HARDWARE SOFTWARE CO-SYNTHESIS OF HETEROGENEOUS HYPERCUBE ARCHITECTURES FOR FAULT TOLERANT EMBEDDED SYSTEMS

by

Jacob Levman, BASc, Toronto, September 17th 2004

A thesis

presented to Ryerson University

in partial fulfillment of the

requirement for the degree of

Master of Applied Science

in the program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2004

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Hardware Software Co-Synthesis of Heterogeneous Hypercube Architectures for Fault Tolerant Embedded Systems

Jacob Levman,
Masters of Applied Science, 2004,
Electrical and Computer Engineering,
Ryerson University

Abstract

The hardware-software co-synthesis of an embedded system's architecture involves the partitioning of a system specification into hardware and software modules so as to meet various non-functional requirements. A designer can specify many non-functional requirements including cost, performance, reliability etc. In this thesis, we present an approach to the hardware-software co-synthesis of embedded systems targeting hypercube topologies. Hypercube topologies provide a flexible and reliable architecture for an embedded device with multiple processing elements. To the best of our knowledge, this is the first time that hypercube topologies have been supported in a co-synthesis algorithm. The co-synthesis approach presented here supports the following features: 1) input in the form of an acyclic periodic task graph with real-time constraints, 2) the pipelining of task graphs, 3) the use of a heterogeneous set of processing elements, 4) Support for fault tolerance through our newly developed group based fault tolerance technique. The co-synthesis algorithm has been applied to two case studies to demonstrate its efficacy.

Acknowledgements

The author would like to thank his two supervising professors, Dr. Gul Khan and Dr. Javad Alirezaie, for providing their guidance, knowledge and support. The author would also like to thank the members of the review committee for their participation. The author would like to thank the National Science and Engineering Research Council of Canada (NSERC) for providing funding support for this research project in the form of multiple grants to my supervising professors. The author would like to thank Ontario Graduate Scholarships (OGS) for funding this research through a scholarship. The author would also like to thank Canadian Microelectronics Corporation (CMC) for providing the ARM rapid prototyping platform that was used in the case studies of this thesis.

Dedication

I would like to dedicate this thesis to my wife whose drive to accomplish is inspiring. Without her love and support I would not be where I am today.

Table of Contents

List of Tables

List of Figures

Glossary of Acronyms

CAD - Computer Aided Design

GBFT – Group Based Fault Tolerance

CBFT – Cluster Based Fault Tolerance

TBFT – Task Based Fault Tolerance

CPU – Central Processing Unit

ASIC – Application Specific Integrated Circuit

FPGA – Field Programmable Gate Array

MPEG – Motion Picture Experts Group

PE – Processing Element

IP – Intellectual Property

SW – Software

HW – Hardware

VHDL – Very high speed integrated circuit Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 Overview

The average Canadian equates the idea of a computer with a desktop or laptop. In actuality, the definition of a computer is much broader. It is estimated that in the average Canadian's home there are 30 to 40 embedded systems. Television, audio systems, refrigerators, telephones, temperature controls and stoves all use embedded computers. Embedded computers also play an integral role in many of the assisted devices that aid disabled individuals in performing daily activities, e.g. power wheelchairs and communication devices. Additionally, many embedded devices are responsible for protecting human life; embedded computers control modern medical instrumentation, airplanes, air traffic control systems, anti-lock breaking systems (ABS) and even the "fasten your seatbelt" light on the dashboard of your car. By many estimates, embedded devices account for 99% of worldwide computers while desktops and laptops account for just 1%. Embedded computers are small-scale application-specific computing devices. Embedded computers already permeate our society and their growth is expected to continue indefinitely.

It is common knowledge that technological developments are producing increasingly efficient and compact computers. This applies to embedded computers as well. The more powerful and complicated the components of an embedded device, the more difficult it is for a computer engineer to produce a product that meets safety, performance, cost and power consumption requirements within a reasonable amount of time. The production of the high performance embedded devices of the future will require tools and formal methods to aid engineers in system design and development.

The research presented in this thesis is centred around the development of computer aided design (CAD) software tools that will be used to aid in the design and development of future embedded devices. Essentially, an engineer will tell the CAD tool what the desired device should be capable of doing; the tool will analyze the given information and recommend a reliable and efficient design. This can aid a product engineer in developing systems that are far more reliable, cost, time and power efficient. Reliability is key for the development of safety-oriented devices that are responsible for protecting human life. Minimal production cost is important for ensuring the final products are accessible to all people regardless of socio-economic status. Similarly, decreased labour costs as a result of more expedient design, results in a cheaper product. Finally, minimizing a device's power consumption is essential to reducing energy costs, both financial and environmental. These CAD tools will aid in the development of new devices that will continue to play a central role in our lives.

Within electrical and computer engineering, embedded systems research is in its infancy. This makes it an exciting field to work in as it provides unique and novel opportunities. It is also attractive due to its broad range of applications (aerospace, automotive, communications etc.). Developing these software tools can be technically challenging and thus intellectually rewarding.

1.2 Original Contributions

This thesis presents a new hardware-software co-synthesis tool to aid in the design and development of high performance embedded devices. The proposed approach focuses primarily on computationally intensive computing systems requiring high levels of fault tolerance. A full hardware-software co-synthesis approach is presented with comparisons to a fully exhaustive technique. Implementation results are also provided in order to further demonstrate the algorithm's efficacy.

The major contributions of this thesis are as follows:

- Development of a hardware-software co-synthesis algorithm capable of generating hypercube architecture based embedded devices
- Development of group based fault tolerance (GBFT), a technique designed to efficiently add support for fault tolerance in embedded systems at the task graph level
- Comparative analysis between the newly developed GBFT algorithm and other existing methods
- Comparative analysis between the co-synthesis algorithm presented here and

the optimal case

- Implementation of a security/navigation device in simulation in order to demonstrate the efficacy of the co-synthesis algorithm
- Implementation of a parallel block matching device in order to demonstrate the efficacy of the co-synthesis algorithm
- Prototype construction of the block matching case study

1.3 Thesis Organization

This thesis consists of five chapters. The second chapter encompasses a survey of hardware-software co-design, and a thorough survey of hardware-software partitioning and co-synthesis. Chapter 2 also includes background on hypercube architectures. These two chapters are intended to provide the basic understanding of the design issues of hardware-software embedded systems and to survey the existing research in this field.

The third chapter is the main component of this thesis. It consists of a full description of all of the components of the co-synthesis algorithm. These include group based fault tolerance, a pipelined scheduling technique, a method for adding processing elements to the current system design, placing existing processing elements within a hypercube topology and synthesizing all required communication links.

The fourth chapter describes the two case studies implemented in order to demonstrate the algorithm's efficacy. The first case study performs the decoding of parallel MPEG-2 video streams and compares the algorithm's results with that of an exhaustive technique. The second case study performs parallel block matching and a final prototype device is constructed. The fourth chapter also includes a discussion of the experimental results obtained from both of the case studies. The fifth chapter concludes this thesis.

CHAPTER 2

DESIGN OF EMBEDDED SYSTEMS

2.1 Introduction to Hardware-Software Co-Design

This section surveys hardware software co-design. Hardware-software co-design is an active area of research that involves the development of tools and methodologies to aid in the design of embedded computer systems. Embedded computers are processing devices used in areas as diverse as wireless communications, medical instrumentation, transportation and food preparation. Although these fields are widely different, the embedded device components of the products share common design techniques. This is an outline of hardware-software co-design: a method for designing and developing an embedded computer device. The motivation behind co-design is that both hardware and software components should be addressed simultaneously in order to ensure that the final device meets cost, performance, reliability and power consumption goals.

Separate software and hardware design methods have been the subject of a great deal of research over the years [21, 41]. However, the design of both hardware and software as a joint venture remains an area of rapidly growing research. Most of the embedded systems research has been stimulated by the development of fairly inexpensive high performance microprocessors [8]. When embedded processors were exclusively small and responsible for the execution of minimal amounts of software, simple techniques were more than sufficient to create devices that satisfied performance and functional goals within a reasonable time to market. With the number of transistors on a chip increasing exponentially, embedded devices have the potential to utilize far more sophisticated circuits and architectures [48]. The embedded engineer requires CAD tools to aid in the design and development of embedded computers and to predict implementation costs.

The rest of the subsection introduces the motivation behind hardware-software co-design of embedded systems. Additionally, it intends to introduce the various components of the hardware-software co-design process.

Co-Design Overview

The embedded system design process will vary considerably with respect to the type of product under development. However, commonalities can be identified and the ability to abstract hardware and software components to the same level is greatly exploited in hardware-software co-design. The traditional approach to the design of an embedded computer system is to enforce hardware-software partitioning at an early stage. This results in well-defined design tracks for both the hardware and software components. The major weakness in traditional embedded systems design lies in the early partitioning process. A graphical overview of the traditional design approach is provided in Figure 2.1 below.

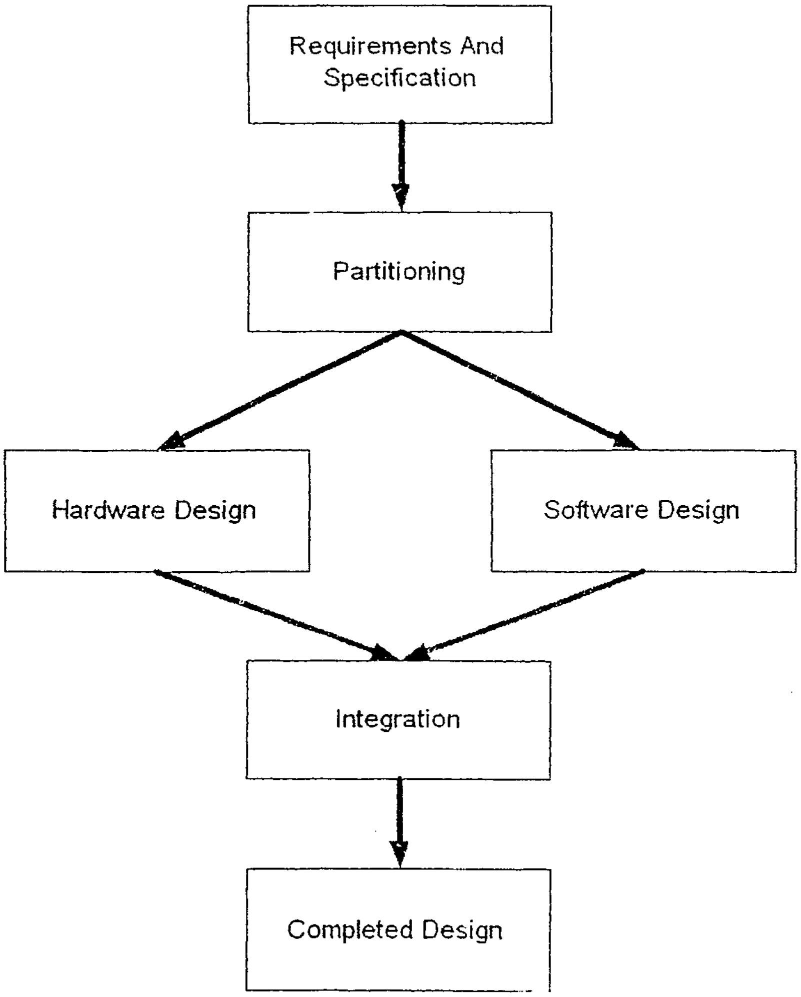

Figure 2.1: Traditional Design

One of the major flaws of such an approach is the inability of the design and development flow to correct mistakes made in the partitioning phase. If during integration, an embedded systems engineer discovers that the product will not meet various non-functional requirements (performance, power consumption, etc.), the cost imposed on re-evaluation of the design will be extremely high.

The hardware-software co-design process begins with the creation of device requirements, which leads to a formal specification. Both functional and non-functional requirements such as performance, cost and power consumption are specified. This can then be converted into a standardized system description or specification. Embedded system specification requires detailed models to aid in the abstract description of component functionality. Abstract modeling that does not differentiate between hardware and software is known as co-specification. Further research into the high level modeling of embedded devices would be greatly beneficial.

It is common for this standardized description to be converted into a task graph format. Hardware-software partitioning is performed on this task graph. Partitioning is concerned with assigning an execution location (software or hardware) to each task. After partitioning, co-synthesis is performed and typically, the co-synthesis and partitioning phases are closely knit. Co-synthesis is broken down into the assignment, allocation and scheduling phases. Finally, the generated software, hardware and interface modules are integrated. Feedback in the design process can occur at system integration by returning to the partitioning phase, thus allowing the designer to refine the given solution. At integration, the overall system can be evaluated for functional and non-functional requirements by using hardware software co-simulation. Co-simulation allows for both hardware and software components to be tested congruently. A visual overview of the co-design process is provided in Figure 2.2.

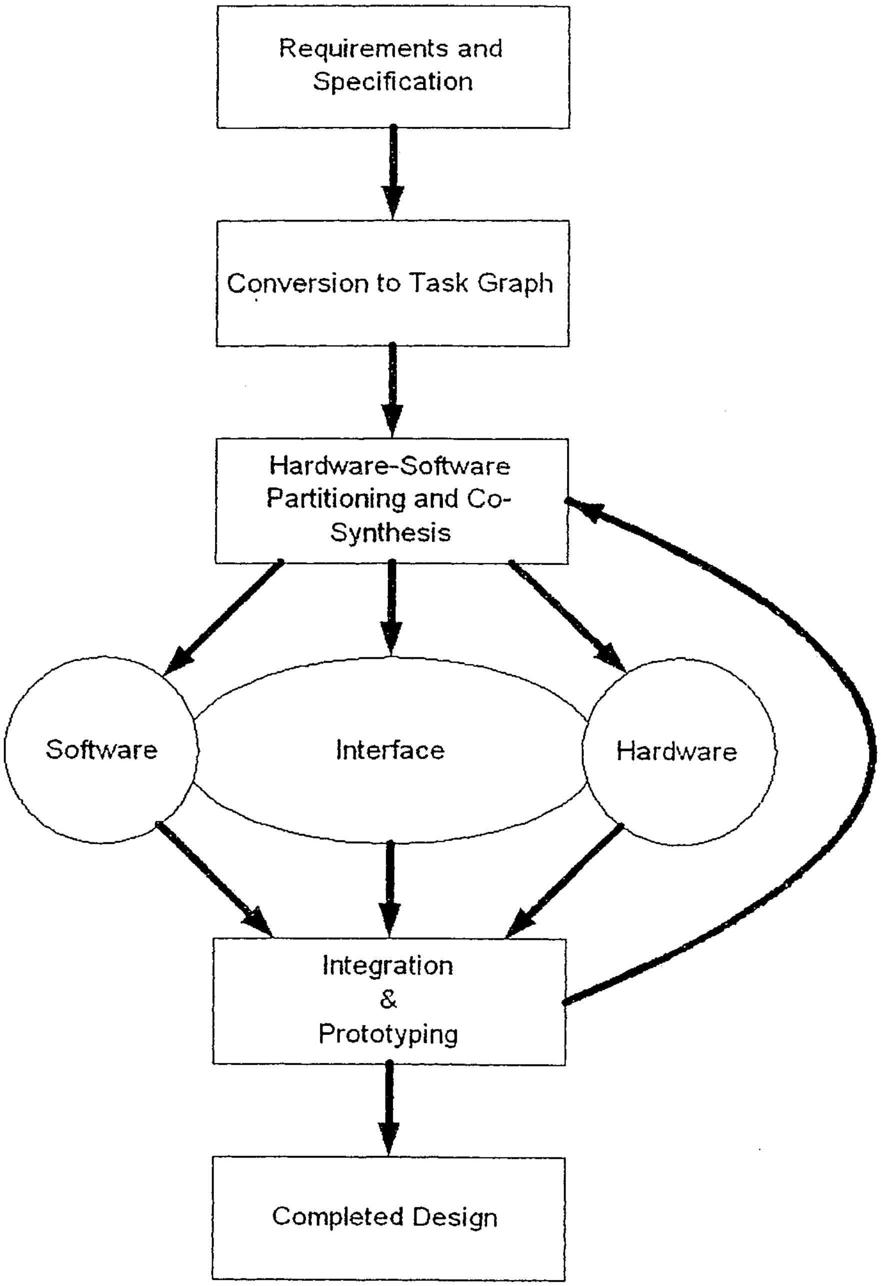
Requirements and Specification
Conversion to Task Graph
Hardware-Software Partitioning and Co-Synthesis
Software
Interface
Hardware
Integration & Prototyping
Completed Design

Figure 2.2: Hardware-Software Co-Design

HW/SW Partitioning

The partitioning process is concerned with deciding what system functionality will be implemented as hardware and what will be implemented as software. Typically, an embedded device will need to meet a number of non-functional requirements. These would often include performance, price, reliability and power consumption. With more components implemented in hardware, system price and power consumption will increase. However, with a large number of components implemented in software, system performance and reliability can degrade. It is important to balance the selection of hardware and software components to ensure that all system requirements are met. Significant research has been conducted with respect to partitioning algorithms in order to automate the process of obtaining an efficient hardware-software layout for an embedded device.

HW/SW Co-Synthesis

Hardware-software co-synthesis of an embedded device is the process by which the hardware-software architecture of the system is automatically derived to satisfy multiple goals. These goals can include factors such as cost, performance and power consumption. Hardware-software co-synthesis is inseparable from the process of partitioning. The hardware-software co-synthesis design flow consists of three main components: allocation, assignment and scheduling. Allocation is concerned with selecting the number and type of communication links and processing elements in the system. The assignment component is concerned with the mapping of tasks to processing elements. The scheduling component is concerned with the timing of task execution and communications. Typically, the partitioner will iteratively adapt its hardware mappings based largely on the results of the scheduler. The scheduler is typically the final phase of co-synthesis. A visual overview of a common approach to co-synthesis is provided in Figure 2.3.

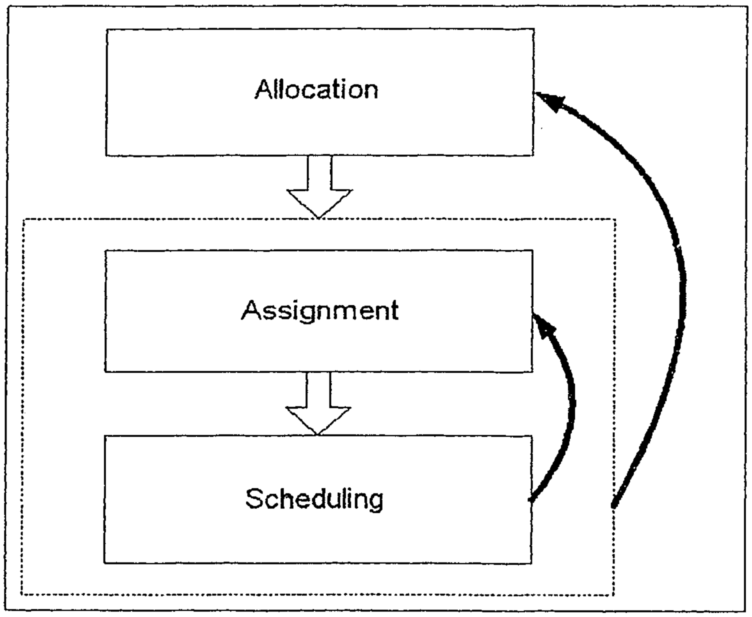

Figure 2.3: HW/SW Co-Synthesis

HW/SW Co-Simulation

High performance embedded system components can be extremely complex. It is difficult to develop comprehensive analytic systems to model the performance of an embedded device that consists of complicated components. Concurrently simulating components with differing behavioural models is referred to as co-simulation. Typically, a co-simulation environment will model multiple components, both software and hardware. This can be a difficult task as software simulation consists of modeling a processor executing a series of instructions. However, hardware simulation can consist of modeling something completely different, such as an analog or digital circuit. In an embedded device it is common for the execution of application specific integrated circuits to depend on commands issued by one of the system's processors. One should note that co-simulation requires the hardware simulator to react to input from the software simulator and vice versa.

A number of co-simulation tools have been developed. One of the most well-known co-simulation tools is Seamless from Mentor Graphics. Seamless allows the user to tie in various hardware and software simulators. Seamless coordinates the communications between simulators to ensure that the overall behaviour reflects that of an embedded device. Although Seamless performs as an effective co-simulation tool, it can also be applied for hardware-software co-verification [44].

HW/SW Co-Verification

The complexity of embedded systems prevents the designer from relying on traditional validation techniques such as simulation and testing. These techniques are insufficient to properly verify the correctness of such a system. To address this problem, new formal verification methods are needed to overcome the limitations of traditional validation techniques.

Multiple methods have been developed for performing hardware-software co-verification. A common approach involves the use of a Petri-net based representation of embedded systems, as in the system named PRES [9, 10]. The PRES model proves the correctness of an embedded system by determining the truth of computation tree logic and timed computation tree logic. These research projects make use of model checking to prove the correctness of embedded systems and have used an automatic teller machine server to demonstrate the feasibility of their approach. Another approach developed by Hsiung involves the use of linear hybrid automata and employs a simplification strategy to address the state-space explosion that occurs in the formal verification of complex systems [26].

Presently, our knowledge of the joint hardware-software design process is far more limited than that of either of the two separately. While embedded systems development can be performed as separate processes of hardware and software design, this route can be far more challenging when attempting to meet various price and performance requirements. Additionally, this design path has a tendency to result in more error prone products due to difficulties in embedded system verification.

Research into system modeling is a key element to our understanding of hardware-software co-design. While many abstract models for embedded system components exist, there is a lack of accurate models to address the detailed characteristics of these components. Embedded systems are always being developed with various performance and cost metrics in mind. In order to properly meet those requirements, it is essential to develop a thorough understanding of modeling that includes both the intricacies of a component as well as its high level properties.

In the current embedded systems market, designers and developers can mostly produce devices that meet requirements within a reasonable amount of time. However, soon the utilization of sophisticated hardware-software co-design techniques will be required in order to meet future device demands. As advanced processors and ASICs become less expensive, the need for tools to aid the design and development process will increase dramatically in order to ensure the development of high quality devices with a minimum time to market.

2.2 Hardware-Software Partitioning

Hardware-software partitioning is the process by which the various components of an embedded computer's functionality are placed in either hardware or software. The motivation behind the partitioning process is to produce a reliable embedded device that meets performance, cost and power consumption requirements.

The partitioning process is a subset of hardware-software co-design. Computer components suitable for use in embedded devices have increased in ability and complexity dramatically in recent years. The job of an embedded systems engineer involves selecting appropriate components and integrating them to produce an embedded device. The rapid increase in both complexity and performance of these components has resulted in an increase in the difficulty of component selection and integration. These difficulties have fuelled demand for tools and design methodologies to aid in the creation of embedded devices that are comprised of both hardware and software components.

The typical embedded system design approach is significantly restrictive. The main flaw revolves around the lack of design flow after system integration and prototyping. If the system is integrated and an expensive prototype is produced, further design changes can be extremely costly. These further design changes may have to occur if prototyping reveals that the system will not meet non-functional requirements (such as performance).

This subsection's goal is to introduce the reader to the motivation behind hardware-software partitioning for embedded devices. Additionally, this subsection intends to introduce the reader to the existing research in hardware-software partitioning.

2.2.1 Standard Approach

The partitioning process is concerned with deciding which system functionality will be implemented as hardware and which will be implemented as software. It is important to balance the selection of hardware and software components to ensure that all system requirements are met. Significant research has been conducted with respect to partitioning algorithms in order to automate the process of obtaining an efficient hardware-software layout for an embedded device.

The standard approach to hardware-software partitioning involves the use of a heuristic to prioritize a task set. This prioritization aids in the determination of task mapping (to hardware or software). Often much of the job of the partitioning algorithm researcher is simply to develop an effective heuristic that will result in an optimal partitioning algorithm.

The earliest partitioning algorithms proposed, began by implementing all components in software and proceeded to move components to hardware implementations until various system requirements were optimized [18]. Other early approaches proposed to implement all components in hardware and to proceed to move components to software until system requirements were met [22]. A more recent approach involves making an educated guess with reference to whether a given task should first be mapped to software or to hardware [6]. Afterwards, the algorithm would iteratively attempt re-mapping tasks from their original locations until requirements are satisfied. These techniques and versions thereof are still in use. In all cases the approach is similar, the algorithm attempts a default or initial configuration, analyzes its effectiveness and iteratively alters the current layout if non-functional requirements are not met. Other techniques to aid in the heuristics for partitioning decisions include linear integer programming [38], simulated annealing [40] and petri-nets [19].

2.2.2 Partitioning Granularity

Granularity defines the size of system components that can be implemented in either hardware or software. A partitioning algorithm that operates at a high level of granularity (also referred to as coarse-grained) uses only large blocks of system functionality to be implemented on any given processing element (PE). It can be beneficial for a large segment of functionality to be implemented on the same PE. This is particularly evident in larger scale distributed embedded devices, where dividing a high grained system into smaller components and then mapping them across the system can yield an unnecessarily high communication overhead. This occurs when tightly coupled components are "spread out" throughout the system. Using the granularity that is specified by the programmer of an application is often referred to as a high level of granularity. Gupta and DeMicheli developed an approach to the coarse grained hardware-software partitioning problem [23]. They present a partitioning procedure to identify potential hardware and software components of a system. Their technique also utilizes the Olympus Synthesis System for digital design [39] for the synthesis of dedicated hardware within their system. Yen and Wolf have also developed an approach to the coarse grained partitioning problem [52]. They present an automatic iterative improvement technique for simultaneously performing concurrency optimization and hardware-software tradeoffs. By considering both concurrency and hardware-software tradeoffs, their approach is able to identify cost/performance points that may not have been identified otherwise.

A partitioning algorithm that operates at a low level of granularity (also referred to as fine-grained) will often divide an embedded system's functionality into extremely small components. Fine-grained partitioning algorithms have the disadvantage that separating system functionality on such a small scale can dramatically increase communication overheads, which has the effect of decreasing system performance. Some systems use this approach when they are dealing with partially re-configurable processors (processors whose IP cores can be modified during the design process).

Fine-grained systems can provide a better solution than coarse-grained algorithms because they are more flexible in terms of mapping the correct computationally intensive components to the appropriate processing elements. Fine-grained systems reduce the potential negative effect of having to deal with a poorly defined system functional specification. A fine-grained system can refine the design on such a small scale that some will take one single computationally intensive CPU instruction and treat it as a separate task [1]. It is common for tasks (individual components that can be mapped to hardware) to be called base blocks when dealing with fine grained partitioning. Ernst et al. have developed a heuristic approach to the fine grained hardware-software partitioning problem [18]. They have developed an iterative partitioning process which is based on hardware extraction and is controlled by a cost function. This technique is in use in the COSYMA system [40]. Knudsen and Madsen have also presented an approach to the fine grained partitioning problem [30]. This approach uses dynamic programming to solve both the problems of system execution time and hardware area constraints. This technique is in use in the LYCOS system [38].

Finally, there is one research project known that involves merging these two ideas of varying granularity [24]. This concept is known as flexible granularity. Depending on the characteristics of the specific application and the system's non-functional requirements, the selected granularity can span from a low level of base blocks all the way to the user-defined functions. This approach is intended to overcome the shortcomings of both of the previous approaches. This work also includes estimation methodologies adapted to different levels of granularity, which help to determine the final system granularity.

2.2.3 Dynamic Programming

Dynamic programming is a technique ideal for problems where calculating all possible outcomes is not computationally feasible. This makes dynamic programming well suited to the partitioning problem, which can be extremely computationally intensive. Typically, a dynamic solution is recursive and iterative in nature.

Dynamic programming problems can always be divided into stages where a decision will be required at each stage. Typically there are a number of states associated with each stage. Decisions made at one stage will alter the current state into a new state in the next stage. The decision made at a given state does not depend on the decisions made in the previous state. It can be seen that dynamic programming extends itself easily to hardware-software partitioning which can be approached as a recursive, iterative, state based problem. Often a dynamic programming solution will effectively process a task graph and improve algorithm execution speed by avoiding testing infeasible combinations.

Shrivastava et al. [45] have used dynamic programming to develop an algorithm that can effectively solve the partitioning problem with extremely fast execution times. Chang and Pedram [5] have also used dynamic programming to determine the solution to the coarse-grained partitioning problem of a generic task graph. Knudsen and Madsen [31] have used dynamic programming to determine the solution to the fine-grained partitioning problem.

2.2.4 Fault Tolerance

Fault tolerance is a large area of computing, whereby computational devices are developed which must meet various mission critical and safety critical requirements. These types of systems are common in aerospace and biomedical applications. Fault tolerant computing has been a large area of study [46], however, incorporating some of these ideas into hardware-software partitioning is an extremely young discipline.

It is common in the development of fault tolerant devices to incorporate a system's fault tolerant components late in the design process. This often creates a significant overhead in terms of implementation cost. Incorporating fault tolerance at an earlier stage of design is likely to be very beneficial in reducing this overhead. This line of reasoning has sparked research in incorporating fault tolerance into earlier phases in the design process, such as hardware-software partitioning.

Fault tolerant hardware-software partitioning generally involves adapting the partitioning process to automatically accommodate systems requiring fault tolerance. The most predominant work in this field is that of Dave and Jha [13]. These researchers have developed a system that will input a functional task graph specification and perform task clustering for fault tolerance. Their system will choose an error recovery topology that is optimized to use a small number of extra processing elements. Additionally, Bolchini et al. [4] have developed a partitioning algorithm that incorporates fault detection capabilities. This may not be as robust as full fault tolerance, however fault detection is relevant as it is a subset of fault tolerance.

Presently, our knowledge of the joint hardware-software design process is far more limited than that of either of the two separately. While embedded system development can be performed as a separate process of hardware and software design, this route can be far more challenging when attempting to meet various price and performance requirements. Additionally, this design path has a tendency to result in more error prone products due to a lack of formalism in design methods.

Hardware-software co-design is a burgeoning field of research and one of its most active subsets is hardware-software partitioning. Partitioning is the process of deciding what functional system components will be implemented as hardware and what will be implemented as software. Effective hardware-software partitioning is essential for creating embedded devices that meet non-functional requirements such as performance, power consumption and cost.

Since hardware-software partitioning has begun to flourish as a field of research, greatly varying approaches to the problem have been taken. Researchers have attempted to tackle many issues and incorporate many varying ideas. The concept of varying degrees of granularity allows an algorithm to separate a system's functionality into components of various size. Dynamic programming is an iterative, recursive technique that has been effectively incorporated into approaches to partitioning. Fault tolerance is an extremely important issue, especially in embedded systems, and partitioning approaches have been developed to incorporate these abilities.

In the current embedded systems market, designers and developers can mostly produce devices that meet requirements within a reasonable amount of time. However, soon the utilization of sophisticated hardware-software co-design techniques will be required in order to meet future device demands. As advanced processors and ASICs become less expensive, the need for tools to aid the design and development process will increase dramatically in order to ensure the development of high quality devices with a minimum time to market. Hardware-software partitioning will be an integral component of these tools.

2.3 Hardware-Software Co-Synthesis

An embedded system's architecture is typically determined by the intuition of a design engineer. This process adds time to the development cycle and sometimes results in architectures that do not meet non-functional requirements. It can also result in an over-designed architecture with excessive hardware that results in overly expensive devices. Consequently, incorporating hardware-software co-synthesis in design automation tools is essential for producing optimal devices with an accelerated time-to-market. The hardware-software co-synthesis problem is concerned with determining optimal hardware and software architectures. This involves the selection of processors (CPUs), application specific integrated circuits (ASICs) and communication links in order to produce a device that meets non-functional requirements. The co-synthesis problem typically involves the selection of the quantity and type of processing elements and communication links (allocation), task assignment from a task graph to processing elements and confirmation of whether the system meets requirements (usually through scheduling). A task graph is a collection of tasks and communication dependencies that describe device functionality. The allocation and scheduling phases are known to be NP-complete [20, 33], thus determining an optimal solution in the co-synthesis phase can be extremely computationally intensive.

This work is motivated by the need to automate the embedded system design process while simultaneously producing high quality devices. The research presented in this thesis is further aimed at producing high performance fault tolerant embedded systems. Embedded applications that would particularly benefit from high levels of fault tolerance include aerospace, medical instruments and high performance telecommunication systems. In fact, it is believed that within the next two or three decades probes will be sent to orbit nearby stars [34]. Such embedded devices would be responsible for considerable system control and measurement instrumentation in addition to unparalleled fault tolerance requirements.

The primary focus of most of the co-synthesis research problem has concentrated on the simplistic single processor and ASIC architecture [3, 6, 18, 28]. Some approaches assume more complicated architectures, such as two CPUs and hardware-accelerated circuitry [37]. Various approaches have been attempted that involve moving tasks from hardware to software or vice versa in order to meet the system requirements. In the co-synthesis of distributed systems, target architectures can incorporate multiple processors, ASICs or FPGAs (field programmable gate arrays). Two main techniques have been utilized to handle the co-synthesis of distributed systems: the optimal and heuristic approaches.

The optimal approach can be divided into three sections: exhaustive, mixed integer linear programming, and constraint solving. The exhaustive approach is characterized by attempting all possible mappings to provide an optimal solution. It can be very computationally intensive and is only suitable for smaller systems. D'Ambrosio and Hu have presented an exhaustive technique for hardware-software partitioning [11]. However, their approach is limited to single processor architectures and ignores the communication overhead. A hardware-software partitioning technique using mixed integer linear programming is presented by Prakash and Parker [42]. The execution time of this technique is prohibitive for large task graphs. Moreover, this approach is limited to bus based architectures or pre-determined point-to-point interconnection topologies. A constraint solving approach has been presented by Kuchcinski in the JaCoP system that concentrates on scheduling and resource assignment [32].

The heuristic-based co-synthesis approach can be divided into two methods: constructive and iterative. The iterative scheme is characterized by having an initial solution, which is iteratively improved. Kirovski and Potkonjak presented an iterative algorithm that includes power as a cost function but their approach ignored the inter-task communication time [30]. Other iterative techniques have been developed [25] but they are limited in that they allow for only one type of communication link. Li and Wolf have presented an iterative co-synthesis algorithm capable of synthesizing memory hierarchies for bus architecture topologies [36]. MOGAC is an iterative approach to the co-synthesis problem that uses genetic algorithms [16]. Experimental results show that for large systems MOGAC suffers due to large execution times. Wolf has also presented an iterative approach to the co-synthesis problem producing generic device architectures [47]. Generic device architectures can be tuned to a particular application, but it can also result in disorganized and difficult to understand designs as the interconnection topology may not conform to an established architecture. The constructive heuristic method is characterized by building the solution step by step, where the final output is not available until the algorithm terminates. Constructive co-synthesis algorithms are presented in the COSYN [15] and COFTA [13] systems. Bakshi and Gajski have also presented a constructive partitioning approach that supports pipelining at varying degrees of granularity [2]. Although their approach allows the addition of multiple software processors, it does not account for the hardware cost of adding each CPU. All of the above co-synthesis algorithms support distributed embedded systems but none have utilized hypercube topologies as a target architecture.

2.4 Hypercube Architectures

Hypercube topologies are useful for high performance embedded systems and have a number of advantages over other architectures [12, 27, 35, 43]. Hypercube nodes represent processing elements (PE) and a link between nodes represents a communication interface (serial, parallel, Ethernet link, etc.). Hypercube topologies are very flexible, versatile and generic. Figure 2.4 shows a hypercube network of degree four, constructed from two 3D hypercubes. The high level of interconnectivity in a hypercube architecture results in a system that is suited to fault tolerance. Additionally, hypercube systems support high performance computation while limiting the communication overheads and/or bottlenecks associated with large-scale systems consisting of many PEs. These features make hypercube architectures an excellent choice for multiple PE embedded systems that need to couple fault-tolerance with complex computation. Additional advantages of hypercube architectures include topologies with small diameters and high levels of symmetry. Finally, routing in hypercube networks is well researched and efficient routing algorithms are available [29].

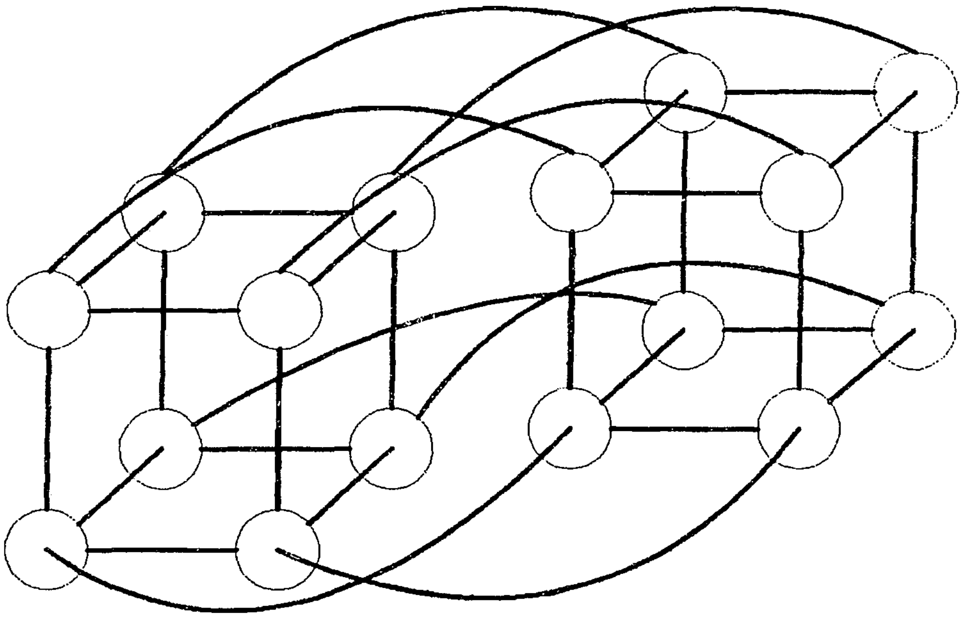

Figure 2.4: A 4D Hypercube

The main disadvantage of hypercube topologies is their poor upward scalability. It can be a difficult and complex process to add nodes to a hypercube network. This criticism applies more directly to hypercube computer systems, which are likely to be reconfigured and expanded regularly. It is uncommon for a distributed embedded device that has already been manufactured to require any additional processing elements. Such a circumstance could be used in reconfigurable space systems. Another disadvantage of hypercube systems is the large number of communication links, but additional links support fault tolerance. These communication links and their interfaces to the processing elements result in a significant cost overhead. However, it should be noted that this research targets high performance fault-tolerant embedded devices. As a result, the associated cost to produce systems that meet the flexibility, reliability and performance requirements of demanding applications is knowingly accepted.

Many co-synthesis methods for distributed embedded systems target varying architectures. Hypercube topologies can be considered a superset of a number of other hierarchical architectures. Topologies such as mesh, torus, binary and quad trees can be partially represented by suitable sized hypercube topologies. Binary trees have even been embedded in incomplete hypercube systems [51]. A hierarchical architecture (e.g. tree topology) is one of the most prevalent system targets in the high performance distributed co-synthesis research projects [13, 14].

Much of the work on automatic architecture generation in co-synthesis algorithms for distributed embedded systems has concentrated on hierarchical topologies [14]. Although hierarchical systems can be adapted to enhance their fault tolerant capabilities [13], a comparison of the two architectures is provided to illustrate the capabilities of hypercube topologies. Consider a 3D hypercube with eight processing elements presented in Figure 2.5. If the communication link between PE^0 and PE^1

fails, the device could still operate correctly by routing messages along alternate channels (e.g. $PE^0 \rightarrow PE^4 \rightarrow PE^5 \rightarrow PE^1$). System performance would decrease as communication between nodes connected by a faulty link (e.g. $PE^0 - PE^1$) would take longer (three transfers instead of one). However, most importantly, the system will continue to function. Now consider a possible hierarchical architecture implementation of the same embedded device shown in Figure 2.6. If the equivalent communication link (connecting PE^0 and PE^1) fails, PE^1, PE^3, and PE^5 of the embedded device would cease to function as they cannot communicate with any of the other PEs. This would result in a catastrophic system failure. The benefits over alternative architectures were a major motivation for developing the first hardware-software co-synthesis algorithm targeting hypercube topologies.

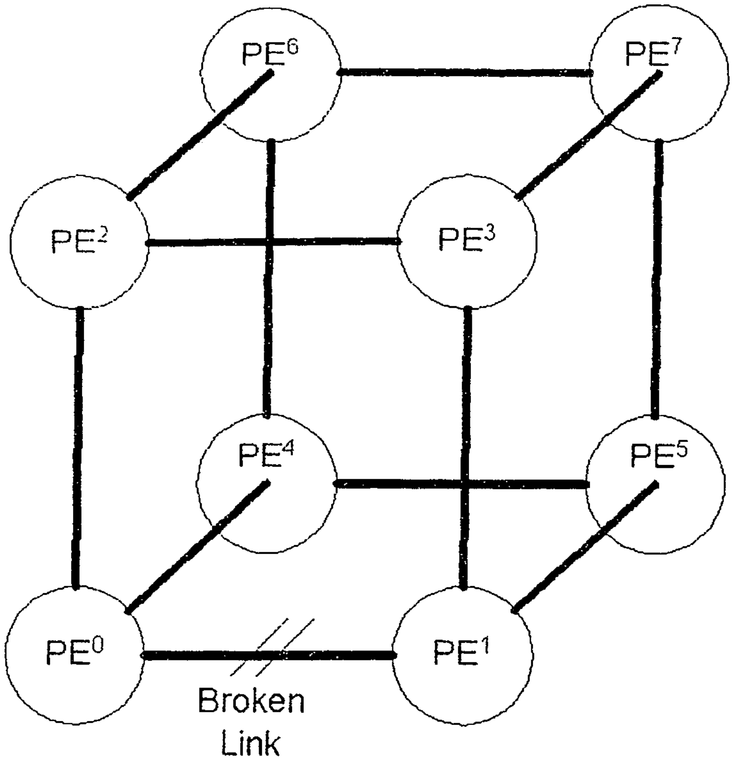

Figure 2.5: Example Hypercube Architecture

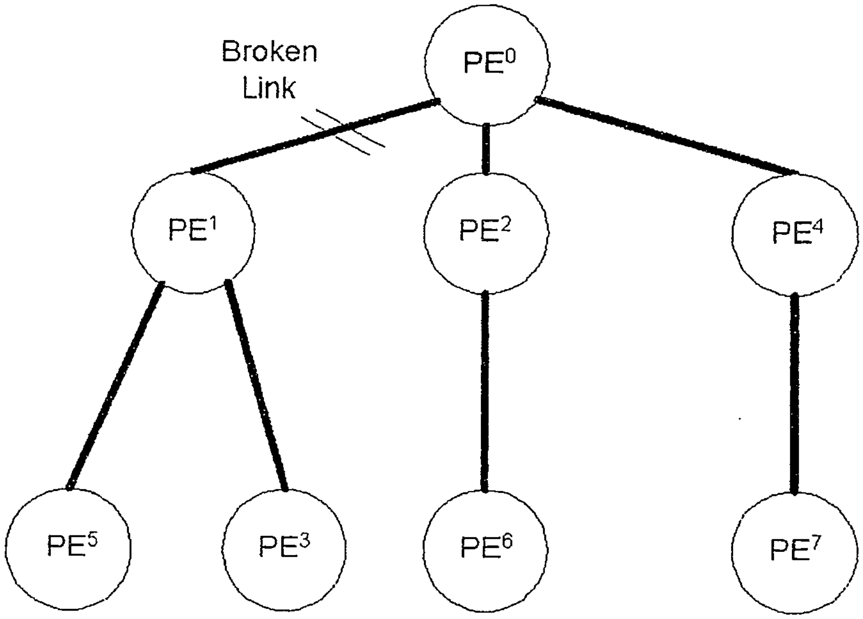

Figure 2.6: Example Hierarchical Architecture

CHAPTER 3

CO-SYNTHESIS FOR HYPERCUBE SYSTEMS

3.1 Introduction

A constructive co-synthesis approach that targets a hypercube topology as the final system architecture is presented. The algorithm uses a library of processing elements (PEs) including processor cores (CPUs) and application specific integrated circuits (ASICs), which provides relevant data such as hardware area requirements and performance information. The library can consist of many CPU types and ASICs. The library also provides information related to various types of communication links available with their interconnectivity costs. The algorithm takes in a task graph representing the functionality of the device. The communication links are generic and the co-synthesis algorithm supports all types of communication links including serial, parallel, etc. The algorithm assumes that each PE in the hypercube system consists of either a CPU or an ASIC. In addition to this, each PE consists of some local memory for computational purposes and interface circuitry for communication links.

The constructive co-synthesis approach presented in this thesis is provided in Figure 3.1 and consists of six main steps:

1) **Specification:** Defining the required device functionality and performance and area requirements.
2) **Profiling:** Evaluating each functional unit in the specification for performance and area utilization data.
3) **Group Based Fault Tolerance (GBFT):** A heuristic technique for adding fault tolerance to an embedded device at the task graph level.
4) **Scheduling:** A technique for evaluating the current device architecture for performance.
5) **Add Processing Element:** A heuristic technique for adding an additional processing element (CPU or ASIC) to the current device architecture.
6) **Synthesize Communication Links:** A technique for arranging the system's processing elements into a hypercube topology and synthesizing all connecting communication links.

The first phase of the approach is concerned with defining the device requirements. Although any specification language can be used, the experimentation presented in this thesis has been specified in C language. The second phase or the algorithm involves the profiling of the device specification. The specification is manually converted into task graph form. Each task in the task graph represents a functional section of the overall device. Each of these functional sections are timed for execution on each type of processor available in the library. Additionally hardware alternatives to the software implementation are developed and are profiled for both performance and area utilization. The profiling stage is complete once the software and hardware timing data and the hardware area data have been collected for each task in the task graph.

The third phase is the group based fault tolerance (GBFT) method which is applied to the input task graph. This heuristic was developed to add a minimal fault detection mechanism to the system and to simplify fault recovery. The algorithm adds additional assertion and duplicate/compare tasks to the task graph. It minimizes the fault detection overhead by exploiting a task's error transparency and combining tasks into groups. The quantity and type of spare PEs in the final device is set by the user. If a fault is detected by one of the added tasks, an additional processing element is signaled to commence execution of the failed task group. This simple heuristic provides a low overhead method for performing node-fault detection and recovery.

The fourth phase is the scheduling technique (see the "Scheduling" block in Figure 3.1). This heuristic method was developed to efficiently evaluate the current device architecture to determine if it meets performance requirements. In order to improve device throughput, the scheduling technique utilizes the established RECOD retiming heuristic to support pipelining of the task graph [7]. The scheduler accurately predicts overall device performance by scheduling tasks based on data dependencies. If the scheduler finds a task execution configuration that allows the current device architecture to meet performance requirements then the co-synthesis algorithm terminates successfully. If the scheduler is unable to schedule the task graph within device performance constraints the co-synthesis algorithm proceeds to the fifth phase.

The fifth phase is concerned with the addition of another processing element to the system (see the "Add Processing Element" block in Figure 3.1). This section of the co-synthesis algorithm analyzes the current device architecture, the current task mappings, each task's hardware/software performance data and each task's hardware area data to determine the ideal type of processing element to add to the current system. If this phase is successful in adding a processing element to the system, the co-synthesis algorithm proceeds to phase six. If this phase is unsuccessful in adding another processing element (unable to add more hardware while still meeting the device hardware area constraint) the co-synthesis algorithm terminates unsuccessfully and provides the user with the partial solution generated.

The sixth phase is concerned with arranging the system's processing elements within a hypercube configuration and synthesizing all of the communication links (see the "Synthesize Communication Links" block in Figure 3.1). This phase arranges all of the processing elements within a hypercube topology while attempting to keep PEs with high levels of intercommunication within close proximity of each other. Once all of the system's PEs have been arranged, communication links connecting the PEs are synthesized. Once this phase has completed, the co-synthesis algorithm proceeds to the scheduling phase.

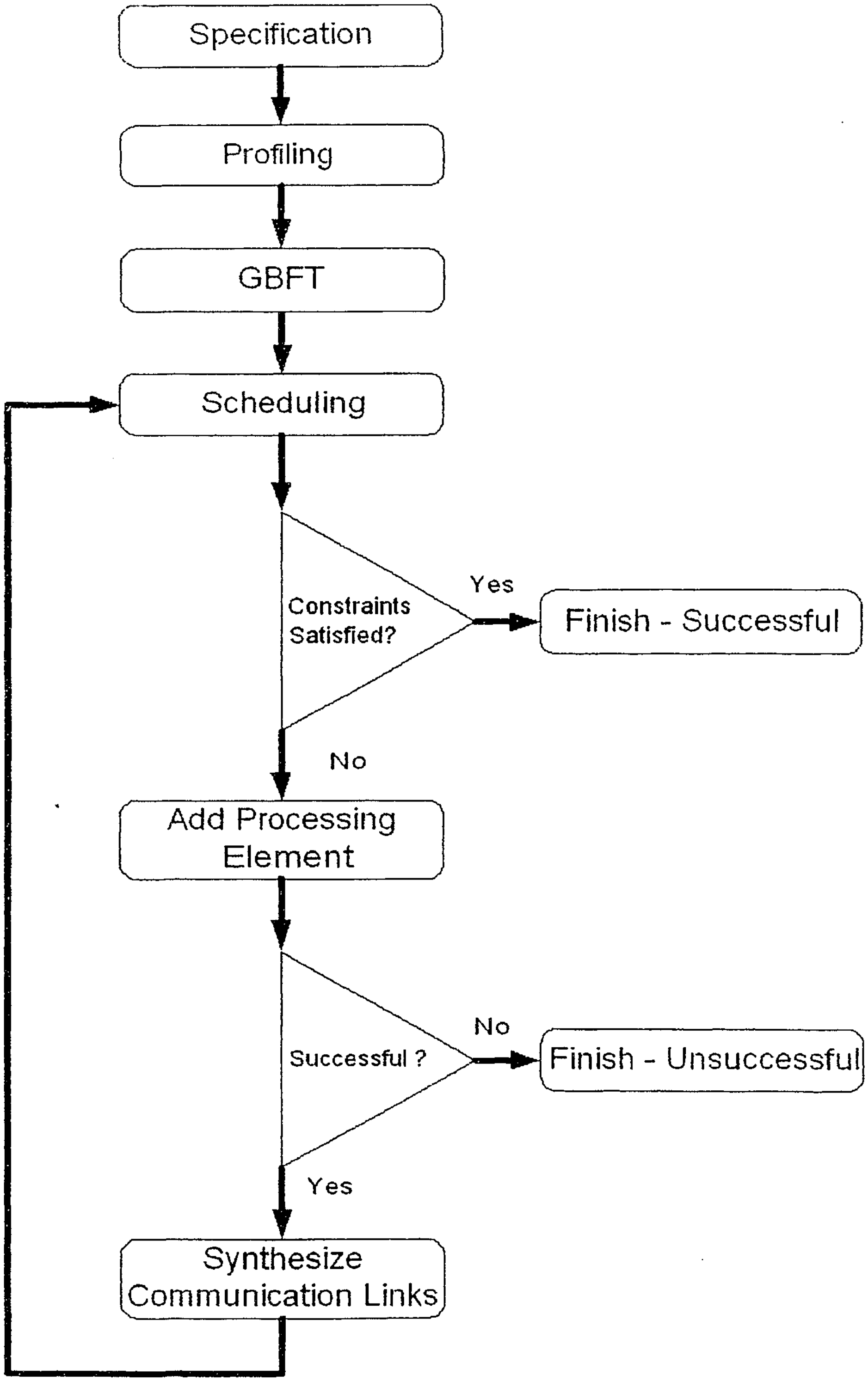

Figure 3.1: Hypercube Co-Synthesis Algorithm Flow

3.2 Fault Tolerance at the Task Graph Level

3.2.1 Overview

This subsection discusses the two main preexisting techniques for adding fault detection/tolerance at the task graph level. Additionally, section 3.2.4 presents an original contribution to task graph based fault tolerance techniques. In all cases fault detection is accomplished through the addition of fault detection tasks to the task graph. There are three types of tasks that any of the approaches presented may add to a task graph: assertion tasks, duplicate tasks and compare tasks. An assertion task will analyze another task's output in order to determine whether the generated results are erroneous. A duplicate task reproduces the same work as another task in the graph. Typically a duplicate task will utilize an alternative implementation to that used by the task it is duplicating. A compare task will examine the results of two tasks to detect any inconsistencies. The addition of an assertion task typically requires considerably less computational overhead than the addition of a duplicate and compare set. However, assertion tasks are not always feasible. For an assertion task to be used, error states must be able to be detected by analyzing the results. An example assertion task could be the analysis of a checksum or checking that the generated results lie within an expected range. Duplicate and compare tasks tend to require a much higher computational overhead. First the entire task's functionality needs to be duplicated and then both generated results need to be compared.

3.2.2 Task Based Fault Tolerance

Task based fault tolerance (TBFT) was developed by Yajnik et al. [49]. Task based fault tolerance is a technique designed to add fault detection capabilities at the task graph level. In task based fault tolerance, some form of error detection must be performed for the results generated by each node in the task graph. Due to the significant difference in fault tolerant overhead, assertion tasks are favoured in the task based fault tolerance algorithm. If a given task is capable of supporting assertions then an assertion task is added. Duplicate and compare tasks are only added if assertions are unavailable for the given task. To demonstrate these concepts Figure 3.2 provides an example input task graph. Figure 3.2 also illustrates the resultant task graph after processing by the task based fault tolerance algorithm. Here all of the tasks in the graph support the use of assertion tasks for error detection with the exception of task T3. Task T3 has had duplicate and compare tasks added to provide support for fault detection.

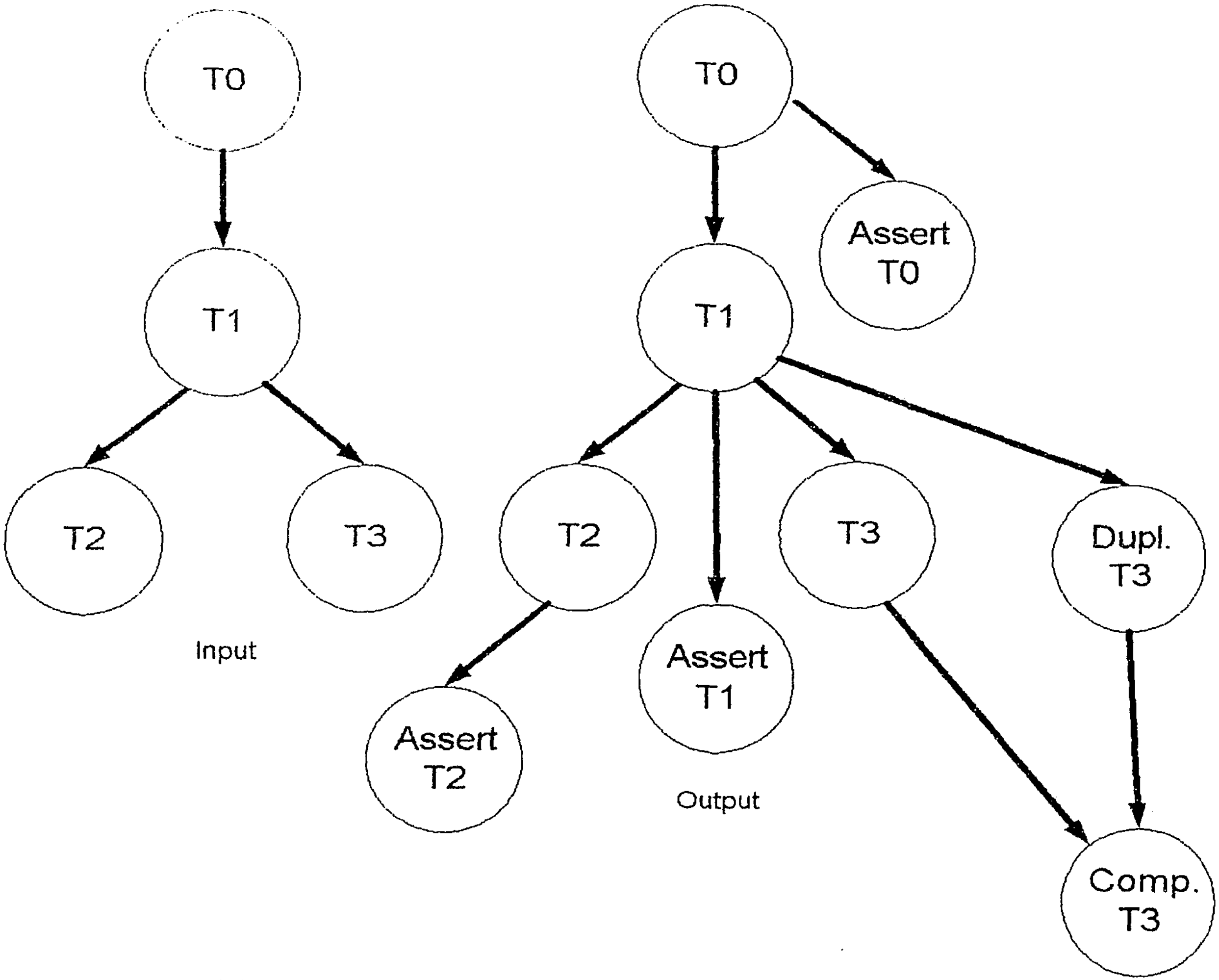

Figure 3.2: Task Based Fault Tolerance Example

3.2.3 Cluster Based Fault Tolerance

Cluster based fault tolerance was developed by Dave and Jha [13]. This technique was developed as an extension of the ideas presented in the task based fault tolerance algorithm. Modifications were made to the approach in order to reduce the substantial fault tolerant overhead prevalent in task based fault tolerance. In cluster based fault tolerance, Dave and Jha have introduced the concept of error transparency [13]. If a task provided with an erroneous input always produces an erroneous output then that task is said to be error transparent. This effect is exploited in their algorithm by grouping error transparent tasks into clusters which only require one assertion or duplicate/compare task. Figure 3.3 shows an example task graph input and the results generated by one iteration of the cluster based fault tolerance algorithm. Figure 3.4 shows the results after both 2 and 3 iterations of the algorithm. Figure 3.5 shows the final clusters for the given input task graph. After all of the tasks have been grouped into clusters, each cluster is given an assertion or duplicate/compare task to perform error detection. Figure 3.6 shows the final clustered task graph with the addition of error detecting assertion tasks. Each cluster is now treated as a single task in order to ensure that all tasks within a cluster are executed on the same processing element. In the example graph provided, all of the tasks are assumed to be error transparent.

The CBFT algorithm traverses a task graph based on task priority levels that favour tasks that are higher in the graph. The algorithm only allows one of a given task's children to be added to that task's cluster. Cluster based fault tolerance also introduces the concept of a maximum tolerated error detection time. If a large number of tasks were grouped into one cluster and an error occurs in the uppermost task, the error state would not be detected until all tasks in that cluster have completed execution. This may be undesirable as it will adversely affect performance. To avoid this problem, Dave and Jha have incorporated a user specified maximum tolerated error detection time into the cluster based fault tolerance algorithm [13]. The algorithm will not group more tasks into one cluster if the sum of the software execution times of those tasks exceeds the user specified maximum error detection time. In the cluster based fault tolerance algorithm, if an error is detected on a given cluster, the entire functionality of that cluster is moved to a spare processing element and signaled to recommence execution.

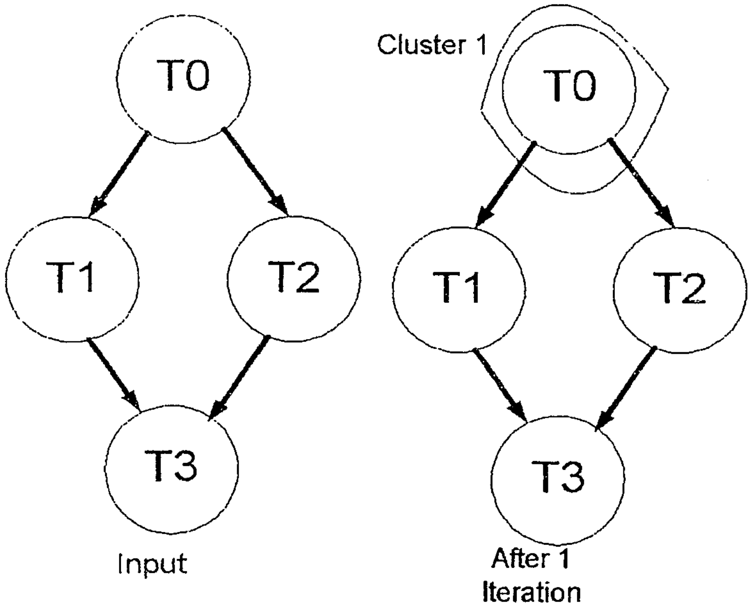

Figure 3.3: Cluster Based Fault Tolerance – Input & 1 Iteration

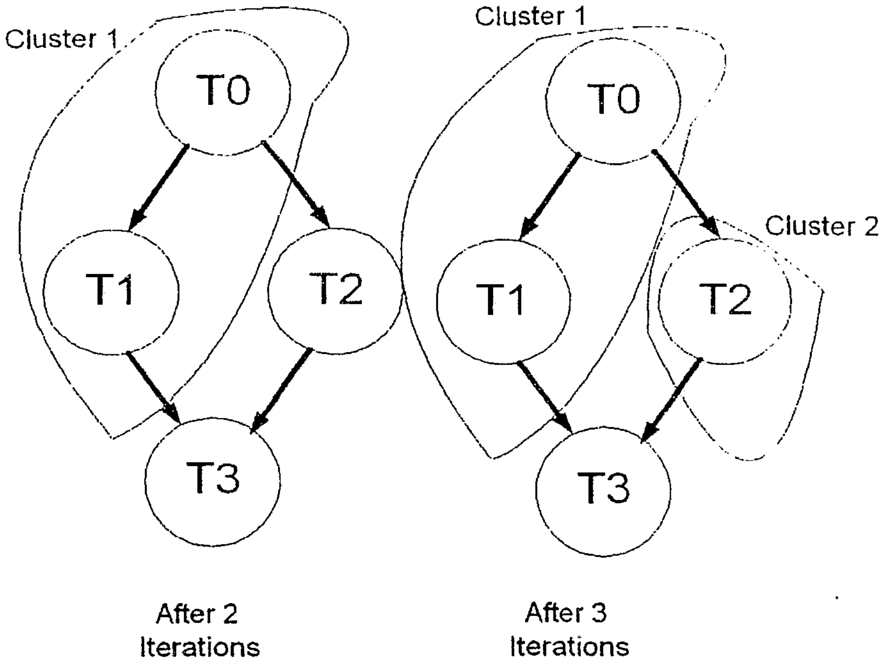

Figure 3.4: Cluster Based Fault Tolerance – 2 and 3 Iterations

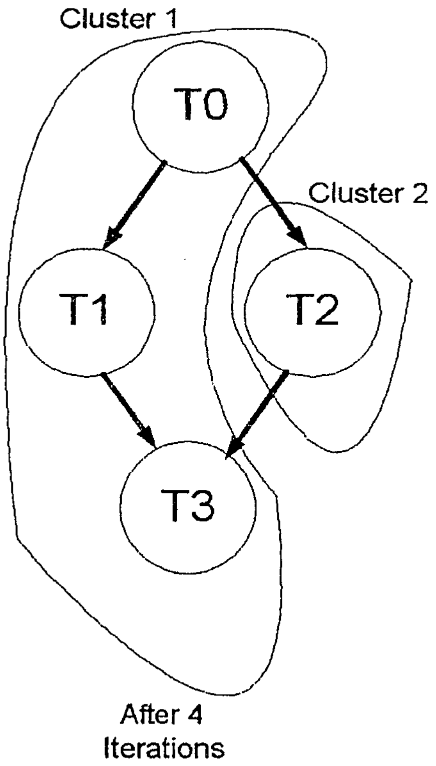

Figure 3.5: Cluster Based Fault Tolerance – Final Clustering

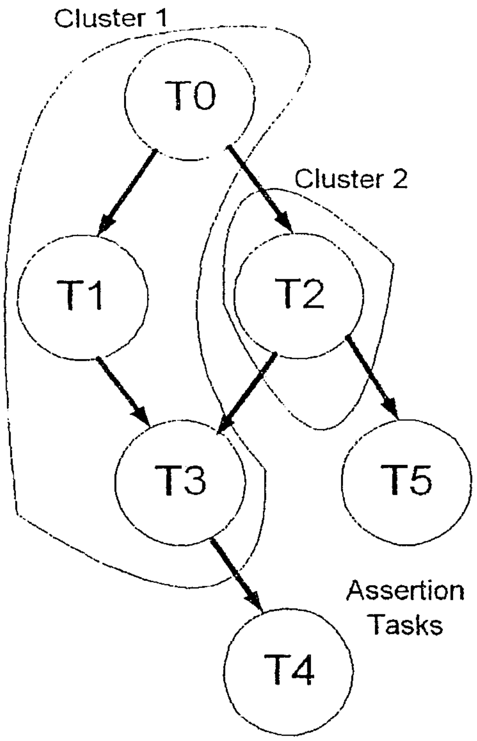

Figure 3.6: Final Added Assertion Tasks After Clustering

3.2.4 Group Based Fault Tolerance

This section presents a new and original technique for adding fault detection and tolerance at the task graph level named group based fault tolerance (GBFT). This technique was developed as an extension of the ideas presented in both the task based fault tolerance [49] and cluster based fault tolerance algorithms [13]. Modifications were made to the approaches in order to reduce the fault tolerant overhead prevalent. This algorithm utilizes the concept of error transparency. Group based fault tolerance also uses the concept of a user specified maximum tolerated error detection time.

In group based fault tolerance, the task graph is traversed from the lowest nodes to the highest nodes. Each leaf node (bottom level node) is assigned to its own group. Each of the leaf's parents are analyzed to determine whether they can potentially be added to the given leaf's group. A parent is considered a possibility for grouping in its child's group if it is not already grouped and if adding it to its child's group will not violate the user imposed error detection timing constraint. Once the set of parents eligible for grouping have been assembled for a given task, the parents are iteratively added to the group in order of decreasing fault tolerant overhead. This process is ended if the addition of another parent task to the group will violate the user defined error detection constraint. If a task has no children, its fault tolerant overhead is set to its assertion overhead. If a task *i* does have children, its fault tolerant overhead is calculated to be:

$$\max[assertion\_overhead(children(i)) + Com(i, children(i))] + assertion\_overhead(i)$$

where,

assertion_overhead(i) = the assertion overhead of task i children(i) = set of the child tasks of i

Com(u, v) = communication time from task u to v across a communication link

Figure 3.7 shows an example task graph input and the results generated by one iteration of the group based fault tolerance algorithm. Figure 3.8 shows the results after both 2 and 3 iterations of the algorithm. Figure 3.9 shows the final groupings for the given input task graph. Finally, Figure 3.10 shows the final clustered task graph with the addition of error detecting assertion tasks. A more complex example of GBFT based task groupings is provided in Figures 4.6, 4.7, 4.8 and 4.9 in section 4.2. After all of the tasks have been grouped, each group is given an assertion or duplicate/compare task to perform error detection. Each group is now treated as a single task in order to ensure that all tasks within a group are executed on the same processing element. In the example graph provided, all of the tasks are assumed to be error transparent. In the group based fault tolerance algorithm, if an error is detected on a given cluster, the entire functionality of that cluster is moved to a spare processing element and signaled to recommence execution.

The group based fault tolerance algorithm adds assertion and duplicate/compare tasks to a task graph. It does not add assertion or duplicate/compare tasks to perform checks on the fault detection tasks that it adds. If fault detection of faults occurring in the GBFT added tasks is wanted, then it must be added manually after the GBFT algorithm has completed execution.

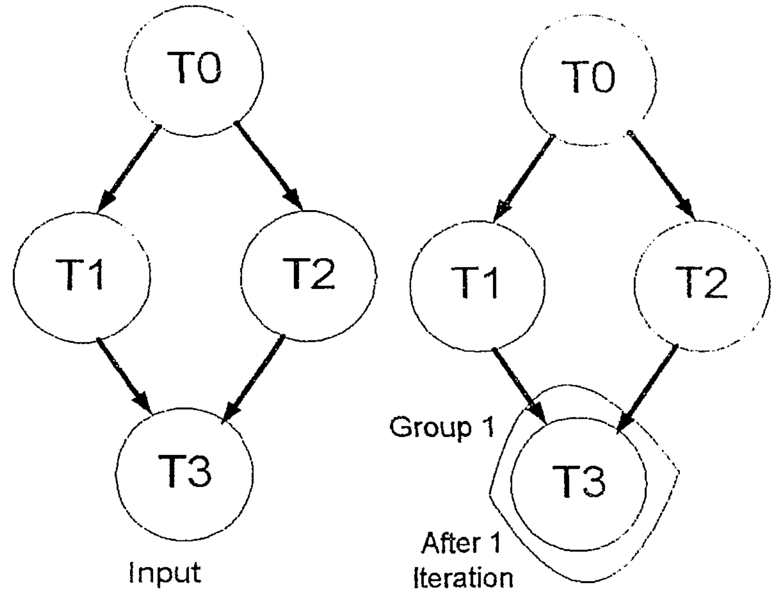

Figure 3.7: Group Based Fault Tolerance – Input & 1 Iteration

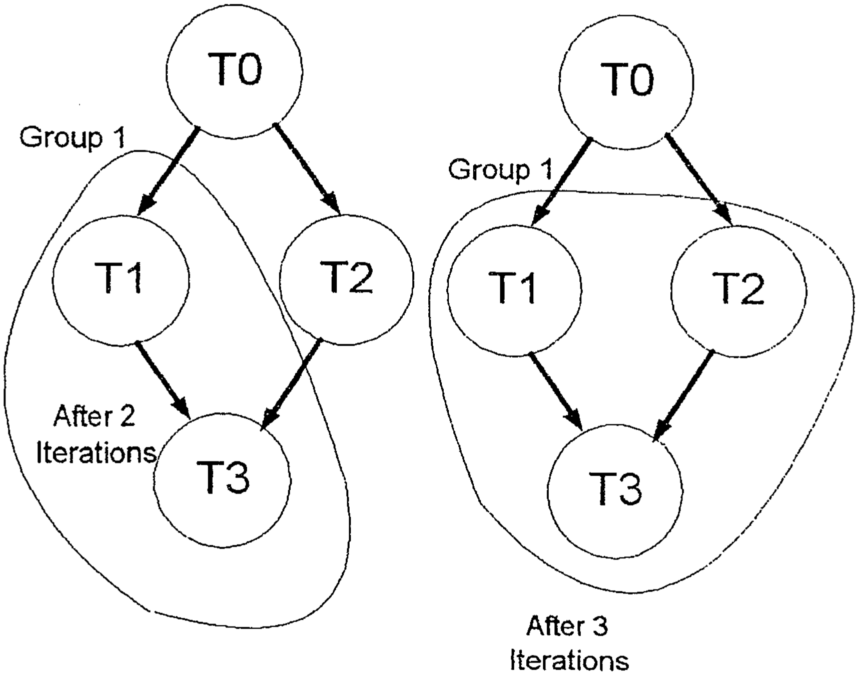

Figure 3.8: Group Based Fault Tolerance – 2 & 3 Iterations

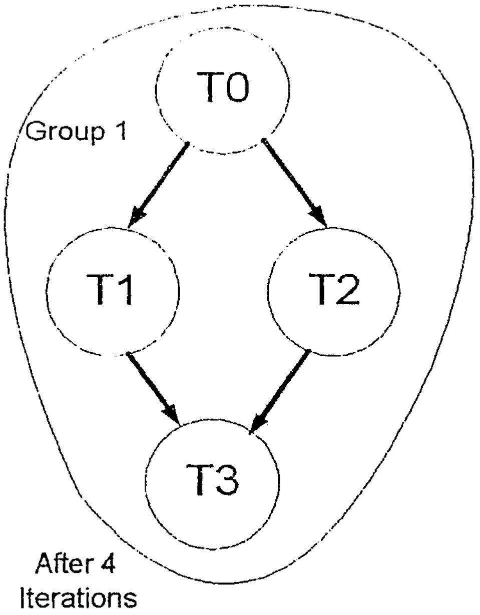

Figure 3.9: Group Based Fault Tolerance – Final Grouping

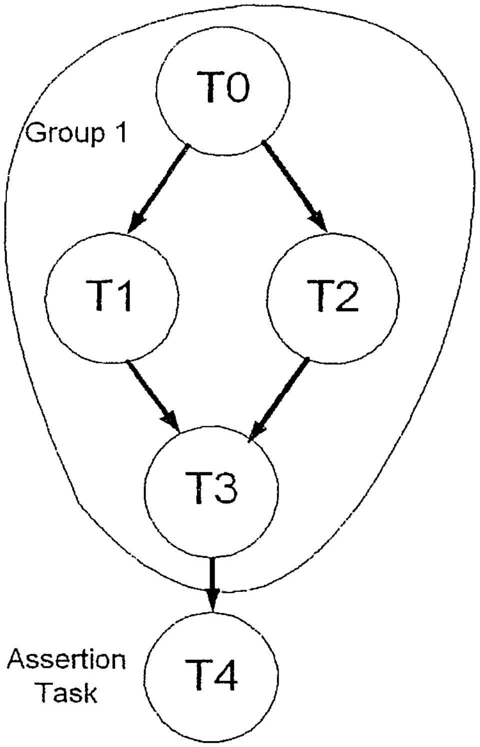

Figure 3.10: Final Added Assertion Task After Grouping

3.2.5 Task Graph Based Fault Tolerance Comparison

A comparison of the results of the group based fault tolerance algorithm with both the cluster based and task based techniques is provided in order to evaluate the newly developed GBFT technique. Thirteen random test cases were generated and the output of all three algorithms is compared. The test cases have varying task graph configurations and varying user defined tolerated error detection times. The outputted error detection information from each algorithm is compared. The data collected has been assembled in Table 3.1. Here the values under the GBFT, CBFT and TBFT columns are the counts of error detection tasks added to the task graph. The fewer number of error detection tasks added to a task graph results in a smaller fault tolerance overhead. A smaller fault tolerance overhead is desirable as it will yield a fault tolerant device that utilizes less hardware and/or less computation time.

The first randomly generated task graph has been provided in Figure 3.11. Task graph 4 corresponds to the task graph from the MPEG decoding case study. The task graph can be found in Figure 4.1. Task graph 5 corresponds to the task graph from the block matching case study and can be found in Figure 4.6.

The information gathered in the table reveals that the group based fault tolerance technique yields a 18.75% improvement in fault tolerance overhead over the cluster based fault tolerance technique and a 45.83% improvement over the task based fault tolerance technique.

Table 3.1: Fault Tolerance Comparison Data

Task Graph #	Tolerated Error Detection Time (msec)	GBFT	CBFT	TBFT
1	4	2	3	8
1	3	3	3	8
1	2	4	4	8
2	3	7	7	11
2	2	7	8	11
3	9	1	3	9
3	8	2	3	9
3	7	2	3	9
3	6	3	3	9
3	5	3	3	9
3	4	3	3	9
3	3	4	3	9
3	2	5	5	9
4	800	1	16	22
4	300	15	16	22
4	120	14	17	22
4	80	17	17	22
4	40	21	21	22
4	34	22	22	22
5	150 000	1	16	22
5	40 000	13	16	22
5	30 000	14	16	22
5	20 000	15	16	22
5	15 000	16	16	22

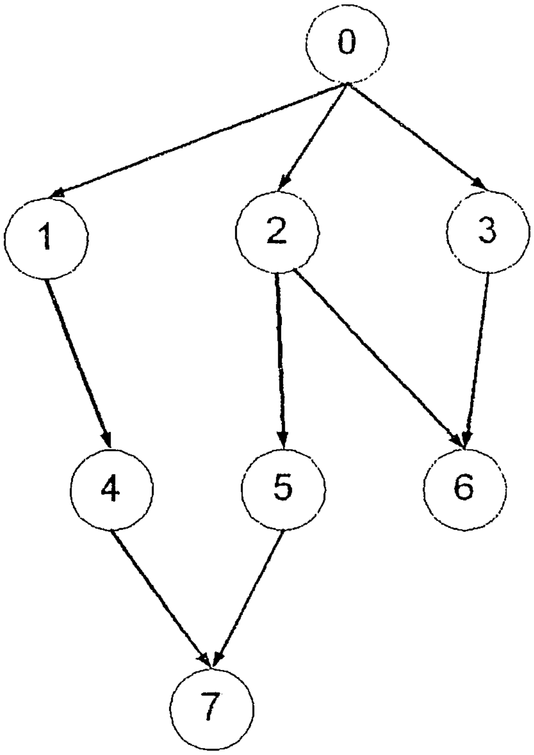

Figure 3.11: Randomly Generated Task Graph (Graph #1)

3.3 Pipelined Scheduler

The functionality described in this sub-section occurs in the "Scheduling" block in Figure 3.1. The effectiveness of a given architecture alternative is evaluated by obtaining a pipelined schedule that is executed in the algorithm's scheduling phase. A pipelined schedule with a minimum period P is an assignment of completion times of all tasks, $F(v)$, such that for all tasks v in the system $0 <= F(v) <= P$. For a task v

with a data dependence $e=(u, v)$, where u is a parent of v, the schedule time of v must honour the following equation:

$$F(v) \geq F(u) + v_{exec} + C(u, v) \qquad \text{if } PSMatch(u, v)$$

$$F(v) \geq v_{exec} \qquad \text{otherwise}$$

where,

v_{exec} = execution time of task v

$C(u, v)$ = the overall communication time between mapped tasks u and v (0 if no communication)

$PSMatch(u, v)$ = true if tasks u and v are located on the same pipelined stage, false otherwise

This definition requires that a task will not commence execution before receiving the required data from all parent tasks. The scheduler takes communication delays and resource usage into account when assigning tasks to processing elements. The algorithm utilizes the established RECOD retiming transformation to divide the task graph into multiple pipelined stages [6, 7].

The RECOD retiming transformation divides a task graph into multiple pipelined stages by inserting a cut-line which separates two tasks (parent and child) and defines the separation between two pipelined stages. The location of this cut-line is dependent on the parent and child's current pipeline stage, the parent and child's mapping, the length of the constraining path of the parent and the amount of information passed between the parent and child.

Initially, the RECOD transformation is repeatedly used to divide the task graph into a maximum number of pipelined stages. After performing the RECOD retiming transformation, the scheduler attempts to assign completion times to all tasks to satisfy device performance requirements. If unsuccessful, the algorithm attempts to improve the task allocations iteratively in order to minimize device communication overhead. A flow diagram of the scheduler's operation is provided in Figure 3.12.

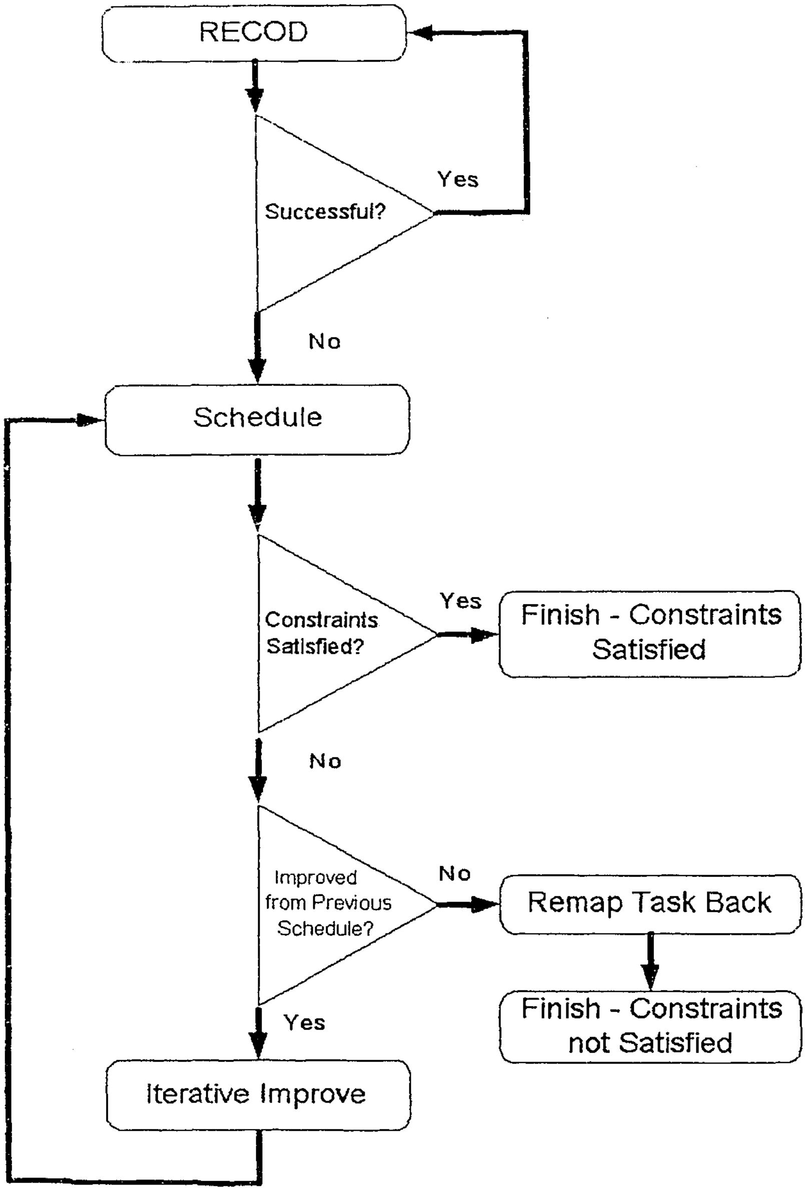
RECOD
Successful?
Yes
No
Schedule
Constraints Satisfied?
Yes
Finish - Constraints Satisfied
No
Improved from Previous Schedule?
No
Remap Task Back
Finish - Constraints not Satisfied
Yes
Iterative Improve

Figure 3.12: Hypercube Co-Synthesis Scheduler

For the purposes of scheduling ("Scheduling" block in Figure 3.1), each task from the input task graph is assigned to one of the three sets, *m*, *n* and *p*. In order to specify the first set, *pathLoadChild*, *pathLoadParent* and *pathLoad* variable values are assigned to each task *v*. These values are defined as:

$$pathLoadChild(v) = v_{exec} + \max\{PSMatch(v, children(v)) * pathLoadChild(children(v))\}$$

$$pathLoadParent(v) = v_{exec} + \max\{PSMatch(v, parents(v)) * pathLoadParent(parents(v))\}$$

$$pathLoad(v) = pathLoadChild(v) + pathLoadParent(v) - v_{exec}$$

where,

PSMatch(u, v) = 1 if tasks u and v are located on the same pipelined stage, 0 otherwise

children(v) = set of the child tasks of v

parent(v) = set of the parent tasks of v

As a result, a task v's *pathLoad* is the value of the heaviest loaded path containing task v.

The first set of tasks m is defined as:

$$\exists\ m\ AT \rightarrow (\ pathLoad(m) == max\{pathLoad(AT)\}\ )$$

where,

$\exists$, $\rightarrow$, and $==$ represent *there exists, such that* and *equivalent* respectively

AT is the set of all tasks

Set m is the set of tasks that are located on the constraining path and are scheduled first. Set w consists of all the tasks having a path to m. The second set n, is defined as: $n = w - m$.

Set n consists of tasks that are ancestors of all the tasks of set m excluding the members of set m.

The final set p, is defined as: $p = AT - m - n$. Set p consists of the remaining ungrouped tasks and will be scheduled last.

These sets are used to prioritize the tasks for scheduling. The constraining path is the longest path through the graph in terms of the execution times of its tasks. When scheduling a task of set m that has unscheduled parents, the parents are scheduled first. Dividing tasks into these set configurations prioritizes tasks located on the constraining path. The tasks on the constraining path are most likely to adversely affect the target device performance and so the motivation is to schedule them first. If a system schedule is obtained that meets the performance requirements, the schedule function exits successfully, otherwise the scheduler attempts to improve iteratively. Excessive communications is a potential problem in hypercube topologies as they can slow down overall system performance by causing PEs to wait for data. When two tasks are located far from each other in the architecture, communication between them may require multiple hops. In order to alleviate this potential problem, the algorithm includes an iterative improver (as labeled in Figure 3.12) whose goal is to refine the task mapping. The value *commLevel* for each task is defined as:

$$commLevel(v, PE) = \sum Com(v,x) * Nh(v,x) + \sum Com(u,v) * Nh(u,v)$$

where,

$Com(u, v)$ = communication time from task u to v across a communication link

$N(u, v)$ = number of links (hops) that must be traversed to communicate between tasks u and v

$x = children(v)$

$u = parents(v)$

A task q is selected for re-mapping based on the following equation:

$$q = \arg\max \left[\frac{execTime(v, execLoc(v)) + commLevel(v, execLoc(v))}{execTime(v, APE) + commLevel(v, APE)} \right]$$

where,

$exec\ Loc(v)$ = the execution location (PE) of task v

$exec\ Time(v,\ PE^0)$ = the execution time of task v on PE^0

APE = the set of all processing elements

The above equation selects task q for mapping based on total performance gain. Task q is subsequently remapped to a new processing element to improve the overall execution and communication times. This process is repeated as long as the scheduler reveals continued improvements in the overall solution. When this iterative process does not provide any further improvements, the scheduler exits unsuccessfully. This technique was incorporated in order to help minimize any unnecessary communications.

3.4 Device Expansion

The functionality described here is executed as part of the "Add Processing Element" block of the design flow given in Figure 3.1. This co-synthesis section is responsible for expanding the current system architecture to include an additional processing element (PE). The PE type can be a CPU or an application specific integrated circuit (ASIC). The primary focus of this function is to determine the most effective type of PE to be added. The addition of a PE should provide a maximum performance enhancement with a minimum additional hardware area. If the addition of a PE causes a violation of the hardware area constraint, the function exits unsuccessfully. Otherwise, the function will add a new PE and exit successfully. In order to make a knowledgeable decision on which PE to add, the algorithm estimates the ratio of expected performance improvement due to the increase in hardware area for all the available options. For these calculations the variable, *speedupSW* is included as a preliminary estimate of performance improvement due to the addition of another CPU and defined as:

$$speedupSW = TSW_{exec}(AT) * (1 - TP/(TP + 1))$$

where,

$TSW_{exec}(v)$ = the total execution time of all tasks mapped to SW from set v

TP = the count of total CPUs in the current system architecture

The variable *ratioSW* is defined that estimates the performance to hardware area ratio for adding a CPU and is defined as:

$$ratioSW = ExL * speedupSW / PAC$$

where,

PAC = the area cost for the addition of another CPU

ExL = the expected load of the new processor

ExL is provided to weight the decision of adding a new processor based on the expected load that processor would receive. ExL is defined as:

$ExL = 0$ if $CST <= TP$

$ExL = CST/(k*TP)$ if $CST > TP$ and $CST < k*TP$

$ExL = 1$ otherwise

where,

CST = the count of tasks currently mapped to software

As an estimation, it is assumed that if the current system architecture contains k times as many tasks mapped to software as there are CPUs, then the newly added processor will be provided with a full task load. The value k is user set. These equations were selected to quickly approximate the expected amount of speedup obtained by adding an additional CPU to the system.

With respect to the speedup and ratio factors for adding new ASICs, a task set v is declared as the set of all tasks currently mapped to software. The *speedupHW[v]* and *ratioHW[v]* array variables are then calculated for the set of tasks in v. Each individual *ratioHW* value represents an expected improvement factor for moving the given task from its current location in software to an ASIC implementation. The *speedupHW[v]* represents the difference between the execution time of task v in software and in hardware. In order to define *ratioHW* the variables *tCouldBen* and *tWillBen* are created. The *tCouldBen[v]* variable is defined as the count of tasks having an identical hardware configuration pattern as task *v*. The *tWillBen[v]* variable is defined as the count of tasks having an identical hardware configuration pattern as task *v* and is currently mapped to software. The *ratioHW[v]* is defined as:

$$ratioHW[v] = \frac{speedupHW[v]}{A(v)} + \left(1 - \frac{speedupHW[v]}{A(v)}\right) * \left(\frac{tWillBen[v] - 1}{tCouldBen[v]}\right)$$

where,

$A(v)$ = the area cost for the hardware implementation of task v

The above formulation encourages the selection of hardware solutions that can be reused by multiple tasks. All of the ratio values (both hardware and software) obtained are compared and the maximum value is used to decide the PE type to be employed in the expansion. When the PE being added is an ASIC (PE_{ASIC}), the function assigns a maximum number of tasks with matching configuration patterns to the newly added ASIC, given that the following constraints are met:

$$THW_{exec}(PT(PE_{ASIC})) \leq timCons$$

where,

$timCons$ = the user specified device timing constraint

$PT(PE)$ = the set of tasks allocated to processing element PE

$THW_{exec}(v)$ = the total execution time of all tasks mapped to HW from set v

When the added PE is a CPU, the function must decide which tasks will be initially allocated to the new CPU. The task to be allocated to the new processor P_{new}, is selected from the processor identified by the variable P_{donor} in the following equation:

$$P_{donor} = \max(TSW_{exec}(PT(AP)))$$

where,

AP is the set of all processors

The above equation ensures that the donor processor would always be the most heavily loaded. The task with the lowest communication overhead with respect to the other locally allocated tasks is selected for remapping from the donor process. This is intended to minimize the communication overhead and simplify the iterative process for the scheduler. After this task has been reallocated to the new PE, the selection of a donor processor and a task is repeated until the new CPU load reaches the average processor load. To accomplish this, the variable $loadAverage = TSW_{exec} / TP$ is defined, where TP is the total number of processors. This process is repeated until an allocation is obtained such that the difference between the load on the new processor and the average load is a minimum.

While adding new hardware, the algorithm tracks the overall area of the device to ensure an accurate prediction of the final system cost. The hardware area overhead associated with each communication interface located at each PE node is recorded, in addition to the cost of each communication link.

3.5 Communication Link Integration

The functionality described here is executed in the “Synthesize Communication Links” block of the design flow provided in Figure 3.1. This section of the co-synthesis algorithm is responsible for connecting all the system PEs and assignment of communication links to form a hypercube topology. To limit the communication overhead. the processing elements with high levels of inter-communication are placed as close as possible in a hypercube topology. An array of communication coefficients is defined, *comCoeff[TPE]* (where TPE = the count of total processing elements), for each processing element. The *commCoeff* for processing element PE_n with respect to PE_m is defined as:

$$PEArray[PE_n].comCoeff[PE_m] = \sum Com(PET(PE_n), PET(PE_m))$$

where,

PET(PE) = the set of tasks currently allocated to *PE*

PEArray = the array of all processing elements

In order to clarify the process of assigning specific locations to each processing element within the framework of a hypercube architecture, the established binary naming convention for labeling nodes is utilized [29]. The binary naming convention in a 3D hypercube is shown in Figure 3.13. This naming convention is convenient for message routing and the number of communication hops between two nodes can be quickly determined by the number of bit-wise mismatches between the binary identifiers for each node.

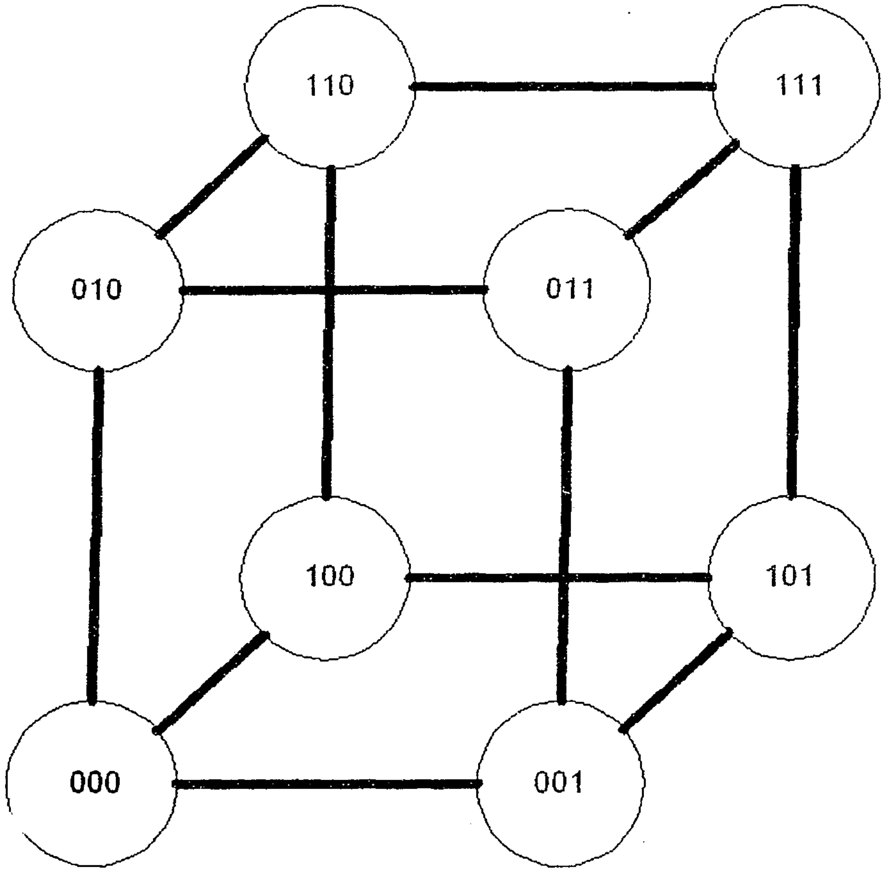

Figure 3.13: Binary Naming for Hypercube Nodes

Initially, the first CPU is placed at location 000. Subsequently, each location (h*N*) is filled by the PE named in the variable *selectedPE* by the following definitions:

$$\exists\ pe \text{ in } APE \rightarrow he(hN,\ BL(pe))$$

$$selectedPE = \arg\max(\sum^{n \in pe} PEArray[n].comCoeff[UPE])$$

where,

APE = the set of all processing elements currently assigned to a specific hypercube location

BL(pe) = the binary number that processing element *pe* is assigned to

he(hN, hM) = the hypercube edge connecting binary location *hN* to binary location *hM*

UPE = the set of all processing elements currently unassigned to a specific hypercube location

The next PE to place in the hypercube is selected from the set of unassigned PEs. The PE is selected such that it has the largest communication coefficient with respect to the already assigned PEs that will be its neighbour. This process is repeated until all the PEs are allocated to hypercube nodes. This heuristic was selected to minimize the occurrence of long communication delays.

The co-synthesis algorithm does not require the hypercube architecture to be perfectly symmetrical. If all the processing elements are not a power of two, some of the PEs will have fewer communication links than the others. This is generally considered undesirable as one of the advantages of hypercube architectures lies in its innately fault-tolerant topology. Consequently, if the total number of processing elements is not a power of two, an additional communication link to compensate for the shortage in the most recently added PE is introduced. This communication link connects the PE in the highest binary location (*finalPE*), with a *destinationPE*. In order to determine the *destinationPE*, the set of processing elements, *neigh* is defined as the set of PEs that share a neighbour with the *finalPE*. The processing element *destinationPE*, is selected to be the task identified by:

$$\max(PEArray[finalPE].comCoeff[n]) \quad \text{while } n \in neigh$$

The above equation was selected to connect the final PE with the compatible PE with which it communicates most often. To illustrate this example consider Figure 3.14, where a 5-node hypercube is provided and the final communication link will connect the PE at node (100) to the shaded PE with which it communicates most. As previously mentioned, one of the advantages of hypercube topologies is its inherent fault tolerant capabilities. The motivation for the above heuristic is to eliminate the potential catastrophic failure that could occur if the communication link connecting node (000) and (100) fails. This heuristic preserves fault tolerance in non-symmetrical hypercube systems.

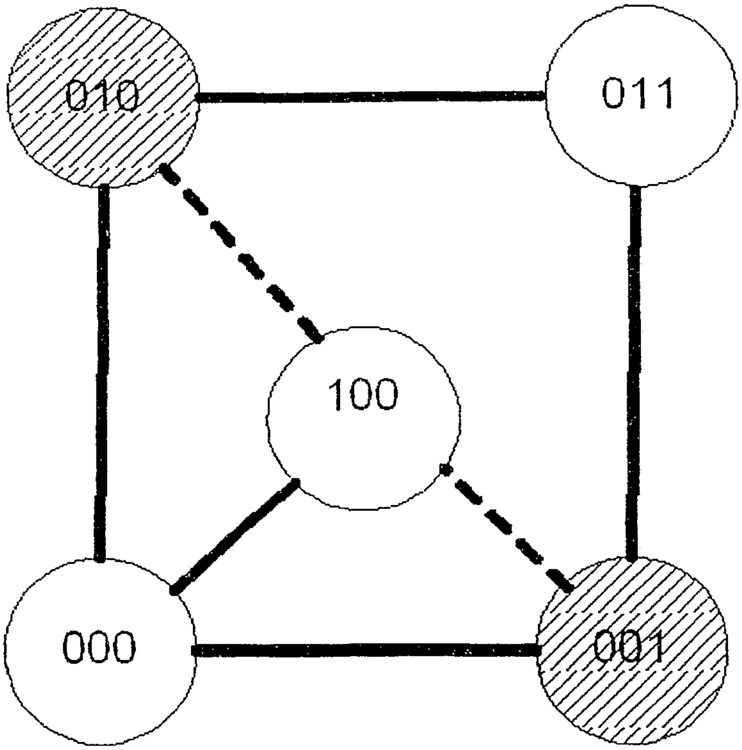

Figure 3.14: Non-Symmetrical 3-D Hypercube

In summary the approach consists of six phases. The first three are performed once, and the final three are repeatedly executed until a final device architecture has been generated. The first phase is responsible for clearly defining the device's functional and non-functional requirements. The second phase is concerned with converting the functional requirements into task graph form and gathering all of the relevant data needed in the later phases. The third phase adds fault detection tasks to the task graph, thus facilitating low overhead fault tolerance. The fourth phase involves evaluating the device's performance and determining if it meets non-functional performance requirements. The fifth phase is concerned with adding a processing element to the system. Finally, the sixth phase is responsible for synthesizing all of the device's communication links.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTAL RESULTS

4.1 Parallel MPEG-2 Decoding

An effective demonstration of the capabilities of the co-synthesis algorithm would have to involve an application that is computationally intensive. Additionally, for demonstrative purposes it is beneficial to implement a device that is easily understood. A multiple PE embedded device was implemented, which is responsible for decoding 16 MPEG-2 video streams and simultaneously comparing their decoded images to predefined data. Such a device could be used in autonomous navigation or security systems. The device was fully specified in C language encompassing 9500 lines of code. A task graph representation was obtained by analysis of the specification and is provided in Figure 4.1. The functionality of the system is expressed as a graph of 22 tasks. The design space of the application has been tested in terms of device area and timing constraints in order to evaluate the performance of the presented approach. The test and experimental results are provided in Figure 4.2. The results of an optimal technique that exhaustively attempts each possible combination is also included. This is useful for demonstrating the quality of the algorithm and its results as compared to the optimal solution.

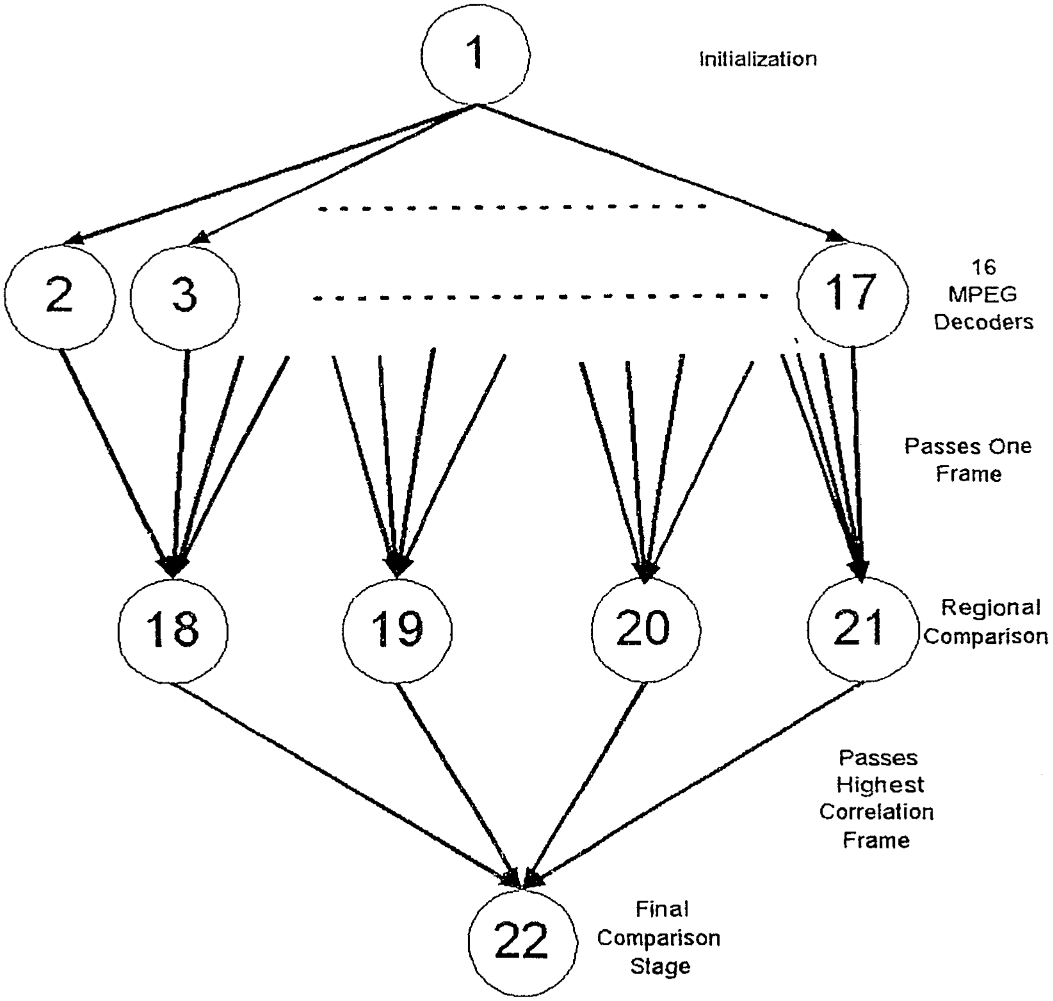

Figure 4.1: Parallel MPEG Decoding: Functional Task Graph

A set of hardware and software implementation data is provided to the algorithm. In order to establish the software execution times for each task, the C language based specification was profiled on a Pentium II 450 MHz CPU. The algorithm library can consist of many CPU types however, for simplicity and practicality only the Intel Pentium II 450 MHz CPU was included for this case study. Each of the tasks were fully implemented in hardware and profiled for performance and hardware area requirements on an Altera FLEX10KE FPGA. The only exception is the MPEG-2 decoding tasks, where the pre-existing IP core from Amphion Semiconductor was utilized for the hardware implementation. Table 4.1 provides information regarding the quantity of each type of PE used for each of the design constraints studied.

The co-synthesis algorithm provides support for fault tolerant communications by generating the target device into a hypercube architecture. By using the group-based fault tolerance techniques (GBFT) previously outlined, a fault tolerant embedded device has been synthesized. The GBFT algorithm adds assertion and duplicate/compare tasks to the task graph. Upon detection of a failure, the assertion/compare task signals one of the spare PEs to commence execution of the failed task. The type and number of spare PEs are defined by the user. It should be noted that in order to compare the algorithm with the optimal case the GBFT section of the algorithm had to be disabled. The GBFT algorithm would substantially increase the total number of tasks in the task graph. The optimal approach is extremely computationally intensive and unable to generate results for larger task graphs within a reasonable amount of time. Although node fault tolerant versions of this case study have been simulated, the data presented illustrates results obtained without the use of the GBFT portion of the algorithm. Further analysis that includes the error detection tasks added by the GBFT portion of the algorithm is provided in section 4.2.

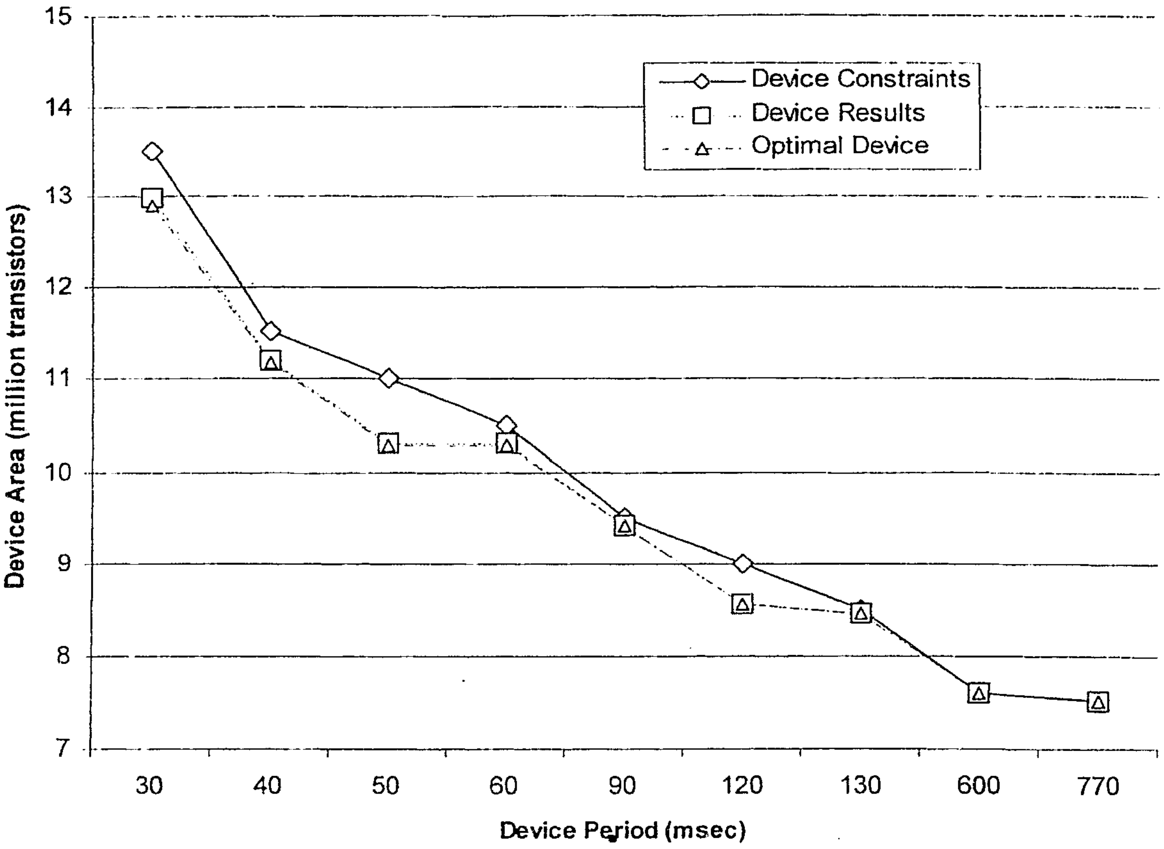

Figure 4.2: Design Space Exploration of Parallel MPEG Decoding

The results provided in Figure 4.2 demonstrate the ability of the method to synthesize embedded devices with varying design constraints. A comparison of hardware area to performance trade-offs can be very useful to an embedded engineer in order to examine differences between design alternatives. The results from using an exhaustive (optimal) mapping technique is also provided. The exhaustive approach attempts all the possible combinations and is very computationally intensive. Comparison results from the co-synthesis algorithm with those obtained by the optimal technique revealed that the algorithm produces embedded devices with 96.25% of the performance obtained by the optimal technique on average.

Additionally, the embedded devices generated by the algorithm only utilized 0.62% (average) more hardware than the devices obtained by the optimal method.

Table 4.1: Processing Element Utilization of Design Space for MPEG Decoding

Area Constraint (millions of transistors)	Timing Constraint (msec)	Initial HW - Task 1	MPEG-2 Decoding Cores Tasks 2-17	HW – Reg. Comparison Tasks 18-21	HW - Final Comparison Task 22	Total PEs	Exec. Time (msec)	Optimal case Execution Time
13.5	30	1	6	1	1	10	35.2	~24 hrs
11.5	40	0	4	1	1	7	14.8	~24 hrs
11	50	0	3	1	1	6	11.2	~24 hrs
10.5	60	0	3	1	1	6	13.5	~24 hrs.
9.5	90	0	2	1	1	5	7.9	~24 hrs
9	120	0	1	1	1	4	4.2	~24 hrs
8.5	130	0	1	1	0	3	3.4	~24 hrs
7.6	600	0	0	1	0	2	1.6	~24 hrs.
7.5	770	0	0	0	0	1	0.8	~24 hrs

Table 4.1 provides detailed information regarding the configuration of each solution that the co-synthesis algorithm produced. Given the design constraints provided in the first two columns, the algorithm generated a device that could meet specifications. The next four columns (3-6) provide the count of various ASIC circuits present in the final synthesized system. The total PEs column provides a count of the total number of PEs present in the given device. The count of CPUs in the table have not been included as there is only one CPU for each case. The execution time column shows

the length of time the algorithm took to execute on a Pentium IV in order to synthesize the device. The final column provides the approximate execution times from performing exhaustive co-synthesis. Figure 4.3 provides the final device architecture for the first test case provided in table 4.1. Figure 4.4 provides the final device architecture for the second test case. The architecture for the fifth test case is provided in Figure 4.5.

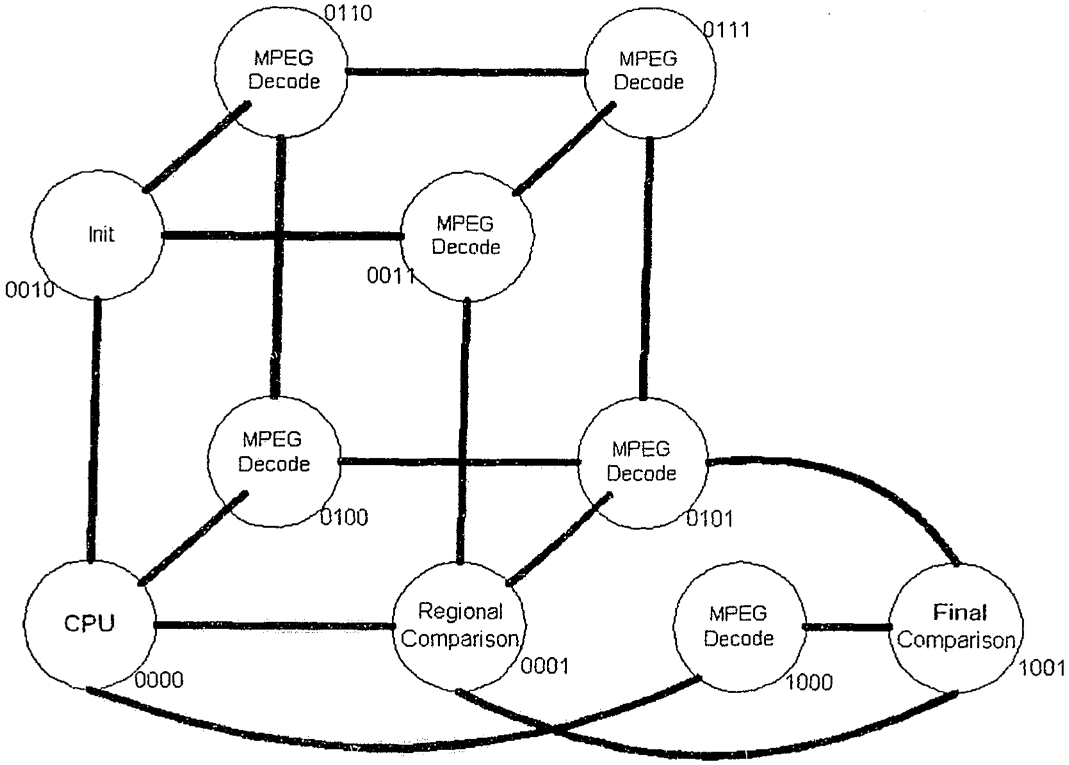

Figure 4.3: Parallel MPEG Decoding: Architecture for 1st Test Case

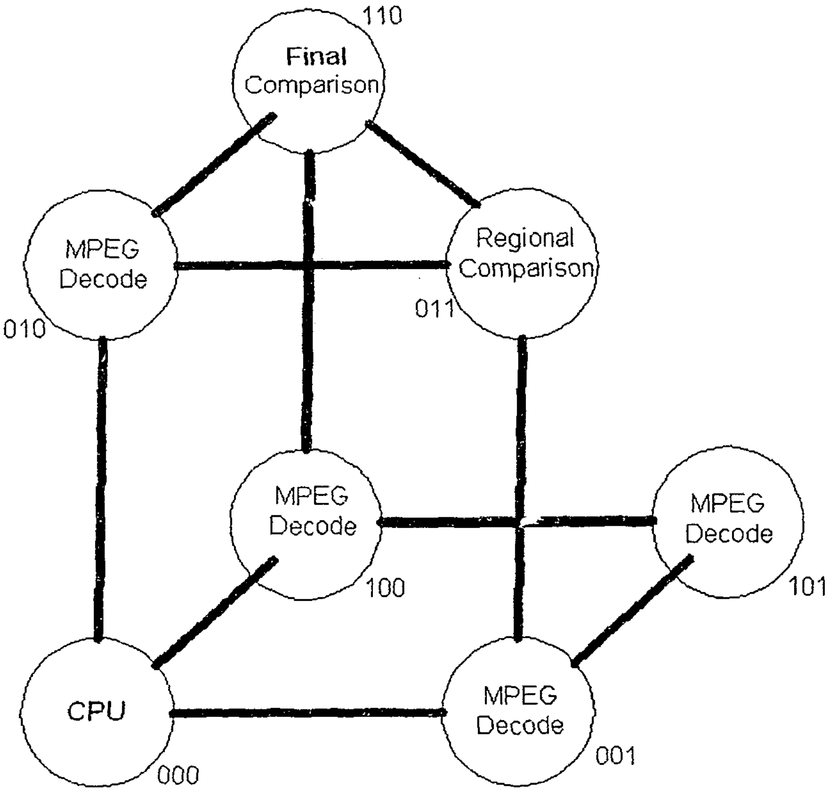

Figure 4.4: Parallel MPEG Decoding: Architecture for 2nd Test Case

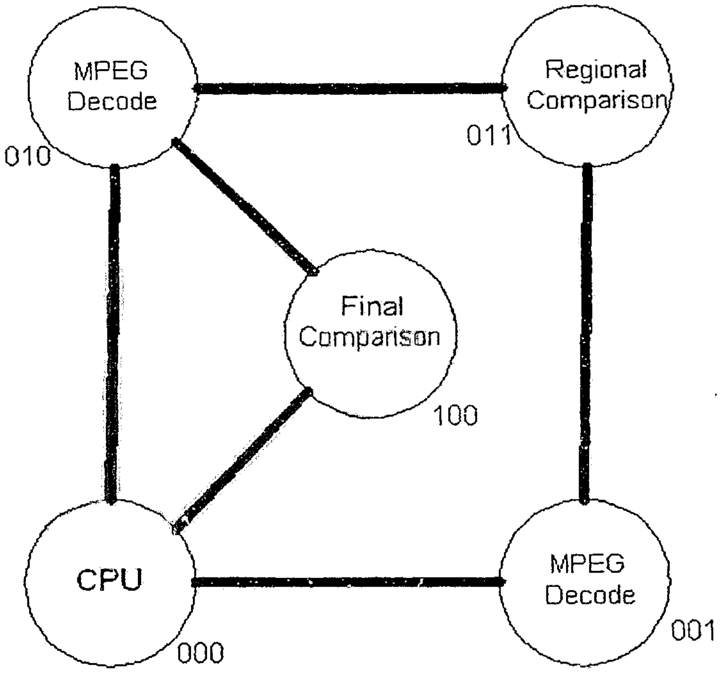

Figure 4.5: Parallel MPEG Decoding: Architecture for 5th Test Case

4.2 Parallel Block Matching

An effective demonstration of the capabilities of the co-synthesis algorithm would have to involve an application that is computationally intensive. Additionally, for demonstrative purposes it is beneficial to implement a device that is easily understood. In the previous case study a device has been implemented that utilizes 16 MPEG decoders. MPEG decoding is the process by which a compressed video file is uncompressed for the purpose of viewing or accessing the raw image data. For the second case study, the block matching algorithm has been implemented. MPEG and other video compression formats are designed for fast transmission / easy storage of video data. In order for such a format to be useful, the decoding process must be relatively fast. In order to efficiently compress the video data for easy decoding, a computationally intensive encoding algorithm must be run. Block matching is the most time consuming component of MPEG encoding. Block matching analyzes the macroblocks from the input image and compares them with the surrounding area on the reference image. By analyzing all possible locations, an exhaustive block matching algorithm will generate motion vector data indicating the movement of a given macroblock from one image to the next.

A multiple PE embedded device responsible for performing the block matching algorithm has been implemented. Such a device could be used for any application involving the acquisition of video data. The device is fully specified in C language. A task graph representation is obtained by analysis of the specification and is provided in Figure 4.6. The functionality of the system is expressed as a graph of 22 tasks. The group based fault tolerance (GBFT) techniques previously outlined have been used and the resultant task grouping is provided in Figure 4.7. The GBFT grouping results in the addition of the assertion tasks shown in Figure 4.8. The final resultant task graph after GBFT has been performed and each group has merged to become a single task is provided in Figure 4.9. Further discussion of how the GBFT algorithm groups tasks together is provided in section 3.2.4.

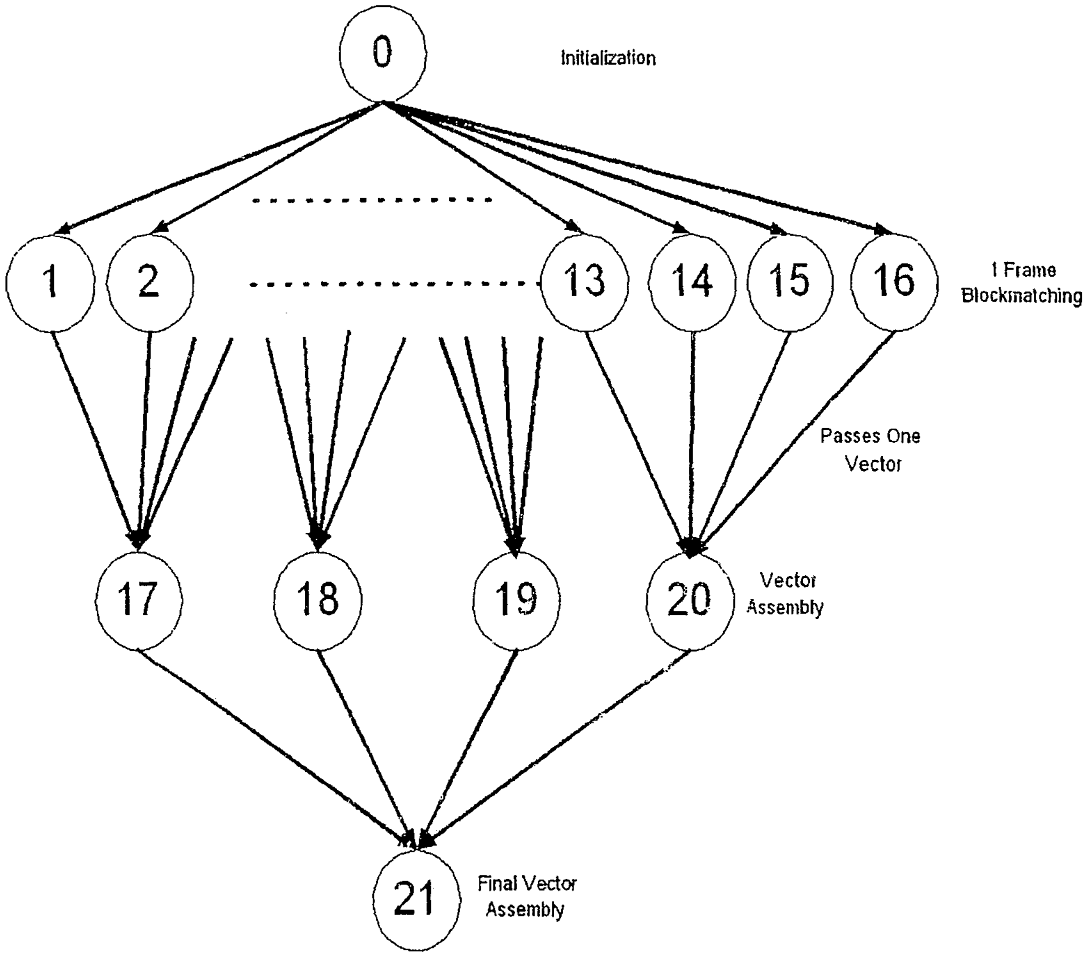

Figure 4.6: Block Matching: Functional Task Graph

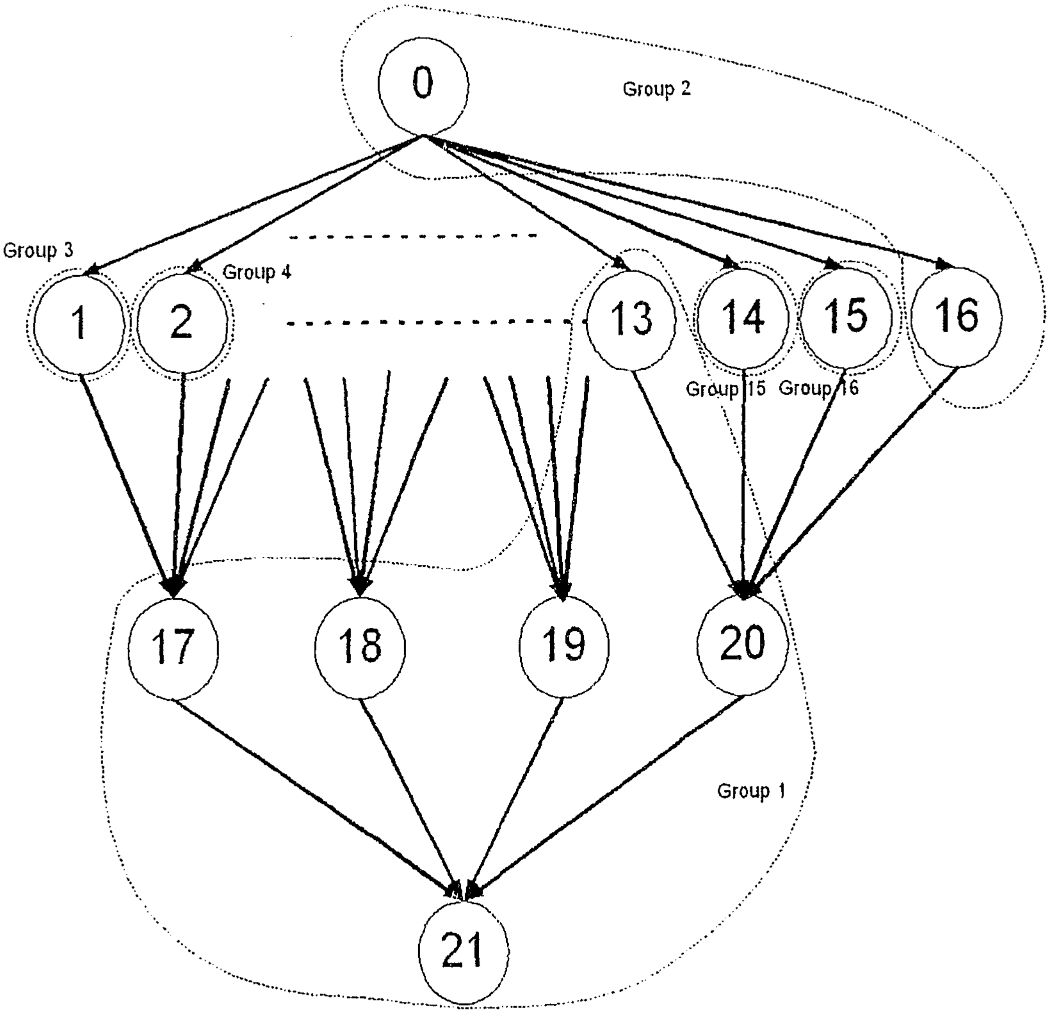

Figure 4.7: Block Matching Task Graph: GBFT Grouping

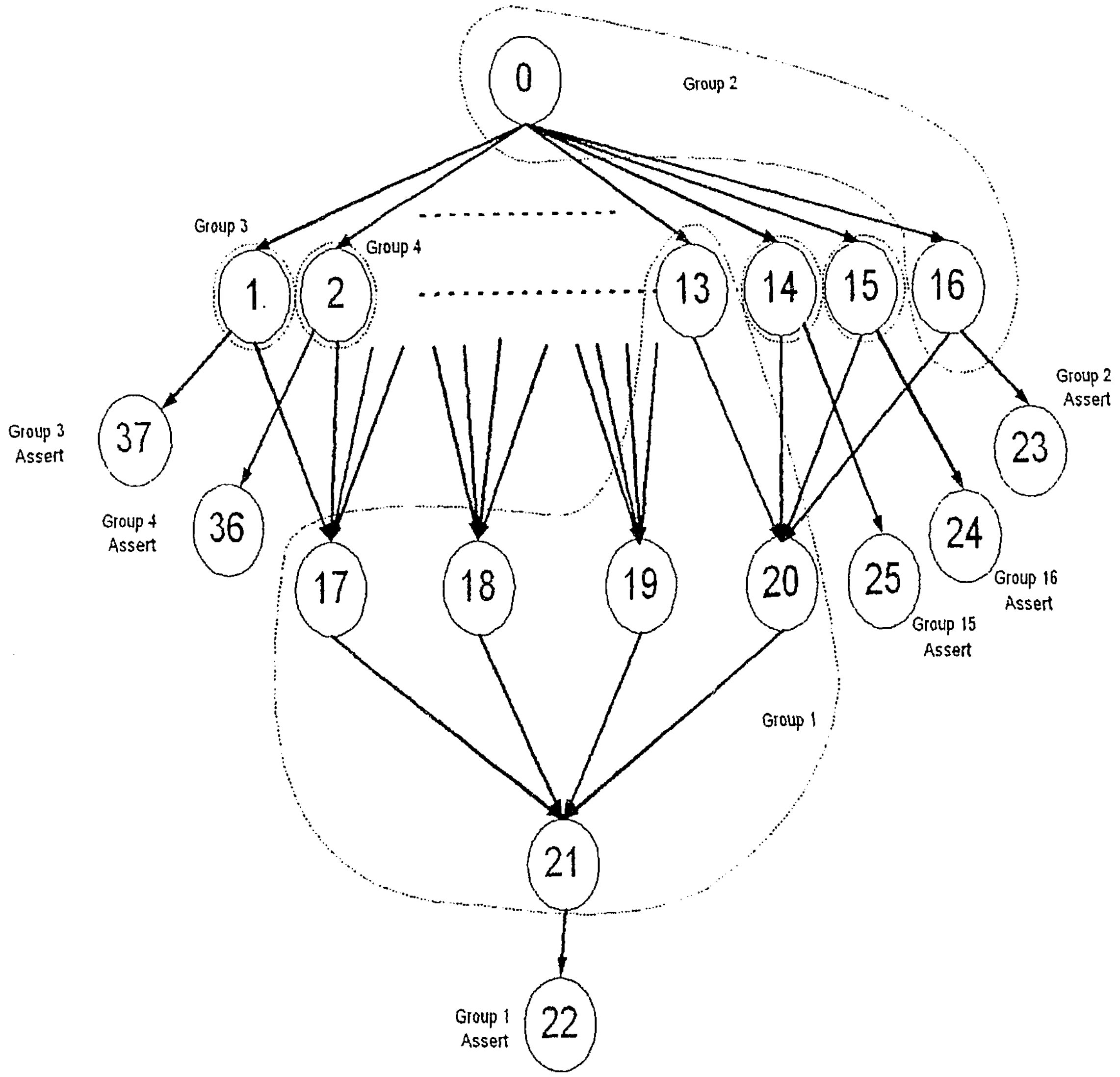

Figure 4.8: Block Matching Task Graph with Added Assertion Tasks

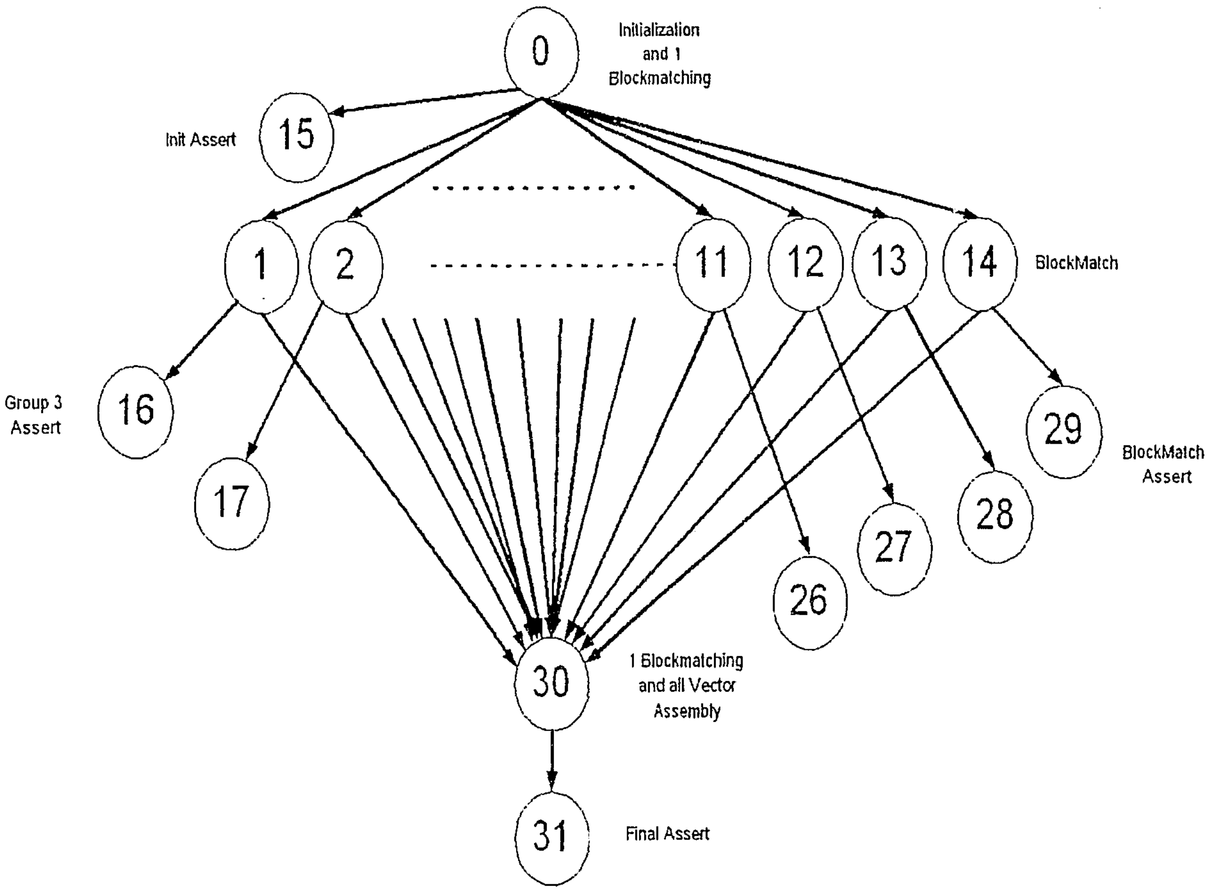

Figure 4.9: Block Matching: Final Resultant Task Graph

The most complex task in the device outlined is the block match task. In order to obtain pertinent hardware performance and area information for all tasks, the functionality needs to be implemented in hardware. This task has been implemented with the design provided in Figure 4.10. Yang et al. have presented an entire motion estimation architecture which utilizes a sub-circuit responsible for error calculations that is similar to the implementation in Figure 4.10 [50]. Dutta and Wolf have presented a flexible motion estimation architecture [17] based on the research of Yang et al. In the design presented, the diff circuit calculates the absolute difference between the two input values. The absolute difference between the corresponding pixel values of the two input images is the error at the given point. The adder circuit adds the current error to the previously calculated errors to sum the overall error for a given vector. The comparison circuit compares the two input values and if the new value is lower than the old value, the circuit outputs the new value and latches the current I and J values into an internal register. These I and J values represent the current motion vector. The control unit tracks the current state of the circuit and signals all sub circuits, registers and outputs accordingly. A formal description of the functionality of the device is provided in the VHDL code in Appendix A.

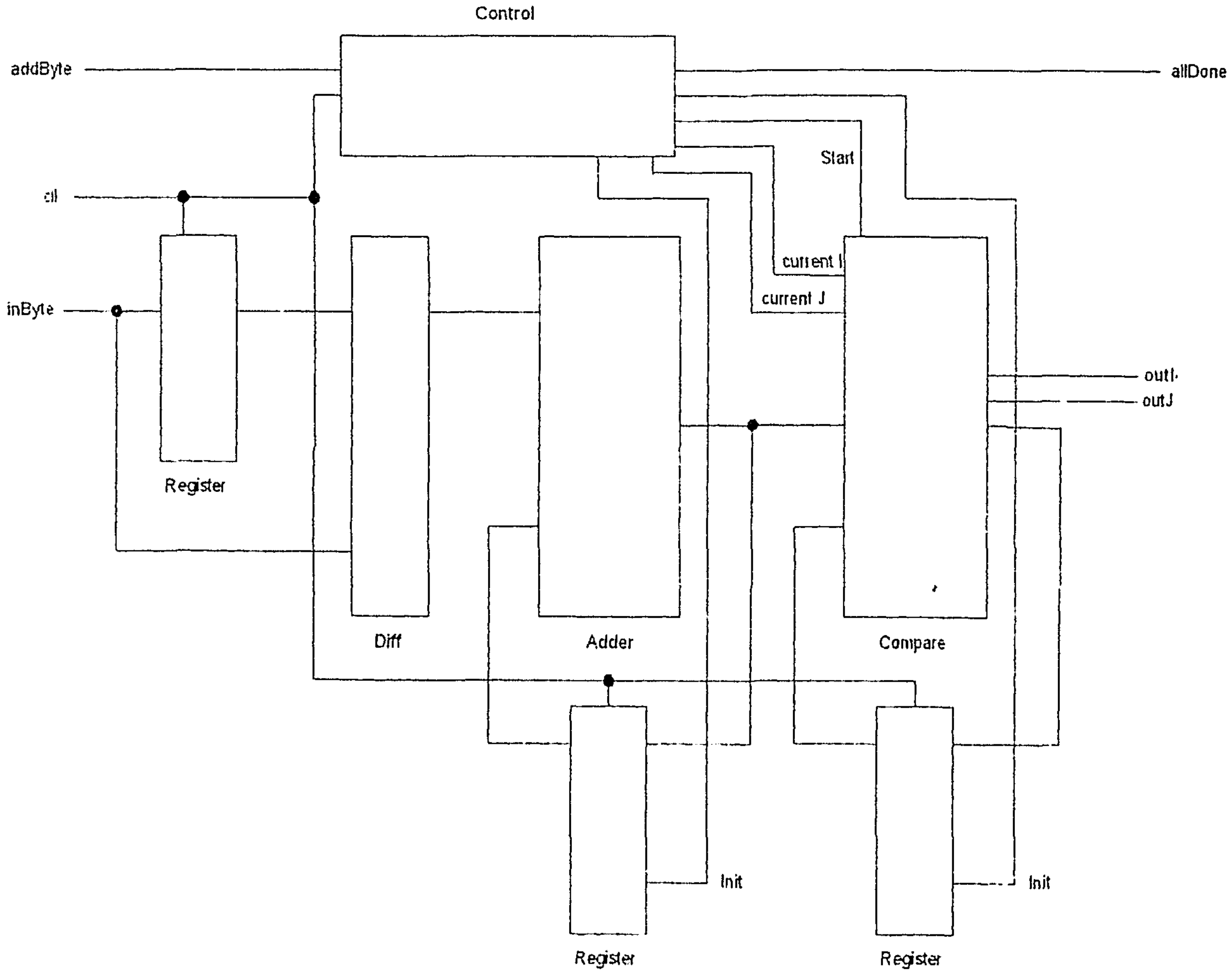

Figure 4.10: Block Match Circuit Design

This block match circuit operates using a state machine which is administered in the control unit and is provided in Figure 4.11. State 0 is the initial state when the circuit is waiting for input. In state 1 the circuit has just received an even numbered count of inputs and as a result, the adder register is signaled to load the adder circuit generated value. In state 2 the circuit has just received an uneven number of inputs. If the circuit has received 512 input values, the circuit transitions to state 3 and signals the compare sub-circuit to begin execution, otherwise the circuit returns to state 1. In state 3 the compare register is signaled to load the compare circuit generated error value. In state 4 the adder register is initialized and the incoming data is loaded into the input register.

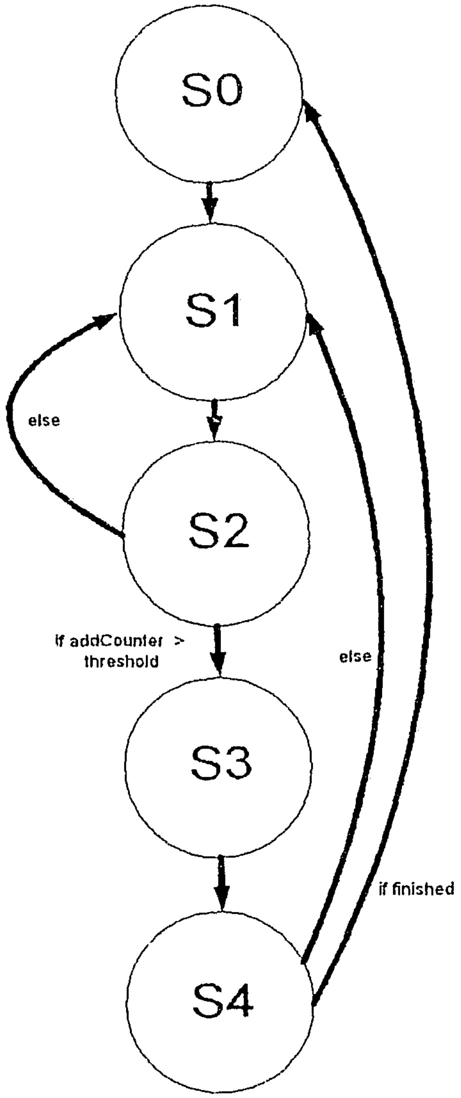

Figure 4.11: Block Match State Machine

The design space of the application in terms of device area and timing constraints has been tested in order to evaluate the performance of the approach. The test and experimental results are provided in Figure 4.12. A set of hardware and software implementation data was provided to the algorithm. In order to establish the software execution times for each task, the C language based specification was profiled on an ARM7TDMI processor. The algorithm library can consist of many CPU types, however, for simplicity and practicality only the ARM7TDMI has been included for this case study. Each of the tasks were fully implemented in hardware and profiled for performance and hardware area requirements on a Xilinx VirtexE XCV2000E FPGA. Table 4.2 provides information regarding the quantity of each type of PE used for each of the design constraints studied.

The co-synthesis algorithm provides support for fault tolerant inter-PE communications by generating the target device into a hypercube architecture. By using the group-based fault tolerance (GBFT) method, fault tolerant embedded devices have been synthesized. The GBFT algorithm adds assertion and duplicate/compare tasks to the task graph. Upon detection of a failure, the assertion/compare task signals one of the spare PEs to commence execution of the failed task. The type and number of spare PEs are defined by the user.

In the previous case study the results were compared with those of an exhaustive technique. The exhaustive technique is very computationally intensive. On the 22 node task graph provided in the previous case study, the exhaustive technique required approximately 24 hours of execution time. The parallel block matching case study consists of 32 nodes (after GBFT) and as a result the exhaustive technique is far too computationally intensive to gather comparative results within a reasonable amount of time. As a result, in Figure 4.12 only the device constraints have been provided to our program and the resultant output provided by the algorithm have been included.

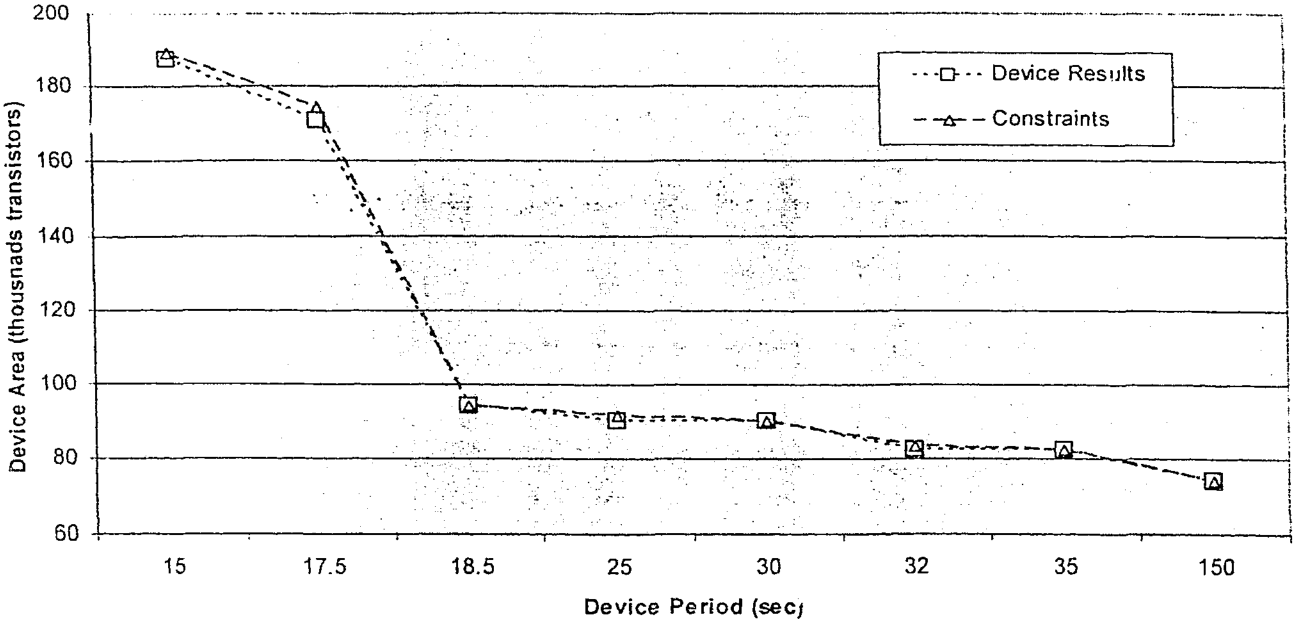

Figure 4.12: Design Space Exploration of Parallel Block Matching

The results provided in Figure 4.12 demonstrate the algorithm's ability to synthesize embedded devices with varying design constraints. A comparison of hardware area to performance trade-offs can be very useful to an embedded engineer in order to examine differences between design alternatives.

Table 4.2: Processing Element Utilization of Design Space for Block Matching

Area Constraint (thousands of transistors)	Timing Constraint (sec)	CPU cores	Block Matching Cores	Block match Assert Cores	Final Assert Cores	Total PEs	Exec. Time (msec)
189	15	2	3	1	1	7	342.4
174	17.5	2	2	1	1	6	233.7
94	18.5	1	2	1	1	5	144.0
91	25	1	2	1	0	4	112.5
90	30	1	2	1	0	4	153.7
83	32	1	1	1	0	3	100.3
82	35	1	1	1	0	3	112.5
74	150	1	0	0	0	1	2.4

Table 4.2 provides detailed information regarding the configuration of each solution that the co-synthesis algorithm produced. Given the design constraints provided in the first two columns, the algorithm generated a device that could meet specifications. The next four columns (3-6) provide the count of various ASIC circuits present in the final synthesized system. The total PEs column provides a count of the total number of PEs present in the given device. The count of initialization cores and final cores have not been included as there are none implemented in hardware for each case. The final column provides the length of time the algorithm took to execute in order to synthesize the device.

In order to further demonstrate the efficacy of the approach a prototype device has been constructed based on the design generated by the hardware-software co-synthesis algorithm. A device based on the third set of design constraints provided in table 4.2 has been constructed. The algorithm was provided with an area constraint of 20000 gates in excess of the area of the initial CPU (74209 gates). The algorithm was also provided with a performance (period) constraint of 18.5 seconds. After co-synthesis, the algorithm generated a final device architecture as seen in Figure 4.13. The design consists of a single CPU, two block matching PE circuits. one block match assert circuit and one final assert circuit. Additionally, in order to support fault tolerance one additional spare block matching circuit and two repeater circuits have been included in the design.

The algorithm predicted a final device area of 19683 gates in excess of the initial CPU area (74209). One block matching processing element and two repeater processing elements were added to support fault tolerance. A single block matching circuit has been synthesized and determined to utilize 3451 gates. The addition of three more processing elements also results in the addition of six new communication links. Each additional link results in the addition of two communication interfaces. Each additional communication interface has been synthesized and determined to utilize 354 gates. Finally, in order to support software to hardware communications on the ARM rapid prototyping platform, additional interface circuitry is required. This circuitry was synthesized and determined to utilize 2866 gates. The algorithm also predicted a device period of 18.49 seconds.

The sum of all of the hardware area predictions are as follows:

	19863	Gate count of initial hardware in excess of initial CPU
	74209	Gate count of initial CPU
	3451	Gate count of 1 additional block match circuit
	4248	Gate count of 12 additional interface circuits (12 * 354)
+	2866	Gate count of ARM circuitry required for HW/SW comm.
=	104637	Total gate count

The entire device represented by Figure 4.13 was implemented in over 2500 lines of VHDL and C code. A listing of the code is provided in Appendix A. Synthesis of the final device's non CPU hardware revealed an actual gate count of 28368 transistors. Adding the total gate count of the initial CPU (74209) yields a final device actual gate count of 102577 transistors.

The completed hardware-software device was profiled for performance. It was determined to have a period of 17.02 seconds.

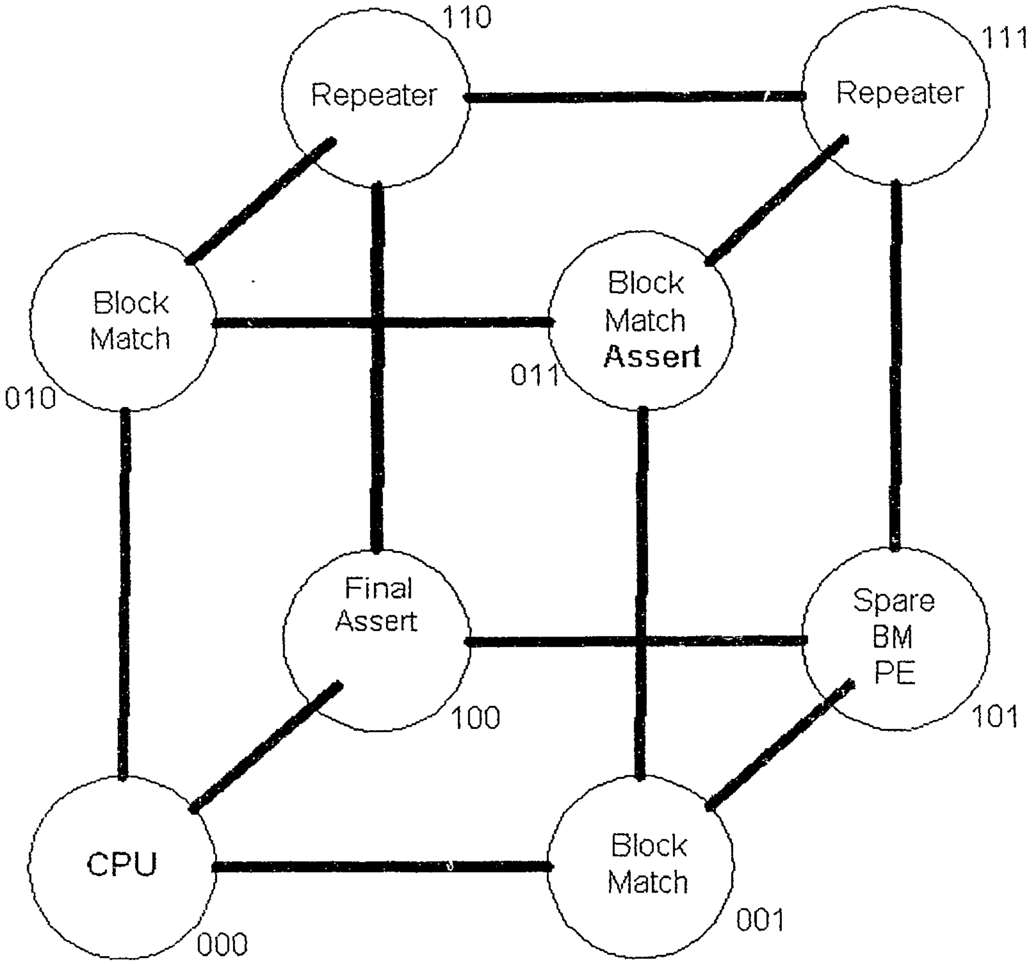

Figure 4.13: Final Prototype Device Architecture

The group based fault tolerance method developed allows the user to manually select the quantity and type of spare processing elements. In order to support fault tolerance but to also limit the fault tolerance overhead one spare block matching processing element has been included. Additionally, in order to simplify the hypercube topology and for ease of link fault tolerance, two repeater processing elements have been included. The repeater processing elements consist solely of communication interface circuitry, and are only responsible for repeating incoming data on the appropriate outgoing link.

In order to test the device's functionality, an input image and a reference image have been provided as input. The device then calculated the motion vectors for each of the 16 macroblocks in the images. The reference image is provided in Figure 4.14. The input image is provided in Figure 4.15. The macroblock numbering convention used is provided in Figure 4.16. The generated vector data is provided in Table 4.3.

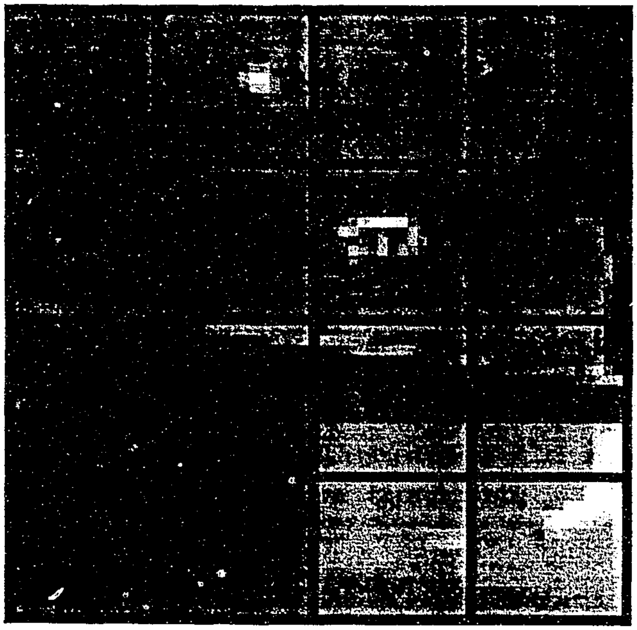

Figure 4.14: Reference Image

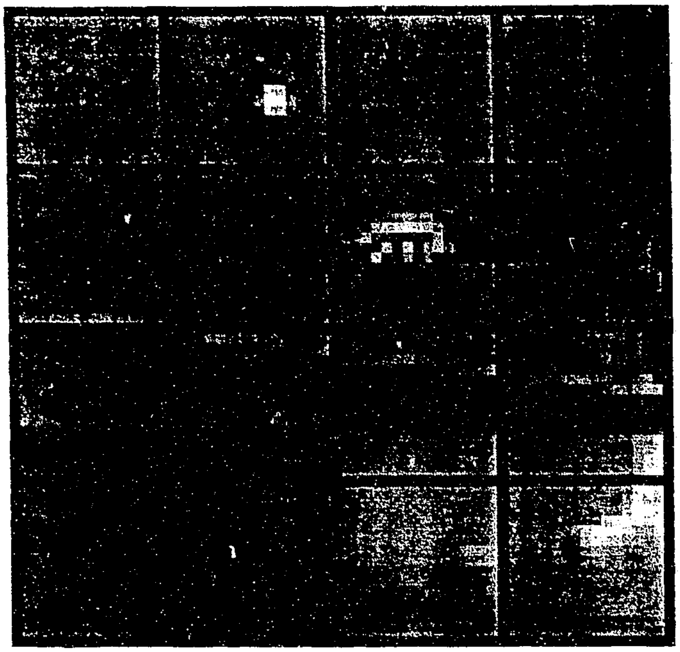

Figure 4.15: Input Image

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Figure 4.16: Macroblock Identification

Table 4.3: Generated Motion Vectors

Macroblock	Row Shift	Column Shift
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	-1	-1
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	-2	0
14	0	0
15	0	0

In order to confirm that the device correctly supports fault tolerance error states were simulated. Random error conditions in various block matching subcomponents were simulated by using a random number generator. After an error state has been detected, all further communication to the block matching PE where the error was detected is rerouted to one of the spare block matching PEs. Additionally, the set of tasks executed on the PE where the error was detected must be re-executed on the newly activated spare PE. This results in a degradation of overall system performance for the period in which the error was detected. The average measured period of the device when an error was detected is 20.94 seconds. The output results from the cases where an error was detected is identical to those generated by the device when no error was detected.

4.3 Discussion of Experimental Results

A design methodology for medium to large-scale fault-tolerant embedded systems has been presented. The objective has been to introduce a tool to aid the embedded systems developer to create hypercube devices meeting performance, cost and reliability constraints within a reasonable amount of time. To achieve this goal efficiently and effectively, a series of equations were developed to govern pipelined scheduling, task reallocation, addition of processing elements, configuration of processing elements within a hypercube topology and synthesis of inter-PE communication links. One of the predominant motivations is to minimize undesirable multi-hop communications that can occur in hypercube systems. Additionally, the innate link-fault tolerant nature of hypercube architectures is preserved. Support for fault tolerance has been developed through the use of the group based fault tolerance technique.

In order to demonstrate the efficacy of the new group based fault tolerance technique, the algorithm was compared to both cluster and task based fault tolerance. To the best of our knowledge cluster and task based fault tolerance are the only other published methods governing the addition of error detection functionality at the task graph level. The experimental results showed that on average the group based fault tolerance method yields a 9.8% improvement in fault tolerant overhead over the cluster based fault tolerance method and a 61% improvement over the task based fault tolerance method. These results demonstrate the efficacy of the group based fault tolerance method.

In order to evaluate the efficacy of the co-synthesis approach the results of an optimal algorithm have been provided for the test cases of the first case study. The exhaustive technique is a simple approach that attempts all possible combinations and is extremely computationally intensive. This exhaustive technique was implemented for the sole purpose of evaluating the results of the algorithm with the optimal solution. On average the approach yielded devices that under-performed the optimal case by 3.75%. Additionally, the algorithm resulted in designs that use 0.62% more hardware area than that of the optimal approach. These results show only small deviations between the algorithm's solutions and the ideal ones, indicating that the proposed method yields high quality solutions. To the best of our knowledge, this approach is the first co-synthesis algorithm developed targeting embedded hypercube systems. Comparing the results of this approach with, for instance, a co-synthesis algorithm developed to target bus systems would be inconclusive due to the differences in the target architectures. Consequently, the comparison has been limited to that of the optimal method.

Another common approach for comparing various co-synthesis or partitioning techniques is to evaluate their respective execution times. Some timing analysis of the algorithm has been performed with relation to the case studies presented in this paper (see Tables 4.1 & 4.2). It can be observed that as the hardware constraint rises, so does the execution time. As more hardware is added to the system, more iterations of the algorithm must be executed. The average execution time for the algorithm (across both case studies) is 84.92 ms. Synthesizing the 22-task embedded device (first case study) for the given set of input requirements would take approximately 24 hours to complete using the optimal approach. The execution time of such an approach is impractical for the comparable task graphs of the case study devices. Additionally, it is expected that the execution time would increase exponentially if the number of tasks in the device specification increases. This was verified when attempting to use the optimal method on the 32-node task graph from the second case study. The optimal execution time was observed to be prohibitive and as a result no optimal cases were completed. In the cases where the optimal technique could be used, it was found to produce marginally better designs but its execution time was too large and impractical. Both algorithms were executed on a Pentium IV 2.8 GHz system with 512MB of memory. The execution timing results indicate that the approach is efficient and its execution time is not a serious impediment to its performance. The first case study presented in this paper is fully implemented and executed in simulation.

In order to further confirm the effectiveness of the devices that the algorithm produces, a prototype of the second case study device has been built using a rapid prototyping platform based on the ARM CPU and Xilinx VirtexE FPGA.

The results show a final device gate count of 102577 transistors. As discussed in the previous section, the algorithm predicted a total device gate count of 104637 transistors. These predicted results yield a 2.0% error. This error has been attributed to the estimates on the cost of the communication link interfaces. The interface gate count was estimated by synthesizing a standard communication interface. The largest source of the discrepancy comes from the communication link interface circuitry that accompanies the CPU processing element. Due to the complex nature of the CPU, this interface circuitry deviates considerably from that of the other processing elements. Additionally, the VHDL compiler attempts to optimize the device area which may reduce the overall device gate count. The existence of these discrepancies was not realized until the final device was completed, as a result the modeling of these costs into the actual system was not possible.

The results show a final device performance period of 17.02 seconds. The algorithm predicted a final device period of 18.49 seconds. This is an 8.6% error. This error has been attributed to the observed inconsistency in the time measurement functions available for use with the ARM CPU. During profiling, the same functionality was repeatedly measured for timing. The execution time of the same software functional block was observed to typically vary by approximately 10% and in some cases as much as 40%. Due to this discrepancy, estimated execution times based on the maximum measured values were selected. This is believed to be the main source of error contributing to the overall device period estimate being 8.6% higher than the final measured value.

It should be noted that with a period of 17.02 seconds, the prototype device is slow. This is due to us using an ARM7TDMI processor. The ARM7 is widely used in small scale embedded systems and at 74,209 transistors, it is an extremely small processor. By comparison, an old desktop CPU such as the Intel Pentium II, consists of 7.5 million transistors which is approximately 100 times larger. Additionally the communication bus on the rapid prototyping platform is also slow.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The hardware-software co-synthesis of an embedded system architecture involves the partitioning of a system specification or functional description into hardware and software modules so as to meet a series of non-functional requirements such as cost and performance. Various distributed embedded device architectures have been presented previously, including bus, hierarchical and hypercube. Hypercube architectures are particularly suitable for use in fault tolerant and high performance devices. To the best of our knowledge, the first co-synthesis algorithm has been presented that will automatically generate fault tolerant hypercube architecture based embedded devices. The newly developed group based fault tolerance, a technique for adding fault tolerance to an embedded device at the task graph level has also been presented. The co-synthesis algorithm consists of six main steps: specification - defining device functional and non-functional requirements, profiling – evaluating the functional specification for performance and area utilization, group based fault tolerance – adding fault tolerance to the device at the task graph level, scheduling – evaluating the performance of the current device architecture, addition of processing elements – improving device performance by adding more hardware, and the synthesis of communication links – arranging all processing elements within a hypercube topology and synthesizing all communication links. The algorithm attempts to minimize the occurrence of multiple hops during inter-task communications. This helps to reduce the overall system communication overhead and thus increases device performance.

A security/navigation device has been implemented that is responsible for decoding 16 MPEG video streams in parallel. The co-synthesis algorithm's final architecture results were compared with that of the optimal case. This comparison revealed only minor deviations between the co-synthesis algorithm's generated devices and that of the optimal technique. These minor deviations illustrate how the co-synthesis algorithm is capable of generating high quality solutions.

A second device has been implemented to illustrate the algorithm's ability to generate fault tolerant devices. This device is responsible for performing the block matching algorithm which is a computationally intensive and an essential component of MPEG encoding. Additionally, in order to further demonstrate the efficacy of the approach, a prototype device based on the algorithm's results has been implemented. The final prototype device's area and timing values were compared with the algorithm's predicted values and revealed only modest deviations from the anticipated results. Error conditions were also simulated on the prototype device to confirm correct fault tolerant device functionality. This demonstrates the co-synthesis algorithm's ability to generate fault tolerant devices and to accurately predict device area and performance costs.

During the synthesis and development of the block matching case study, available equipment was limited to a small ARM7TDMI CPU connected to a Xilinx FPGA via a slow communication medium. This resulted in a slow performing but inexpensive device. Future work will entail the development of an additional prototype based on a faster and larger CPU. Additionally, the system will have a faster FPGA for the application specific circuitry and higher speed hardware-software communications. This will further demonstrate our co-synthesis algorithm's capabilities by synthesizing and building a prototype device with more practical performance and area constraints.

The co-synthesis algorithm presented, targets fault tolerant embedded devices and takes as input functional and non-functional requirements. The non-functional requirements include performance and area. In order to further enhance the algorithm's ability to synthesize fault tolerant devices, future work will entail adapting the algorithm to include reliability as a non-functional requirement. CPUs, ASICs and communication links all have an inherent reliability factor and the overall device reliability will be a function of the selection of differing processing elements, links and how they are interconnected.

REFERENCES

[1] P. Athanas, H. F. Silverman, "Processor reconfiguration through instruction-set metamorphosis", *IEEE Computer*, vol. 26, no. 3, pp. 11-18, March 1993.

[2] S. Bakshi and D. Gajski, "Partitioning and Pipelining for Performance-Constrained Hardware/Software Systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, no. 4, Dec. 1999.

[3] E. Barros, W. Rosenstiel, and X. Xiong, "A method for partitioning UNITY language to hardware and software," *Proceedings European Design Automation Conference*, Sept 1994, pp. 220-225, Grenoble, France.

[4] C. Bolchini, L. Pomante, F. Salice, D. Sciuto, "Online Fault Detection in a Hardware/Software Co-Design Environment: System Partitioning", *Proceedings IEEE International Symposium on System Synthesis*, Oct. 2001, pp. 51-56, Montreal, Canada.

[5] Jui-Ming Chang, Massoud Pedram, "Codex-dp: Co-Design of Communicating Systems Using Dynamic Programming", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, July 2000.

[6] K. S. Chatha and R. Vemuri, "Hardware-Software Partitioning and Pipelined Scheduling of Transformative Applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 3, June 2002.

[7] K. S. Chatha and R. Vemuri, "RECOD: a retiming heuristic to optimize resource and memory utilization in HW/SW codesigns," *Proceedings International Workshop on Hardware/Software Codesign*, March 1998, pp. 139-143, Seattle, USA.

[8] J. Choquette, M. Gupta, D. McCarthy, J. Veenstra, "High performance RISC microprocessors", *IEEE Micro*, pp. 48-55, Aug 1999.

[9] L.A. Cortes, P. Eles, Z. Peng, "Formal coverification of embedded systems using model checking", *Proceedings Euromicro Conference*, Sept. 2000, vol. 1, pp. 106-113, Maastricht, Netherlands.

[10] L.A. Cortes, P. Eles, Z. Peng, "Verification of embedded systems using a Petri net based representation", *Proceedings International Symposium on System Synthesis*, Sept. 2000, pp. 149-155, Madrid, Spain.

[11] J. G. D'Ambrosio and X. Hu, "Configuration-level hardware/software partitioning for real-time systems," *Proceedings International Workshop Hardware-Software Co-Design*, Sept. 1994, pp. 34-41, Grenoble, France.

[12] S. K. Das, M. C. Pinotti and F. Sarkar, "Optimal and load balanced mapping of parallel priority queues in hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, issue 6, pp. 555-564, June 1996.

[13] B. P. Dave and N. K. Jha, "COFTA: Hardware-software co-synthesis of heterogeneous distributed embedded system architectures for low overhead fault tolerance," *IEEE Transactions on Computers*, vol. 48, no. 4, April 1999.

[14] B. P. Dave and N. K. Jha, "COHRA: Hardware-Software Cosynthesis of Hierarchical Heterogeneous Distributed Embedded Systems," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, Oct. 1998.

[15] B. P. Dave, G. Lakshminarayana and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Transactions on VLSI Systems*, vol. 7, no. 1, March 1999.

[16] R. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920-935, Oct. 1998.

[17] S. Dutta and W. Wolf "A Flexible Parallel Architecture Adapted to Block-Matching Motion-Estimation Algorithms", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, No. 1, pp. 74-86, Feb. 1996.

[18] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test*, vol. 10, pp. 64-75, Dec. 1993.

[19] F. C. Filho, P. Maciel, E. Barros, "A Petri Net Based Approach For Hardware/Software Partitioning", *Symposium on Integrated Circuits and Systems Design*, Sept. 2001, pp. 72-77, Pirenopolis, Brazil.

[20] M.R. Garey and D. S. Johnson, *Computers and Interactability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[21] C. Ghezzi, M. Jazayeri, D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, Upper Saddle River, NJ, 1991.

[22] R. Gupta and G. DeMicheli, "Hardware/software cosynthesis for digital systems," *IEEE Design & Test of Computers*, pp. 29-41, Sept. 1993.

[23] R. Gupta and G. DeMicheli, "System-level synthesis using re-programmable components", *Proceedings European Conference on Design Automation*, Mar. 1992, pp. 2-7, Brussels, Belgium.

[24] J. Henkel, R. Ernst, "An Approach to Automated Hardware/Software Partitioning Using a Flexible Granularity that is Driven by High-Level Estimation Techniques", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9, no. 2, April 2001.

[25] J. Hou and W. Wolf, "Process partitioning for distributed embedded systems," *Proceedings International Workshop on Hardware/Software Codesign*, Sept. 1996. pp. 70-76, Pittsburgh, USA.

[26] P.-A. Hsiung, "Hardware-software timing coverification of concurrent embedded real-time systems", *IEE Proceedings Computers and Digital Techniques*, pp. 83-92, March 2000.

[27] B. A. Izadi, F. Ozguner, "Real-time fault-tolerant hypercube multicomputer," *IEE Proceedings Computers and Digital Techniques*, vol. 149, no. 5, pp. 197-202, Sep. 2002.

[28] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Design & Test*, vol. 10, pp. 16-28, Sept. 1993.

[29] G. N. Khan, G. S. Hura, G. Wei, "Distributed Recovery Block Based Fault-tolerant Routing in Hypercube Networks," *Proceedings IEEE Canadian Conference on Electrical and Computer Engineering*, May 2002, pp. 603-608, Winnipeg, Canada.

[30] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power real-time systems," *Proceedings Design Automation Conference*, June 1997, pp. 697-702, Anaheim, USA.

[31] P. V. Knudsen, J. Madsen, "PACE: A dynamic programming algorithm for hardware/software partitioning", *Proceedings 4th International Workshop on Hardware/Software Codesign*, pp. 85-92, 1996, Pittsburgh, USA.

[32] K. Kuchcinski, "Constraints-driven scheduling and resource assignment," *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 3, pp. 355-383, July 2003.

[33] Yu-Kwong Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel Distributed Systems*, vol. 7, pp. 506-521, May 1996.

[34] L. E. LaForge, "Self-Healing Avionics for Starships," *Proceedings IEEE Aerospace Conference*, March 2000, vol. 5, pp. 499-519, Big Sky, USA.

[35] L. E. LaForge, K. F. Korver and M. S. Fadali, "What Designers of Bus and Network Architectures Should Know about Hypercubes," *IEEE Transactions on Computers*, vol. 52, no. 4, April 2003.

[36] Y. Li and W. Wolf, "Hardware/Software Co-Synthesis with Memory Hierarchies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 10 pp. 1405-1417, Oct. 1999.

[37] Huiqun Liu, D. F. Wong, "Integrated Partitioning and Scheduling for Hardware/Software Co-Design", *Proceedings International Conference on VLSI in Computers and Processors*, Oct. 1998, pp. 609-614, Austin, USA.

[38] J. Madsen, J. Grode, P.V. Knudsen, M.E. Peterson, A. Haxthausen, "LYCOS: the lyngby co-synthesis system", *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 195-236, 1997.

[39] D.D. Mitchell, D.C. Ku, F. Mailhot, and T. Truong, "The Olympus Synthesis System for digital design", *IEEE Design and Test Magazine*, pp.37-53, Oct. 1990.

[40] A. Osterling, T. Benner, R. Ernst, D. Herrmann, T. Scholz, and W. Ye, "The COSYMA system", *Hardware/Software Co-Design: Principles and Practice*, pp. 263-281. Kluwer Academic Publishers, Amsterdam, 1997.

[41] D. A. Patterson, J. L. Hennessy, *Computer Architecture A Quantitative Approach 2nd Edition*, Morgan Kaufman Publishers Inc., San Francisco, CA, 1996.

[42] S. Prakash and A. Parker, "SOS: Synthesis of application-specific heterogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 338-351, Dec. 1992.

[43] K. Rajan, L. M. Patnaik and J. Ramakrishna, "High-speed parallel implementation of a modified PBR algorithm on DSP-based EH topology," *IEEE Transactions on Nuclear Science*, vo [illegible], issue 4, pp. 1658-1672, Aug. 1997.

[44] E. Salminen, T. Hamalainen, T. Kangas, K. Kuusilinna, J. Saarinen, "Interfacing multiple processors in a system-on-chip video encoder", *International Symposium on Circuits and Systems*, May 2001, vol. 4, pp. 478-481, Sydney, Australia.

[45] Aviral Shrivastava, Mohit Kumar, Sanjiv Kapoor, Shashi Kumar, M. Balakrishnan, "Optimal Hardware/Software Partitioning for Concurrent Specification using Dynamic Programming", *International Conference on VLSI Design*, Jan. 2000, pp. 110-113, Calcutta, India.

[46] S. Srinivasan and N. K. Jha, "Hardware-Software Co-Synthesis of Fault-Tolerant Real-Time Distributed Embedded Systems," *Proceedings European Design Automation Conference*, Sept. 1995, pp. 334-339, Brighton, UK.

[47] W. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems," *IEEE Transactions on VLSI Systems*, vol. 5, pp. 218-229, June 1997.

[48] W. Wolf, *Computers as Components*, Morgan Kaufman Publishers Inc., San Diego, CA, 2001.

[49] S. Yajnik, S. Srinivasan, N. K. Jha, "TBFT: A Task Based Fault Tolerance Scheme for Distributed Systems," *Proceedings International Conference on Parallel and Distributed Computing Systems*, Oct. 1994, pp. 483-489, Las Vegas, USA.

[50] Kun-Min Yang, Ming-Ting Sun and Lancelot Wu, "A Family of VLSI Designs for the Motion Compensation Block-Matching Algorithm", *IEEE Transactions on Circuits and Systems*, vol.36, no.10, pp. 1317-1325, Oct. 1989.

[51] Yao-Ming Yeh and Yiu-Cheng Shyu, "Efficient distributed schemes for embedding binary trees into incomplete hypercubes," *Proceedings International Conference IEEE Region 10*, Aug. 1994, vol. 1, pp. 182-186, Singapore.

[52] T.Y. Yen, W. Wolf, "Multiple-process behavioral synthesis for mixed hardware-software systems", *Proceedings 8th International Symposium on System Synthesis*, Sept. 1995, pp. 10-15, Cannes, France.

Appendix A - Prototype Device Code Listing

The following VHDL and C code define the final prototype device described in section 4.2 and illustrated in figure 4.13. The device is a symmetrical three dimensional hypercube that performs the block matching algorithm.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY reg8 IS
  PORT(load            : IN std_logic;
              clock    : in std_logic;
      datain : IN std_ulogic_vector(7 downto 0);

      dataout : OUT std_ulogic_vector(7 downto 0)); --parallel outputs
END reg8;

ARCHITECTURE reg5 OF reg8 IS
  --SIGNAL int_reg : integer range 0 to 65535;

BEGIN
  process(clock)
              --variable vec: std_ulogic_vector (1 to size);


  BEGIN
            if( clock'event and clock = '1') then--rising_edge(clock) and load = '1' ) then
                        if(load = '1') then
            --int_reg <= datain;
                        dataout <= datain;
                        end if;

            end if;
  END PROCESS;
  --connect internal register to dataout port
  --dataout <= int_reg;
END reg5;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

      --our register definition
ENTITY reg65535Int2 IS
  PORT(load            : IN std_logic;
              clock    : in std_logic;
              init     : IN std_logic;
      datain : IN integer range 0 to 65535;
              --datain2 : IN std_logic_vector(15 downto 0);--integer range 0 to 65535;
      dataout : OUT integer range 0 to 65535); --parallel outputs
END reg65535Int2;

ARCHITECTURE reg1 OF reg65535Int2 IS
  --SIGNAL int_reg : integer range 0 to 65535;

BEGIN
  process(clock, init)
              --variable vec: std_ulogic_vector (1 to size);


  BEGIN
    if(init = '1') then
              dataout <= 65535;
             else
            if( clock'event and clock = '1' ) then
                                if(load = '1') then
                                --int_reg <= datain;
                                        dataout <= datain;
                                end if;
                        end if;
            end if;
  END PROCESS;
  --connect internal register to dataout port
  --dataout <= int_reg;
END reg1;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY reg65535Int IS
  PORT(load           : IN std_logic;
            clock     : in std_logic;
            init      : IN std_logic;
     datain : IN integer range 0 to 65535;

     dataout : OUT integer range 0 to 65535); --parallel outputs
END reg65535Int;

ARCHITECTURE reg4 OF reg65535Int IS
  --SIGNAL int_reg : integer range 0 to 65535;

BEGIN
  process(clock, init)
            --variable vec: std_ulogic_vector (1 to size);


  BEGIN
        if(init = '1') then
          dataout <= 0;
        else
        if( clock'event and clock = '1' ) then
                                if(load = '1') then
                                --int_reg <= datain;
                                        dataout <= datain;
                                end if;
                        end if;
        end if;
  END PROCESS;
  --connect internal register to dataout port
  --dataout <= int_reg;
END reg4;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

  --our compare circuit definition
ENTITY comp IS

  PORT(inVal1 : IN integer range 0 to 65535;  --parallel inputs
            inI1      : in integer range 0 to 32;
            inJ1      : in integer range 0 to 32;
            inVal2 : IN integer range 0 to 65535;  --parallel inputs
            start : in std_logic;
            outI      : out integer range 0 to 32;
            outJ      : out integer range 0 to 32;
     out1 : OUT integer range 0 to 65535); --parallel outputs
END comp;

ARCHITECTURE comp1 OF comp IS
--signal outTemp : INTEGER;
```

```
BEGIN
          process(start)

                    begin
                              if(start'event and start = '1') then
                                        if(inVal1 < inVal2) then
                                                  out1 <= inVal1;
                                                  outI <= inI1;
                                                  outJ <= inJ1;
                                        end if;
                              end if;


          end process;
          --out1 <= outTemp;

END comp1;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

   --our add circuit definition
ENTITY add IS

  PORT(in1 : IN integer range 0 to 65535;  --parallel inputs
            in2 : IN integer range 0 to 65535;   --parallel inputs
     out1 : OUT integer range 0 to 65535); --parallel outputs
END add;

ARCHITECTURE add1 OF add IS
signal outTemp : INTEGER;

BEGIN
          process(in1, in2)

                    begin
                              outTemp <= in1 + in2;
          end process;
          out1 <= outTemp;

END add1;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

   --our difference circuit definition
ENTITY diff IS

  PORT(in1 : IN Std_ulogic_vector(7 DOWNTO 0);  --parallel inputs
            in2 : IN Std_ulogic_vector(7 DOWNTO 0);  --parallel inputs
     out1 : OUT integer range 0 to 65535); --parallel outputs
END diff;

ARCHITECTURE diff1 OF diff IS
signal outTemp : INTEGER;


BEGIN
          process(in1, in2)
                    variable val : INTEGER;
                    variable val1 : INTEGER;
                    variable val2 : INTEGER;

                    variable b : INTEGER;
```

```
                BEGIN
                        val := 0;
                        b := 1;
                        for y in 0 to 7 loop

                                if(in1(y) = '1') then
                                        val := val + b;
                                end if;
                                b:= b * 2;
                        end loop;--for y
                        val1 := val;

                        val := 0;
                        b := 1;
                        for y in 0 to 7 loop
                                if(in2(y) = '1') then
                                        val := val + b;
                                end if;
                                b := b * 2;
                        end loop;--for y
                        val2 := val;

                if(val1 < val2) then
                                outTemp <= (val2 - val1);
                        else
                                outTemp <= (val1 - val2);
                        end if;

        END PROCESS;

        out1 <= outTemp;

END diff1;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

   --our register definition
ENTITY reg IS

  PORT(load           : IN std_logic;
          clock       : in std_logic;
    datain : IN Std_ulogic_vector(7 DOWNTO 0);   --parallel inputs
    dataout : OUT Std_ulogic_vector(7 DOWNTO 0)); --parallel outputs
END reg;

ARCHITECTURE v1 OF reg IS
  SIGNAL int_reg : Std_ulogic_vector(7 DOWNTO 0);

BEGIN
  process(clock)
  BEGIN
    if(clock'event and clock = '1') then
                if(load = '1') then
                int_reg <= datain;
          end if;
        end if;
  END PROCESS;
  --connect internal register to dataout port
  dataout <= int_reg;-- when timeToOutput = '1' else "ZZZZZZZZ";

END v1;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;
```

```
ENTITY blockmatch IS
                PORT(
                        --stopAllDone                       : IN STD_LOGIC;
                        addByte     : IN        STD_LOGIC;--goes high when it is time to add a byte to the arrays
                        clk         : IN STD_LOGIC;
                        inByte      : IN STD_ULOGIC_VECTOR(7 DOWNTO 0);
                        allDone     : OUT STD_LOGIC;--goes high when EVERYTHING is finished
                        outI        : OUT integer range 0 to 32; --the i shift value (SW will make it -16 to +16)
                        outJ        : OUT integer range 0 to 32); --the j shift value (SW will make it -16 to +16)
END blockmatch;


ARCHITECTURE description OF blockmatch IS

COMPONENT reg IS
  PORT(load         : IN std_logic;
           clock    : in std_logic;
    datain : IN Std_ulogic_vector(7 DOWNTO 0);  --parallel inputs
    dataout : OUT Std_ulogic_vector(7 DOWNTO 0)); --parallel outputs
END component;

COMPONENT diff IS
  PORT(in1 : IN Std_ulogic_vector(7 DOWNTO 0);  --parallel inputs
       in2 : IN Std_ulogic_vector(7 DOWNTO 0);  --parallel inputs
    out1 : OUT integer range 0 to 65535); --parallel outputs
END component;

component add IS

  PORT(in1 : IN integer range 0 to 65535;  --parallel inputs
       in2 : IN integer range 0 to 65535;  --parallel inputs
    out1 : OUT integer range 0 to 65535); --parallel outputs
END component;

component comp IS

  PORT(inVal1 : IN integer range 0 to 65535;  --parallel inputs
       inI1     : in integer range 0 to 32;
       inJ1     : in integer range 0 to 32;
       inVal2 : IN integer range 0 to 65535;  --parallel inputs
       start : in std_logic;
       outI     : out integer range 0 to 32;
       outJ     : out integer range 0 to 32;
    out1 : OUT integer range 0 to 65535); --parallel outputs
END component;

component reg65535Int2 IS
  PORT(load         : IN std_logic;
       clock        : in std_logic;
       init         : IN std_logic;
    datain : IN integer range 0 to 65535;
       --datain2 : IN std_logic_vector(15 downto 0);--integer range 0 to 65535;
    dataout : OUT integer range 0 to 65535); --parallel outputs
END component;



component reg65535Int IS
  PORT(load         : IN std_logic;
       clock        : in std_logic;
       init         : IN std_logic;
    datain : IN integer range 0 to 65535;
    dataout : OUT integer range 0 to 65535); --parallel outputs
END component;

--signal load : STD_LOGIC_VECTOR(2559 downto 0);--the load lines for all the registers
--signal regOutput : STD_LOGIC_VECTOR(2559 downto 0);--the line indicates if this register should be outputting
```

```
signal regActiveOut : STD_ULOGIC_VECTOR(7 downto 0);
--signal regOldOut : STD_ULOGIC_VECTOR(7 downto 0);
signal diff1Out : integer range 0 to 65535;
signal addRegOut : integer range 0 to 65535;
--signal load2ndReg : STD_LOGIC_VECTOR(1088 downto 0);
--signal reg2ndOutput: STD_LOGIC_VECTOR(1088 downto 0);
signal iVal : integer range 0 to 32; --generated by control circuitry to tell comparator what index is being compared
signal jVal : integer range 0 to 32; --as above
signal compOut : integer range 0 to 65535;
signal iOut: integer range 0 to 32;
signal jOut: integer range 0 to 32;
signal bestErrorYet : integer range 0 to 65535;
signal loadCompReg : STD_LOGIC;
signal initCompReg : STD_LOGIC;
--signal regAdderIn2 : integer range 0 to 65535;
--signal regActiveTimeToOutput : std_logic_vector(255 downto 0);
--signal regOldTimeToOutput : std_logic_vector(2303 downto 0);

signal addOut : integer range 0 to 65535;
signal loadAddReg : std_logic;
signal initAddReg : std_logic;
signal loadInReg : std_logic;
signal reg2ndOut : std_logic_vector(15 downto 0);
--subtype WORD8 is STD_LOGIC_VECTOR (7 downto 0);
--type ARRAY1089 is array (1088 downto 0) of WORD8;
--signal reg2ndOut : ARRAY1089;
signal compStart : std_logic;


BEGIN

        inRegActiveLabel: reg PORT MAP(loadInReg, clk, inByte, regActiveOut);
        diffCircuit: diff port map(regActiveOut, inByte, diff1Out);
        add1: add port map(diff1Out, addRegOut, addOut);
        addRegLabel: reg65535Int port map(loadAddReg, clk, initAddReg, addOut, addRegOut);
        comp1: comp port map(addOut, iVal, jVal, bestErrorYet, compStart, iOut, jOut, compOut);
        compRegister: reg65535Int2 port map(loadCompReg, clk, initCompReg, compOut, bestErrorYet);

        --tempInRegOut <= regActiveOut;
        --tempDiffOut  <= diff1Out;
        --tempAddRegOut <= addRegOut;
        --tempAddOut <= addOut;
        --tempCompRegOut <= bestErrorYet;
        --tempCompOut                  <= compOut;


        process(clk)
                variable addCounter   : integer;
                variable compCounter: integer;
                variable i : integer;
                variable j : integer;
                variable state                  : integer := 0; --0 means we are loading values. 1 means we are doing
the adding
                begin
                        if(rising_edge(clk)) then
                                --set all registers to no load....

                                allDone <= '0';
                                loadInReg <= '0';
                                loadAddReg <= '0';
                                initAddReg <= '0';
                                loadCompReg <= '0';
                                initCompReg <= '0';
                                compStart <= '0';
                                if(addByte = '1') then
                                        --then we are adding another byte to the circuit flow
--                                      tempSignal <= '1';
--                                      tempClk <= '1';
                                        addCounter := addCounter + 1;
                                        if(state = 1) then
                        loadAddReg <= '1';
                        state := 2;
                    elsif(state = 2) then
                        loadInReg <= '1';
                        if(addCounter > 511) then --254) then
                            state := 3;
                            addCounter := -1;

                            j := j + 1;
                            if(j > 32) then
                                j := 0;
                                i := i + 1;
                            end if;
                            if(i > 32) then
                        --reset i - we are done - don't update the output i's and j's
                                i := 0;
                            else
                                iVal <= i;
                                jVal <= j;
                                compStart <= '1';
                            end if;
                        else
                            state := 1;
                        end if;
                    elsif(state = 3) then
                        loadCompReg <= '1';
                        state := 4;

                    elsif(state = 4) then
                        initAddReg <= '1';
                        loadInReg <= '1';
                        compCounter := compCounter + 1;
                        if(compCounter > 1088) then --we just did the last one
                            state := 0;
                            allDone <= '1';
                        else
                            state := 1;
                        end if;
                    else        --we are in state 0
                        loadInReg <= '1';
                        initAddReg <= '1';
                        initCompReg <= '1';
                        i := 0;
                        j := 0;
                        --compCounter := 0;
                        state := 1;
                    --so we are probably in state 0 (loading of the registers so do nothing....
                    end if;--if state = 1,2....
                end if;--if addByte = '1'
--              tempState <= state;
            end if;--if rising_edge clock
    end process;

    outI <= iOut;
    outJ <= jOut;

END description;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


ENTITY PE_BlockMatch IS
        PORT(
            clk                 : in std_logic;
            inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
            inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
            inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
```

```
                    inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
                    inStart1            : in std_logic; --goes high when the three things above are ready
                    inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
                    inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
                    inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
                    inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
                    inStart2            : in std_logic; --goes high when the three things above are ready
                    inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
                    inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
                    inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
                    inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
                    inStart3            : in std_logic; --goes high when the three things above are ready
                    myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
                    ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
                    ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
                    ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
                    outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
                    outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
                    outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
                    outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
                    outStart1           : out std_logic;
                    outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
                    outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
                    outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
                    outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
                    outStart2           : out std_logic;
                    outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
                    outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
                    outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
                    outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                    outStart3           : out std_logic);
END PE_BlockMatch;

ARCHITECTURE PEdescription OF PE_BlockMatch IS

component blockmatch IS
                PORT(
                    --stopAllDone                                   : IN STD_LOGIC;
                    addByte   : IN      STD_LOGIC;--goes high when it is time to add a byte to the arrays
                    clk       : IN STD_LOGIC;
                    inByte    : IN STD_ULOGIC_VECTOR(7 DOWNTO 0);
                    allDone   : OUT STD_LOGIC;--goes high when EVERYTHING is finished
                    outI      : OUT integer range 0 to 32; --the i shift value (SW will make it -16 to +16)
                    outJ      : OUT integer range 0 to 32); --the j shift value (SW will make it -16 to +16)
END component;

        signal circuitStart : std_logic;--tied to clk
        signal addingAByte   : std_logic;--tied to addByte
        signal inData                : std_ulogic_vector(7 downto 0);--tied to inByte
        signal done                  : std_logic;--listen if circuit is done
        signal outDataI_Int : integer range 0 to 32;
        signal outDataJ_Int : integer range 0 to 32;
        signal outDataI      : std_ulogic_vector(7 downto 0);--the value of outI from circuit converted to vector to send
        signal outDataJ      : std_ulogic_vector(7 downto 0);--the value of outJ from circuit converted to vector to send
        signal outDataI_LOGIC        : std_logic_vector(7 downto 0);
        signal outDataJ_LOGIC        : std_logic_vector(7 downto 0);


                --will be sent on outID lines a flag will have to be set in the top 2 bits

        begin

                BM: blockmatch port map(addingAByte, clk, inData, done, outDataI_Int, outDataJ_Int);

                process(clk)

                variable destCounter1 : integer range 0 to 3;
                variable destCounter2 : integer range 0 to 3;
                variable destCounter3 : integer range 0 to 3;
                variable iTemp                : integer range 0 to 32;
variable jTemp                    : integer range 0 to 32;

begin
 if(rising_edge(clk)) then
          outStart1 <= '0';
          outStart2 <= '0';
          outStart3 <= '0';
          addingAByte <= '0';
          circuitStart <= '0';
--        tempOut3<= '0';


          if(done = '1') then
                    --tempOut <= '1';
                    --so the BM circuit just finished!
                    --convert outDataI_Int and outDataJ_Int to outDataI and outDataJ
                    --set top 2 bits of outDataJ to zero (it won't overwrite any info)
                    --destination "000" figure out which link to send the data down
                    --assign outDestination# <= "000", outData# <= outDataI, outID# <= outDataJ,
                    --outStart# <= '1';
                    --same data must also be sent to ASSERT task          at location "011"
                    iTemp := outDataI_Int;
                    jTemp := outDataJ_Int;
                    -- they have been latched, tell the BM circuit to stop having allDone be high!
                    addingAByte <= '0';--without adding a byte
                    circuitStart <= '1';
                    outDataI_LOGIC <= CONV_STD_LOGIC_VECTOR(iTemp, 8);
                    outDataJ_LOGIC <= CONV_STD_LOGIC_VECTOR(jTemp, 8);

                    for i in 0 to 7 loop
                              outDataI(i) <= outDataI_LOGIC(i);
                              outDataJ(i) <= outDataJ_LOGIC(i);
                    end loop;
                    outDataJ(7) <= '0';
                    outDataJ(6) <= '0';--flags to indicate what this info is
                    --which link is closest to 000 and which to 011?
                    destCounter1 := 0;
                    destCounter2 := 0;
                    destCounter3 := 0;
                    for i in 0 to 2 loop
                              if(ID1(i) = '0') then
                                        destCounter1 := destCounter1 + 1;
                              end if;
                              if(ID2(i) = '0') then
                                        destCounter2 := destCounter2 + 1;
                              end if;
                              if(ID3(i) = '0') then
                                        destCounter3 := destCounter3 + 1;
                              end if;
                    end loop;
                    if(destCounter1 > destCounter2) then
                              if(destCounter1 > destCounter3) then
                                        --send down link!
                                        outDestination1 <= "000";
                                        outSource1 <= myID;
                                        outData1 <= outDataI;
                                        outID1 <= outDataJ;
                                        outStart1   <= '1';
                                        --is assert closer to link 2 or 3?
                                        destCounter2 := 0;
                                        destCounter3 := 0;
                                        for i in 0 to 1 loop
                                                  if(ID2(i) = '1') then
                                                            destCounter2 := destCounter2 + 1;
                                                  end if;
                                                  if(ID3(i) = '1') then
                                                            destCounter3 := destCounter3 + 1;
                                                  end if;
                                        end loop;
                                        if(ID2(2) = '0') then
```

```
                destCounter2 := destCounter2 + 1;
            end if;
            if(ID3(2) = '0') then
                destCounter3 := destCounter3 + 1;
            end if;
            if(destCounter2 > destCounter3) then
                --send packet down link2
                outDestination2 <= "000";
                outSource2 <= myID;
                outData2 <= outDataI;
                outID2 <= outDataJ;
                outStart2   <= '1';
            else
                --send packet down link3
                outDestination3 <= "000";
                outSource3 <= myID;
                outData3 <= outDataI;
                outID3 <= outDataJ;
                outStart3   <= '1';
            end if;
        else
            --send down link3
            outDestination3 <= "000";
            outSource3 <= myID;
            outData3 <= outDataI;
            outID3 <= outDataJ;
            outStart3   <= '1';
            --is assert closer to link 2 or 1?
            destCounter2 := 0;
            destCounter1 := 0;
            for i in 0 to 1 loop
                if(ID2(i) = '1') then
                    destCounter2 := destCounter2 + 1;
                end if;
                if(ID1(i) = '1') then
                    destCounter1 := destCounter1 + 1;
                end if;
            end loop;
            if(ID2(2) = '0') then
                destCounter2 := destCounter2 + 1;
            end if;
            if(ID1(2) = '0') then
                destCounter1 := destCounter1 + 1;
            end if;
            if(destCounter2 > destCounter1) then
                --send packet down link2
                outDestination2 <= "000";
                outSource2 <= myID;
                outData2 <= outDataI;
                outID2 <= outDataJ;
                outStart2   <= '1';
            else
                --send packet down link1
                outDestination1 <= "000";
                outSource1 <= myID;
                outData1 <= outDataI;
                outID1 <= outDataJ;
                outStart1   <= '1';
            end if;
        end if;
    else
        if(destCounter2 > destCounter3) then
            --send down link2
            outDestination2 <= "000";
            outSource2 <= myID;
            outData2 <= outDataI;
            outID2 <= outDataJ;
            outStart2   <= '1';
            --is assert closer to link 1 or 3?
            destCounter1 := 0;
    destCounter3 := 0;
    for i in 0 to 1 loop
        if(ID1(i) = '1') then
            destCounter1 := destCounter1 + 1;
        end if;
        if(ID3(i) = '1') then
            destCounter3 := destCounter3 + 1;
        end if;
    end loop;
    if(ID1(2) = '0') then
        destCounter1 := destCounter1 + 1;
    end if;
    if(ID3(2) = '0') then
        destCounter3 := destCounter3 + 1;
    end if;
    if(destCounter1 > destCounter3) then
        --send packet down link2
        outDestination1 <= "000";
        outSource1 <= myID;
        outData1 <= outDataI;
        outID1 <= outDataJ;
        outStart1   <= '1';
    else
        --send packet down link3
        outDestination3 <= "000";
        outSource3 <= myID;
        outData3 <= outDataI;
        outID3 <= outDataJ;
        outStart3   <= '1';
    end if;
else
    --send down link3
    outDestination3 <= "000";
    outSource3 <= myID;
    outData3 <= outDataI;
    outID3 <= outDataJ;
    outStart3   <= '1';
    --is assert closer to link 2 or 1?
    destCounter2 := 0;
    destCounter1 := 0;
    for i in 0 to 1 loop
        if(ID2(i) = '1') then
            destCounter2 := destCounter2 + 1;
        end if;
        if(ID1(i) = '1') then
            destCounter1 := destCounter1 + 1;
        end if;
    end loop;
    if(ID2(2) = '0') then
        destCounter2 := destCounter2 + 1;
    end if;
    if(ID1(2) = '0') then
        destCounter1 := destCounter1 + 1;
    end if;
    if(destCounter2 > destCounter1) then
        --send packet down link2
        outDestination2 <= "000";
        outSource2 <= myID;
        outData2 <= outDataI;
        outID2 <= outDataJ;
        outStart2   <= '1';
    else
        --send packet down link1
        outDestination1 <= "000";
        outSource1 <= myID;
        outData1 <= outDataI;
        outID1 <= outDataJ;
        outStart1   <= '1';
    end if;
end if;
```

```
        end if;
end if;
if(inStart1 = '1') then
        --something has come in off of the 1 lines
        if(inDestination1 = myID) then
                --this packet is for here!  Pass it along to the BM circuit
                addingAByte <= '1';
                inData <= inData1;
                circuitStart <= '1';
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter2 := 0;
                destCounter3 := 0;
                if(inDestination1(0) = ID2(0)) then

                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(1) = ID2(1)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(2) = ID2(2)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter2) then
                        --forward packet to ID3
                        outDestination3 <= inDestination1;
                        outSource3 <= inSource1;
                        outData3 <= inData1;
                        outID3 <= inID1;
                        outStart3   <= '1';
                else
                        --forward packet to ID2
                        outDestination2 <= inDestination1;
                        outSource2 <= inSource1;
                        outData2 <= inData1;
                        outID2 <= inID1;
                        outStart2   <= '1';
                end if;
        end if;
end if;
if(inStart2 = '1') then
        if(inDestination2 = myID) then
                --this packet is for here!  Pass it along to the BM circuit
                addingAByte <= '1';
                inData <= inData1;
                circuitStart <= '1';
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination2(0) = ID1(0)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter1) then
                        --forward packet to ID3
                        outDestination3 <= inDestination2;
                        outSource3 <= inSource2;
                        outData3 <= inData2;
                        outID3 <= inID2;
                        outStart3   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination2;
                        outSource1 <= inSource2;
                        outData1 <= inData2;
                        outID1 <= inID2;
                        outStart1   <= '1';
                end if;
        end if;
end if;
if(inStart3 = '1') then
        if(inDestination3 = myID) then
                --this packet is for here!  Pass it along to the BM circuit
                tempOut3<= '1';
                addingAByte <= '1';
                inData <= inData1;
                circuitStart <= addingAByte;--'1';
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination3(0) = ID1(0)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(0) = ID2(0)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(1) = ID2(1)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(2) = ID2(2)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(destCounter2 > destCounter1) then
                        --forward packet to ID3
                        outDestination2 <= inDestination3;
                        outSource2 <= inSource3;
                        outData2 <= inData3;
                        outID2 <= inID3;
                        outStart2   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination3;
                        outSource1 <= inSource3;
                        outData1 <= inData3;
                        outID1 <= inID3;
                        outStart1   <= '1';
```

```
                                        end if;
                                end if;
                        end if;

                end if;
                end process;
end PEdescription;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


ENTITY PE_BM_Assert IS
                PORT(
                        clk                     : in std_logic;
                        inDestination1          : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource1               : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData1                 : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID1                   : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart1                : in std_logic; --goes high when the three things above are ready
                        inDestination2          : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource2               : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData2                 : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID2                   : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart2                : in std_logic; --goes high when the three things above are ready
                        inDestination3          : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource3               : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData3                 : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID3                   : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart3                : in std_logic; --goes high when the three things above are ready
                        myID                    : in std_logic_vector(2 downto 0);--what this PE's ID is
                        ID1                     : in std_logic_vector(2 downto 0);--what PE 1's ID is
                        ID2                     : in std_logic_vector(2 downto 0);--what PE 2's ID is
                        ID3                     : in std_logic_vector(2 downto 0);--what PE 3's ID is
                        outDestination1         : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource1              : out STD_LOGIC_VECTOR(2 downto 0);
                        outData1                : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID1                  : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart1               : out std_logic;
                        outDestination2         : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource2              : out STD_LOGIC_VECTOR(2 downto 0);
                        outData2                : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID2                  : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart2               : out std_logic;
                        outDestination3         : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource3              : out STD_LOGIC_VECTOR(2 downto 0);
                        outData3                : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID3                  : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart3               : out std_logic);
END PE_BM_Assert;

ARCHITECTURE PEAssertdescription OF PE_BM_Assert IS

begin
        process(clk)

                variable destCounter1 : integer range 0 to 3;
                variable destCounter2 : integer range 0 to 3;
                variable destCounter3 : integer range 0 to 3;

                variable sendError  : integer range 0 to 1;--goes high if error is detected (so a packet must be sent to the
                                                                --control so that that PE is now neglected)
                variable badBM          : std_logic_vector(2 downto 0);--represents which BM is faulty

                begin
                 if(rising_edge(clk)) then
                        outStart1 <= '0';
```

```
outStart2 <= '0';
outStart3 <= '0';
sendError := 0;
if(inStart1 = '1') then
        --something has come in off of the 1 lines
        if(inDestination1 = myID) then
                --this packet is for here!
                --is there an error in the packet?
                if( (inID1(7) = '1') and (inID2(7) = '1') ) then
                        --then it is an error packet
                        sendError := 1;
                        badBM := inSource1;
                end if;

        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter2 := 0;
                destCounter3 := 0;
                if(inDestination1(0) = ID2(0)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(1) = ID2(1)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(2) = ID2(2)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination1(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter2) then
                        --forward packet to ID3
                        outDestination3 <= inDestination1;
                        outData3 <= inData1;
                        outID3 <= inID1;
                        outSource3 <= inSource1;
                        outStart3   <= '1';
                else
                        --forward packet to ID2
                        outDestination2 <= inDestination1;
                        outData2 <= inData1;
                        outID2 <= inID1;
                        outSource2 <= inSource1;
                        outStart2   <= '1';
                end if;
        end if;
end if;
if(inStart2 = '1') then
        if(inDestination2 = myID) then
                --this packet is for here!  Pass it along to the BM circuit
                --was there an error in the packet?
                if( (inID2(7) = '1') and (inID2(7) = '1') )then
                        --then it is an error packet
                        sendError := 1;
                        badBM := inSource2;
                end if;
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination2(0) = ID1(0)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter1) then
                        --forward packet to ID3
                        outDestination3 <= inDestination2;
                        outSource3 <= inSource2;
                        outData3 <= inData2;
                        outID3 <= inID2;
                        outStart3   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination2;
                        outSource1 <= inSource2;
                        outData1 <= inData2;
                        outID1 <= inID2;
                        outStart1   <= '1';
                end if;
        end if;
end if;
if(inStart3 = '1') then
        if(inDestination3 = myID) then
                --this packet is for here! Pass it along to the BM circuit
                --was there an error in the packet?
                if( (inID3(7) = '1') and (inID3(7) = '1') )then
                        --then it is an error packet
                        sendError := 1;
                        badBM := inSource3;
                end if;
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination3(0) = ID1(0)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(0) = ID2(0)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(1) = ID2(1)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(2) = ID2(2)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(destCounter2 > destCounter1) then
                        --forward packet to ID3
                        outDestination2 <= inDestination3;
                        outSource2 <= inSource3;
                        outData2 <= inData3;
                outID2 <= inID3;
                outStart2   <= '1';
            else
                --forward packet to ID1
                outDestination1 <= inDestination3;
                outSource1 <= inSource3;
                outData1 <= inData3;
                outID1 <= inID3;
                outStart1   <= '1';
            end if;
        end if;
end if;
if(sendError = 1) then
        --send a packet to "000" containing a message that says:don't use BM located in
        --badBM

        --which link is closest to 000?
        destCounter1 := 0;
        destCounter2 := 0;
        destCounter3 := 0;
        for i in 0 to 2 loop
            if(ID1(i) = '0') then
                destCounter1 := destCounter1 + 1;
            end if;
            if(ID2(i) = '0') then
                destCounter2 := destCounter2 + 1;
            end if;
            if(ID3(i) = '0') then
                destCounter3 := destCounter3 + 1;
            end if;
        end loop;
        if(destCounter1 > destCounter2) then
            if(destCounter1 > destCounter3) then
                --send down link1
                outDestination1 <= "000";
                outData1(7) <= '0';
                outData1(6) <= '0';
                outData1(5) <= '0';
                outData1(4) <= '0';
                outData1(3) <= '0';
                outData1(2) <= badBM(2);
                outData1(1) <= badBM(1);
                outData1(0) <= badBM(0);
                outID1 <= "00000000";
                outSource1 <= myID;
                outStart1   <= '1';

            else
                --send down link3
                outDestination3 <= "000";
                outData3(7) <= '0';
                outData3(6) <= '0';
                outData3(5) <= '0';
                outData3(4) <= '0';
                outData3(3) <= '0';
                outData3(2) <= badBM(2);
                outData3(1) <= badBM(1);
                outData3(0) <= badBM(0);
                outID3 <= "00000000";
                outSource3 <= myID;
                outStart3   <= '1';
            end if;
        else
            if(destCounter2 > destCounter3) then
                --send down link2
                outDestination2 <= "000";
                outData2(7) <= '0';
                outData2(6) <= '0';
                outData2(5) <= '0';
                outData2(4) <= '0';
```

```
                    outData2(3) <= '0';
                    outData2(2) <= badBM(2);
                    outData2(1) <= badBM(1);
                    outData2(0) <= badBM(0);
                    outID2 <= "00000000";
                    outSource2 <= myID;
                    outStart2   <= '1';
                else
                    --send down link3
                    outDestination3 <= "000";
                    outData3(7) <= '0';
                    outData3(6) <= '0';
                    outData3(5) <= '0';
                    outData3(4) <= '0';
                    outData3(3) <= '0';
                    outData3(2) <= badBM(2);
                    outData3(1) <= badBM(1);
                    outData3(0) <= badBM(0);
                    outID3 <= "00000000";
                    outSource3 <= myID;
                    outStart3   <= '1';
                end if;
            end if;
        end if;
    end if;
    end process;
end PEAssertdescription;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


ENTITY PE_Final_Assert IS
    PORT(
        clk                 : in std_logic;
        inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
        inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
        inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
        inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
        inStart1            : in std_logic; --goes high when the three things above are ready
        inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
        inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
        inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
        inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
        inStart2            : in std_logic; --goes high when the three things above are ready
        inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
        inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
        inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
        inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
        inStart3            : in std_logic; --goes high when the three things above are ready
        myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
        ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
        ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
        ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
        outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
        outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
        outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
        outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
        outStart1           : out std_logic;
        outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
        outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
        outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
        outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
        outStart2           : out std_logic;
        outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
        outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
        outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
```

```
                    outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                    outStart3           : out std_logic);
END PE_Final_Assert;

ARCHITECTURE PEFinalAssertdescription OF PE_Final_Assert IS


begin
        process(clk)

                variable destCounter1 : integer range 0 to 3;
                variable destCounter2 : integer range 0 to 3;
                variable destCounter3 : integer range 0 to 3;

                variable sendError  : integer range 0 to 1;--goes high if error is detected (so a packet must be sent to the
                                                            --control so that that PE is now neglected)
                variable badBM                  : std_logic_vector(2 downto 0);--represents which BM is faulty

                begin
                 if(rising_edge(clk)) then
                        outStart1 <= '0';
                        outStart2 <= '0';
                        outStart3 <= '0';
                        sendError := 0;
                        if(inStart1 = '1') then
                                --something has come in off of the 1 lines
                                if(inDestination1 = myID) then
                                        --this packet is for here!
                                        --is there an error in the packet?
                                        if( (inID1(7) = '1') and (inID2(7) = '1') ) then
                                                --then it is an error packet
                                                sendError := 1;
                                                badBM := inSource1;
                                        end if;

                                else
                                        --then we must forward the packet
                                        --forward the packet to the neighbour with the closest ID
                                        destCounter2 := 0;
                                        destCounter3 := 0;
                                        if(inDestination1(0) = ID2(0)) then
                                                destCounter2 := destCounter2 + 1;
                                        end if;
                                        if(inDestination1(1) = ID2(1)) then
                                                destCounter2 := destCounter2 + 1;
                                        end if;
                                        if(inDestination1(2) = ID2(2)) then
                                                destCounter2 := destCounter2 + 1;
                                        end if;
                                        if(inDestination1(0) = ID3(0)) then
                                                destCounter3 := destCounter3 + 1;
                                        end if;
                                        if(inDestination1(1) = ID3(1)) then
                                                destCounter3 := destCounter3 + 1;
                                        end if;
                                        if(inDestination1(2) = ID3(2)) then
                                                destCounter3 := destCounter3 + 1;
                                        end if;
                                        if(destCounter3 > destCounter2) then
                                                --forward packet to ID3
                                                outDestination3 <= inDestination1;
                                                outData3 <= inData1;
                                                outID3 <= inID1;
                                                outSource3 <= inSource1;
                                                outStart3   <= '1';
                                        else
                                                --forward packet to ID2
                                                outDestination2 <= inDestination1;
                                                outData2 <= inData1;
                        outID2 <= inID1;
                        outSource2 <= inSource1;
                        outStart2   <= '1';
                end if;
        end if;
end if;
if(inStart2 = '1') then
        if(inDestination2 = myID) then
                --this packet is for here! Pass it along to the BM circuit
                --was there an error in the packet?
                if( (inID2(7) = '1') and (inID2(7) = '1') )then
                        --then it is an error packet
                        sendError := 1;
                        badBM := inSource2;
                end if;
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination2(0) = ID1(0)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter1) then
                        --forward packet to ID3
                        outDestination3 <= inDestination2;
                        outSource3 <= inSource2;
                        outData3 <= inData2;
                        outID3 <= inID2;
                        outStart3   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination2;
                        outSource1 <= inSource2;
                        outData1 <= inData2;
                        outID1 <= inID2;
                        outStart1   <= '1';
                end if;
        end if;
end if;
if(inStart3 = '1') then
        if(inDestination3 = myID) then
                --this packet is for here! Pass it along to the BM circuit
                --was there an error in the packet?
                if( (inID3(7) = '1') and (inID3(7) = '1') )then
                        --then it is an error packet
                        sendError := 1;
                        badBM := inSource3;
                end if;
        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination3(0) = ID1(0)) then
```

```
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(1) = ID1(1)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(2) = ID1(2)) then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination3(0) = ID2(0)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(1) = ID2(1)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(inDestination3(2) = ID2(2)) then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(destCounter2 > destCounter1) then
                        --forward packet to ID3
                        outDestination2 <= inDestination3;
                        outSource2 <= inSource3;
                        outData2 <= inData3;
                        outID2 <= inID3;
                        outStart2   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination3;
                        outSource1 <= inSource3;
                        outData1 <= inData3;
                        outID1 <= inID3;
                        outStart1   <= '1';
                end if;
        end if;
end if;
if(sendError = 1) then
        --send a packet to "000" containing a message that says:don't use BM located in
        --badBM


        --which link is closest to 000?
        destCounter1 := 0;
        destCounter2 := 0;
        destCounter3 := 0;
        for i in 0 to 2 loop
                if(ID1(i) = '0') then
                        destCounter1 := destCounter1 + 1;
                end if;
                if(ID2(i) = '0') then
                        destCounter2 := destCounter2 + 1;
                end if;
                if(ID3(i) = '0') then
                        destCounter3 := destCounter3 + 1;
                end if;
        end loop;
        if(destCounter1 > destCounter2) then
                if(destCounter1 > destCounter3) then
                        --send down link1
                        outDestination1 <= "000";
                        outData1 <= "00000000";
                        outID1 <= "00000000";
                        outSource1 <= myID;
                        outStart1   <= '1';

                else
                        --send down link3
                        outDestination3 <= "000";
                        outData3 <= "00000000";
                        outID3 <= "00000000";
                        outSource3 <= myID;
                        outStart3   <= '1';
```

```
                                        end if;
                            else
                                        if(destCounter2 > destCounter3) then
                                                    --send down link2
                                                    outDestination2 <= "000";
                                                    outData2 <= "00000000";
                                                    outID2 <= "00000000";
                                                    outSource2 <= myID;
                                                    outStart2   <= '1';
                                        else
                                                    --send down link3
                                                    outDestination3 <= "000";
                                                    outData3 <= "00000000";
                                                    outID3 <= "00000000";
                                                    outSource3 <= myID;
                                                    outStart3   <= '1';
                                        end if;
                            end if;
                  end if;
         end if;
        end process;
end PEFinalAssertdescription;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


ENTITY PE_CPU IS--the interface and control circuitry "around" the cpu
                    PORT(
                              clk                  : in std_logic;
                              inSignalFromCPU      : in std_logic;--there is data available from the CPU
                              inDataFromCPU        : IN STD_ULOGIC_VECTOR(16 downto 0);--8 downto 0);--the data
                                                                                     --from the CPU
                              inDestination1       : IN STD_LOGIC_VECTOR(2 downto 0);
                              inSource1            : IN STD_LOGIC_VECTOR(2 downto 0);
                              inData1              : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inID1                : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inStart1             : in std_logic; --goes high when the three things above are ready
                              inDestination2       : IN STD_LOGIC_VECTOR(2 downto 0);
                              inSource2            : IN STD_LOGIC_VECTOR(2 downto 0);
                              inData2              : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inID2                : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inStart2             : in std_logic; --goes high when the three things above are ready
                              inDestination3       : IN STD_LOGIC_VECTOR(2 downto 0);
                              inSource3            : IN STD_LOGIC_VECTOR(2 downto 0);
                              inData3              : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inID3                : IN STD_ULOGIC_VECTOR(7 downto 0);
                              inStart3             : in std_logic; --goes high when the three things above are ready
                              myID                 : in std_logic_vector(2 downto 0);--what this PE's ID is
                              ID1                  : in std_logic_vector(2 downto 0);--what PE 1's ID is
                              ID2                  : in std_logic_vector(2 downto 0);--what PE 2's ID is
                              ID3                  : in std_logic_vector(2 downto 0);--what PE 3's ID is
                              outSignalToCPU       : out std_logic;--there is data available from the CPU
                              --outDataToCPU       : out STD_ULOGIC_VECTOR(8 downto 0);--the data from the CPU
                              outDataToCPU         : out STD_ULOGIC_VECTOR(16 downto 0);--the data from the CPU
                              outDestination1      : out STD_LOGIC_VECTOR(2 downto 0);
                              outSource1           : out STD_LOGIC_VECTOR(2 downto 0);
                              outData1             : out STD_ULOGIC_VECTOR(7 downto 0);
                              outID1               : out STD_ULOGIC_VECTOR(7 downto 0);
                              outStart1            : out std_logic;
                              outDestination2      : out STD_LOGIC_VECTOR(2 downto 0);
                              outSource2           : out STD_LOGIC_VECTOR(2 downto 0);
                              outData2             : out STD_ULOGIC_VECTOR(7 downto 0);
                              outID2               : out STD_ULOGIC_VECTOR(7 downto 0);
                              outStart2            : out std_logic;
                              outDestination3      : out STD_LOGIC_VECTOR(2 downto 0);
                              outSource3           : out STD_LOGIC_VECTOR(2 downto 0);
                    outData3                : out STD_ULOGIC_VECTOR(7 downto 0);
                    outID3                  : out STD_ULOGIC_VECTOR(7 downto 0);
--temp temp
                    tempBMCounter : out integer range 0 to 13;
                    tempDestinationBM : out std_logic_vector(2 downto 0);
                    outStart3                                           : out std_logic);
END PE_CPU;

ARCHITECTURE PECPU OF PE_CPU IS

component reg8 IS
  PORT(load           : IN std_logic;
            clock     : in std_logic;
     datain : IN std_ulogic_vector(7 downto 0);

     dataout : OUT std_ulogic_vector(7 downto 0)); --parallel outputs
END component;

signal loadIReg : std_logic_vector(13 downto 0);
signal loadJReg : std_logic_vector(13 downto 0);
signal BMIntermediateI : std_ulogic_vector(7 downto 0);
signal BMIntermediateJ : std_ulogic_vector(7 downto 0);
subtype WORD8 is STD_ULOGIC_VECTOR (7 downto 0);
type ARRAYOFBYTES14 is array (13 downto 0) of WORD8;
signal BMIData : ARRAYOFBYTES14;
signal BMJData : ARRAYOFBYTES14;
--signal BM1Destination : std_logic_vector(2 downto 0) := "001";
--signal BM2Destination : std_logic_vector(2 downto 0) := "010";
--signal SpareBM1     : std_logic_vector(2 downto 0) := "101";
--signal SpareBM2     : std_logic_vector(2 downto 0) := "110";
--signal SpareBM3     : std_logic_vector(2 downto 0) := "111";
--signal destinationBM : std_logic_vector(2 downto 0) := "001";
--signal BMcounter : integer range 0 to 13 := 0;

begin
        --reg1: for i in 0 to 13 generate
        --          reg1Label: reg8 port map(loadIReg(i), clk, BMIntermediateI, BMIData(i));
        --          reg2Label: reg8 port map(loadJReg(i), clk, BMIntermediateJ, BMJData(i));
        --end generate;

        process(clk)

                variable destCounter1 : integer range 0 to 3;
                variable destCounter2 : integer range 0 to 3;
                variable destCounter3 : integer range 0 to 3;
                variable BMcounter : integer range 0 to 13 := 0;

                variable BM1Destination : std_logic_vector(2 downto 0) := "001";
                variable BM2Destination : std_logic_vector(2 downto 0) := "010";
                variable SpareBM1     : std_logic_vector(2 downto 0) := "101";
                variable SpareBM2     : std_logic_vector(2 downto 0) := "110";
                variable SpareBM3     : std_logic_vector(2 downto 0) := "111";
                variable destinationBM : std_logic_vector(2 downto 0) := "001";

                variable BMOutCounter : integer := 0;

                variable packetToGo : integer range 0 to 1 := 0;
                variable toGoDestination : std_logic_vector(2 downto 0);
                variable toGoData : std_ulogic_vector(7 downto 0);
                variable toGoLink : integer range 1 to 3;

                begin

                 tempDestinationBM <= destinationBM;
                 tempBMCounter <= BMCounter;

                 if(rising_edge(clk)) then
                        outStart1 <= '0';
                        outStart2 <= '0';
                        outStart3 <= '0';
```

```
outSignalToCPU <= '0';
loadIReg <= (others => '0');
loadJReg <= (others => '0');
if(inStart1 = '1') then
          --something has come in off of the 1 lines
          if(inDestination1 = myID) then
                    --this packet is for here!
                    --if it is from a BM circuit we need to have that data ready to give
                    --to the LEDS register
                    --if it is from the BM assert then we need to cancel use of the given BM
                    --circuit
                    if(inSource1 = "011") then
                              --then it is an assert signal to block the use of a BM circuit
                              --so simply change the BM#Destination signals and spareBM
                              --stuff
                              if(SpareBM1 = "000") then
                                        --there are no spares left - do nothing
                              else
                              if( (inData1(2) = BM1Destination(2)) and (inData1(1) =
          BM1Destination(1)) and (inData1(0) = BM1Destination(0)) ) then
                                                  --BM1 is to be replaced
                                                  BM1Destination := SpareBM1;
                                                  --now move the others down the line
                                                  SpareBM1 := SpareBM2;
                                                  SpareBM2 := SpareBM3;
                                                  SpareBM3 := "000";
                                        else
                                                  --BM2 is to be replaced
                                                  BM2Destination := SpareBM1;
                                                  --now move the others down the line
                                                  SpareBM1 := SpareBM2;
                                                  SpareBM2 := SpareBM3;
                                                  SpareBM3 := "000";
                                        end if;
                              end if;
                    elsif(inSource1 = "100") then
                              --then it is an error on the final task, we are not supporting
                              --error recovery in this case since we have no spare general
                              --purpose CPU
                    else
                              --it is info from a BM circuit - that needs to be stored in the
                              --corresponding place
                              BMIntermediateI <= inData1;
                              BMIntermediateJ <= inID1;
                              loadIReg(BMCounter) <= '1';
                              loadJReg(BMCounter) <= '1';
                              BMCounter := BMCounter + 1;
                              if(BMCounter > 13) then
                                        BMCounter := 0;
                              end if;
                              --now just send it directly to the LEDS for reading by
                              --software
                              outDataToCPU(15 downto 8) <= inID1(7 downto 0);
                              outDataToCPU(7 downto 0) <= inData1(7 downto 0);
                              outSignalToCPU <= '1';
                    end if;

          else
                    --then we must forward the packet
                    --forward the packet to the neighbour with the closest ID
                    destCounter2 := 0;
                    destCounter3 := 0;
                    if(inDestination1(0) = ID2(0)) then
                              destCounter2 := destCounter2 + 1;
                    end if;
                    if(inDestination1(1) = ID2(1)) then
                              destCounter2 := destCounter2 + 1;
                    end if;
                    if(inDestination1(2) = ID2(2)) then
                              destCounter2 := destCounter2 + 1;
```

```
                end if;
                if(inDestination1(0) = ID3(0)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(1) = ID3(1)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination1(2) = ID3(2)) then
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter2) then
                        --forward packet to ID3
                        outDestination3 <= inDestination1;
                        outData3 <= inData1;
                        outID3 <= inID1;
                        outSource3 <= inSource1;
                        outStart3   <= '1';
                else
                        --forward packet to ID2
                        outDestination2 <= inDestination1;
                        outData2 <= inData1;
                        outID2 <= inID1;
                        outSource2 <= inSource1;
                        outStart2   <= '1';
                end if;
        end if;
end if;
if(inStart2 = '1') then
        if(inDestination2 = myID) then
                --this packet is for here!
                --if it is from a BM circuit we need to have that data ready to give
                --to the LEDS register
                --if it is from the BM assert then we need to cancel use of the given BM
                --circuit
                if(inSource2 = "011") then
                        --then it is an assert signal to block the use of a BM circuit
                        --so simply change the BM#Destination signals and spareBM
                        --stuff
                        if(SpareBM1 = "000") then
                                --there are no spares left - do nothing
                        else
                        if( (inData2(2) = BM1Destination(2)) and (inData2(1) =
        BM1Destination(1)) and (inData2(0) = BM1Destination(0)) ) then
                                        --BM1 is to be replaced
                                        BM1Destination := SpareBM1;
                                        --now move the others down the line
                                        SpareBM1 := SpareBM2;
                                        SpareBM2 := SpareBM3;
                                        SpareBM3 := "000";
                                else
                                        --BM2 is to be replaced
                                        BM2Destination := SpareBM1;
                                        --now move the others down the line
                                        SpareBM1 := SpareBM2;
                                        SpareBM2 := SpareBM3;
                                        SpareBM3 := "000";
                                end if;
                        end if;
                elsif(inSource1 = "100") then
                        --then it is an error on the final task, we are not supporting
                        --error recovery in this case since we have no spare general
                        --purpose CPU
                else
                        --it is info from a BM circuit - that needs to be stored in the
                        --corresponding place
                        BMIntermediateI <= inData2;
                        BMIntermediateJ <= inID2;
                        loadIReg(BMCounter) <= '1';
                        loadJReg(BMCounter) <= '1';
                        BMCounter := BMCounter + 1;
                        if(BMCounter > 13) then
                                BMCounter := 0;
                        end if;
                        --now just send it directly to the LEDS for reading by
                        --software
                        --outDataToCPU(8) <= '0';
                        outDataToCPU(15 downto 8) <= inID2(7 downto 0);
                        outDataToCPU(7 downto 0) <= inData2(7 downto 0);
                        outSignalToCPU <= '1';
                end if;

        else
                --then we must forward the packet
                --forward the packet to the neighbour with the closest ID
                destCounter1 := 0;
                destCounter3 := 0;
                if(inDestination2(0) = ID1(0)) then
                else
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(1) = ID1(1)) then
                else
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(2) = ID1(2)) then
                else
                        destCounter1 := destCounter1 + 1;
                end if;
                if(inDestination2(0) = ID3(0)) then
                else
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(1) = ID3(1)) then
                else
                        destCounter3 := destCounter3 + 1;
                end if;
                if(inDestination2(2) = ID3(2)) then
                else
                        destCounter3 := destCounter3 + 1;
                end if;
                if(destCounter3 > destCounter1) then
                        --forward packet to ID3
                        outDestination3 <= inDestination2;
                        outSource3 <= inSource2;
                        outData3 <= inData2;
                        outID3 <= inID2;
                        outStart3   <= '1';
                else
                        --forward packet to ID1
                        outDestination1 <= inDestination2;
                        outSource1 <= inSource2;
                        outData1 <= inData2;
                        outID1 <= inID2;
                        outStart1   <= '1';
                end if;
        end if;
end if;
if(inStart3 = '1') then
        if(inDestination3 = myID) then
                --this packet is for here!
                --if it is from a BM circuit we need to have that data ready to give
                --to the LEDS register
                --if it is from the BM assert then we need to cancel use of the given BM
                --circuit
                if(inSource3 = "011") then
                        --then it is an assert signal to block the use of a BM circuit
                        --so simply change the BM#Destination signals and spareBM
                        --stuff
                        if(SpareBM1 = "000") then
                                --there are no spares left - do nothing
```

```
            else
                    if( (inData3(2) = BM1Destination(2)) and
(inData3(1) = BM1Destination(1)) and (inData3(0) = BM1Destination(0))) then
                            --BM1 is to be replaced
                            BM1Destination := SpareBM1;
                            --now move the others down the line
                            SpareBM1 := SpareBM2;
                            SpareBM2 := SpareBM3;
                            SpareBM3 := "000";
                    else
                            --BM2 is to be replaced
                            BM2Destination := SpareBM1;
                            --now move the others down the line
                            SpareBM1 := SpareBM2;
                            SpareBM2 := SpareBM3;
                            SpareBM3 := "000";
                    end if;
            end if;
        elsif(inSource1 = "100") then
            --then it is an error on the final task, we are not supporting
            --error recovery in this case since we have no spare general
            --purpose CPU
        else
            --it is info from a BM circuit - that needs to be stored in the
            --corresponding place
            BMIntermediateI <= inData3;
            BMIntermediateJ <= inID3;
            loadIReg(BMCounter) <= '1';
            loadJReg(BMCounter) <= '1';
            BMCounter := BMCounter + 1;
            if(BMCounter > 13) then
                    BMCounter := 0;
            end if;
            --now just send it directly to the LEDS for reading by
            --software
            --outDataToCPU(8) <= '0';
            outDataToCPU(15 downto 8) <= inID3(7 downto 0);
            outDataToCPU(7 downto 0) <= inData3(7 downto 0);
            outSignalToCPU <= '1';
        end if;
else
        --then we must forward the packet
        --forward the packet to the neighbour with the closest ID
        destCounter1 := 0;
        destCounter3 := 0;
        if(inDestination3(0) = ID1(0)) then
            destCounter1 := destCounter1 + 1;
        end if;
        if(inDestination3(1) = ID1(1)) then
            destCounter1 := destCounter1 + 1;
        end if;
        if(inDestination3(2) = ID1(2)) then
            destCounter1 := destCounter1 + 1;
        end if;
        if(inDestination3(0) = ID2(0)) then
            destCounter2 := destCounter2 + 1;
        end if;
        if(inDestination3(1) = ID2(1)) then
            destCounter2 := destCounter2 + 1;
        end if;
        if(inDestination3(2) = ID2(2)) then
            destCounter2 := destCounter2 + 1;
        end if;
        if(destCounter2 > destCounter1) then
            --forward packet to ID3
            outDestination2 <= inDestination3;
            outSource2 <= inSource3;
            outData2 <= inData3;
            outID2 <= inID3;
            outStart2  <= '1';
```

```
                        else
                                --forward packet to ID1
                                outDestination1 <= inDestination3;
                                outSource1 <= inSource3;
                                outData1 <= inData3;
                                outID1 <= inID3;
                                outStart1   <= '1';
                        end if;
                end if;
        end if;
        if(inSignalFromCPU = '1') then
                --then there is incoming data from the CPU it needs to be distributed to a BM
                --circuit
                --or if it is a signalling packet we need to put info on the LEDS for reading....
                if(inDataFromCPU(16) = '1') then
                        --then it is signalling and we should load the contents of the register
                        --indexed
                        --in the lower 7 bits to the LEDS
                        if(inDataFromCPU(15) = '1') then
                                if(inDataFromCPU(14) = '1') then
                                        --signal from CPU to change the BM destination
                                        --circuit!
                                        if(SpareBM1 = "000") then
                                                --there are no spares left - do nothing
                                        else
                                                if( (inDataFromCPU(2) =
                                        BM1Destination(2)) and (inDataFromCPU(1) =
                                        BM1Destination(1)) and (inDataFromCPU(0) =
                                        BM1Destination(0))) then
                                                        --BM1 is to be replaced
                                                        BM1Destination :=
                                                                SpareBM1;
                                                        --now move the others
                                                        --down the line
                                                        SpareBM1 := SpareBM2;
                                                        SpareBM2 := SpareBM3;
                                                        SpareBM3 := "000";
                                                else
                                                        --BM2 is to be replaced
                                                        BM2Destination :=
                                                                SpareBM1;
                                                        --now move the others
                                                        --down the line
                                                        SpareBM1 := SpareBM2;
                                                        SpareBM2 := SpareBM3;
                                                        SpareBM3 := "000";
                                                end if;
                                        end if;

                                end if;
                        else

                        end if;
                        --outSignalToCPU <= '1';

                else
                        --then data needs to be distributed to the BM circuit!
                        --immediately send data to destinationBM
                        --which link do I send it out on?
                        packetToGo := 1;
                        toGoDestination := destinationBM;
                        toGoData := inDataFromCPU(7 downto 0);

                        destCounter1 := 0;
                        destCounter2 := 0;
                        destCounter3 := 0;

                        if(destinationBM(2) = ID1(2)) then
                                destCounter1 := destCounter1 + 1;
```

```
end if;
if(destinationBM(1) = ID1(1)) then
        destCounter1 := destCounter1 + 1;

end if;
if(destinationBM(0) = ID1(0)) then
        destCounter1 := destCounter1 + 1;

end if;
if(destinationBM(2) = ID2(2)) then
        destCounter2 := destCounter2 + 1;

end if;
if(destinationBM(1) = ID2(1)) then
        destCounter2 := destCounter2 + 1;

end if;
if(destinationBM(0) = ID2(0)) then
        destCounter2 := destCounter2 + 1;

end if;
if(destinationBM(2) = ID3(2)) then
        destCounter3 := destCounter3 + 1;

end if;
if(destinationBM(1) = ID3(1)) then
        destCounter3 := destCounter3 + 1;

end if;
if(destinationBM(0) = ID3(0)) then
        destCounter3 := destCounter3 + 1;

end if;
if(destCounter1 > destCounter2) then
        if(destCounter1 > destCounter3) then
                --send to ID1
                outDestination1 <= destinationBM;
                outData1 <= inDataFromCPU(7 downto 0);
                outSource1 <= "000";
                outID1 <= "00000000";
                outStart1 <= '1';
                toGoLink := 1;
        else
                --send to ID3
                outDestination3 <= destinationBM;
                outData3 <= inDataFromCPU(7 downto 0);
                outSource3 <= "000";
                outID3 <= "00000000";
                outStart3 <= '1';
                toGoLink := 3;
        end if;
else
        if(destCounter2 > destCounter3) then
                --send to ID2
                outDestination2 <= destinationBM;
                outData2 <= inDataFromCPU(7 downto 0);
                outSource2 <= "000";
                outID2 <= "00000000";
                outStart2 <= '1';
                toGoLink := 2;
        else
                --send to ID3
                outDestination3 <= destinationBM;
                outData3 <= inDataFromCPU(7 downto 0);
                outSource3 <= "000";
                outID3 <= "00000000";
                outStart3 <= '1';
                toGoLink := 3;
        end if;
end if;
```

```
                    BMOutCounter := BMOutCounter + 1;
                    if(BMOutCounter = 1951509) then--3903018) then
                        --it is time to change destinationBM! (swap...)
                        if(destinationBM = BM1Destination) then
                            destinationBM := BM2Destination;
                        else
                            destinationBM := BM1Destination;

                        end if;
                        BMOutCounter := 0;--start again!
                    end if;

                end if;
            end if;
            if(packetToGo = 1) then
                if(toGoLink = 1) then
                    outDestination1 <= toGoDestination;
                    outData1 <= toGoData;
                    outSource1 <= "000";
                    outID1 <= "00000000";
                    outStart1 <= '1';
                elsif(toGoLink = 2) then
                    outDestination2 <= toGoDestination;
                    outData2 <= toGoData;
                    outSource2 <= "000";
                    outID2 <= "00000000";
                    outStart2 <= '1';
                else
                    outDestination3 <= toGoDestination;
                    outData3 <= toGoData;
                    outSource3 <= "000";
                    outID3 <= "00000000";
                    outStart3 <= '1';
                end if;


            end if;
        end if;
        end process;
end PECPU;

-- --------------------------------------------------------------------------
-- Purpose :
--        This APB peripheral contains registers
--
-- --=========================================================================

library IEEE;
use IEEE.std_logic_1164.all;

-- --------------------------------------------------------------------------

entity APBRegs is
  port (
-- Inputs
      PCLK          : in   std_logic; -- APB clock
      nRESET        : in   std_logic; -- AMBA reset
      PENABLE       : in   std_logic; -- APB enable
      PSEL          : in   std_logic; -- APB select
      PWRITE        : in   std_logic; -- APB read/write
      nPBUTT        : in   std_logic; -- input that will be latched for
                           -- an interrupt example
      SW            : in   std_logic_vector(7 downto 0);
                           -- switches
      PWDATA        : in   std_logic_vector(31 downto 0);
                           -- APB write data
      PA            : in   std_logic_vector(4 downto 2);
                           -- APB address bus
-- Outputs
      CTRLCLK1      : out  std_logic_vector(18 downto 0);
```

```
                        -- sets frequency of CLK1
    CTRLCLK2        : out   std_logic_vector(18 downto 0);
                        -- sets frequency of CLK2
    REGSINT         : out   std_logic; -- interrupt output
    LED           : out   std_logic_vector(8 downto 0);
                        -- LED control
    PRDATA          : out   std_logic_vector(31 downto 0)
                        -- APB read data
    );
end APBRegs;

-- ----------------------------------------------------------------------------
--
--                      APBRegs
--                      =======
--
-- ----------------------------------------------------------------------------
--
-- Overview
-- ========
-- This APB peripheral contains registers to...
-- * program & lock the two clock oscillators
-- * write to the general purpose LEDs
-- * clear push button interrupt
-- * read the general purpose switches
--
-- Certain registers are protected by the LOCK register. You must write 0xA05F
-- to the lock register to enable the following registers to be modified:
--
--  LM_OSC1
--  LM_OSC2
--
-- Provides nLMINT to the top level & registers all interrupt sources
--
-- ----------------------------------------------------------------------------

-- --================================ ARCHITECTURE ==================================--

architecture synth of APBRegs is



-- ----------------------------------------------------------------------------
-- Component declarations
-- ----------------------------------------------------------------------------

component PE_BlockMatch IS
                    PORT(
                            clk                 : in std_logic;
                            inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
                            inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
                            inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inStart1            : in std_logic;  --goes high when the three things above are ready
                            inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
                            inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
                            inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inStart2            : in std_logic;  --goes high when the three things above are ready
                            inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
                            inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
                            inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
                            inStart3            : in std_logic;  --goes high when the three things above are ready
                            myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
                            ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
                            ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
                            ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
                            outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart1           : out std_logic;
                        outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart2           : out std_logic;
                        outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart3           : out std_logic);
END component;

component PE_BM_Assert IS
                PORT(
                        clk                 : in std_logic;
                        inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart1            : in std_logic; --goes high when the three things above are ready
                        inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart2            : in std_logic; --goes high when the three things above are ready
                        inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart3            : in std_logic; --goes high when the three things above are ready
                        myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
                        ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
                        ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
                        ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
                        outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart1           : out std_logic;
                        outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart2           : out std_logic;
                        outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart3           : out std_logic);
END component;


component PE_Final_Assert IS
                PORT(
                        clk                 : in std_logic;
                        inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart1            : in std_logic; --goes high when the three things above are ready
                        inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart2            : in std_logic; --goes high when the three things above are ready
                        inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart3            : in std_logic; --goes high when the three things above are ready
                        myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
                        ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
                        ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
                        ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
                        outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart1           : out std_logic;
                        outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart2           : out std_logic;
                        outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart3           : out std_logic);
END component;

component PE_CPU IS--the interface and control circuitry "around" the cpu
                  PORT(
                        clk                 : in std_logic;
                        inSignalFromCPU     : in std_logic;--there is data available from the CPU
                        inDataFromCPU       : IN STD_ULOGIC_VECTOR(16 downto 0);--the data from the CPU
                        inDestination1      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource1           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData1             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID1               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart1            : in std_logic; --goes high when the three things above are ready
                        inDestination2      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource2           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData2             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID2               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart2            : in std_logic; --goes high when the three things above are ready
                        inDestination3      : IN STD_LOGIC_VECTOR(2 downto 0);
                        inSource3           : IN STD_LOGIC_VECTOR(2 downto 0);
                        inData3             : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inID3               : IN STD_ULOGIC_VECTOR(7 downto 0);
                        inStart3            : in std_logic; --goes high when the three things above are ready
                        myID                : in std_logic_vector(2 downto 0);--what this PE's ID is
                        ID1                 : in std_logic_vector(2 downto 0);--what PE 1's ID is
                        ID2                 : in std_logic_vector(2 downto 0);--what PE 2's ID is
                        ID3                 : in std_logic_vector(2 downto 0);--what PE 3's ID is
                        outSignalToCPU      : out std_logic;--there is data available from the CPU
--                      outDataToCPU        : out STD_ULOGIC_VECTOR(8 downto 0);--the data from the CPU
                        outDataToCPU        : out STD_ULOGIC_VECTOR(16 downto 0);--the data from the CPU
                        outDestination1     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource1          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData1            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID1              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart1           : out std_logic;
                        outDestination2     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource2          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData2            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID2              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart2           : out std_logic;
                        outDestination3     : out STD_LOGIC_VECTOR(2 downto 0);
                        outSource3          : out STD_LOGIC_VECTOR(2 downto 0);
                        outData3            : out STD_ULOGIC_VECTOR(7 downto 0);
                        outID3              : out STD_ULOGIC_VECTOR(7 downto 0);
                        outStart3           : out std_logic);
END component;
-- ---------------------------------------------------------------------------
-- Constant declarations
-- ---------------------------------------------------------------------------
-- 1 MHz default clock values
constant OSC1_VECTOR     : std_logic_vector(18 downto 0)
                 := "1100111110000000100";

constant OSC2_VECTOR     : std_logic_vector(18 downto 0)
                 := "1100111110000000100";

-- Lock register key 0xA05F
constant LOCK_KEY        : std_logic_vector(15 downto 0)
                 := "1010000001011111";

-- Address decoding
constant LM_OSC1         : std_logic_vector(4 downto 2) := "000";
-- read/write

constant LM_OSC2         : std_logic_vector(4 downto 2) := "001";
-- read/write

constant LM_LOCK         : std_logic_vector(4 downto 2) := "010";
-- read/write

constant LM_LEDS         : std_logic_vector(4 downto 2) := "011";
-- read/write

constant LM_INT          : std_logic_vector(4 downto 2) := "100";
-- read/write

constant LM_SW           : std_logic_vector(4 downto 2) := "101";
-- read only

-- ---------------------------------------------------------------------------
-- Signal declarations
-- ---------------------------------------------------------------------------
signal LmOscReg1       : std_logic_vector(18 downto 0);
-- Oscillator register1

signal LmOscReg2       : std_logic_vector(18 downto 0);
-- Oscillator register2

signal LmLckReg        : std_logic_vector(15 downto 0);
-- Lock register

--signal LmLedsReg       : std_logic_vector(8 downto 0);
signal LmLedsReg       : std_logic_vector(16 downto 0);
-- LED register

signal LmIntReg        : std_logic;
-- INT register

signal LmSwReg         : std_logic_vector(7 downto 0);
-- Switch register

signal Locked          : std_logic;
-- Registers are Locked

signal NextPRDATA      : std_logic_vector(31 downto 0);
-- read data

-- a duplex bus based commlink with a bunch a necessary signals (from PE 000 to PE 001)
signal commDest000_001 : std_logic_vector(2 downto 0);
signal commSource000_001 : std_logic_vector(2 downto 0);
signal commData000_001 : std_ulogic_vector(7 downto 0);
signal commID000_001 : std_ulogic_vector(7 downto 0);
signal commStart000_001 : std_logic;

signal commDest001_000 : std_logic_vector(2 downto 0);
signal commSource001_000 : std_logic_vector(2 downto 0);
signal commData001_000 : std_ulogic_vector(7 downto 0);
signal commID001_000 : std_ulogic_vector(7 downto 0);
signal commStart001_000 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest000_010 : std_logic_vector(2 downto 0);
signal commSource000_010 : std_logic_vector(2 downto 0);
signal commData000_010 : std_ulogic_vector(7 downto 0);
signal commID000_010 : std_ulogic_vector(7 downto 0);
signal commStart000_010 : std_logic;

signal commDest010_000 : std_logic_vector(2 downto 0);
signal commSource010_000 : std_logic_vector(2 downto 0);
signal commData010_000 : std_ulogic_vector(7 downto 0);
signal commID010_000 : std_ulogic_vector(7 downto 0);
signal commStart010_000 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest000_100 : std_logic_vector(2 downto 0);
signal commSource000_100 : std_logic_vector(2 downto 0);
signal commData000_100 : std_ulogic_vector(7 downto 0);
signal commID000_100 : std_ulogic_vector(7 downto 0);
signal commStart000_100 : std_logic;

signal commDest100_000 : std_logic_vector(2 downto 0);
signal commSource100_000 : std_logic_vector(2 downto 0);
signal commData100_000 : std_ulogic_vector(7 downto 0);
signal commID100_000 : std_ulogic_vector(7 downto 0);
signal commStart100_000 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest001_101 : std_logic_vector(2 downto 0);
signal commSource001_101 : std_logic_vector(2 downto 0);
signal commData001_101 : std_ulogic_vector(7 downto 0);
signal commID001_101 : std_ulogic_vector(7 downto 0);
signal commStart001_101 : std_logic;

signal commDest101_001 : std_logic_vector(2 downto 0);
signal commSource101_001 : std_logic_vector(2 downto 0);
signal commData101_001 : std_ulogic_vector(7 downto 0);
signal commID101_001 : std_ulogic_vector(7 downto 0);
signal commStart101_001 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest001_011 : std_logic_vector(2 downto 0);
signal commSource001_011 : std_logic_vector(2 downto 0);
signal commData001_011 : std_ulogic_vector(7 downto 0);
signal commID001_011 : std_ulogic_vector(7 downto 0);
signal commStart001_011 : std_logic;

signal commDest011_001 : std_logic_vector(2 downto 0);
signal commSource011_001 : std_logic_vector(2 downto 0);
signal commData011_001 : std_ulogic_vector(7 downto 0);
signal commID011_001 : std_ulogic_vector(7 downto 0);
signal commStart011_001 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest010_011 : std_logic_vector(2 downto 0);
signal commSource010_011 : std_logic_vector(2 downto 0);
signal commData010_011 : std_ulogic_vector(7 downto 0);
signal commID010_011 : std_ulogic_vector(7 downto 0);
signal commStart010_011 : std_logic;

signal commDest011_010 : std_logic_vector(2 downto 0);
signal commSource011_010 : std_logic_vector(2 downto 0);
signal commData011_010 : std_ulogic_vector(7 downto 0);
signal commID011_010 : std_ulogic_vector(7 downto 0);
signal commStart011_010 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest010_110 : std_logic_vector(2 downto 0);
signal commSource010_110 : std_logic_vector(2 downto 0);
signal commData010_110 : std_ulogic_vector(7 downto 0);
signal commID010_110 : std_ulogic_vector(7 downto 0);
signal commStart010_110 : std_logic;

signal commDest110_010 : std_logic_vector(2 downto 0);
signal commSource110_010 : std_logic_vector(2 downto 0);
signal commData110_010 : std_ulogic_vector(7 downto 0);
signal commID110_010 : std_ulogic_vector(7 downto 0);
signal commStart110_010 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest110_111 : std_logic_vector(2 downto 0);
signal commSource110_111 : std_logic_vector(2 downto 0);
signal commData110_111 : std_ulogic_vector(7 downto 0);
signal commID110_111 : std_ulogic_vector(7 downto 0);
signal commStart110_111 : std_logic;

signal commDest111_110 : std_logic_vector(2 downto 0);
signal commSource111_110 : std_logic_vector(2 downto 0);
signal commData111_110 : std_ulogic_vector(7 downto 0);
signal commID111_110 : std_ulogic_vector(7 downto 0);
signal commStart111_110 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest110_100 : std_logic_vector(2 downto 0);
signal commSource110_100 : std_logic_vector(2 downto 0);
signal commData110_100 : std_ulogic_vector(7 downto 0);
signal commID110_100 : std_ulogic_vector(7 downto 0);
signal commStart110_100 : std_logic;

signal commDest100_110 : std_logic_vector(2 downto 0);
signal commSource100_110 : std_logic_vector(2 downto 0);
signal commData100_110 : std_ulogic_vector(7 downto 0);
signal commID100_110 : std_ulogic_vector(7 downto 0);
signal commStart100_110 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest101_111 : std_logic_vector(2 downto 0);
signal commSource101_111 : std_logic_vector(2 downto 0);
signal commData101_111 : std_ulogic_vector(7 downto 0);
signal commID101_111 : std_ulogic_vector(7 downto 0);
signal commStart101_111 : std_logic;

signal commDest111_101 : std_logic_vector(2 downto 0);
signal commSource111_101 : std_logic_vector(2 downto 0);
signal commData111_101 : std_ulogic_vector(7 downto 0);
signal commID111_101 : std_ulogic_vector(7 downto 0);
signal commStart111_101 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest100_101 : std_logic_vector(2 downto 0);
signal commSource100_101 : std_logic_vector(2 downto 0);
signal commData100_101 : std_ulogic_vector(7 downto 0);
signal commID100_101 : std_ulogic_vector(7 downto 0);
signal commStart100_101 : std_logic;

signal commDest101_100 : std_logic_vector(2 downto 0);
signal commSource101_100 : std_logic_vector(2 downto 0);
signal commData101_100 : std_ulogic_vector(7 downto 0);
signal commID101_100 : std_ulogic_vector(7 downto 0);
signal commStart101_100 : std_logic;

-- a duplex bus based commlink with a bunch a necessary signals
signal commDest011_111 : std_logic_vector(2 downto 0);
signal commSource011_111 : std_logic_vector(2 downto 0);
signal commData011_111 : std_ulogic_vector(7 downto 0);
signal commID011_111 : std_ulogic_vector(7 downto 0);
signal commStart011_111 : std_logic;

signal commDest111_011 : std_logic_vector(2 downto 0);
signal commSource111_011 : std_logic_vector(2 downto 0);
signal commData111_011 : std_ulogic_vector(7 downto 0);
signal commID111_011 : std_ulogic_vector(7 downto 0);
signal commStart111_011 : std_logic;

--signal dataToLEDs : std_ulogic_vector(8 downto 0);
signal dataToLEDs : std_ulogic_vector(16 downto 0);
signal dataFromLEDs : std_ulogic_vector(16 downto 0);
signal signalToLEDs : std_logic;
signal signalFromLEDs : std_logic;

signal VAL000 : std_logic_vector(2 downto 0) := "000";
signal VAL001 : std_logic_vector(2 downto 0) := "001";
signal VAL010 : std_logic_vector(2 downto 0) := "010";
signal VAL011 : std_logic_vector(2 downto 0) := "011";
signal VAL100 : std_logic_vector(2 downto 0) := "100";
signal VAL101 : std_logic_vector(2 downto 0) := "101",
signal VAL110 : std_logic_vector(2 downto 0) := "110";
signal VAL111 : std_logic_vector(2 downto 0) := "111";
-- ---------------------------------------------------------------------------
-- Function declarations
-- ---------------------------------------------------------------------------


-- ---------------------------------------------------------------------------
--
-- Main body of code
-- ==================
--
-- ---------------------------------------------------------------------------

begin
        --if you are circuit 000 then your inputs start with 000 and your outputs end with 000
        PE000: PE_CPU port map(PCLK, signalFromLEDs, dataFromLEDs, commDest000_001, commSource000_001,
commData000_001, commID000_001, commStart000_001,
                commDest000_100, commSource000_100, commData000_100, commID000_100, commStart000_100,
                commDest000_010, commSource000_010, commData000_010, commID000_010, commStart000_010,
                VAL000, VAL001, VAL100, VAL010, signalToLEDs, dataToLEDs,
                commDest001_000, commSource001_000, commData001_000, commID001_000, commStart001_000,
                commDest100_000, commSource100_000, commData100_000, commID100_000, commStart100_000,
                commDest010_000, commSource010_000, commData010_000, commID010_000, commStart010_000);

        PE001: PE_BlockMatch port map(PCLK,
                commDest001_101, commSource001_101, commData001_101, commID001_101, commStart001_101,
                commDest001_011, commSource001_011, commData001_011, commID001_011, commStart001_011,
                commDest001_000, commSource001_000, commData001_000, commID001_000, commStart001_000,
                VAL001, VAL101, VAL011, VAL000,
                commDest101_001, commSource101_001, commData101_001, commID101_001, commStart101_001,
                commDest011_001, commSource011_001, commData011_001, commID011_001, commStart011_001,
                commDest000_001, commSource000_001, commData000_001, commID000_001, commStart000_001);

        PE010: PE_BlockMatch port map(PCLK,
                commDest010_000, commSource010_000, commData010_000, commID010_000, commStart010_000,
                commDest010_011, commSource010_011, commData010_011, commID010_011, commStart010_011,
                commDest010_110, commSource010_110, commData010_110, commID010_110, commStart010_110,
                VAL010, VAL000, VAL011, VAL110,
                commDest000_010, commSource000_010, commData000_010, commID000_010, commStart000_010,
                commDest011_010, commSource011_010, commData011_010, commID011_010, commStart011_010,
                commDest110_010, commSource110_010, commData110_010, commID110_010, commStart110_010);

        PE011: PE_BM_Assert port map(PCLK,
                commDest011_010, commSource011_010, commData011_010, commID011_010, commStart011_010,
                commDest011_111, commSource011_111, commData011_111, commID011_111, commStart011_111,
                commDest011_001, commSource011_001, commData011_001, commID011_001, commStart011_001,
                VAL011, VAL010, VAL111, VAL001,
                commDest010_011, commSource010_011, commData010_011, commID010_011, commStart010_011,
                commDest111_011, commSource111_011, commData111_011, commID111_011, commStart111_011,
                commDest001_011, commSource001_011, commData001_011, commID001_011, commStart001_011);

        PE100: PE_Final_Assert port map(PCLK,
                commDest100_101, commSource100_101, commData100_101, commID100_101, commStart100_101,
                commDest100_000, commSource100_000, commData100_000, commID100_000, commStart100_000,
                commDest100_110, commSource100_110, commData100_110, commID100_110, commStart100_110,
                VAL100, VAL101, VAL000, VAL110,
                commDest101_100, commSource101_100, commData101_100, commID101_100, commStart101_100,
                commDest000_100, commSource000_100, commData000_100, commID000_100, commStart000_100,
                commDest110_100, commSource110_100, commData110_100, commID110_100, commStart110_100);

        PE101: PE_BlockMatch port map(PCLK,
                commDest101_001, commSource101_001, commData101_001, commID101_001, commStart101_001,
                commDest101_111, commSource101_111, commData101_111, commID101_111, commStart101_111,
                commDest101_100, commSource101_100, commData101_100, commID101_100, commStart101_100,
                VAL101, VAL001, VAL111, VAL100,
                commDest001_101, commSource001_101, commData001_101, commID001_101, commStart001_101,
                commDest111_101, commSource111_101, commData111_101, commID111_101, commStart111_101,
                commDest100_101, commSource100_101, commData100_101, commID100_101, commStart100_101);

        PE110: PE_BlockMatch port map(PCLK,
                commDest110_010, commSource110_010, commData110_010, commID110_010, commStart110_010,
                commDest110_111, commSource110_111, commData110_111, commID110_111, commStart110_111,
                commDest110_100, commSource110_100, commData110_100, commID110_100, commStart110_100,
                VAL110, VAL010, VAL111, VAL100,
                commDest010_110, commSource010_110, commData010_110, commID010_110, commStart010_110,
                commDest111_110, commSource111_110, commData111_110, commID111_110, commStart111_110,
                commDest100_110, commSource100_110, commData100_110, commID100_110, commStart100_110);

        PE111: PE_BlockMatch port map(PCLK,
                commDest111_101, commSource111_101, commData111_101, commID111_101, commStart111_101,
                commDest111_011, commSource111_011, commData111_011, commID111_011, commStart111_011,
                commDest111_110, commSource111_110, commData111_110, commID111_110, commStart111_110,
                VAL111, VAL101, VAL011, VAL110,
                commDest101_111, commSource101_111, commData101_111, commID101_111, commStart101_111,
                commDest011_111, commSource011_111, commData011_111, commID011_111, commStart011_111,
                commDest110_111, commSource110_111, commData110_111, commID110_111, commStart110_111);

-- Locked signal protects registers that could be accidently changed
Locked <= '0' when (LmLckReg = LOCK_KEY)
    else
      '1';

-- switch register is read only
LmSwReg  <= SW;

-- -----------------------------------------------------------------------
-- Lock register is read/write
-- -----------------------------------------------------------------------
p_LdLckRegSeq : process(PCLK, nRESET)
begin
  if (nRESET = '0') then
    LmLckReg     <= (others => '0');
  elsif (PCLK'event and PCLK = '1') then
    if ((PSEL and PWRITE and PENABLE) = '1') then
      if (PA = LM_LOCK) then
        LmLckReg <= PWDATA(15 downto 0);
      end if;
    end if;
  end if;
end process p_LdLckRegSeq;

-- -----------------------------------------------------------------------
-- Oscillator1 register is read/write, protected by lock register
-- -----------------------------------------------------------------------
p_LdOscRegSeq1 : process(PCLK, nRESET)
begin
  if (nRESET = '0') then
    LmOscReg1     <= OSC1_VECTOR;
  elsif (PCLK'event and PCLK = '1') then
    if ((PSEL and PWRITE and PENABLE and not Locked) = '1') then
      if (PA = LM_OSC1) then
        LmOscReg1 <= PWDATA(18 downto 0);
      end if;
```

```
   end if;
  end if;
end process p_LdOscRegSeq1;

CTRLCLK1 <= LmOscReg1;

-- --------------------------------------------------------------------------
-- Oscillator2 register is read/write, protected by lock register
-- --------------------------------------------------------------------------
p_LdOscRegSeq2 : process(PCLK, nRESET)
begin
  if (nRESET = '0') then
    LmOscReg2     <= OSC2_VECTOR;
  elsif (PCLK'event and PCLK = '1') then
    if((PSEL and PWRITE and PENABLE and not Locked) = '1') then
      if (PA = LM_OSC2) then
        LmOscReg2 <= PWDATA(18 downto 0);
      end if;
    end if;
  end if;
end process p_LdOscRegSeq2;

CTRLCLK2 <= LmOscReg2;

-- --------------------------------------------------------------------------
-- LEDS register is read/write
-- --------------------------------------------------------------------------



p_LdLEDSRegSeq : process(PCLK, nRESET)
begin
  if (nRESET = '0') then
  -- put a pattern on them
    LmLedsReg(8 downto 0)  <= "101010101";
    signalFromLEDs <= '0';
  elsif (PCLK'event and PCLK = '1') then
    signalFromLEDs <= '0';
    if ((PSEL and PWRITE and PENABLE) = '1') then
      if (PA = LM_LEDS) then
                --we need to place this info in data from LEDS
       --LmLedsReg(8 downto 0) <= PWDATA(8 downto 0);
                LmLedsReg(16 downto 0) <= PWDATA(16 downto 0);

                dataFromLEDs(16) <= PWDATA(16);
                dataFromLEDs(15) <= PWDATA(15);
                dataFromLEDs(14) <= PWDATA(14);
                dataFromLEDs(13) <= PWDATA(13);
                dataFromLEDs(12) <= PWDATA(12);
                dataFromLEDs(11) <= PWDATA(11);
                dataFromLEDs(10) <= PWDATA(10);
                dataFromLEDs(9) <= PWDATA(9);
                dataFromLEDs(8) <= PWDATA(8);
                dataFromLEDs(7) <= PWDATA(7);
                dataFromLEDs(6) <= PWDATA(6);
                dataFromLEDs(5) <= PWDATA(5);
                dataFromLEDs(4) <= PWDATA(4);
                dataFromLEDs(3) <= PWDATA(3);
                dataFromLEDs(2) <= PWDATA(2);
                dataFromLEDs(1) <= PWDATA(1);
                dataFromLEDs(0) <= PWDATA(0);
                signalFromLEDs <= '1';
      end if;
    end if;
    if( signalToLEDs = '1') then
                LmLedsReg(16) <= dataToLEDs(16);
       LmLedsReg(15) <= dataToLEDs(15);
                LmLedsReg(14) <= dataToLEDs(14);
                LmLedsReg(13) <= dataToLEDs(13);
                LmLedsReg(12) <= dataToLEDs(12);
                LmLedsReg(11) <= dataToLEDs(11);
                LmLedsReg(10) <= dataToLEDs(10);
                LmLedsReg(9) <= dataToLEDs(9);
                LmLedsReg(8) <= dataToLEDs(8);
                LmLedsReg(7) <= dataToLEDs(7);
                LmLedsReg(6) <= dataToLEDs(6);
                LmLedsReg(5) <= dataToLEDs(5);
                LmLedsReg(4) <= dataToLEDs(4);
                LmLedsReg(3) <= dataToLEDs(3);
                LmLedsReg(2) <= dataToLEDs(2);
                LmLedsReg(1) <= dataToLEDs(1);
                LmLedsReg(0) <= dataToLEDs(0);
   end if;
  end if;
end process p_LdLEDSRegSeq;

LED <= LmLedsReg(8 downto 0);

-- ---------------------------------------------------------------------------
-- interrupt is latched on rising edge of nPBUTTutton input
-- INT register is read/write(to clear int)
-- ---------------------------------------------------------------------------
p_LdIntRegSeq : process(PCLK, nRESET, nPBUTT)
begin
  if(nRESET = '0') then
   LmIntReg     <= '0';
  elsif(nPBUTT = '0') then
   LmIntReg     <= '1';
  elsif(PCLK'event and PCLK = '1') then
   if((PSEL and PWRITE and PENABLE) = '1') then
    if(PA = LM_INT) then
     LmIntReg <= PWDATA(0);
    end if;
   end if;
  end if;
end process p_LdIntRegSeq;

REGSINT <= LmIntReg;

-- ---------------------------------------------------------------------------
-- Read registers
-- ---------------------------------------------------------------------------
p_GenNPRDATAComb : process (PA, LmOscReg1, LmOscReg2, LmLckReg, Locked,
                    LmLedsReg, LmIntReg, LmSwReg)
begin
  NextPRDATA <= (others => '0');
  case PA is
   when LM_OSC1 =>
    NextPRDATA(18 downto 0) <= LmOscReg1;
   when LM_OSC2 =>
    NextPRDATA(18 downto 0) <= LmOscReg2;
   when LM_LOCK =>
    NextPRDATA(15 downto 0) <= LmLckReg;
    NextPRDATA(16)        <= Locked;
   when LM_LEDS =>
    --NextPRDATA(8 downto 0)  <= LmLedsReg;
             NextPRDATA(16 downto 0)  <= LmLedsReg;
   when LM_INT =>
    NextPRDATA(0)         <= LmIntReg;
   when LM_SW =>
    NextPRDATA(7 downto 0)  <= LmSwReg;
   when others =>
    NextPRDATA(31 downto 0) <="00000000000000000000000000000000";
  end case;
end process p_GenNPRDATAComb;

-- ---------------------------------------------------------------------------
-- When the peripheral is not being accessed, '0's are driven
-- on the Read Databus (PRDATA) so as not to place any restrictions
-- on the method of external bus connection. The external data buses of the
-- peripherals on the APB may then be connected to the ASB-to-APB bridge using
-- Muxed or ORed bus connection method.
-- ---------------------------------------------------------------------------
p_RdSeq : process (PCLK, nRESET)
begin
 if (nRESET = '0') then
   PRDATA  <= (others => '0');
 elsif (PCLK'event and PCLK = '1') then
   PRDATA  <= NextPRDATA;
 end if;
end process p_RdSeq;

end synth;

-- --================================== End ==================================--


#include <stdio.h>
#include <time.h>

//as defined in irqint.s
extern void             uHALir_EnableInt(void);
extern void             uHALir_DisableInt(void);

extern void word_write(int addr, int data);
extern int  word_read(int addr);
extern void hword_write(int addr, int data);
extern int  hword_read(int addr);
extern void byte_write(int addr, int data);
extern int  byte_read(int addr);

/*****************DEFINES****************************/
#define FRAME_DIMENSION 16
#define SCANNING_RADIUS 16
#define IMAGE_ROWS 64
#define IMAGE_COLS 64
#define FRAMES_IN_ROW 4//must be IMAGE_ROWS / FRAME_DIMENSION
#define FRAMES_IN_COL 4//must be IMAGE_COLS / FRAME_DIMENSION

#define LM_LEDS   0xC000000C//0xC0000008 //temp set to lock register instead//  0xC000000C    --end with 8 = lock reg. end
with C = LED

/* Timer register informations*/
#define                 TIMER1_CTRL             (0x13000108)
#define                 TIMER1_VALUE            (0x13000104)
#define                 TIMER1_LOAD             (0x13000100)
#define                 TIMER1_CLR              (0x1300010C)
#define                 TIMER2_LOAD             (0x13000200)
/* Register set for IRQ controller ... pp 4-32 */
#define IRQ_STATUS                      (0x14000000)
#define IRQ_RAWSTAT             (0x14000004)
#define IRQ_ENABLESET           (0x14000008)
#define IRQ_ENABLECLR           (0x1400000C)
/* bit assignment for interrupts for interrupt controller 0 (first CPU) */
#define SOFTINT                         (0x1 << 0)
#define TIMERINT1               (0x1 << 6)
#define TIMERINT2               (0x1 << 7)
/* IRQ Vector address for Integrator/AP platform */
#define IRQ_VECT                (0x18)

#define AND_FILTER 0x0000FFFF//0x0000003F

unsigned char activeFrame[FRAME_DIMENSION][FRAME_DIMENSION];
unsigned char
oldFrameArea[FRAME_DIMENSION+2*SCANNING_RADIUS][FRAME_DIMENSION+2*SCANNING_RADIUS];
unsigned char wholeImage1[IMAGE_ROWS][IMAGE_COLS];
unsigned char wholeImage2[IMAGE_ROWS][IMAGE_COLS];

unsigned char bitmapHeaderInfo[14];
unsigned char infoHeaderInfo[40];
```

```
int counter;//used by timing functions

struct bitmapHeader{
   unsigned short int type;               /* Magic identifier           */

   unsigned int size;                     /* File size in bytes         */
   unsigned short int reserved1, reserved2;
//   unsigned short int padding;//I've been getting this padded so that offset is alligned causing problems!!

   unsigned int offset;                   /* Offset to image data, bytes */

};


struct infoHeader{
   unsigned int size;              /* Header size in bytes      */
   int width,height;               /* Width and height of image */
   unsigned short int planes;      /* Number of colour planes   */
   unsigned short int bits;        /* Bits per pixel            */
   unsigned int compression;       /* Compression type          */
   unsigned int imagesize;         /* Image size in bytes       */
   int xresolution,yresolution;    /* Pixels per meter          */
   unsigned int ncolours;          /* Number of colours         */
   unsigned int importantcolours;  /* Important colours         */
};

struct vector{
               int rowMovement;
               int columnMovement;
};

struct vector MasterOut[FRAMES_IN_ROW][FRAMES_IN_COL];


/*********FUNCTION PROTOTYPES********/
struct vector blockMatch(void);
int init(void);
int absolute(int in);
void counter_start(void);
double counter_stop(void);
void c_enableinterrupt(void);
void IRQenable_interrupts(void);
unsigned IRQ_install_handler(unsigned location, unsigned *vector);
__irq void IRQHandler(void);
void loadArrays(void);


float dumbCounter = 0;

int main(){
               int i, j, m, n, p, q;
               struct vector temp;
               double timer;
               int blockMatchCounter = -1;
               int upperVal;
               int lowerVal;
               int intTemp;
               unsigned char toCircuit[2];
               int wlm2First = 0;
               int wlm2Second = 0;
               int wlm1First = 0;
               int wlm1Second = 0;
               int errorDetectFlag = 1;

               //tempInt = word_read(LM_LEDS);
               //printf("tempInt: %d\r\n", tempInt);
               //return 0;
               printf("starting up\r\n");
               counter_start();
```

```
if(!init()){
        return 0;
}
//test test
//return 0;//did the LEDS go off?
//now load up one frame from each image and call blockmatch, then switch to the next frame..


for(i = 0; i < FRAMES_IN_ROW; i++){
        for(j = 0; j < FRAMES_IN_COL; j++){
                //now load the whole corresponding section of each image into the frame!
                for(m = 0; m < FRAME_DIMENSION; m++){
                        for(n = 0; n < FRAME_DIMENSION; n++){
                                activeFrame[m][n] =
                                wholeImage2[i*FRAME_DIMENSION+m][j*FRAME_DIMENSION+
                                        n];
                                //printf("activeFrame: %d\r\n", activeFrame[m][n]);
                                wIm2First = i*FRAME_DIMENSION+m;
                                wIm2Second = j*FRAME_DIMENSION+n;
                        }//end for n
                }//end for m
                //printf("wholeImage2[%d][%d]\r\n", wIm2First, wIm2Second);
                for(m = 0; m < FRAME_DIMENSION+2*SCANNING_RADIUS; m++){
                        for(n = 0; n < FRAME_DIMENSION+2*SCANNING_RADIUS; n++){
                                if( (i*FRAME_DIMENSION-SCANNING_RADIUS+m) < 0 ){
                                        oldFrameArea[m][n] = wholeImage1[0][0];//0;
                                        wIm1First = 0;
                                        wIm1Second = 0;
                                }
                                else if( (i*FRAME_DIMENSION-SCANNING_RADIUS+m) >
                                                IMAGE_ROWS ){
                                        oldFrameArea[m][n] = wholeImage1[0][0];//0;
                                        wIm1First = 0;
                                        wIm1Second = 0;
                                }
                                else if( (j*FRAME_DIMENSION-SCANNING_RADIUS+n) < 0 ){
                                        oldFrameArea[m][n] = wholeImage1[0][0];//0;
                                        wIm1First = 0;
                                        wIm1Second = 0;
                                }
                                else if( (j*FRAME_DIMENSION-SCANNING_RADIUS+n) >
                                        IMAGE_COLS ){
                                        oldFrameArea[m][n] = wholeImage1[0][0];//0;
                                        wIm1First = 0;
                                        wIm1Second = 0;
                                }
                                else{
                                        oldFrameArea[m][n] =
                                                wholeImage1[i*FRAME_DIMENSION-
                                                SCANNING_RADIUS+m][j*
                                                FRAME_DIMENSION-
                                                SCANNING_RADIUS+n];
                                        wIm1First = i*FRAME_DIMENSION-
                                                        SCANNING_RADIUS+m;
                                        wIm1Second = j*FRAME_DIMENSION-
                                                        SCANNING_RADIUS+n;
                                }
                        }//end for n
                }//end for m
                //printf("wholeImage1[%d][%d]\r\n", wIm1First, wIm1Second);
                blockMatchCounter++;
                printf("working on blockmatch task #%d\r\n", blockMatchCounter);



                //if( (blockMatchCounter < 13) || (blockMatchCounter == 14) || (blockMatchCounter == 15) ){
                if( (blockMatchCounter < 12) || (blockMatchCounter == 13) || (blockMatchCounter == 14) ){
                        //then do it in HW
                        //so just write all the data to HW
                        for(p = 0; p < (2*SCANNING_RADIUS+1); p++){
```

```
                    for(q = 0; q < (2*SCANNING_RADIUS+1); q++){
                        for(m = 0; m < FRAME_DIMENSION; m++){
                            for(n = 0; n < FRAME_DIMENSION; n++){
                                short * shortTempPtr;
                                int integerTemp;
                                toCircuit[0] = activeFrame[m][n];
                                toCircuit[1] =
                                    oldFrameArea[p+m][q+n];
                                shortTempPtr = (short
                                    *)(&toCircuit[0]);
                                integerTemp = *shortTempPtr;
                                word_write(LM_LEDS,
                                    integerTemp);
                                intTemp =
                                    integerTemp;
                            }//end for n
                        }//end for m
                    }//end for q
                }//end for p
//              printf("intTemp: %d\r\n", intTemp);

                //wait to be sure HW circuit has generated the results
                for(p = 0; p < 20; p++){
                    dumbCounter = dumbCounter * 7;
                    dumbCounter = dumbCounter / 5;
                }//end for p
                upperVal = word_read(LM_LEDS);
                lowerVal = upperVal;
                //printf("initial value: %d\r\n", upperVal);

                //remove unwanted upper bits
                lowerVal = lowerVal & 0x000000FF;
                upperVal = upperVal & 0x0000FF00;
                upperVal = upperVal >> 8;
                MasterOut[i][j].columnMovement = upperVal;
                MasterOut[i][j].rowMovement = lowerVal;


            }
            else{
                temp = blockMatch();
                MasterOut[i][j].columnMovement = temp.columnMovement;
                MasterOut[i][j].rowMovement = temp.rowMovement;
            }

            if(errorDetectFlag == 1){
                if(i == 1){
                    if(j == 0){
                        errorDetectFlag = 0;//don't do this again!
                        //send an error reconfigure packet to the circuit
                        //word_write(LM_LEDS, 0x1C001);//tell them to stop using
                        //BM 001
                        //now we must resend everything from the beginning!
                        //i = -1;
                        //j = -1;//so that they will be incremented to 0
                    }
                }

            }

        }//end for j
    }//end for i



    for(i = 0; i < FRAMES_IN_ROW; i++){
        for(j = 0; j < FRAMES_IN_ROW; j++){
            //first shift the values.....
            MasterOut[i][j].rowMovement = MasterOut[i][j].rowMovement - 16;
            MasterOut[i][j].columnMovement = MasterOut[i][j].columnMovement - 16;
            printf("BM task #%u - row shift:%d, column shift:%d\r\n", 4*i+j,
                                                MasterOut[i][j].rowMovement, MasterOut[i][j].columnMovement);
                        }//end for j
                }//end fori

                timer = counter_stop();
                printf("timer for everything is %f\n", timer);
                printf("finished\r\n");
                return 1;
}

int init(){
                //time to open the two big images
/*              FILE* first, *second;
                int i, j;
                int firstIndex, secondIndex;
                first = fopen("c:\\0.bmp", "r");
                second = fopen("c:\\1.bmp", "r");
                if( (first == NULL) || (second == NULL) ){
                                printf("read error on input files\r\n");
                                return 0;
                }
                fseek(first, 0L, SEEK_SET);
                fseek(second, 0L, SEEK_SET);
                for(i = 0; i < 14; i++){
                                fscanf(first, "%c", &(bitmapHeaderInfo[i]));
                }//end for i
                for(i = 0; i < 40; i++){
                                fscanf(first, "%c", &(infoHeaderInfo[i]));
                }//end for i
                //jump to the start of the image data
                fseek(second, 44L, SEEK_SET);
                fseek(first, 44L, SEEK_SET);
                firstIndex = 44L;
                secondIndex = 44L;

                //now load up the overall image
                for(i = 0; i < IMAGE_ROWS; i++){
                                for(j = 0; j < IMAGE_COLS; j++){
                                                //char charTemp;
                                                //char charTemp1, charTemp2;
                                                //unsigned char uCharTemp1, uCharTemp2;
                                                int outputTemp;
                                                int fails;

                                                //fscanf(first, "%c", &(charTemp1));//(wholeImage1[IMAGE_COLS*i + j]));
                                                //fscanf(second, "%c", &(charTemp2));(wholeImage2[IMAGE_COLS*i + j]));
                                                fscanf(first, "%c", (wholeImage1[IMAGE_COLS*i + j]));
                                                fscanf(second, "%c", (wholeImage2[IMAGE_COLS*i + j]));
                                                outputTemp = (int)(wholeImage1[IMAGE_COLS*i + j]);
                                                printf("wholeImage1[%d] = %d\r\n", IMAGE_COLS*i + j, outputTemp);
                                                outputTemp = (int)(wholeImage2[IMAGE_COLS*i + j]);
                                                printf("wholeImage2[%d] = %d\r\n", IMAGE_COLS*i + j, outputTemp);
//                                              charTemp1 = fgetc(first);
//                                              if(charTemp1 == EOF)
//                                                              printf("fgetc 1 returned null\r\n");
//                                              charTemp2 = fgetc(second);
//                                              if(charTemp2 == EOF)
//                                                              printf("fgetc 2 returned null\r\n");

                                                //charTemp = (unsigned char)(wholeImage1[IMAGE_COLS*i + j]);
//                                              uCharTemp1 = (unsigned char)(charTemp1);
//                                              uCharTemp2 = (unsigned char)(charTemp2);
//                                              outputTemp = (int)(charTemp1);
                                                //printf("wholeImage1: %d\r\n", outputTemp);//wholeImage1[IMAGE_COLS*i + j]);
                                                //charTemp = (unsigned char)(wholeImage2[IMAGE_COLS*i + j]);
//                                              outputTemp = (int)(charTemp2);
                                                //printf("wholeImage2: %d\r\n", outputTemp);//wholeImage1[IMAGE_COLS*i + j]).
//                                              wholeImage1[IMAGE_COLS*i + j] = (unsigned char)(uCharTemp1);
//                                              wholeImage2[IMAGE_COLS*i + j] = 0;//uCharTemp2.
```

```
                        firstIndex++;
                        secondIndex++;
                        //printf("that was for i: %d, j: %d\r\n", i, j);
                        fails =     fseek(second, firstIndex, SEEK_SET);
//                      printf("fails: %d\r\n", fails);
                        fseek(first, secondIndex, SEEK_SET);
//                      printf("fails: %d\r\n", fails);
                }//end for j
                //printf("one row of loading it up done\r\n");
        }//end for i


        //word_write(LM_LEDS,0x10F);

        //load up the arrays*/
        loadArrays();
        printf("init is done\r\n");
        return 1;
}
int absolute(int in){
        if(in < 0)
                return -in;
        else
                return in;
}

struct vector blockMatch(){
        int bestRowLocation, bestColumnLocation;
        int lowestError = 400000000;
        int currentError = 0;
//      int halfWay = SCANNING_RADIUS;//used to be + 1 but I think that is wrong
        struct vector out;
        int i, j, m, n;

        for(i = 0; i < (2*SCANNING_RADIUS+1); i++){
                for(j = 0; j < (2*SCANNING_RADIUS+1); j++){
                        for(m = 0; m < FRAME_DIMENSION; m++){
                                for(n = 0; n < FRAME_DIMENSION; n++){
                                        currentError += absolute((activeFrame[m][n]-
oldFrameArea[m+i][n+j]));
                                }//end for n
                        }//end for m
                        if(currentError < lowestError){
                                lowestError = currentError;
                                bestRowLocation = i;
                                bestColumnLocation = j;
                        }
                        currentError = 0;
                }//end for j
        }//end for i
        //set them to be -ive if below the half way mark.....
        out.rowMovement = bestRowLocation;// - halfWay;
        out.columnMovement = bestColumnLocation;// - halfWay;
        return out;
}

void counter_start()
{
        c_enableinterrupt();
        IRQenable_interrupts();
        counter = 0;
        *(int *) TIMER1_CTRL = 0x000000C8;
}



double counter_stop()
{
        double total_time;
        //printf("stop\n");
        *(int *) TIMER1_CTRL = 0x00000000;
        // The last number 0.01936 is the overhead for starting and finishing the function calls.
        total_time = (double) 1/(24000000/256) * (65535 - *(int *) TIMER1_VALUE + counter * 65535)- 0.019;
        //printf("%d\n", *(int *) TIMER1_VALUE);
        //printf("time %f\n", total_time);
        return total_time;
}
// This function will enable timer1.
void c_enableinterrupt(void)
{

        *(int *) TIMER1_VALUE = 65535;
        *(int *) TIMER1_LOAD = 65535;
}

void IRQenable_interrupts(void)
{
        unsigned original_vector = 0x0;

        original_vector = IRQ_install_handler((unsigned) IRQHandler, (unsigned *) IRQ_VECT);
        *(int *) (IRQ_ENABLESET) = 0x0;          //disable ALL interrupts
        *(int *) (IRQ_ENABLECLR) = 0xFFFFFFFF;          //clear ALL interrupts

        *(int *) (IRQ_ENABLESET) |= TIMERINT1;
        *(int *) (IRQ_ENABLECLR) &= ~TIMERINT1;
        *(int *) (IRQ_ENABLESET) |= TIMERINT2;
        *(int *) (IRQ_ENABLECLR) &= ~TIMERINT2;

        //printf("IRQ_ENABLESET = 0x%X\n", *(int *) (IRQ_ENABLESET) );
        //printf("IRQ_ENABLECLR = 0x%X\n", *(int *) (IRQ_ENABLECLR) );

        //printf("calling Enabling ints ...\n");
        uHALir_EnableInt();
        //printf("Finished Enabling ints ...\n");
        //printf("original_vector = 0x%X\n", original_vector);

}

unsigned IRQ_install_handler(unsigned location, unsigned *vector)
{
        unsigned vec, oldvec;

#ifdef DEBUG
        printf("location %%p: 0x%p\n", location);
        printf("location %%X: 0x%X\n", location);

        printf("vector %%p: 0x%p\n", vector);
        printf("vector %%X: 0x%X\n", vector);

        printf("location - vector %%X: 0x%X\n", location - (unsigned)vector);
        printf("location - vector - 0x8 %%X: 0x%X\n", location - (unsigned)vector - 0x8);
        printf("(location - vector - 0x8)>>2 %%X: 0x%X\n", (location - (unsigned)vector - 0x8)>>2);
        printf("((location - vector - 0x8)>>2) | 0xea000000) %%X: 0x%X\n", ((location - (unsigned)vector - 0x8)>>2) |
0xea000000);
#endif

        vec = ((location - (unsigned)vector -0x8) >>2);
        if(vec & 0xff000000)
        {
                printf("\nInstallation of handler failed !!\n");
                return(1);
        }
        vec = 0xea000000 | vec;
        oldvec = *vector;
        *vector = vec;
                                                        /* Install new vector */
        return(oldvec);

}
```

```
__irq void IRQHandler(void)
{
        //int n;
        unsigned int *base = (unsigned int *) IRQ_STATUS;        /* IRQ status register for processor 0 */

/* Determine interrupt type and call appropriate handler */

        if(*base & TIMERINT1)
        {
                *(int *) TIMER1_CTRL = 0x00000000;
                counter++;
                //for(n=0; n<10; n++)
                        //printf("%d\n", *(int *)TIMER1_VALUE);
                *(int *) TIMER1_CLR = 0;
                *(int *) TIMER1_CTRL = 0x000000C8;
        }

}
```