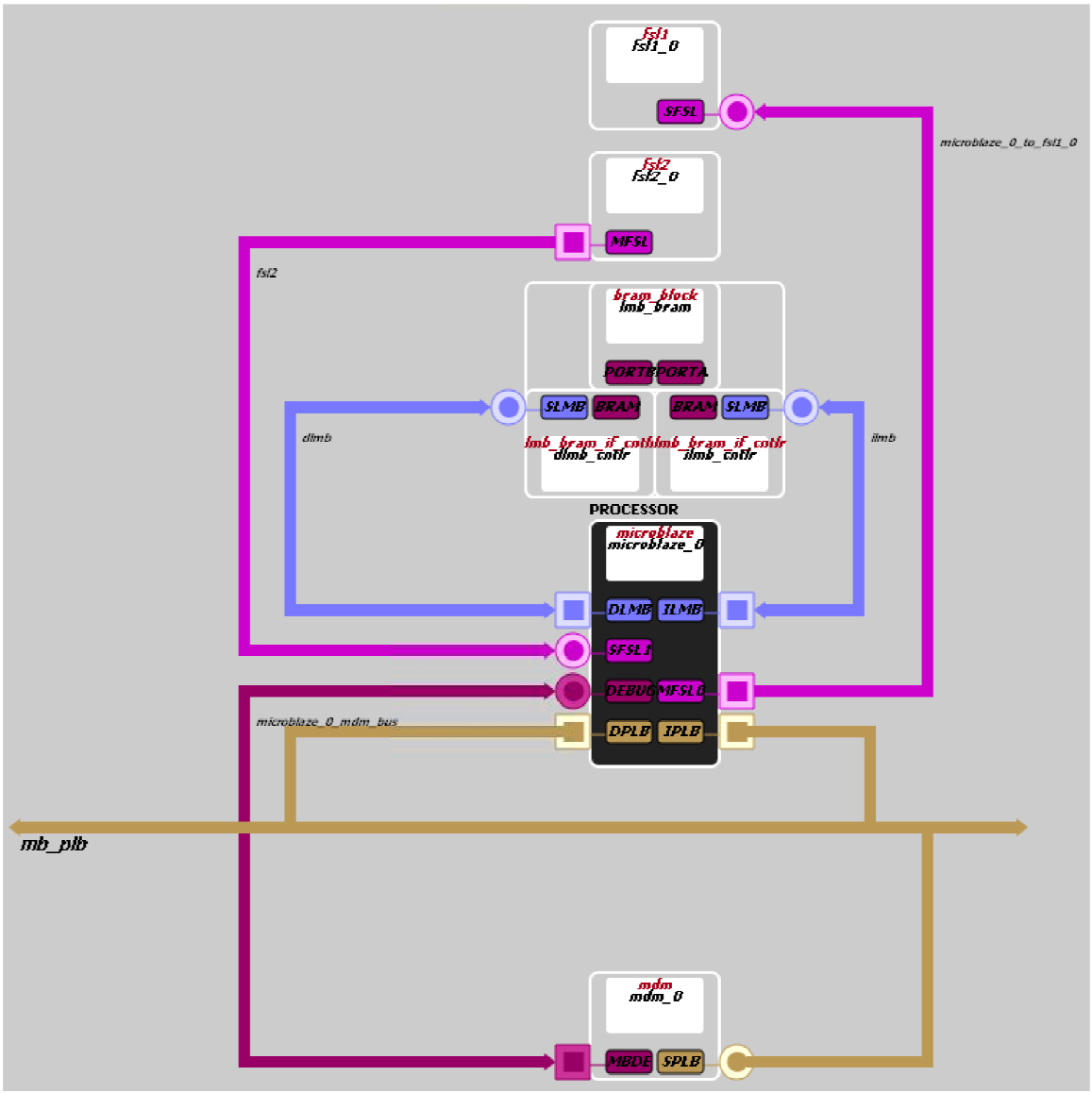

Figure A.1: Microblaze mb2 internal component organization for for Sobel mask and YC_BC_R Hybrid DVC Experiments

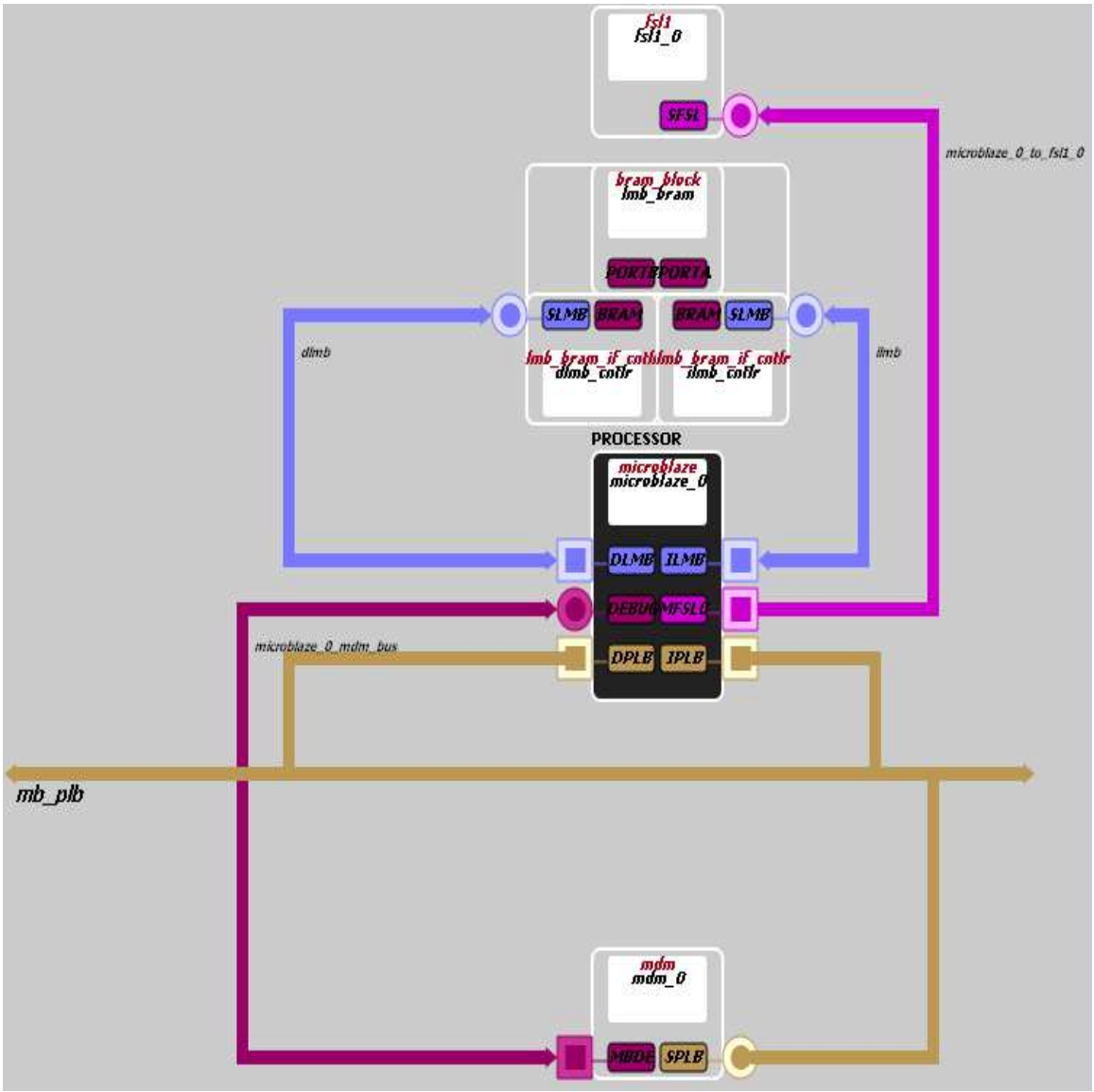

Figure A.2: mb2 Microblaze mb2 internal component organization for Sobel mask and YC_BC_R Software DVC Experiments

B NUI-ARDS Component I/O Signals

Table B.1: Input signals required by NUI-ARDS

Name	Size	Description
Clk_65	1 bit	This is the 65MHz clock that will be used by the "NUI-ARDS" as the clock frequency for some internal clocked components such as Block Rams (BRAMs).
H_count	1 bit	This signal provides the "NUI-ARDS" with the current horizontal display count. This count is used by the "NUI-ARDS" components to determine when the horizontal display count is aligned with the user input device horizontal count ("col_obj").
vga_line_count	10 bits	This signal provides the "NUI-ARDS" with the current vertical display count. This count is used by the "NUI-ARDS" components to determine when the vertical display count is aligned with the user input device vertical count ("row_obj").
h_data_flag	1 bit	This is the horizontal data flag signal. It is used by the "NUI-ARDS" components to determine if the horizontal count ("H_count") is within the display area visible to the human operator.
v_data_flag	1 bit	This is the vertical data flag signal. It is used by the "NUI-ARDS" components to determine if the vertical count ("vga_line_count") is within the display area visible to the human operator.
disparity_from_calc2	7 bits	This is the disparity signal obtained by the disparity computation component. It is used by "Microblaze" software to properly display a virtual objects size and shape.
VGA1_byte_R	8 bits	This is the 8bit red pixel information obtained from the buffering and display component for vga camera 1.
VGA1_byte_G	8 bits	This is the 8bit green pixel information obtained from the buffering and display component for vga camera 1.
VGA1_byte_B	8 bits	This is the 8bit blue pixel information obtained from the buffering and display component for vga camera 1.
VGA2_byte_R	8 bits	This is the 8bit red pixel information obtained from the buffering and display component for vga camera 2.
VGA2_byte_G	8 bits	This is the 8bit green pixel information obtained from the buffering and display component for vga camera 2.
VGA2_byte_B	8 bits	This is the 8bit blue pixel information obtained from the buffering and display component for vga camera 2.
col_obj	12 bits	This signal is the horizontal count location of the human operator's pointer device. This signal is used by the "NUI-ARDS" components to determine if the horizontal count of the pointer device is aligned with the projector horizontal display count ("H_count").
row_obj	10 bits	This signal is the vertical count location of the human operator's pointer device. This signal is used by the "NUI-ARDS"

Table B.2: Output signals produced by NUI-ARDS

Name	Size	Description
VGA1_R	8 bits	This is the (8 bit) red colour component pixel data to be sent to the display 1 device.
VGA1_G	8 bits	This is the (8 bit) green colour component pixel data to be sent to the display 1 device.
VGA1_B	8 bits	This is the (8bit) blue colour component pixel data to be sent to the display 1 device.
VGA2_R	8 bits	This is the (8 bit) red colour component pixel data to be sent to the display 2 device.
VGA2_G	8 bits	This is the (8 bit) green colour component pixel data to be sent to the display 2 device.
VGA2_B	8 bits	This is the (8bit) blue colour component pixel data to be sent to the display 2 device.
fsl5_sig	31bits	This is the new robotic agent destination coordinate selected by the human operator. Bits 0 to 5 contain the x coordinate and bits 6 to 11 contain the y coordinate.

C CEF Calculations

C.1 Optimized Code

Table C.1: Sobel Dedicated DVC CEF Calculations

Ai = Dedicated Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	80	34.19	2735.20	2735.2	10.51	0.00002067	48379.29	17.62	4603.17
5	80	34.19	2735.20	547.04	10.51	0.00002067	48379.29	86.77	4603.17
10	80	34.19	2735.20	273.52	10.51	0.00002067	48379.29	170.33	4603.17
50	80	34.19	2735.20	54.704	10.51	0.00002067	48379.29	741.85	4603.17
100	80	34.19	2735.20	27.352	10.51	0.00002067	48379.29	1277.78	4603.17
200	80	34.19	2735.20	13.676	10.51	0.00002067	48379.29	2000.30	4603.17
500	80	34.19	2735.20	5.4704	10.51	0.00002067	48379.29	3027.41	4603.17
1000	80	34.19	2735.20	2.7352	10.51	0.00002067	48379.29	3652.59	4603.17
5000	80	34.19	2735.20	0.54704	10.51	0.00002067	48379.29	4375.43	4603.17
10000	80	34.19	2735.20	0.27352	10.51	0.00002067	48379.29	4486.41	4603.17
100000	80	34.19	2735.20	0.027352	10.51	0.00002067	48379.29	4591.22	4603.17

Table C.2: Sobel Software DVC CEF Calculations

A_i = Software Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	6	34.19	205.14	205.14	67.56	0.000484	2066.12	7.58	30.58
5	6	34.19	205.14	41.03	67.56	0.000484	2066.12	19.03	30.58
10	6	34.19	205.14	20.51	67.56	0.000484	2066.12	23.46	30.58
50	6	34.19	205.14	4.10	67.56	0.000484	2066.12	28.83	30.58
100	6	34.19	205.14	2.05	67.56	0.000484	2066.12	29.68	30.58
200	6	34.19	205.14	1.03	67.56	0.000484	2066.12	30.12	30.58
500	6	34.19	205.14	0.41	67.56	0.000484	2066.12	30.40	30.58
1000	6	34.19	205.14	0.21	67.56	0.000484	2066.12	30.49	30.58
5000	6	34.19	205.14	0.04	67.56	0.000484	2066.12	30.56	30.58
10000	6	34.19	205.14	0.02	67.56	0.000484	2066.12	30.57	30.58
100000	6	34.19	205.14	0.00	67.56	0.000484	2066.12	30.58	30.58

Table C.3: Sobel Hybrid DVC CEF Calculations

A_i = Hybrid Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	30	34.19	1025.70	1025.70	67.56	0.000556	1798.56	1.65	26.62
5	30	34.19	1025.70	205.14	67.56	0.000556	1798.56	6.60	26.62
10	30	34.19	1025.70	102.57	67.56	0.000556	1798.56	10.57	26.62
50	30	34.19	1025.70	20.51	67.56	0.000556	1798.56	20.42	26.62
100	30	34.19	1025.70	10.26	67.56	0.000556	1798.56	23.11	26.62
200	30	34.19	1025.70	5.13	67.56	0.000556	1798.56	24.74	26.62
500	30	34.19	1025.70	2.05	67.56	0.000556	1798.56	25.84	26.62
1000	30	34.19	1025.70	1.03	67.56	0.000556	1798.56	26.22	26.62
5000	30	34.19	1025.70	0.21	67.56	0.000556	1798.56	26.54	26.62
10000	30	34.19	1025.70	0.10	67.56	0.000556	1798.56	26.58	26.62
100000	30	34.19	1025.70	0.01	67.56	0.000556	1798.56	26.62	26.62

Table C.4: YC_BC_R Dedicated DVC CEF Calculations

A_i = Dedicated RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	40	34.19	1367.60	1367.6	14.50	0.00002166	46168.05	33.40	3184.00
5	40	34.19	1367.60	273.52	14.50	0.00002166	46168.05	160.29	3184.00
10	40	34.19	1367.60	136.76	14.50	0.00002166	46168.05	305.22	3184.00
50	40	34.19	1367.60	27.352	14.50	0.00002166	46168.05	1103.13	3184.00
100	40	34.19	1367.60	13.676	14.50	0.00002166	46168.05	1638.56	3184.00
200	40	34.19	1367.60	6.838	14.50	0.00002166	46168.05	2163.65	3184.00
500	40	34.19	1367.60	2.7352	14.50	0.00002166	46168.05	2678.71	3184.00
1000	40	34.19	1367.60	1.3676	14.50	0.00002166	46168.05	2909.58	3184.00
5000	40	34.19	1367.60	0.27352	14.50	0.00002166	46168.05	3125.05	3184.00
10000	40	34.19	1367.60	0.13676	14.50	0.00002166	46168.05	3154.25	3184.00
100000	40	34.19	1367.60	0.013676	14.50	0.00002166	46168.05	3181.00	3184.00

Table C.5: YC_BC_R Software DVC CEF Calculations

A_i = Software RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	6	34.19	205.14	205.14	67.56	0.122	8.20	0.03	0.12
5	6	34.19	205.14	41.028	67.56	0.122	8.20	0.08	0.12
10	6	34.19	205.14	20.514	67.56	0.122	8.20	0.09	0.12
50	6	34.19	205.14	4.1028	67.56	0.122	8.20	0.11	0.12
100	6	34.19	205.14	2.0514	67.56	0.122	8.20	0.12	0.12
200	6	34.19	205.14	1.0257	67.56	0.122	8.20	0.12	0.12
500	6	34.19	205.14	0.41028	67.56	0.122	8.20	0.12	0.12
1000	6	34.19	205.14	0.20514	67.56	0.122	8.20	0.12	0.12
5000	6	34.19	205.14	0.041028	67.56	0.122	8.20	0.12	0.12
10000	6	34.19	205.14	0.020514	67.56	0.122	8.20	0.12	0.12
100000	6	34.19	205.14	0.0020514	67.56	0.122	8.20	0.12	0.12

Table C.6: YC_BC_R Hybrid DVC CEF Calculations

A_i = Hybrid RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	30	34.19	1025.70	1025.7	67.56	0.0008	1250.00	1.14	18.50
5	30	34.19	1025.70	205.14	67.56	0.0008	1250.00	4.58	18.50
10	30	34.19	1025.70	102.57	67.56	0.0008	1250.00	7.35	18.50
50	30	34.19	1025.70	20.514	67.56	0.0008	1250.00	14.19	18.50
100	30	34.19	1025.70	10.257	67.56	0.0008	1250.00	16.06	18.50
200	30	34.19	1025.70	5.1285	67.56	0.0008	1250.00	17.20	18.50
500	30	34.19	1025.70	2.0514	67.56	0.0008	1250.00	17.96	18.50
1000	30	34.19	1025.70	1.0257	67.56	0.0008	1250.00	18.23	18.50
5000	30	34.19	1025.70	0.20514	67.56	0.0008	1250.00	18.45	18.50
10000	30	34.19	1025.70	0.10257	67.56	0.0008	1250.00	18.47	18.50
100000	30	34.19	1025.70	0.010257	67.56	0.0008	1250.00	18.50	18.50

C.2 Non-Optimized Code

Table C.7: Sobel dedicated DVC CEF calculations (Non-Optimized Code)

A_i = Dedicated Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	80	34.19	2735.20	2735.2	10.51	0.00002067	48379.29	17.62	4603.17
5	80	34.19	2735.20	547.04	10.51	0.00002067	48379.29	86.77	4603.17
10	80	34.19	2735.20	273.52	10.51	0.00002067	48379.29	170.33	4603.17
50	80	34.19	2735.20	54.704	10.51	0.00002067	48379.29	741.85	4603.17
100	80	34.19	2735.20	27.352	10.51	0.00002067	48379.29	1277.78	4603.17
200	80	34.19	2735.20	13.676	10.51	0.00002067	48379.29	2000.30	4603.17
500	80	34.19	2735.20	5.4704	10.51	0.00002067	48379.29	3027.41	4603.17
1000	80	34.19	2735.20	2.7352	10.51	0.00002067	48379.29	3652.59	4603.17
5000	80	34.19	2735.20	0.54704	10.51	0.00002067	48379.29	4375.43	4603.17
10000	80	34.19	2735.20	0.27352	10.51	0.00002067	48379.29	4486.41	4603.17
100000	80	34.19	2735.20	0.027352	10.51	0.00002067	48379.29	4591.22	4603.17

Table C.8: Sobel Software DVC CEF Calculations (Non-Optimized Code)

A_i = Software Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	6	34.19	205.14	205.14	67.56	0.00629	158.98	0.58	2.35
5	6	34.19	205.14	41.03	67.56	0.00629	158.98	1.46	2.35
10	6	34.19	205.14	20.51	67.56	0.00629	158.98	1.81	2.35
50	6	34.19	205.14	4.10	67.56	0.00629	158.98	2.22	2.35
100	6	34.19	205.14	2.05	67.56	0.00629	158.98	2.28	2.35
200	6	34.19	205.14	1.03	67.56	0.00629	158.98	2.32	2.35
500	6	34.19	205.14	0.41	67.56	0.00629	158.98	2.34	2.35
1000	6	34.19	205.14	0.21	67.56	0.00629	158.98	2.35	2.35
5000	6	34.19	205.14	0.04	67.56	0.00629	158.98	2.35	2.35
10000	6	34.19	205.14	0.02	67.56	0.00629	158.98	2.35	2.35
100000	6	34.19	205.14	0.00	67.56	0.00629	158.98	2.35	2.35

Table C.9: Sobel Hybrid DVC CEF Calculations (Non-Optimized Code)

A_i = Hybrid Sobel Mask DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 trans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	30	34.19	1025.70	1025.70	67.56	0.0048	208.33	0.19	3.08
5	30	34.19	1025.70	205.14	67.56	0.0048	208.33	0.76	3.08
10	30	34.19	1025.70	102.57	67.56	0.0048	208.33	1.22	3.08
50	30	34.19	1025.70	20.51	67.56	0.0048	208.33	2.37	3.08
100	30	34.19	1025.70	10.26	67.56	0.0048	208.33	2.68	3.08
200	30	34.19	1025.70	5.13	67.56	0.0048	208.33	2.87	3.08
500	30	34.19	1025.70	2.05	67.56	0.0048	208.33	2.99	3.08
1000	30	34.19	1025.70	1.03	67.56	0.0048	208.33	3.04	3.08
5000	30	34.19	1025.70	0.21	67.56	0.0048	208.33	3.07	3.08
10000	30	34.19	1025.70	0.10	67.56	0.0048	208.33	3.08	3.08
100000	30	34.19	1025.70	0.01	67.56	0.0048	208.33	3.08	3.08

Table C.10: YC_BC_R Dedicated DVC CEF Calculations (Non-Optimized Code)

A_i = Dedicated RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	40	34.19	1367.60	1367.6	14.50	0.00002166	46168.05	33.40	3184.00
5	40	34.19	1367.60	273.52	14.50	0.00002166	46168.05	160.29	3184.00
10	40	34.19	1367.60	136.76	14.50	0.00002166	46168.05	305.22	3184.00
50	40	34.19	1367.60	27.352	14.50	0.00002166	46168.05	1103.13	3184.00
100	40	34.19	1367.60	13.676	14.50	0.00002166	46168.05	1638.56	3184.00
200	40	34.19	1367.60	6.838	14.50	0.00002166	46168.05	2163.65	3184.00
500	40	34.19	1367.60	2.7352	14.50	0.00002166	46168.05	2678.71	3184.00
1000	40	34.19	1367.60	1.3676	14.50	0.00002166	46168.05	2909.58	3184.00
5000	40	34.19	1367.60	0.27352	14.50	0.00002166	46168.05	3125.05	3184.00
10000	40	34.19	1367.60	0.13676	14.50	0.00002166	46168.05	3154.25	3184.00
100000	40	34.19	1367.60	0.013676	14.50	0.00002166	46168.05	3181.00	3184.00

Table C.11: YC_BC_R Software DVC CEF Calculations (Non-Optimized Code)

A_i = Software RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	6	34.19	205.14	205.14	67.56	0.149	6.71	0.02	0.10
5	6	34.19	205.14	41.028	67.56	0.149	6.71	0.06	0.10
10	6	34.19	205.14	20.514	67.56	0.149	6.71	0.08	0.10
50	6	34.19	205.14	4.1028	67.56	0.149	6.71	0.09	0.10
100	6	34.19	205.14	2.0514	67.56	0.149	6.71	0.10	0.10
200	6	34.19	205.14	1.0257	67.56	0.149	6.71	0.10	0.10
500	6	34.19	205.14	0.41028	67.56	0.149	6.71	0.10	0.10
1000	6	34.19	205.14	0.20514	67.56	0.149	6.71	0.10	0.10
5000	6	34.19	205.14	0.041028	67.56	0.149	6.71	0.10	0.10
10000	6	34.19	205.14	0.020514	67.56	0.149	6.71	0.10	0.10
100000	6	34.19	205.14	0.0020514	67.56	0.149	6.71	0.10	0.10

Table C.12: YC_BC_R Hybrid DVC CEF Calculations (Non-Optimized Code)

A_i = Hybrid RGB to YC_BC_R DVC									
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	exe_time in Sec	$P(A_i)$ 32x32 tans/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	30	34.19	1025.70	1025.7	67.56	0.0034	294.12	0.27	4.35
5	30	34.19	1025.70	205.14	67.56	0.0034	294.12	1.08	4.35
10	30	34.19	1025.70	102.57	67.56	0.0034	294.12	1.73	4.35
50	30	34.19	1025.70	20.514	67.56	0.0034	294.12	3.34	4.35
100	30	34.19	1025.70	10.257	67.56	0.0034	294.12	3.78	4.35
200	30	34.19	1025.70	5.1285	67.56	0.0034	294.12	4.05	4.35
500	30	34.19	1025.70	2.0514	67.56	0.0034	294.12	4.23	4.35
1000	30	34.19	1025.70	1.0257	67.56	0.0034	294.12	4.29	4.35
5000	30	34.19	1025.70	0.20514	67.56	0.0034	294.12	4.34	4.35
10000	30	34.19	1025.70	0.10257	67.56	0.0034	294.12	4.35	4.35
100000	30	34.19	1025.70	0.010257	67.56	0.0034	294.12	4.35	4.35

C.3 NUI-ARDS CEF

Table C.13: NUI-ARDS Hybrid DVC CEF Calculations

A_i = NUI-ARDS Hybrid DVC								
N_{units}	$DT(A_i)$ in Hours	$AC(A_i)$ in $	$DC(A_i)$ in $	$DC(A_i)/N_{units}$	UC in $	$P(A_i)$FPS/sec	$CEF(A_i)$ in P(Ai)/$	CPR
1	1250	34.19	42737.50	42737.5	93.78	60.00	0.00	0.64
5	1250	34.19	42737.50	8547.5	93.78	60.00	0.01	0.64
10	1250	34.19	42737.50	4273.75	93.78	60.00	0.01	0.64
50	1250	34.19	42737.50	854.75	93.78	60.00	0.06	0.64
100	1250	34.19	42737.50	427.375	93.78	60.00	0.12	0.64
200	1250	34.19	42737.50	213.6875	93.78	60.00	0.20	0.64
500	1250	34.19	42737.50	85.475	93.78	60.00	0.33	0.64
1000	1250	34.19	42737.50	42.7375	93.78	60.00	0.44	0.64
5000	1250	34.19	42737.50	8.5475	93.78	60.00	0.59	0.64
10000	1250	34.19	42737.50	4.27375	93.78	60.00	0.61	0.64
100000	1250	34.19	42737.50	0.427375	93.78	60.00	0.64	0.64

Bibliography

[1] D. Menasce and V. Almeida, "Cost-performance analysis of heterogeneity in supercomputer architectures," in *Supercomputing '90., Proceedings of*, 1990, pp. 169–177.

[2] P.-A. Hsiung, P.-H. Lu, and L. C-W, "Energy efficient co-scheduling in dynamically reconfigurable systems," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, 2007, pp. 87–92.

[3] Z. J. Yang, A. Kumar, and H. Yajun, "An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee," in *Field-Programmable Technology (FPT), 2010 International Conference on*, 2010, pp. 389–392.

[4] E.-B. Bourennane, S. Bouchoux, J. Miteran, M. Paindavoine, and S. Bouillant, "Cost comparison of image rotation implantations on static and dynamic reconfigurable FPGAs," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 3, 2002, pp. III–3176–III–3179.

[5] Xilinx. (2008, Sept.) 7 Series FPGAs Overview. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[6] Altera. (2013, Jun.) The breakthrough advantage for FPGAs with Tri-Gate technology. [Online]. Available: http://www.altera.com/literature/wp/wp-01201-fpga-tri-gate-technology.pdf

[7] D. Diaz, V. Dumitriu, and L. Kirischian, "Cost-performance analysis of component architectural designs for dynamic partially reconfigurable systems," in *Electrical Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, 2012, pp. 1–6.

[8] S. Borgio, D. Bosisio, F. Ferrandi, M. Monchiero, M. Santambrogio, D. Sciuto, and A. Tumeo, "Hardware DWT accelerator for multiprocessor system-on-chip on FPGA," in *Embedded Computer Systems: Architectures, Modeling and Simulation, 2006. IC-SAMOS 2006. International Conference on*, 2006, pp. 107–114.

[9] M. Hubner, C. Tradowsky, D. Gohringer, L. Braun, F. Thoma, J. Henkel, and J. Becker, "Dynamic processor reconfiguration," in *Reconfigurable Computing and FPGAs (ReConFig 2011 International Conference on*, 2011, pp. 123–128.

[10] V. Pranav and J. Lee, "Simulation of hybrid computer architectures: simulators, methodologies and recommendations," in *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, 2007, pp. 157–162.

[11] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli, "Hybrid performance modeling and prediction of large-scale computing systems," in *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, March, pp. 132–138.

[12] M. B. Abdelhalim and S. Habib, "Modeling communication cost and hardware alternatives in PSO based HW/SW partitioning," in *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, Dec., pp. 111–114.

[13] P. Saha and T. El-Ghazawi, "Software/hardware co-scheduling for reconfigurable computing systems," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, April, pp. 299–300.

[14] S. Prakash and A. Parker, "A design method for optimal synthesis of application-specific heterogeneous multiprocessor systems," in *Heterogeneous Processing, 1992. Proceedings. Workshop on*, Mar., pp. 75–80.

[15] D. Menasce and V. Almeida, "Cost-performance analysis of heterogeneity in supercomputer architectures," in *Supercomputing '90., Proceedings of*, Nov., pp. 169–177.

[16] H. Muller, P. Stallard, and D. Warren, "The role of associative memory in virtual shared memory architectures: a price-performance comparison," in *Parallel and Distributed Processing, 1996. PDP '96. Proceedings of the Fourth Euromicro Workshop on*, 1996, pp. 41–49.

[17] E. G. Cale, L. L. Gremillion, and J. L. McKenney, "Price/performance patterns of U. S. computer systems," *Commun. ACM*, vol. 22, no. 4, pp. 225–233, Apr. 1979. [Online]. Available: http://doi.acm.org/10.1145/359094.359097

[18] F. Sijstermans, "The TriMedia processor: the price-performance challenge for media processing," in *Multimedia and Expo, 2001. ICME 2001. IEEE International Conference on*, 2001, pp. 222–225.

[19] S. H. Fuller, "Price/performance comparison of C.mmp and the PDP-10," in *Proceedings of the 3rd annual symposium on Computer architecture*, ser. ISCA '76. New York, NY, USA: ACM, 1976, pp. 195–202. [Online]. Available: http://doi.acm.org/10.1145/800110.803580

[20] S. A. Ryan, A. M. Jones, and R. H. Deaves, "A low-cost SoC architecture for the next-generation home-networked set-top box," in *Consumer Electronics, 2009. ICCE '09. Digest of Technical Papers International Conference on*, 2009, pp. 1–2.

[21] J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach 3rd edition," in *Hennessy, John, and Patterson, David*, 2003, pp. 169–177.

[22] M. Majmudar, C. Docan, M. Parashar, and C. Marty, "Cost vs. performance of VaR on accelerator platforms," in *Proceedings of the 2nd Workshop on High Performance Computational Finance*, ser. WHPCF '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:8. [Online]. Available: http://doi.acm.org/10.1145/1645413.1645422

[23] M. Uno. (2008, Sept) FPGAs Leveling with ASICs, ASSPs. [Online]. Available: http://techon.nikkeibp.co.jp/article/HONSHI/20080924/158406/fig2.jpg

[24] R. C. Gonzalez, *Digital Image Processing: Third Edition.* Upper Saddle River, NJ: Pearson Education Inc., 2008.

[25] E. Prathibha, Y. Siva, and A. Manjunath, "Design and implementation of color conversion module RGB to YCbCr and vice versa," *IJCI International Journal of Computer Science Issues, Special Issue, ICVCI-2011*, vol. 1, no. 1, pp. 13–18, 2011.

[26] D. Marcantonio, V. Dumitriu, and S. Artur, "Stereo-panoramic acquisition and display system," in *Stereo-Panoramic Acquisition and Display System*, 2010.

[27] Xilinx. (2010, Aug) Virtex-4 Family Overview. [Online]. Available: http://www.altera.com/literature/wp/wp-01201-fpga-tri-gate-technology.pdf

[28] J. S. Joon, Y. M. Chan, and K. C. Weng, "A case study of integrating principles of photography and photorealistic for 3D rendering," in *Computer Graphics, Imaging and Visualisation, 2007. CGIV '07*, 2007, pp. 62–70.

[29] M. Kefi, V. Barichard, and P. Richard, "A constraint-solver based tool for user-assisted interactive 3D layout," in *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, vol. 1, 2012, pp. 199–206.

[30] Y. Quan, W.-H. Li, Y.-J. Pang, and G.-J. Liu, "An efficient point rendering system for ray tracing," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 9, 2005, pp. 5429–5431 Vol. 9.

[31] L. Szirmay-Kalos, T. Umenhoffer, B. Toth, L. Szecsi, and M. Casasayas, "Volumetric Ambient Occlusion," in *Volumetric Ambient Occlusion*, vol. PP, no. 99, 2009, pp. 1–1.

[32] B.-S. Liang, Y.-C. Lee, W.-C. Yeh, and C.-W. Jen, "Index rendering: hardware-efficient architecture for 3-D graphics in multimedia system," *Multimedia, IEEE Transactions on*, vol. 4, no. 3, pp. 343–360, 2002.

[33] W. Wright, "Parallelization of Bresenham's line and circle algorithms," *Computer Graphics and Applications, IEEE*, vol. 10, no. 5, pp. 60–67, 1990.

[34] D. "Diaz, ""natural user interface augmented reality display system with robotic agent path selection (NUI-ARDS) documentation"," *Ryerson University ERSL Lab*, pp. 1–235, Dec. 2012.