1-1-2008

Co-synthesis of multiple processor embedded systems for real time applications

Anika Awwal
Ryerson University

CO-SYNTHESIS OF MULTIPLE PROCESSOR EMBEDDED SYSTEMS FOR REAL TIME APPLICATIONS

Anika Awwal
BASc, University of Toronto, 2004

A thesis presented to Ryerson University
in partial fulfilment of the requirements
for the degree of

Masters of Applied Science

Graduate Program in Electrical and Computer Engineering
Toronto, Ontario, Canada, 2008

Declaration

I hereby declare that I am the sole author of this thesis. I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

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

Anika Awwal

Abstract

Co-synthesis of Multiple Processor Embedded Systems for Real Time Applications

Masters of Applied Science, 2008

Anika Awwal

Electrical and Computer Engineering

Ryerson University

This thesis presents the methods for automating the synthesis of multiprocessor real-time embedded systems. It describes an evolutionary technique of finding an affordable architecture for a multi-mode multi-task system while meeting the real-time constraints imposed by designers.

First the synthesis problem is introduced and previous co-synthesis approaches to handle this problem are discussed. Then the description of the proposed co-synthesis framework foe real time systems is presented. The co-synthesis framework consists of

four main steps, namely processing element allocation, process assignment, scheduling and evaluation. The method determines a set of feasible solutions with optimized partitioning and real-time schedules for processes and data communication. The framework is capable of producing acceptable solutions for critical systems with hard real-time deadlines by employing process level prioritization and by meeting the process level deadlines. Moreover, the proposed scheduling methodology achieves better PE utilization as compared to the conventional non-preemptive scheduling technique. The co-synthesis method is demonstrated by applying it to examples from the literature and to industrial benchmarks, such as auto industry, telecommunication, networking and office automation.

Acknowledgements

First, I would like to express my gratitude to my supervisor, Gul N. Khan, for his close supervision on all the work presented in this thesis. He is methodical, imaginative and an excellent research advisor. I would like to thank NSERC and Ryerson University for providing financial support. The equipments provided by the Canadian Micro-electronics Corporation were valuable and important for implementing the proposed algorithm. Finally, I would like to thank Robert P. Dick (Northwestern University) for discussions on E3S benchmarks implementation.

Table of Contents

List of Figures

Chapter 1

Introduction

In this chapter we give an overview of embedded systems and co-synthesis method. Here we state the motivation behind our proposed work. We also state our objective, and we present our contribution in the co-synthesis research field. Finally we discuss the thesis organization.

1.1 Overview

A large number of modern electronic devices around us are embedded systems. Average modern cars consist of 10-20 embedded computer systems and these systems are responsible for a major portion of the cost of a car. The microwave oven we use to warm our food, the cell phone we use to communicate, printers, almost every modern device around us are built around embedded systems. Recent implementations of

embedded systems consist of multiple general purpose processors (software processing elements) and dedicated hardware blocks (hardware processing elements), such as Application Specific Integrated Circuit (ASIC) and/or Field Programmable Gate Arrays (FPGAs) [14, 39]. Co-synthesis automates the process of design space exploration for multiple processing element embedded systems.

1.2 Motivation

Designing an embedded computer system, especially the ones with the real-time constraints, is a challenging job. Hard real-time systems are designed to minimize (if not completely eliminate all) possible errors and to produce results that are on time. Life-supporting medical equipments, nuclear power plants, flight control systems are examples of such embedded systems. Many of these systems have task level, as well as explicit process level deadlines. The designer can ensure real-time deadlines are met by using more expensive faster processors. However, they also have to design and develop affordable low cost systems.

To explore the embedded systems design space, the designer needs to design and prototype numerous different embedded systems. Manual design space exploration for embedded system is time consuming and expensive. Though embedded systems containing multiple processing elements are gaining popularity, design automation of

these systems poses a number of challenges. In this thesis, we propose a new design automation method that is capable of producing optimized partitioned solutions for large-scale multiprocessor multi-task embedded systems in finite time.

Initial design automation has been for the low-level stages of design processes and later design automation moved toward the automation of the high-level (architectural) design processes. The problem definition of high level designs is more ambiguous than that of low level ones. As the embedded system synthesis problems become increasingly well defined and solved, embedded system designers will have to spend less time to explore design space.

1.3 Objectives

We wanted to design a heuristic optimization algorithm for our co-synthesis method that can guarantee sub-optimal solutions. The objective of the co-synthesis method was to determine feasible solutions with optimized partitioning and real-time valid schedules for processes and communication events. These solutions should have low resource cost, and they should meet the real-time deadlines as well as schedulability conditions.

1.4 Contribution

The proposed method is fast and efficient to design large scale high performance embedded system . The presented framework is multiobjective: while meeting the real-time constraints it also tries to optimize the resource usage. It supports process level resource sharing, and therefore, it is more flexible than other co-synthesis methods that use task level resource sharing. We consider process level deadline assignment and prioritization method. This prioritization method helps the proposed method to meet the deadlines of critical hard real-time systems.

Unlike many other proposed co-synthesis methods our method does not ignore communication resource allocation and scheduling. It supports both processing elements and communication resource allocation and scheduling. It is also capable of producing efficient solutions for practical modern embedded systems. Moreover, the proposed scheduling methodology achieves better PE utilization as compared to the conventional non-preemptive scheduling technique.

1.5 Thesis Organization

The thesis is organized into five chapters. Chapter 2 provides definitions that are useful when discussing hardware-software co-synthesis, formalizes the basic hardware-

software co-synthesis problem, discusses various optimization methods for scheduling and co-synthesis, and finally surveys the work of other co-design and co-synthesis research groups. In chapter 3, we describe all phases of the proposed co-synthesis algorithm. These include algorithms for initialization of processing elements and communication resources allocation, scheduling of the processes on the allocated processing elements, and evolution method for a new generation of solutions. Chapter 4 presents experimental results. The proposed algorithm is tested with different random graphs from the literature and industrial benchmarks. Results for these systems are discussed here. Moreover these results are compared with previous work. We summarize our contributions, present conclusions, and state some directions for future work in chapter 5.

Chapter 2

Co-synthesis of Real-time Embedded Systems

This chapter provides definitions that will be useful when discussing the proposed co-synthesis method. Here we also discuss general optimization algorithms used in the area of scheduling and partitioning. And finally we provide a survey of previous work in the co-synthesis field.

Hardware-software co-synthesis automates the process of design space exploration for multiple processing element embedded systems. The problem of mapping functionalities into one of the available dedicated hardware processing elements (hardware modules) or implement them as processes on one of the available software processing elements (traditional CPUs) can be viewed as an optimization problem. The goal of this problem is to optimize the implementation cost and the performance while meeting the schedulability condition and real-time constraints imposed on the system. There

are four main steps that are carried out by a co-synthesis CAD tool, namely allocation, assignment, scheduling and evaluation.

- Allocation selects various types of processing elements (PEs) and communication links that can be used in the system.
- Assignment, determines the mapping of processes to PEs and communication events to links. Binding or mapping has been used as synonyms for assignment.
- Scheduling determines the order in which processes will be executed on the PEs they are assigned to.
- Evaluation determines the cost, performance, and amount of deadline violation of the solution.

To achieve optimal solutions it is important to have feedback from one step of co-synthesis to others, because each of the steps affects others.

2.1 Definitions

2.1.1 Constraint Specification: Task Set, Tasks and Processes

The proposed co-synthesis method is intended to design multi-task embedded systems. A multi-task embedded system supports applications with multiple independent exe-

cutable tasks. For our co-synthesis method the behavior, functional requirements and constraints of a system are represented by a task set. Each of the tasks of the task set consists of multiple function modules. Fixed Point Bit Allocation (FPBA) and Convolutional Encoder (CE) [43] are examples of function modules of a telecom application. Functional modules for a system are presented as processes in this thesis. A task can have multiple processes of the same functionality and different tasks can have processes of the same functionality. We assume the processes are coarse grained, which means a process may be complicated enough to require numerous microprocessor instructions.

A directed acyclic graph (DAG) represents a task of a system and the nodes of a DAG represent the processes of a task. A simple example of a system with two tasks and the processes is shown in Figure 2.1. The volume of data transferred between a source and a destination node is associated with the edge (arc) between the two nodes. There are two tasks for this system. First task has four processes, and second task has three processes. A dependant process may start executing after all of its data dependencies have been satisfied. Process P4 of first task in figure 2.1, may begin execution after both process P2 and process P3 complete execution and required data to process P4 (3kbits from process P2 and 1kbits from process P3) have been transferred.

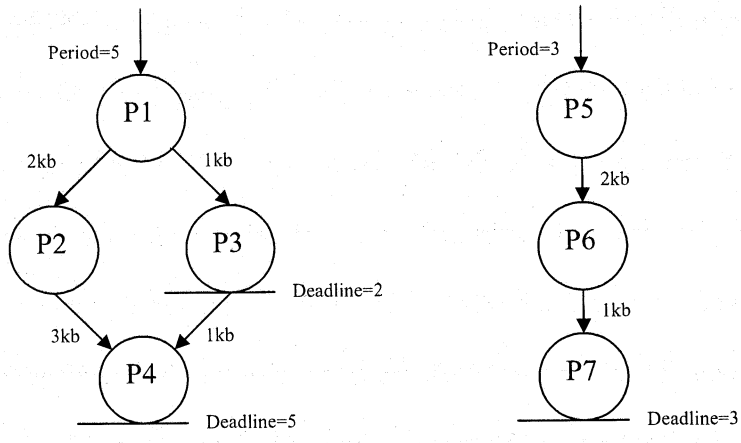


Figure 2.1: Tasks Set Representing a System Specification.

The *period* of a task is the time between one copy of task ready time and consecutive copy of the same task ready time. A system with multiple tasks may have different periods for different tasks. The *hyperperiod* of a task set is the least common multiple of the periods of the tasks. A node without any incoming edges is called a *source* node and a node with no outgoing edges is called a *sink* node. The deadline of a process (or task) is the time by when the process (or task) must complete its execution. A deadline can be either soft deadline or hard deadline. If for a synthesis solution, one or more multiple processes violate the hard deadline, then that solution will be an invalid solution. It is not desirable that a process misses its soft deadline, but meeting the soft deadline is not mandatory.

2.1.2 Processing Elements and Communication Resources

A processing element (PE) execute processes. A PE can be a general-purpose processor, an application specific integrated circuits (ASICs), a field programmable gate array (FPGA), or of any other processor type. A PE library is available as an input to our co-synthesis tool. A PE library for a system describes some characteristics of the PEs, such as price, power consumption etc. It also contains information about the relationships between processes and PEs, such as worst-case execution time for each task-PE pair.

If for a particular system architecture solution, dependent and parent processes are assigned to different PEs, the edge connecting these two processes must be assigned to a communication resource. When there is no communication resource connecting the PEs, then the architecture is invalid. The main attributes of a communication resource are communication controller price, transmission time per bit, the number of contacts, power consumption during operation etc. A communication resource can connect a number of PEs that are equal or less than its number of contacts. A point-to-point communication resource has two contacts and a bus can have more than two contacts. The worst-case communication time for an edge can be calculated from various parameters, such as the amount of data need to be transferred for the edge and transmission time per bit for the communication resource.

2.1.3 Utilization, Load and Laxity factor of a Processor

The *Utilization* factor of a processor is the fraction of time the processor is busy executing the application processes. Utilization factor, U can be expressed with equation 2.1. Load factor, CH of a processor can be calculated with equation (2.2).

$$U = \sum_{i=1}^{n} C_i/T_i \tag{2.1}$$

$$CH = \sum_{i=1}^{n} C_i/D_i \tag{2.2}$$

wehere,

n is total number of processes,

C_i is the execution time of the processor for a specific process i ,

T_i is the period of process i, and

D_i is the deadline of process i.

The laxity factor of a processor at any time t is the maximal time the processor may remain idle after t without causing a process to miss its deadline. The conditional laxity factor, $LC_i(t)$ can be presented by equation (2.3) where the sum in j presents the pending execution time of all the processes (including process i) that are ready to execute at t and have a higher priority than process i. The laxity factor $LP(t)$ is the

smallest value of all $LC_i(t)$.

$$LC_i(t) = D_i - \sum_j C_j(t) \tag{2.3}$$

2.1.4 Scheduling Algorithms Taxonomy

Off-Line scheduling algorithms are executed on the processors before the processes are activated. The schedule list is then used by the dispatcher. The run-time overhead of these algorithms is assumed to be low. However, these algorithms are not appropriate for real-time systems, as process activation for real-time systems is difficult to calculate off-line.

On-Line scheduling algorithms are executed online, during the run-time, and scheduling decision are taken every time a process is ready to execute or a process completes its execution. Each process is prioritized based on its temporal parameters. Processes can be assigned to a fixed priority or a dynamic priority. Fixed priority is assigned based on fixed parameters, before a process is activated. Dynamic priority is assigned based on the dynamic parameters that may change during system evolution. Among the basic on-line algorithms with static prioritization methods, *rate monotonic scheduling* (*RM*) and *deadline monotonic scheduling* (*DM*) algorithms are the most known ones. For *RM scheduling*, priority is assigned based on the period of a process. A process with the shortest period is assigned the highest pri-

ority. If the condition presented in equation (2.4) is satisfied for a PE, then n number of processes are schedulable on that PE using *RM scheduling*. This is a sufficient schedulability condition.

$$U = \sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1) \tag{2.4}$$

For *DM scheduling*, priority is assigned based on the deadline of a processes, the shorter the deadline of a process, the higher its priority is. If the condition presented in equation (2.5) is satisfied for a PE, then n number of processes are schedulable on that PE using *DM scheduling*. This is a sufficient schedulability condition.

$$U = \sum_{i=1}^{n} C_i/D_i \leq n(2^{1/n} - 1) \tag{2.5}$$

Earliest deadline first (EDF) algorithms and *least laxity first (LLF) algorithms* are the most important dynamic priority assignment algorithms. For *EDF algorithms*, processes are prioritized based on their absolute deadlines. A process with the earliest deadline will be prioritized the highest. *LLF algorithm* assigns the highest priority to the process with the smallest laxity (difference between deadline and execution time of a process). A necessary and sufficient schedulability condition for *EDF algorithm* and *LLF algorithm* is

$$U = \sum_{i=1}^{n} C_i/D_i \leq 1 \tag{2.6}$$

Dynamic prioritization methods can achieve higher PE utilization comparing to static prioritization methods, but they have a higher run-time overhead. Whereas static prioritization methods are more predictable and low priority processes have chances to miss their deadlines, dynamic prioritization methods are unpredictable, and a large number of processes may miss deadlines.

2.2 Optimization Techniques for Scheduling and Assignment Methods

Allocation/assignment and scheduling are known to be NP-complete problems [1, 38]. An algorithm that can find guaranteed-optimal solution to an NP-complete problem usually takes an amount of time exponentially dependent on the problem size. Therefore heuristic algorithms have been used in the literature to solve co-synthesis problems for large systems.

Optimization algorithms attempt to minimize a value, the optimized cost of a system. The optimization cost of a system depends on some parameters of a system that are called optimization parameters. The set of solutions around a solution is called the neighborhood of the solution, if they can be reached in one discrete step of the optimization algorithm. A local minima is a solution that has the lowest optimized cost in the neighborhood. A global minima is a solution that has the lowest optimized

cost in the whole problem space. In general optimization algorithms try to avoid getting stuck in the local minima, and attempt to find the global minima. Some of the heuristic algorithms are iterative improvement algorithms. For an iterative algorithm, greediness is the tendency or probability to choose a cost-decreasing change over a cost-increasing change. Greedy algorithms are one of the simplest and the most commonly used heuristic approaches. For greedy algorithms, at a given stage, only the best choice at the present stage is considered, without taking into account any previous and later decisions. Greedy algorithms have a tendency to get trapped in local minima or local maxima. In the following sections we discuss some popular optimization algorithms that can be used to solve scheduling and assignment/allocation problems.

2.2.1 Optimization Algorithms for Scheduling Methods

Scheduling is a very important phase of co-synthesis. Scheduling of a system determines the start time and completion time of the processes. As discussed in the previous section, the start time of a process must satisfy the dependencies of the DAG. This dependency constraint limits the amount of process execution parallelism. Scheduling affects the performance (completion time of a task) of a system for a co-synthesis solution, because it determines the order and concurrency of the process execution. There are scheduling methods for systems without resource constraints and there are

scheduling methods for systems with resource constraints. Our co-synthesis algorithm focuses on systems with resource constraints. If resource constraints are imposed, the number of processes that can execute in parallel depends on the number of available processors. The solution of scheduling problem under resource constraints provides the trade-off points of the design space.

To achieve exact solutions for resource constrained scheduling problems, Integer Linear Programming (ILP) method can be used. To formulate scheduling problems as linear problems, the objectives and constraints need to be defined as linear equations. The objective is to minimize the schedule length under the imposed resource constraints, and the following additional constraints :

- The start time of each process is unique
- The earliest possible start time for a process is when all its parent processes have finished execution and completed required data transfer
- The number of processes executing in parallel depends on the number of available processors
- Maximum schedule length has to be smaller than a specified upper bound.

The advantage of ILP formulation is that it provides an exact solution to scheduling problems. The disadvantage of this method is that ILP formulation is computationally

complex. For medium scale examples ILP schedulers are efficient, but for large scale examples, when the number of variables or constraints of the linear equations become more than several hundreds, ILP schedulers fail to solve the problems.

To simplify the scheduling problem many researchers have reduced the number of variables and made simplified assumptions of less numbers of constraints. It can be assumed that all the processes can be executed by the same type of processors and it takes one unit execution delay to execute all the processes. Under these assumptions, to find the minimum number of processors required to schedule processes with latency constraint, researchers have used Hu's algorithm [18]. If it is also assumed that the DAG as a tree (single paths from each vertex to sink), the problem can be solved in polynomial time. Hu's algorithm applies greedy strategy. At each scheduling step it schedules as many ready-to-execute processes as possible.

The above simplified assumptions are impractical for real multiprocessor embedded systems. Without making these assumptions, minimum-latency resource-constrained scheduling problems and minimum-resource latency-constrained problems are known to be intractable. Therefore, to solve the scheduling problems heuristic algorithms have gained popularity. Among heuristic algorithms, list scheduling has been widely used [27]. For this algorithm a priority list of the processes (based on some heuristic urgency measure) is used to select among the processes. The priority list can be mod-

ified to support the timing constraints, by reflecting the proximity of an unscheduled process to a deadline. Using this technique, it is possible to meet the timing constraints, but due to the heuristic nature of list scheduling, there is no guarantee in finding the optimal solution. The list scheduling method can be applied to find a scheduling solution to minimum resource with latency constrained problems. At the beginning the least numbers of processors are allocated and to meet the latency constraints. If required additional processors are allocated. The processes are prioritized based on a slack based prioritization method [9]. *Slack* for a process is calculated based on the difference between the latest possible start time of the process and the index of the schedule step under consideration. The low computational complexity of the list scheduling method made the algorithm very popular for large systems. For smaller system this method is capable of finding optimal solutions. Among other scheduling algorithms force-directed scheduling [30], trace scheduling [12] and percolation scheduling [32] methods are most well known.

Genetic algorithms, tabu search and dynamic programming algorithms can be used for scheduling methods. We discuss these optimization methods in the following sections, as these methods have been used for assignment/allocation.

2.2.2 Optimization Algorithms for Assignment/Allocation

Different optimal approaches and heuristic approaches have been used to solve assignment/allocation problems. The simplest approach is to try out all possible solutions and then select the best option. However, this approach is not feasible, because design space can be very large. Even for the lowest number of possible PEs (i.e. two PEs) of a system with N processes, there are 2^N number of possible solutions. Hence there is no guarantee to find the best optimal solution manually in a finite amount of time.

Among optimal approaches, integer linear programming (ILP) and dynamic programming have been used widely to solve the partitioning problem [33, 26]. Using ILP for assignment (partitioning) is similar to the use of ILP for scheduling (discussed earlier in this section).

Dynamic Programming approach has also been known as an efficient approach for hardware-software partitioning. In this method, the optimization problem is decomposed into a sequence of stages. The optimal solution to each stage must belong to the optimal solutions of the original problem. The success of this approach depends on how efficiently the problem is divided and complexity to reach an optimal solution at every stage of the problem.

The discussed optimal approaches are suitable for only small assignment problems. For large systems, these methods become computationally too expensive, because of

the exact nature of the solutions. Therefore, heuristic approaches, like simulated annealing, genetic algorithms, tabu search techniques have been widely researched to solve the process assignment problem for co-synthesis algorithms (see section 2.3). These heuristic approaches provide good-quality solutions, but they cannot guarantee the optimality of the solution.

Simulated annealing techniques are iterative improvement algorithms. Greediness in simulated annealing increases during the run of the algorithm [8]. During a simulated annealing algorithm run, probability P of a modified solution to get accepted can be represented with the following equation

$$P = 1/(1 + e^{(N-P)/T}) \tag{2.7}$$

where,

T is the global temperature parameter,

P is the cost of the old solution,

N is the cost of the modified solution.

The temperature parameter begins at a high value (infinity) and decreases as the system stabilizes. Beginning of a simulated annealing algorithm run, changes that increase the cost of a solution are selected with the same probability as changes that decrease the cost, and the algorithm can escape local minima. But this stage does not help to reach the goal of reducing optimized cost. Towards the end of the run

cost-decreasing changes have higher probability to be preferred comparing to the cost-increasing changes and thus the algorithm degrades to a greedy iterative algorithm.

Tabu search algorithms are another type of powerful heuristic algorithms that can be used for efficient hardware-software partitioning. Tabu search uses a local search method to escape a local minima. It maintains a tabu list that keeps track of all the recently visited solutions. The iterative improvement procedure prohibits the solutions on the tabu list to be revisited for the next iteration and thus tries to avoid getting trapped in one neighborhood of the solution space. The tabu condition might be over-ruled if it results in an overall low cost. After a specified number of iterations, the search is restarted from the initial system state to expand the search of the design space.

Genetic algorithms maintain a set of solutions. During the run of the algorithm, for each generation these solutions evolve in parallel over time. For each generation, solutions from the current generation are improved by randomized local changes performed by genetic operators and exchange of information between solutions. The lowest quality solutions of a generation are then removed from the solutions set by ranking and selection process [13]. In genetic algorithm, the term chromosome typically refers to a candidate solution to a problem. All changes to chromosomes are made with two operators that are mutation and crossover. *Mutation* operation ran-

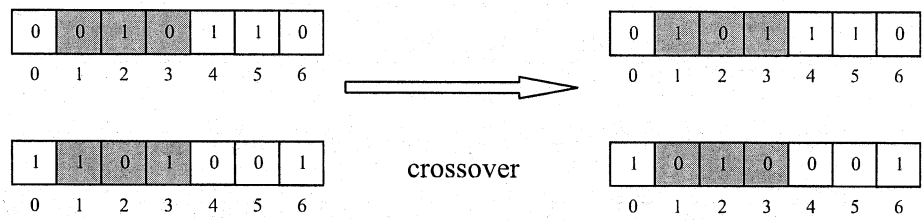


Figure 2.2: Crossover for GA.

domly picks a location on the solution array and changes the entry of the location with a new value. *crossover* randomly picks two solutions and two locations on these solutions, and swaps portions of solutions between these two location. Figure 2.2 shows an example of crossover. For two randomly chosen solutions presented in the figure, the randomly chosen positions are position 1 and 3. After crossover portions of solutions between position 1 and 3 are swapped and one new pair of solutions are evolved.

Genetic algorithms are capable of handling difficult problems composed of multiple NP-hard problems, even if each of these problems has huge solution spaces. But it is difficult to design and implement genetic algorithms. Genetic algorithms can be claimed to be better heuristic comparing to simulated annealing algorithms and tabu search algorithms based on the argument that only GA has the ability to share information between solutions.

2.3 Co-synthesis Approaches

In this section we discuss some of the past co-synthesis approaches. A lot of research has been done in the area of hardware-software co-design. Some related research on hardware-software co-synthesis with real time constraints have also been reported. Some recent work on multi-mode multi-task embedded systems has also been reported. Unfortunately, most of the techniques found in the literature suffer from some problems – either they are computationally intensive, or do not account for real-time constraints, process level deadlines, communication link scheduling, etc.

2.3.1 Pioneering Work in Co-synthesis

Prakash and Parker used Integer Linear Programming (ILP) for SOS to solve partitioning problems [33]. Input to the algorithm is in the form of data flow graph and the algorithm synthesizes a multiprocessor architecture. The constraints and objective for a heterogeneous multiprocessor system are presented with formal mathematical equations. These equations are then linearized and solved by applying simplex technique with Bozo program [15]. It is possible to find the best optimal solution for systems with a limited number of processes by using Prakash and Parker's method. However as pointed out earlier, the computation time to find the solution using ILP increases exponentially with the system size.

Gupta and Micheli presented one of the earlier approaches in the area of optimization for hardware-software co-synthesis [14]. The authors proposed the idea of optimized hardware-software partitioning to meet the constraints imposed by the designer. There are trade-offs between hardware and software implementations while capturing and making use of a partitions effect on system performance. They proposed that a partition cost function should be devised to capture these properties. Then this cost function can be used to direct the partitioning algorithm toward a desired solution, where the optimum solution is defined by the minimum value of the partition cost function. Unlike the previous research done in this area that has only focused on optimizing area and pinout of resulting circuits, the authors emphasized on how capturing the effect of timing behavior in the partition cost function is very crucial. The authors suggested that to find a solution that minimizes this cost function, a large number (exponential to the number of operations under partition) of solutions should be examined.

Ernst et al. developed a co-synthesis system COSYMA [11]. The system specification is represented in C^x, a superset of C language. A simplified assumption of only one software processor and one hardware block was made. Authors used simulated annealing for the partitioning process. The method starts with infeasible solutions that violate the timing constraints, and then improves the solutions by migrating function-

ality from software to hardware. A cost function was used to estimate the cost and also to control partitioning process. Hankel and Ernst proposed a co-synthesis method that dynamically determines the partitioning granularity to achieve better partitioning results [16]. It is assumed that software and hardware parts execute in mutual exclusion. This partitioning process was integrated in COSYMA.

Kalavade and Lee presented a constructive partitioning algorithm [19]. First it is decided whether hardware area is more critical for a system or the time is more critical for the system. This is selected by the threshold based comparison of a global time-criticality measure, called global criticality. In the next phase of the algorithm, named as local phase, each process is mapped to hardware or software based on different criteria of a process. This algorithm works on coarse granularity and partitions a system by traversing a list of processes.

Liu and Wong integrated partitioning with scheduling algorithm for their iterative improvement algorithm [25]. Authors made a simplified assumption of fixed number of PEs for a system, two software PEs and k number of hardware PEs. The algorithm starts with allocating all the processes to the software PEs. Later appropriate processes are migrated to hardware based on the feedback provided by the scheduler, to minimize the completion time and resource cost.

Mooney and Micheli presented a tool that performs real time analysis and priority

assignments [26]. To meet hard real time constraints, authors used dynamic programming algorithms to assign the static priorities to that processes. Proposed real-time scheduler is able to achieve tighter bounds thus squeezing more performance out of the same components as compared to a traditional RTOS. However, as mentioned earlier due to the exact nature of dynamic programming algorithms, they are only suitable for small problems.

Eles et al. presented an approach for system level specification and hardware-software partitioning with VHDL [10]. The authors formulated HW-SW partitioning as a graph partitioning problem and solved it by implementing two iterative improvement heuristics based on simulated annealing and Tabu search. They mainly focused on deriving a perfect cost function that should be optimized. The problem of using a linear weighted sum as the cost function is that the weighing array has to be appropriate for the problem instance, and also for the designer's desired solution [9]. However, since the co-synthesis problem is too complicated without first exploring all the possible solutions, it is hard to find an instance's best weighing array.

Shaha et al. used a genetic algorithm for hardware-software partitioning [36]. Some simplifications are made in their work. For instance, only one software processing element is allowed, and thus it cannot handle multiprocessor distributed systems. Moreover, communication link synthesis is not carried out. The fitness function

presented, only considers the violation of constraints, and does not try to find better solutions among the valid solutions.

Areto et al. provided a formal mathematical analysis of the complexity of the partitioning problems. It was proved the partitioning problems are $NP - hard$ in general case, and the authors presented some efficiently solvable special cases of partitioning problem [1]. Only one software processor was considered. The authors in this paper tried to introduce a simplified model for hardware-software partitioning to make algorithms scalable for systems with hundreds or even thousands of components. Prakash et al., Shaha et al. and Areto et al. did not consider real time constraints in their work [1, 33, 36].

2.3.2 Synthesis of Distributed Embedded Systems and Multiprocessor Systems

Recently, some researchers conducted research on distributed embedded systems. Axelsson compared three heuristic techniques for hardware software co-synthesis of real-time systems: a tabu search algorithm, a simulated annealing algorithm and a genetic algorithm [3]. Dave et al. developed a constructive co-synthesis algorithm (COHRA) to solve the co-synthesis problem for multi-rate hierarchical distributed embedded systems [6]. COHRA supports pipelining of the task graphs and employs the combination of preemptive and non-preemptive scheduling algorithm. Karkowoski and Corporall

presented a design space exploration method for homogeneous processing elements on a single chip [20]. This method employs functional pipelining. Oh and Ha designed an iterative algorithm for the co-synthesis problem targeting system-on-chip [28]. Hou and Wolf presented a process clustering method to achieve better co-synthesis results in lower time [17]. Li and Malik worked on analyzing the extreme (best and worst) case bounds on the running time of a program on a given processor [24].

Some researchers considered real-time constraints for allocation and scheduling problems in distributed multiprocessor systems [31, 34, 41]. Xu presented a scheduling algorithm that solves the problem of finding non-preemptive schedule for system with M identical processors [41]. The simplified assumption of all same type PEs is not realistic for modern multiprocessor systems. Peng et al. also presented a method that finds an optimal solution to the problem of allocating communicating periodic tasks to the heterogeneous processing nodes in a distributed real-time system [31]. The authors assumed that all functional modules (processes) from the same task were assigned to the same PE. Moreover, the presented method does not support systems with deadline greater than the period. Therefore, the method is not efficient for most of the real systems. The method presented by Xu and that presented by Peng. et al. uses a branch-and-bound search algorithm to find the optimal scheduling solutions for a system. The branch-and-bound search method is capable of finding the

exact optimal solution. This optimal approach is suitable for small systems with a limited number of tasks and processes. Due to the exact nature of the solutions, in the worst case this algorithm may take exponential amount of time to find the optimal solutions for large systems. Ramamritham presented a static allocation and scheduling algorithm for periodic tasks and distributed systems [34]. Author used latest start time/maximum immediate successors first (LST/MISF) heuristic to search a feasible allocation and schedule. The prioritization method for this algorithm assigns priority of a process based on the latest start time of a process and the number of successor processes. This priority assignment method is not efficient, because it only considers the number of successors and does not consider priority levels of the successors. For the methods discussed above authors considered constant number of PEs for the multiprocessor systems [31, 34, 41]. Therefore, their allocation and scheduling problem has only one objective that is meeting the time constraints and this makes the optimization problem much simpler compared to the multiobjective optimization problem for embedded systems. In the rest of this section, we present the previous research work that is closely related to the work proposed in this thesis.

2.3.3 Co-synhesis of Multiprocessor Real- time Embedded Systems

A number of researchers considered hardware-software co-synthesis with real-time constraints [7, 9, 23, 39]. Dave et al. presented a new technique for hardware-software co-synthesis algorithm COSYN, for periodic task graphs with real time constraints [7]. COSYN can produce a feasible distributed embedded architecture for real-time systems. It allows task graphs in which different tasks have different deadlines. Authors presented a co-synthesis method to optimize the resource cost and another co-synthesis method (COSYN-LP) to optimize the power. COSYN is efficient for systems where a large number of processes are executable on the same type of PE, and thus clustering the processes speeds up the process assignment [7]. Systems for which a large number of PEs is available and where tasks are executable on different types of PEs, clustering of tasks becomes less advantageous.

Wolf presented an iterative architectural co-synthesis algorithm for heterogeneous multiprocessor embedded systems [39]. The algorithm starts with assigning each process to one PE where the process can be executed efficiently. To minimize the resource cost, processes are re-assigned based on the PE utilization, i.e. by trying to remove less utilized PE from the architecture. In the next stage of the algorithm, process reassignment is performed to minimize communication cost between PEs. In the final stage of the algorithm, communication channels are allocated between PEs. Depending on

whether the communication allocation on an existing channel is possible, the channel is used or a new channel is added. This heuristic algorithm is fast comparing to other methods.

Among all the co-synthesis algorithms presented in literature, only MOGAC uses multiobjective optimization strategy [8, 9]. Dick and Jha presented a hardware-software co-synthesis method that partitions and schedules embedded specifications consisting of multiple periodic task graphs [9]. MOGAC synthesizes real-time heterogeneous distributed architectures, which meets the hard real time constraints. It used a multiobjective genetic algorithm (GA) to optimize the conflicting features of price and power consumption. The requirements of the embedded system are modeled as DAG. The authors used multiple general-purpose processors and multiple cores as the processing element (PE) for co-synthesis. MOGAC was designed such that it accepts a database, which specifies the performance and power consumption of each task on each available PE type. Each task graph edge is assigned to a communication link for which the power consumption and communication time is considered.

Xie and Wolf presented an allocation and scheduling algorithm for systems with control dependencies among the processes [40]. The proposed method handles conditional execution in multirate embedded systems. The authors proposed a new mutual exclusion technique for the conditional branches of the task graph to exploit the re-

source sharing. Author used a scheduling method that is similar to Sih and Lee's proposed compile-time scheduling heuristic [37]. Sih and Lee used dynamic level scheduling that accounts for interprocessor communication overhead when mapping processes onto heterogeneous processor architecture. The authors used a simplified non-preemptive scheduling technique in which lower priority processes might hamper a higher priority process execution.

Chakravarty et al. used a stochastic scheduling algorithm to schedule the processes for their co-synthesis method ESCORTS [4]. Using this scheduling method, the authors are able to assign non-preemptive stochastic start and completion time to the processes for a given allocation, with polynomial time complexity. A hierarchical genetic algorithm is used for resource optimization. The GA chromosomes are evaluated using a cost feasibility function. For this method chromosomes with a fitness value below a certain threshold are removed immediately, and these invalid solutions do not get a chance to evolve into better solutions.

Lee and Ha proposed a co-synthesis algorithm applicable for general multiprocessor systems with diverse operating policies [23]. The proposed algorithm separates the partitioning method from schedulability analysis and thus adaptable to various scheduling and operating policies. The proposed method takes into account the effect of partitioning results of higher-priority process when partitioning a lower-priority

process. This algorithm also adopts the schedule-based schedulability analysis of timed multiplexing model in the performance evaluation step. The authors proposed a schedulability analysis technique to schedule all tasks until their hyper-period. A preemptive scheduling algorithm is used. If any task cannot meet the real-time constraints, the framework reports an error message. Lee et al.'s algorithm supports only task level deadlines and does not support process level deadlines, which can be critical for hard real-time systems [35]. As explained earlier in section 2.1.1 every task of a system consists of one or multiple processes.

2.3.4 Co-synthesis of Multi-mode Embedded Systems

Only a few recent work were done in the area of multi-mode embedded systems. Oh and Ha proposed techniques for multi-mode, multitask embedded systems with real time constraints [29]. This iterative co-synthesis method has three main steps that solve the sub problems separately. In the first step of the algorithm a set of processing elements are allocated. The next step of the algorithm schedules the acyclic graph of each task to the selected processing elements to minimize the schedule length. In the final stage of the algorithm, evaluation is performed to check if the design constraints are satisfied and to compute the utilization factor. The problem with this algorithm is, each task is scheduled independently and to reduce the schedule length, all tasks use

the fastest candidate processor as much as possible. As a result the faster processors tend to be always over utilized and the slower ones always under utilized. Oh and Ha's method assumes that processes in different tasks can not be executed in parallel and consequently the schedule length becomes unnecessarily longer.

Kim and Kim's recent work proposes hardware-software partitioning technique for multi-mode embedded systems [22]. Unlike Oh and Ha, authors considered process-level resource sharing and parallelism rather than task level resource sharing. Processes are scheduled such that tasks can be executed in fully, partially parallel, or completely nonparallel based on the status of resource sharing between the processes in the tasks. Finally a global optimization method for PE allocation for all the tasks and all modes of a system is proposed. Scheduling method used for the proposed algorithm, schedules processes such that a process starts executing as soon as it is ready to execute. Consequently, a lower priority process may keep executing, while a higher priority process becomes ready, hampering the higher priority process execution. This simple scheduling method is not realistic for critical real-time systems.

Process assignment methods for the multi-mode, multi-task co-synthesis methods discussed above, are not efficient for systems with a large number of candidate PEs [22, 23, 29]. For each process assignment, every PE is considered and for large number of PEs the method of repeated re-assignment and re-scheduling can be laborious, and

sometimes not feasible. Some of the proposed co-synthesis algorithms discussed here do not provide allocation or scheduling methods for the communication links [22, 23, 29]. Communication link allocation and scheduling is critical for system optimization, and for accurate evaluation of target system architecture. Most of these co-synthesis methods use conventional task level prioritization method for scheduling, which is not efficient for satisfying real-time process level deadlines [22, 23, 29].

In this chapter we provided some definitions that will be useful when discussing the proposed co-synthesis method. Here we discussed general optimization algorithms used in the area of scheduling and partitioning. This chapter also provides a survey of previous work in the co-synthesis field.

Chapter 3

Co-synthesis of Heterogeneous Multi-task Embedded Systems with Real Time Constraints

In this chapter we describe our co-synthesis framework, and the software implementation associated with it for hardware software partitioning and process scheduling of a real-time system. First we present the main steps of the proposed method, and then we describe these steps in detail.

3.1 Introduction

The co-synthesis problem is formulated to optimize the system cost and performance, while satisfying the schedulability conditions. The proposed co-synthesis algorithm can produce valid system architectures meeting real time deadlines and area con-

straints. Optimized system partitioning (mapping/assignment) is achieved using a genetic algorithm (GA) that can avoid becoming trapped in the local minima. The proposed co-synthesis method employs a new static priority list scheduling technique to schedule the tasks on their assigned PEs and the data communication on the communication resources [2]. It can handle multi-rate systems containing tasks with hyperperiods that are large relative to their periods. A single co-synthesis run of this algorithm produces multiple designs that present the trade-offs points of a multiple PE architectural design space.

In this chapter we explain how we represent the solutions and system specifications. Here we provide detailed description of our co-synthesis steps, such as allocation, assignment, scheduling, initialization and evolution techniques.

3.1.1 System Architecture

The proposed method does not place any limit on the number of PEs (software or hardware) in the architecture. It supports the use of multiple types of communication resource (bus or point to point) in the target architecture. The following assumptions are made for the target multi-processor embedded architecture:

- The target system can consist of multiple software and hardware PEs that can work in parallel.

- A communication resource can support communication for multiple PEs based on the number of contacts available on the link.

- PEs can perform computation concurrently while transferring data to other PEs.

3.1.2 Optimization Algorithm Requirements

The goal of our co-synthesis algorithm is to produce a sub-optimal embedded real-time architecture by allocating PEs, finding process assignment to the allocated PEs, and by finding a scheduling order for the architecture. This architecture should have low cost and must meet the real time constraints. The algorithm requires the application specifications in the form of DAGs and a PE library describing types of PEs and communication resources (see sections 2.1.1 and 2.1.2). As mentioned in section 2.2, partitioning and scheduling problems are NP-complete, and in the worst case they may optimally require an amount of time exponential in the size of problem instance. Hence we were forced to resort to a heuristic optimization algorithm that can guarantee sub-optimal solutions. This algorithm should be able to avoid becoming trapped in the local minima, and it should assimilate problem-specific heuristic into the optimization framework easily.

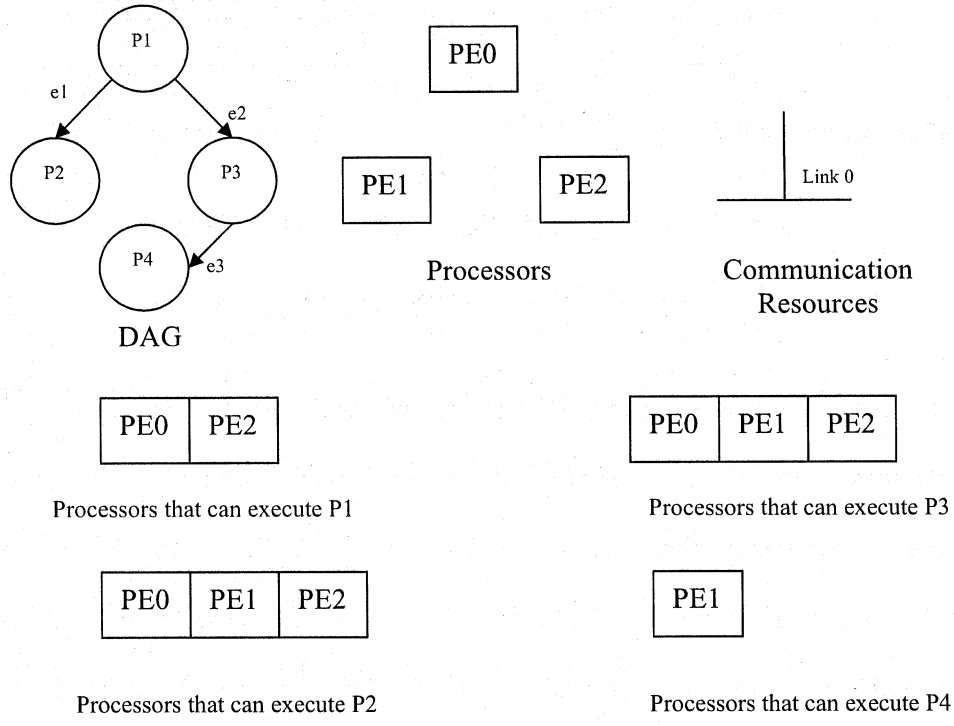


Figure 3.1: PE and Communication Resource Allocation.

3.2 System Specification and Solution Representation

The proposed co-synthesis algorithm accepts a PE library that specifies the execution time of each process on each available PE, a list of processes that are not executable on each PE and cost of each PE. The types of PEs available for a process are represented by an array of integers, and the types of communication resources are represented by another similar array of integers. For example for a system presented in Figure 3.1,

three PEs are available, and a communication resource with three contacts is available. There are two PEs, PE0 and PE2 that can execute process 1, three PEs are suitable for process 2 and 3, and process 4 is executable only on PE1.

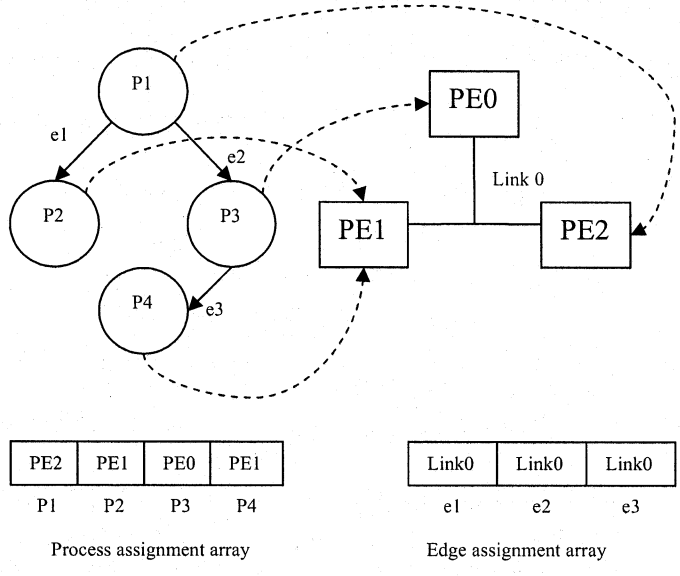


Figure 3.2: PE and Communication Resource Assignment.

Process assignment is presented with a one dimensional array. The offset in this array corresponds to the process. The integer at each offset represents the PE the process is assigned to. For example, for the DAG and PE allocation presented in Figure 3.1, one possible PE assignment and communication resource assignment solution is presented in Figure 3.2. Every process is assigned to one of the available PEs and Link 0 connects all the three PEs that need to be connected for data transfer.

For every communication resource an array of PEs specifies which PEs the communication resource is connected to. In Figure 3.3, Link 0 has two contacts and it connects PE0 and PE1, Link 1 has three contacts and it connects PE0, PE2 and PE3, Link 2 has two contacts that connect PE2 and PE3.

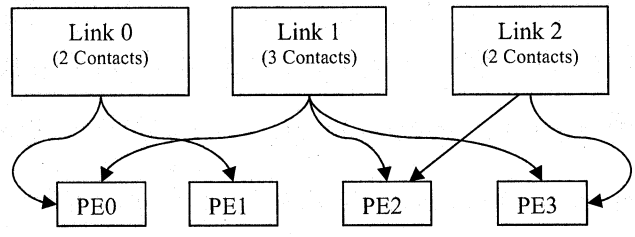


Figure 3.3: Communication Resource Connectivity Array.

For our software implementation of the co-synthesis algorithm, there are four main structures that model the system. The *Node* structure represents processes, the *Edge* structure represents the data communication between processes, the *PE* structure represents processing elements, and the *Link* structure represents communication resources. Some fields of these structures are specified by the user. The user provides the system specification, e.g. available PEs, DAG, and system constraints in the form of cost (hardware area), real-time process deadlines etc.

3.3 Optimization Algorithm for Proposed Co-synthesis Method

The main steps of our co-synthesis algorithm are given in Figure 3.4. The shaded rectangles and diamonds present the genetic algorithm steps. At the first step of the algorithm, the solution pool for the genetic algorithm is initialized with a randomized solution. After this step the algorithm enters the main loop of the algorithm and the next steps of the algorithm are repeated until the halting condition reaches.

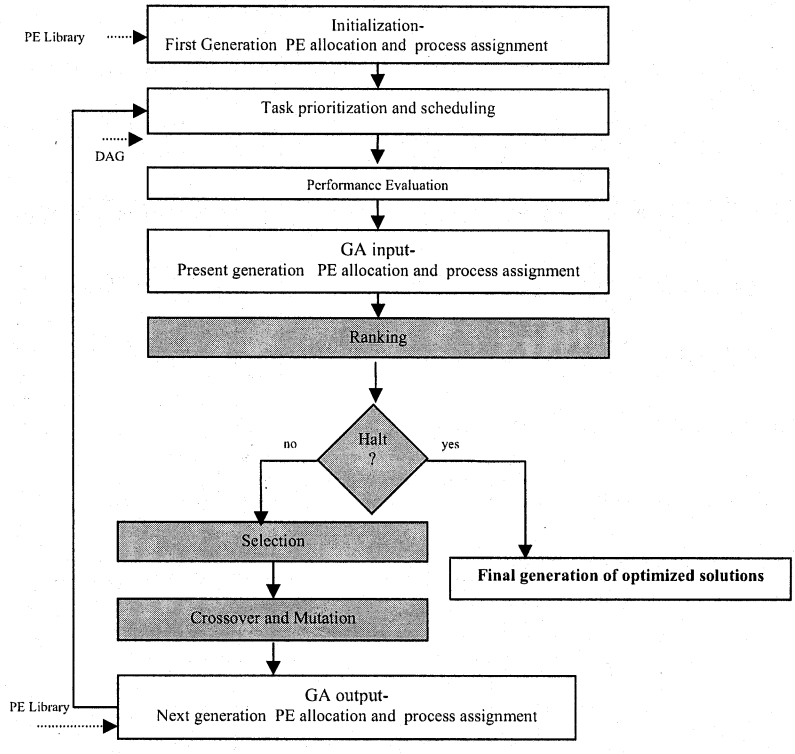


Figure 3.4: The Co-synthesis Framework.

The PE assignment solutions for the generations of the genetic algorithm are presented with a two dimensional array in which the first dimension corresponds to the solution index, and the second dimension corresponds to the process index. For our co-synthesis algorithm the GA chromosome is encoded as an array and the genes in terms of integers (see Figure 3.5). For a specific run of this system, process assignment arrays for the first, second, and nth solution of a generation are presented in Figure 3.5.

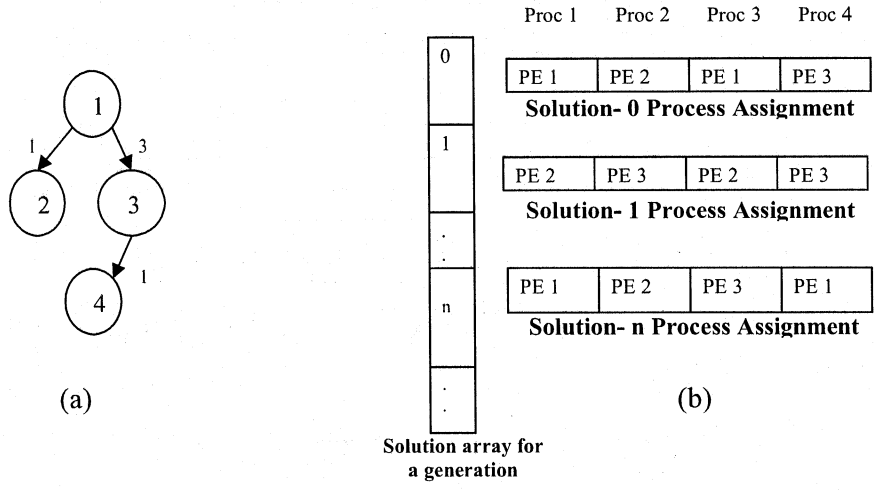


Figure 3.5: Solutions for A Generation.

3.4 Initialization of PE and Communication Resource Allocation

Initial solutions are generated by the initialization method of PE allocation and communication resource allocation. The initialization method is presented below. The first process of the system is assigned to a PE by a randomized selection mechanism. One of the available PEs, capable of executing the first process is chosen randomly. For other processes, first it is checked if one of the already allocated PEs is capable of executing the process. When none of the allocated PEs is suitable for executing the process, the process is assigned to a new PE by the randomized selection procedure. If already allocated multiple PEs are suitable for executing the process, then a randomized selection method is used to choose one of these PEs to execute the process. This initialization procedure, confirms the assignment of all the processes to valid PEs, for a minimum resource cost.

```
1 for each process P_i
2    for every allocated PE
3       if the PE is capable of executing P_i
4          The PE is entered in the suitablePElist
5       endif;
6    endFor;
7    if suitablePElist is not empty
8       Randomly choose one of the PE from suitablePElist
9    else
10       randomly choose one of the PE that is capable to execute P_i
11   endIf;
12 endFor;
```

Every communication event is assigned to a communication resource. A communication resource can support communication between a limited numbers of PEs, depending on the number of the contacts of the communication resource. Communication between any two PEs can use at most two contacts of a resource. The first communication event that need to be assigned to a resource is assigned by a randomized selection method. For other communication event assignments, first already allocated resources are considered. If one (two) of the contacts of one of the available allocated resources has (have) been used for one (or both) of the PEs under consideration, there is no need to allocate a new contact for the PE. Communication event assignment is performed in such a way that the least number of new communication resource contacts are added to the architecture. This assignment technique avoids the addition of extra communication resource contacts to system resource and minimizes the total resource cost. If none of the already used communication resources are economical for the communication event, a new resource is randomly selected to perform the communication event. Communication resource (CR) allocation algorithm is presented below.

1 for *each edge* e_i

```
2  while number of available allocated CR, that have not been checked yet > 0
3     randomly choose one of the allocated communication resource CR_i
4     if the source process of e_i is one of the contacts of CR_i
5        if the sink process of e_i is one of the contacts of CR_i
6           assign e_i to CR_i
7           break
8        else
9           use one of the available contact of CR_i for e_i sink process
10          assign e_i to CR_i
11       endif;
12    else
13       if the sink process of e_i is one of the contacts of CR_i
14          use one of the available contact of CR_i for e_i source process
15          assign e_i to CR_i
16       endif;
17    endif;
18    if e_i is assigned to CR_i
19       break;
20    else
21       check another randomly chosen already allocated CR
22 endWhile;
23 if e_i is not assigned to any one of the allocated CR
24    Randomly choose a CR for e_i from PE library
25 endIF
26 endFor;
```

3.5 Deadline Assignment

The hyperperiod of the system is calculated as the least common multiple (LCM) of the periods of the tasks. Each task has a *hyperperiod/period* number of copies for the duration of the hyperperiod. It is assumed that, for a particular system architecture solution, if all the copies of all the tasks of a system are schedulable within the hyperperiod, then the system is schedulable for the solution [23]. The proposed method supports a deadline longer than the period, which means multiple copies of the same task can be ready to execute simultaneously. Deadlines for the first copy of each task are specified by the user. The proposed method also supports process level deadlines. Deadlines for the copies of a process are calculated as given in equation (3.1):

$$Deadline\ of\ nth\ copy\ of\ a\ process = Process\_period * (n-1) + Deadline\ of\ the\ 1st\ copy \quad (3.1)$$

For example consider a system presented in Figure 3.6 , with a hyperperiod of 6. Task 0 has a period of 3 and there are two copies of task 0 within the hyperperiod. Task 1 has a period of 2 and there are three copies of task 1 within the hyperperiod. The first copy of the sink node of task 0 has a deadline of 4. The second copy of task 0 is ready to execute at time unit 3, while the first copy might be still executing. The second copy of task 0 has a deadline of 7 time unit. The first copy of the sink node of

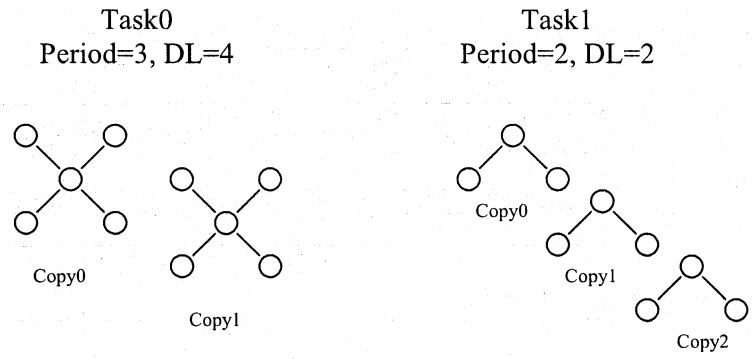


Figure 3.6: Deadlines and Periods of the Tasks.

task 1 has a deadline of 2. The second copy of task 1 is ready to execute at time unit 2, and it has a deadline of 4 time units. The third copy of task 1 is ready at time unit 4 and has a deadline of 6 time units.

3.6 Scheduling

For a co-synthesis algorithm run, scheduling is carried out during each solution evaluation. Our static scheduling method is non-preemptive and allows preemption only for the special cases where preemption time is smaller than the schedule length saved by preemption. This results in a lower context switch overhead delay as compared to traditional preemptive scheduling techniques. In our scheduling algorithm, processes of an input task is scheduled on their assigned PEs in the order of priority. The static prioritization method makes it possible that hard real-time constraints are met. Note

that, data transfer time is added to the parent process end time, to calculate the ready time of a process, when the parent and dependent processes are allocated to different PEs.

3.6.1 Process Prioritization

For each generation and each solution, all the copies of every process are relatively prioritized based on their deadline, execution time, and children priority after these processes are assigned to PEs. If a process does not have a deadline, for the priority assignment purposes, the shortest of the deadlines of the children processes is taken as the deadline. Unlike many other conventional prioritization methods [22, 23, 29], our technique assigns priorities at the process level, as compared to task level, even when the deadlines are not specified at the process level. It guarantees that the ancestors of the sink process complete execution in time and the sink process can meet its deadline. The priority assignment methodology used in our co-synthesis procedure is an improved version of the priority assignment method employed in COSYN [7].

We will use the task graph, PEs and communication resources presented in Figure 3.7 to explain the prioritization method. PE library for the application is presented in Table 3.1, and communication resource library is presented in Table 3.2.

Let us assume a solution is generated by the co-synthesis algorithm, for which pro-

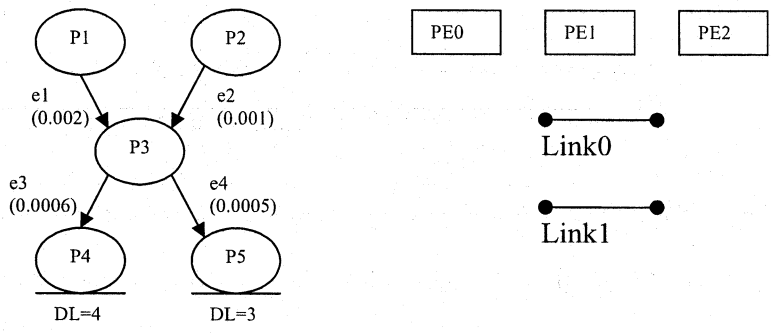


Figure 3.7: Task Graph, PEs, Communication Resources.

cess assignment and communication events (edges) assignment is presented in Figure 3.8. Below the process assignment array for each process, execution time on the assigned PE is presented in parentheses, and below the communication event assignment array for each edge, data transfer time on the assigned communication resource is presented in parentheses. Data transfer time for a communication event is calculated as given in equation (3.2).

$$Data\ Transfer\ Time\ of\ an\ edge = Communication\_resource\_rate * Amount\_of\_data; \quad (3.2)$$

According to equation (3.2) data transfer time between a source and destination process is calculated based on the data transfer rate (time/bit) of the communication resource the event is assigned to, and the amount of data (in bits) needs to be transferred between the processes.

Priority level assignment for the processes can be represented by the following equations (3.3) and (3.4):

Table 3.1: PE library.

process	PE0 (cost:40) Exec. Time	PE1 (cost:20) Exec. Time	PE2 (cost:35) Exec. Time
1	0.7	1.5	1
2	0.5	1.7	1.3
3	–	1	–
4	1	–	0.7
5	1.3	–	0.5

Table 3.2: CR library.

link	cost per contact	contacts	rate
0	6	2	1
1	10	2	0.8

$$Priority\ Level\ of\ a\ sink\ process = Proc\_time - Deadline; \tag{3.3}$$

$$Priority\ Level\ of\ a\ nonsink\ process = Proc\_time + Max\ of(child\_Priority\_level, -Deadline); \tag{3.4}$$

The priority of a process is assigned by employing priority level equations (3.3) and (3.4). For process and communication event assignment presented in Figure 3.8, prior-

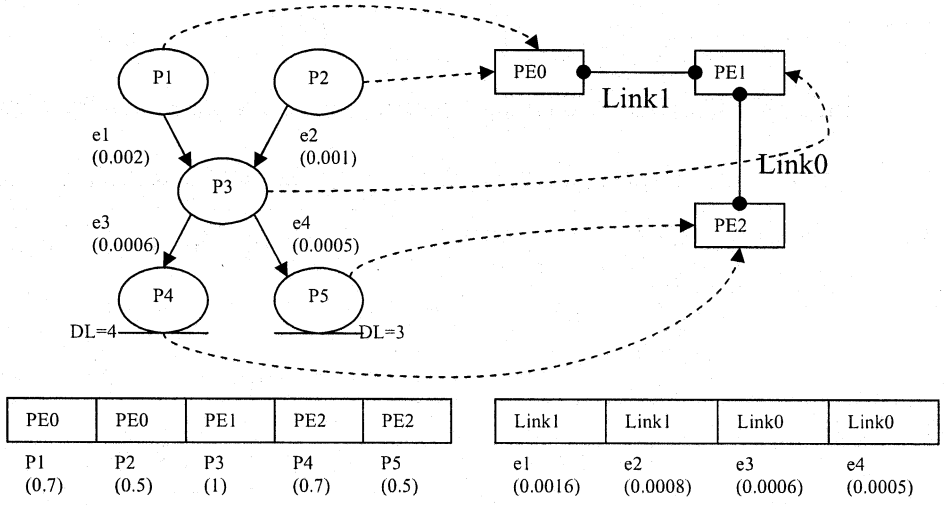


Figure 3.8: Process and Edge Assignment.

ity levels and priorities of all the process are presented in Table 3.3. *Proc_Time* is the execution time for a process on the PE it is assigned to, including the data communication time (maximum of the data transfer time for all incoming edges). For example, for calculating the *proc_time* for process P3, maximum data transfer time (0.00016) for edge e1 is added to the execution of P3. The *proc_time* of a process differs from solution to solution depending on which PE the process is assigned to, and on which communication resource the incoming edges are assigned to. Child process priority for a process is taken as the maximum priority of all the dependent processes. For example child priority for P3 in Table 3.3 is taken as the maximum of P4 priority (-3.2994) and P5 priority (-2.4995). DL in fourth column of Table 3.3 refers to the deadlines of

processes. Priority levels of the processes are calculated by employing equations (3.3) and (3.4). Here P1 has the highest priority, and P4 has the lowest priority.

Table 3.3: Priority assignment.

process (no.)	Proc. Time	child priority (no.)	DL	Priority Level	Priority
P1	0.7	-1.4979	3	-0.7979	5
P2	0.5	-1.4979	3	-0.9979	4
P3	1.0016	-2.495	3	-1.4979	3
P4	0.70006	–	4	-3.2994	1
P5	0.5005	–	3	-2.4995	2

This prioritization method helps to meet the real-time constraints, and achieves better results comparing to prioritization methods used in rate monotonic and deadline monotonic scheduling methods [2, 5]. In the deadline monotonic method, priority assignment is based on the process deadline alone. If there is a situation, where a process has two times longer deadline as compared to a 2nd process, but also has an execution time three times longer than that of the 2nd process, then the first process has a higher probability to miss the deadline. For deadline monotonic scheduling, the first process will have a lower priority than the second process. However, in our

prioritization method, the 2nd process will be prioritized higher. The priorities of the processes with children, depend on the children priority as well. The priority of a process differs from one solution to other, because the priority depends on the process's execution time. Data communication events on the communication links are prioritized as follows. Data communication for a higher priority destination process is assigned a higher priority as compared to the data communication for a lower priority destination process.

The schedule lists for the PEs and communication resources based on the priorities in Table 3.3 are demonstrated in Figure 3.9.

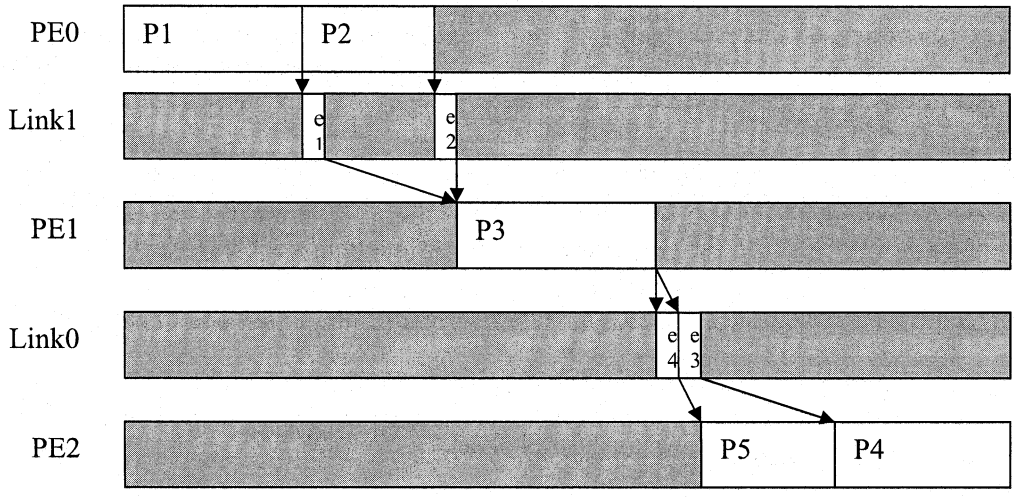


Figure 3.9: Example of PE and Communication Resource Schedule.

3.6.2 Scheduling of Communication Events

The scheduling algorithm of communication events is given in the following pseudo code (*HP* stands for higher priority, and CR_i stands for the assigned communication resource for the edge):

Communication Event Scheduling

```
1 for each process in the order of priority
2  for each incoming edge
3   EdgeReadyTime = SourceProcessEndTime;
4   PresentTime = EdgeReadyTime;
5   for each HP edge in the order of scheduling sequence of CR_i
6      if PresentTime < HP EdgeStartTime
7       if PresentTime + DataTransferTime <= HP EdgeStartTime;
8            The Edge is scheduled in this empty slot;
9            break;
10         else
11            PresentTime = HP EdgeEndTime;
12      if HP EdgeStartTime < PresentTime < HP EdgeEndTime
13         PresentTime = HP EdgeEndTime;
15   endFor;
16   EdgeStartTime = PresentTime;
17   ParentDataSentTime = EdgeStartTime + DataTransferTime;
18  endFor;
19 endFor;
```

Communication events for the incoming edges of the process are scheduled on the assigned communication resources. After the source process of an edge finishes exe-

cuting, communication event for the edge can take place on the assigned resource. To schedule a communication event on a communication resource, data transfer start and end times for all the already scheduled communication events on that communication resource are checked. If the communication event can start and finish data transfer before one of the scheduled communication is ready for transfer, then the communication is scheduled on that empty slot (lines 6,7 and 8). If there is no suitable empty slot among the higher priority edges, then the edge has to wait until all the higher priority edges finish data transfer on the communication resource. Destination process of an edge receives the required data from the source process at $ParentDataSentTime$.

3.6.3 Scheduling of Processes

Process scheduling algorithm for the PEs is presented below. HP stands for higher priority and PE_i stands for the PE on which the process is being scheduled. A process is ready to execute when required data from its parent processes have been received (line 2). Process execution can take place on the assigned PE, when the PE has an available time slot to execute the process. To schedule a process on a PE, execution start and end times for all other already scheduled processes on that PE are checked. If the process can complete execution before one of the scheduled processes is ready to execute, then the process is scheduled on that empty slot of the PE (lines 6 and 7). For

the case of preemptive scheduling option, a portion of a process execution is allowed in an empty slot, only if the preemption time is smaller than the empty slot (lines 12 and 13). The finish time of a process is the process start time added to the execution time for that process.

Process scheduling :

```
1 for each process in the order of priority
2  ProcessReadyTime = max of all ParentDataSentTime;
3  for each scheduled HP process in the order of scheduling sequence of the PE_i
4     EmptyTimeSlot = HP ProcessStartTime – ProcessReadyTime
5     if EmptyTimeSlot > 0
6        if ProcessExecutionTime <= EmptyTimeSlot;
7          The Process is scheduled in this empty slot
8          break; / * No need to check other HP processes * /
9        else
10          if Preemption is allowed
12            if ProcessPreemptionTime < EmptyTimeSlot
13               A portion of the Process is scheduled in the empty slot;
14               ProcessStartTime = ProcessReadyTime
15               ProcessExecituiomTime – = EmptyTimeSlot;
16               ProcessExecituiomTime + = ProcessPreemptionTime;
17          ProcessReadyTime = HP ProcessFinishTime;
18     else/ * no empty slot before the HP process * /
19        ProcessReadyTime = HP ProcessFinishTime;
20 endFor;
21 if the process was not preempted
22    ProcessStartTime = ProcessReadyTime;
23 ProcessFinishTime = ProcessReadyTime + ProcessExecutionTime;
24 endFor;
```

3.6.4 Verification of Schedule List by Utilization Factor Computation

In this section, we will demonstrate that the proposed scheduling methodology makes it possible to achieve better PE utilization as compared to the conventional non-preemptive scheduling technique. We assume that if all copies of all the tasks are schedulable within the hyperperiod, then the schedulability condition is satisfied [23]. Our method has a better chance of satisfying the schedulability condition. For conventional methods, process sequencing is based on priority, which means higher priority processes are always scheduled on a PE before the lower priority processes. As a result, many empty time slots on PEs are wasted, which could be used to execute lower priority processes, which are ready. Our scheduling algorithm allows a lower priority process execution before the higher priority processes, if some conditions (see previous section) are met. In this way, higher priority process execution is not hampered by executing a lower priority process.

We compare the utilization factor for both methods by following equations (3.5), (3.6) and (3.7). Assume that the processes under consideration are assigned to PE_a. The priority of the highest prioritized process is denoted by n and the priority of the process that is the last to be ready for executing within the hyperperiod (due to data

dependency) is denoted by $n - y$. Let us assume, there exist another process, which has priority $n - z$. Please note that y, z are integers where z is greater than y. The process with priority $n - z$, has a lower priority than the process with priority $n - y$. Let us also assume that the process with priority $n - z$ is ready to execute before some higher priority processes (which is often a common case). The process execution time for the processes with priority n is denoted by c_n and processes execution time for the processes with priority less than n, is denoted by c_{n-1}, c_{n-2}.. and so on. Let us also assume that, between process with priority n finish time and process with priority $n - 1$ ready time, there is an empty slot large enough to execute a process with priority $n - z$. Equation (3.5) presents utilization for the conventional method and equation (3.6) presents utilization for our scheduling technique.

$$PE_a\ utilization(conv.) = (c_n + c_{n-1} + c_{n-2} + .. + c_{n-y})/hyperperiod \quad (3.5)$$

$$PE_a\ utilization(prop.) = (c_n + c_{n-z} + c_{n-1} + c_{n-2} + .. + c_{n-y})/hyperperiod \quad (3.6)$$

$$PE_a\ utilization(prop.) - PE_a\ utilization(conv.\ method) = c_{n-z}/hyperperiod \quad (3.7)$$

From equation (3.7) it can be observed that the proposed method achieves better utilization as compared to conventional methods by $c_{n-z}/hyperperiod$. We use the E3S benchmark for telecom to compare PE utilization for our scheduling with the conven-

tional method. The DAG, PE library (Table 4.7) and communication resource library (Table 4.8) for this benchmark are presented in section 4.4.

Consider a solution with three PEs (PE1, PE5 and PE13) for a resource cost of 297 (including the communication resource cost). The hyperperiod for this system is 0.001 time units. Figure 3.10 presents the schedule for the processes of the system that are assigned to PE13, by using the conventional non-preemptive scheduling. One can observe from Figure 3.10 plot that there is a large empty slot on this PE between the processes with priority 35 and 34. All the lower prioritized processes , are waiting for process with priority 34 to finish executing first. As a result of this waiting of the lower priority processes, the schedule length for the processes is extended to 0.00152 time units. PE13 utilization for the hyperperiod in this case is 0.398. The total schedule length is greater than the hyperperiod indicating that the processes assigned to PE13 are not schedulable and this architecture is not a valid solution. If we explore the design space with all other options of process assignment on these three PEs, it can be observed that no valid solution can be produced for the low cost of 297 by using the conventional scheduling.

Figure 3.11 presents the schedule for the system processes that are assigned to PE13, based on our scheduling method. As we can observe from Figure 3.11 that by allowing lower priority processes execute before the higher priority process, the empty

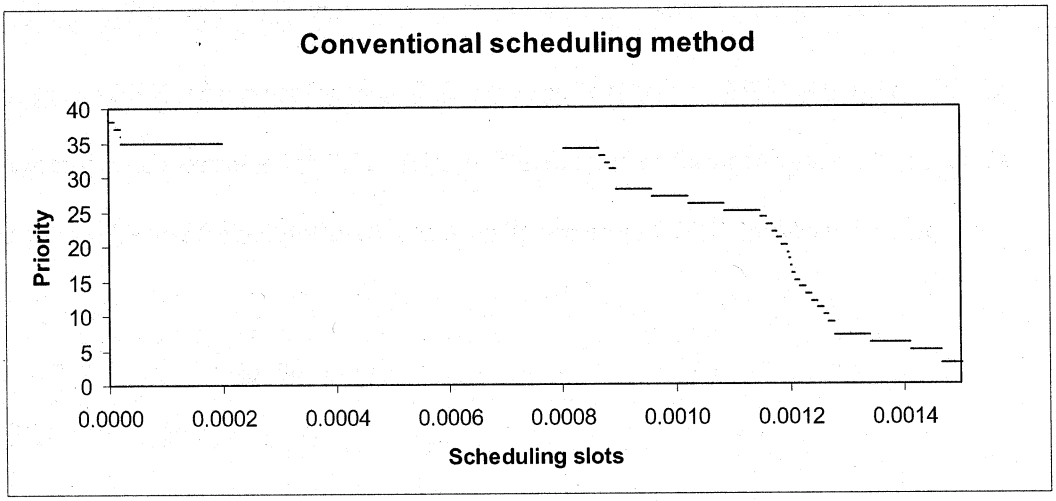


Figure 3.10: Schedule for Conventional Non-Preemptive Method.

slots for PE13 processor have been filled out. In this way PE13 utilization for the hyperperiod is increased to 0.926. Using this technique it has been possible to provide a valid schedule (schedule length less than hyperperiod 0.001) for lower resource cost.

3.7 Solution Evaluation

Solution evaluation is performed by calculating system's cost, total resource (hardware area) used for the system, process graph completion time, and the amount of deadline violation. The completion time of each process is recorded during scheduling. Schedule for every process spans the system's hyperperiod. Therefore, it can be verified

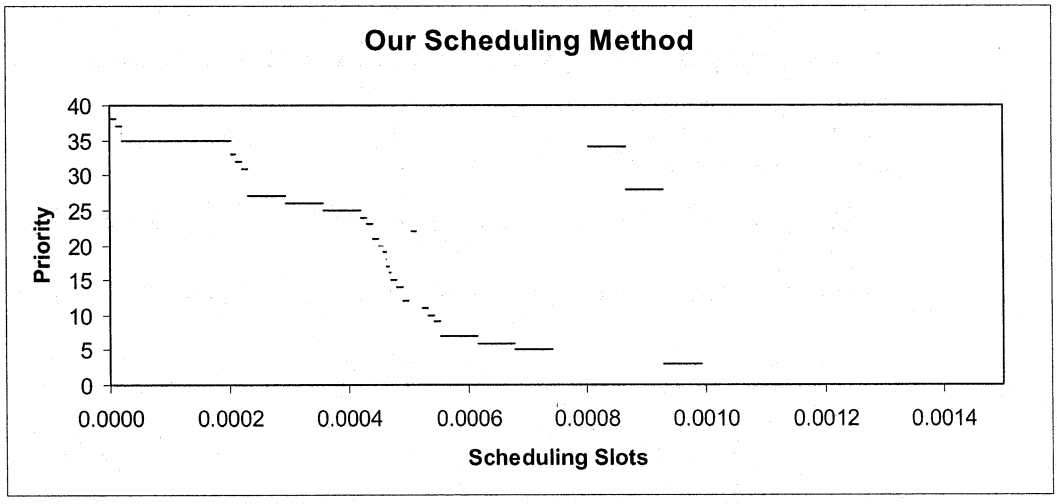


Figure 3.11: Schedule for Proposed Method.

whether a process has missed its deadline by inspecting its completion time. Every task of a system has one or more processes with specified deadlines. Hard real time constraint violation for a system is the sum of the deadline violations of all such process in all task copies in the system. Resource cost for a solution is determined by the sum of the cost of all the PEs and communication resources allocated for the solution. A solution is invalid if it follows any of the following conclusions:

- it violates the resource constraints
- it cannot meet the specified deadline
- it does not have the required communication links available for the specified

architecture

- it cannot satisfy schedulability conditions

These invalid solutions are not terminated immediately. It might be possible to repair the invalid solutions. However, it is also difficult to formulate a repair operation that will guarantee producing all valid repaired solutions [8]. Therefore the invalid solutions are treated the same way as the valid solutions during evolution. These solutions are given a chance to reproduce for the next generations and evolve into high-quality valid solutions by mutation and crossover.

Valid solutions may violate their soft constraints, although it is desirable to reduce a cost or schedule length until it is lower than its soft constraint (section 2.1.1). For a co-synthesis algorithm run, evaluation method is carried out for every solution of each generation.

3.8 Evolution of New Solutions by Genetic Algorithm

In this section we discuss the solutions selection and evolution techniques. The number of solutions for every generation of a co-synthesis algorithm is constant during a particular run of the algorithm. This number is chosen at the start of the run. For our co-synthesis algorithm this number varies depending on the system size and complexity.

During selection and reproduction for every solution created for the new generation, another solution is terminated.

3.8.1 Solutions Ranking

The invalid solutions are ranked based on the amount by which they violate the hard constraints (section 2.1.1). For valid solutions, ranking is performed based on the resource cost of the solutions. The lower the resource cost for a solution, the higher it is ranked. If two valid solutions have the same resource cost, ranking is performed based on the soft deadline violation proportion and schedule length, where solutions with better (smaller) deadline violation proportion and shorter schedule length are ranked higher.

3.8.2 Halt

After ranking, if the halting conditions have not yet reached, the selection process is performed on these solutions for next generation evolution. These steps are repeatedly performed until one of the halting conditions reaches. The halting condition occurs when a number of generations pass without any change in the solution pool. At the end of a run, the feasible solutions of the final generation that meet the imposed constraints, are presented to the user. These solutions present the design space with

trade-off points.

3.8.3 Solutions Selection and Reproduction

During the solution selection process, some solutions in the present generation are selected for reproduction. The higher the rank of a solution, the more likely it is to be selected for reproduction. Once a good solution is produced for a generation, the solution may be lost in the next generation by random selection, mutation or crossover. That is why if the best solution for a generation meets all the constraints, it is passed on to the next generation without any changes. In this way, the best ranked solution is preserved until a better solution is evolved in another generation and ranked as the best solution. Our solution selection algorithm is presented below.

Selection Algorithm

```
1 for Total_Number_of_solutions
2     Generate a random number between 0 and Total_rank;
    / * Total_rank = sum of ranks of all the processes * /
3     for Each solution s_i
4         NextRank = PreviousRank + The_solution_rank;
5         if PreviousRank <= random number < NextRank
6             The solution s_i is selected;
7             break;
8         else
9             PreviousRank = NextRank;
10    endFor;
11 endFor;
```

3.8.4 Crossover and Mutation

Mutation and crossover operations are performed on the selected PE assignment arrays. During the mutation operation , for a randomly selected process, the PE assignment is changed by another PE that is capable to execute the process. During crossover operation on a randomly selected pair of PE assignment arrays, two offsets on the arrays are chosen randomly. Integer presenting PE types for the offsets between these two randomly selected offsets are exchanged between the selected arrays.

For example, consider the assignment arrays shown in Figure 3.5. We assume assignment arrays for solution 0 and 1 are randomly selected for crossover. The randomly chosen offsets on these arrays are offset 2 and 3. Therefore PE assignments are exchanged for offset 2 and offset 3 between PE assignment array 0 and 1. For the new solution 1, process 2 will be assigned to PE2 and process 3 will be assigned to PE1.

To preserve the locality of the solutions towards the end of a run, the probability of crossover and mutation is lowered for the later generations of a run. After these crossover and mutation operations, a new generation of solutions consisting of new system partition is evolved. The solutions of the newly evolved generation are prioritized, scheduled, evaluated, and ranked as described in the earlier sections. The

algorithms for crossover and mutation are presented below.

Crossover Algorithm

```
1 for every generation g_i
2   for i = 0 to i = TotalNumberofSolutions/2
3     randomly choose one pair of solutions(PE assignment Arrays)
4     if g_i ≤ TotalNumberofGenerations/2
5       probability of crossover is 1/2
6     else
7       probability of crossover is 1/5
8     endIF;
9     if the pair is selected for crossover
10      Randomly choose two offsets(processes) of the solutions
11      swap the integers(PEs) at each offset between these offsets
12    endIf;
13  endFor;
14 endFor;
```

Mutation Algorithm

```
1 for every generation g_i
2   for every solution(PE Assignment Array)
2     for every offset(process) of the solution
4       if g_i ≤ TotalNumberofGenerations/2
5         probability of mutation is 1/2
6       else
7         probability of mutation is 1/5
8       endIF;
9       if the offset(process) is selected for mutation
10        Randomly choose a PE that can execute the process
11        change the integer(old PE) at the offset with the new PE
12      endIf;
13    endFor;
13  endFor;
14 endFor;
```

In this chapter we described our co-synthesis framework, and the software implementation associated with it. We described the allocation and assignment technique for the method. Here we presented the initialization, scheduling, prioritization and evolution algorithms for the co-synthesis method in detail.

Chapter 4

Experimental Results

We have implemented the co-synthesis algorithm using C language and results are obtained by executing on a shared 450 MHZ 4 X UltraSPARC-II CPU with 4 Gigabytes of memory under Linux environment. The method has been tested with different random graphs and benchmarks. In this chapter we present the results for MPEG encoder, SOS, Hou and wolf's graphs and E3s benchmarks. We compare our results with the results presented for various real-time co-synthesis techniques in the literature.

4.1 MPEG Encoder

For the experimental purpose, MPEG encoder application was fed as an input to the co-synthesis algorithm.

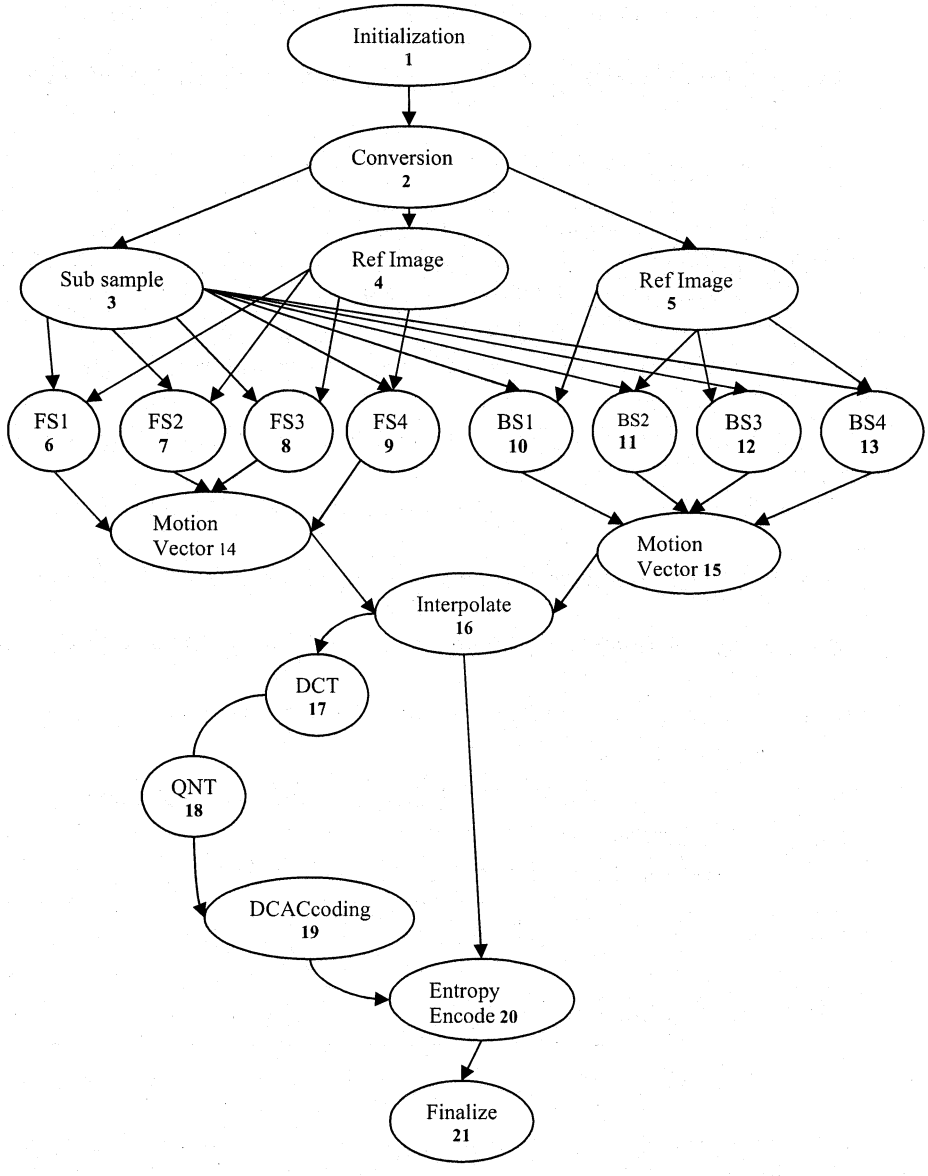


Figure 4.1: MPEG Encoder Task Graph.

The MPEG encoder DAG (Figure 4.1) with 21 coarse-grained processes and PE library used for the system are adapted from a recent co-synthesis research by Khan and Ahmed [21]. The comparison between the first generation of random solutions and the final generation of optimized solutions, for a co-synthesis run of the MPEG encoder system, are plotted in Figure 4.2. Here, the schedule length (performance) of the MPEG2 encoder system is in clock cycles. From these results, it can be observed that our co-synthesis algorithm is able to produce an optimized solution with 85% better performance (shorter task finish time) compared to the first generation random solution with the same area cost (8963 logic block units). For the same schedule length (30000000 time units), the method is able to produce an optimized architecture with 18% better area cost as shown in Table 4.1.

Table 4.1: MPEG2 first and final generation.

Generation	**Resource Cost (Schedule Length of 30000000)**
1	10591 unit
Final	8963 unit

The co-synthesis method running time for this 21 nodes graph was 4.08 seconds. If we compare this time with the results presented by Wolf for a 15 process system [39], we can observe that our synthesis method required almost half of the running

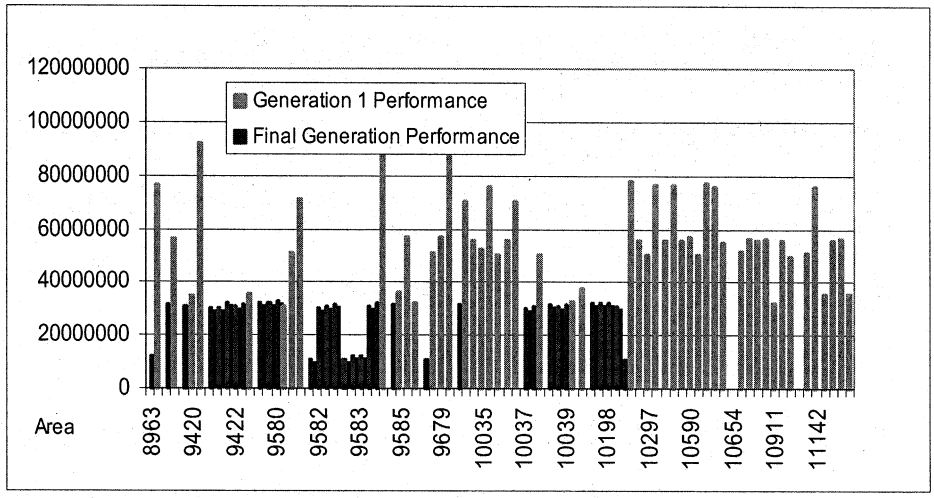
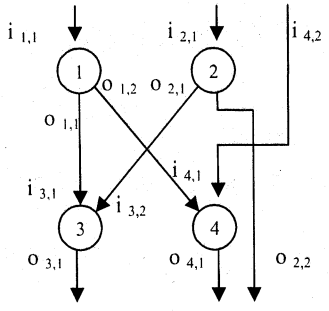


Figure 4.2: Initial and Final Generation Solutions for MPEG2 Encoder.

time for a more complex system consisting of 21 processes. While different computer systems have been used for Wolf's and our example, significant improvement in our algorithm execution time demonstrates that the proposed algorithm is efficient in terms of algorithm execution time.

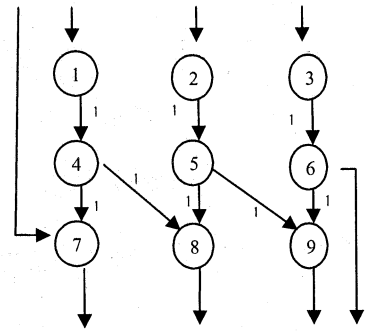
4.2 SOS: Synthesis of Application-specific Heterogeneous Multi-processor Systems

We used Prakash and Parker's example to verify the efficiency of our proposed co-synthesis algorithm [33]. Figure 4.3 demonstrates the DAGs for SOS example 1 and SOS example 2. Table 4.2 presents PE library for example 1 while Table 4.3 shows PE library for example 2. These libraries list the execution time for every process on the PEs. The value in parentheses associated with each PE is the cost of that PE.



$f_R(i_{1,1})=0.25$ $f_A(o_{1,1})=0.50$
$f_R(i_{2,1})=0.25$ $f_A(o_{1,2})=0.75$
$f_R(i_{3,1})=0.25$ $f_A(o_{2,1})=0.50$
$f_R(i_{3,2})=0.50$ $f_A(o_{2,2})=0.75$
$f_R(i_{4,1})=0.25$ $f_A(o_{3,1})=0.75$
$f_R(i_{4,2})=0.50$ $f_A(o_{4,1})=0.75$

SOS example 1



SOS example 2

Figure 4.3: SOS Example 1 and Example 2.

Table 4.2: PE library: SOS example 1.

Process	PE1(4)	PE2(5)	PE3(2)
1	1	3	-
2	1	1	3
3	-	2	1
4	3	1	-

Table 4.3: PE library: SOS example 2.

Process	PE1(4)	PE2(5)	PE3(2)
1	2	3	1
2	2	1	1
3	1	1	2
4	1	3	-
5	1	1	3
6	1	2	1
7	3	1	4
8	-	2	1
9	1	1	3

For these DAGs, the authors assumed that the process may begin execution before all of its input data has arrived [33]. Their model implies that part of each process is independent of the process's input data. As presented in Figure 4.3, each input has a parameter f_R associated with it, which is the fraction of the destination process (code) that can proceed without requiring that particular input. Each output has a parameter f_A associated with it. The output is available when f_A fraction of the source process is completed. For the second example Prakash and Parkar assumed the processes are fully dependent on the processes's input data. Unlike Dick et al.'s work, where this model is changed to a conventional one [9], our algorithm does not make any change to the original Prakash and Parker's graphs. Like Prakash and Parker, we support fraction input or output options for the processes.

Table 4.4 provides the comparison of our algorithm's performance with SOS [33] and MOGAC [9] when they are applied to Prakash and Parker's process graphs. The cost number shown by each process graph is the total PE and communication link cost. For instance, "P&P1.2⟨13⟩" refers to Prakash and Parker's first process graph, design 2, with a resource usage of cost 13. Our algorithm does not guarantee the best optimal solution, but only feasible solutions. However, it can be observed from Table 4.4 that like SOS, in each case the set of solutions also contains the optimal results. Moreover for two of the designs (namely 2.1 and 2.3), our algorithm achieved better

results compared to the other heuristic algorithms presented in literature [9]. Table 4.4 also lists the CPU running time of our algorithm for each case. If we compare the CPU running time of our algorithm, with the ones reported in the literature [7, 9, 33, 39], it is evident that our algorithm requires considerably less CPU running time.

Table 4.4: Results - Prakash&Parker's example.

Example⟨cost⟩	**No. of Proc.**	**Performance(time units)**			**CPU time(s)**
		SOS	**MOGAC**	**Proposed**	
P&P1.1⟨14⟩	4	2.5	2.5	2.5	0.048
P&P1.2⟨13⟩	4	3	3	3	0.054
P&P1.3⟨7⟩	4	4	4	4	0.054
P&P1.4⟨5⟩	4	7	7	7	0.035
P&P2.1⟨15⟩	9	5	**7**	**5**	1.164
P&P2.2⟨12⟩	9	6	6	6	1.179
P&P2.3⟨8⟩	9	7	**8**	**7**	0.139
P&P2.4⟨7⟩	9	8	8	8	0.144
P&P2.5⟨5⟩	9	15	15	15	0.144

4.3 Hou and Wolf's Graph

We have also tested our method using Hou and Wolf's examples [17]. Figure 4.4 presents the DAGs for the Hou and Wolf four tasks, and Table 4.5 presents the PE library for the tasks. The library demonstrates execution time for every process on the PEs. The value in parentheses associated with each PE is the cost of that PE. The cost for a point to point communication link for this system is 20.

Table 4.5: PE library: Hou and Wolf's example.

Process	PEX(100)	PEY(50)	PEZ(20)
a	5	12	18
b	10	18	40
c	5	12	18
d	35	85	95
e	15	22	80
f	30	75	180
g	15	25	85
h	15	35	47
i	7	10	30
j	10	28	35

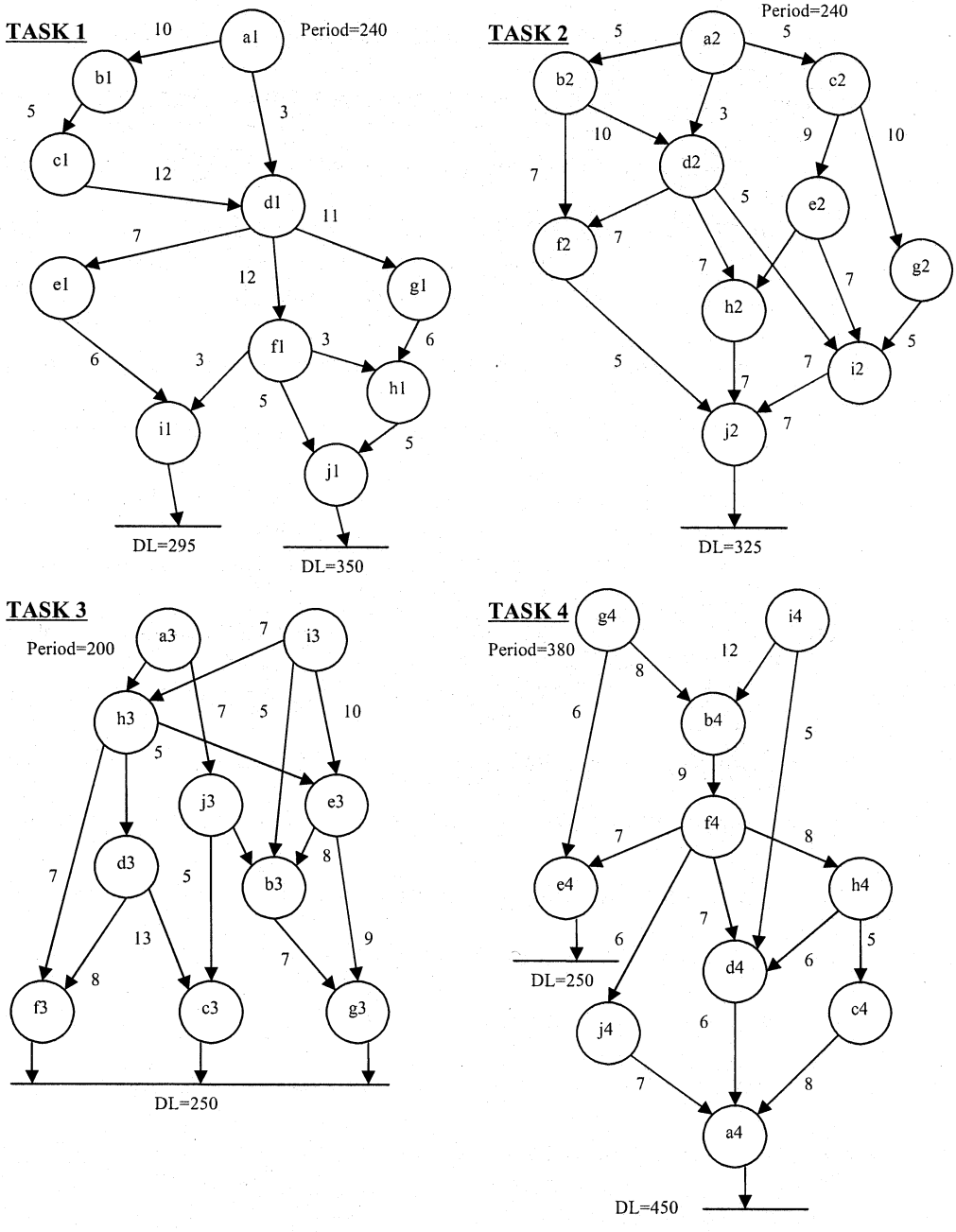


Figure 4.4: Hou and Wolf Task Graphs.

Table 4.6 compares the results produced by our co-synthesis algorithm for Hou and Wolf's examples [17] with those presented in the literature. The first column lists Hou and Wolf's examples. For example, H&W 1,2 refers to a multi-task system that consists of task 1 and task 2 from Figure 4.4. The second column shows the prices of the solutions produced by our algorithm when Hou and Wolf's original task deadlines are used. The third column shows the prices for system solutions produced by our algorithm when deadlines are assumed to be the same as the periods of the tasks. The fourth column provides the results produced by Yen's iterative improvement algorithm [42], fifth column for COSYN [7], sixth column for EMOGAC [8] and seventh for Lee et al.'s algorithm [23]. Lee et al. assumed the deadlines of the tasks are same as the periods of the tasks [23]. When the deadline is relaxed to be the same as the period of the task, our algorithm produces better solutions with prices lower than those produced by previous co-synthesis work. When the deadline is tightened, our algorithm has determined solutions with prices that are equal or lower than those presented in the literature.

Table 4.6: Results - Hou&Wolf's examples.

Example	Proposed (Hou's DL)	Proposed (DL=Period)	Yen	COSYN	EMOGAC (Hou's DL)	Lee (DL=Period)
H&W1,2	100	100	170	170	140	150
H&W1,3	170	140	170	170	170	170
H&W3,4	140	100	170	–	140	170

4.4 E3S Benchmarks

In this section we discuss the experimental results for E3S benchmarks of four industrial systems : telecommunication, networking, office automation and automotive [43]. Figure 4.5 demonstrates E3S benchmark for the telecommunication system. The main functional modules (processes) for this system are autocorrelation (ac), convolutional encoder (ce), fixed-point bit allocation (fpba), fixed point complex fast Fourier transformation (fft) and global system for mobile communications (gsm). The system consists of nine tasks, which share these functional modules. Table 4.7 presents PE library for the telecommunication system. It lists execution time for every process on the PEs. In the first row of the table the value in parentheses associated with each PE is the cost of that PE. For example PE1 has a cost of 77 and process ac has execution time

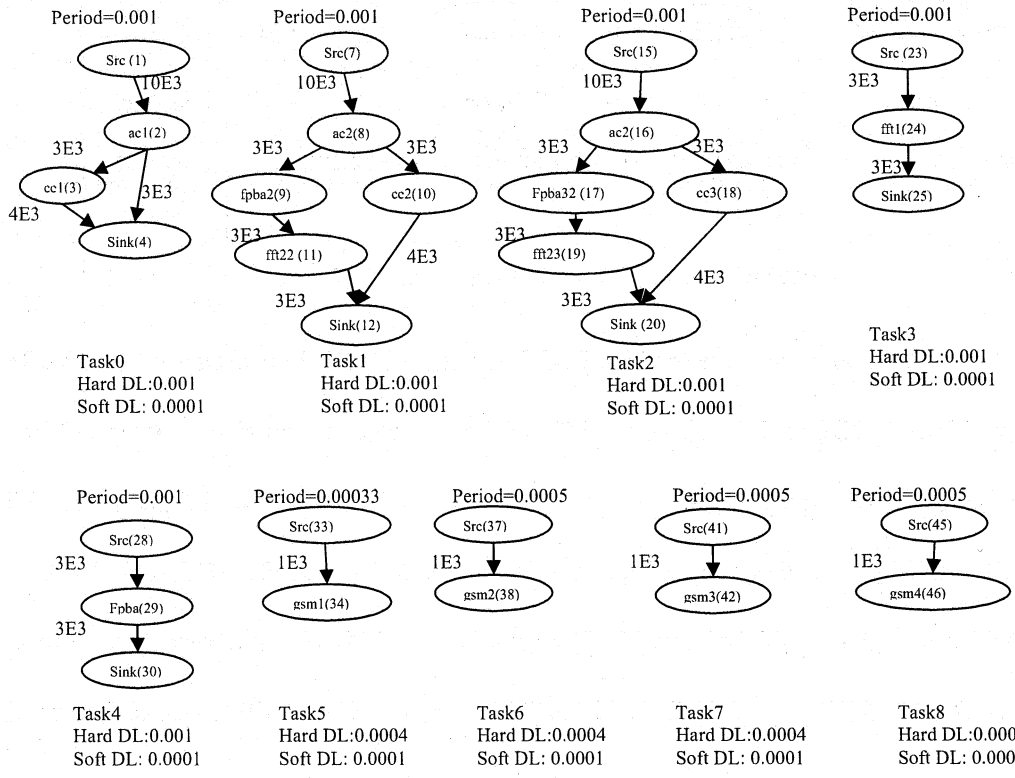


Figure 4.5: Telecommunication E3S Benchmark.

of 2.86e-6 time units on PE1. The process fpba 2,3 is not executable on PE7 or PE9, and the process fft is not executable on PE9. PE0 is an AMD ElanSC520-133MHz, PE1 is an AMD K6-2E 400 MHz/ACR, PE2 an AMD K6-2E+ 500Mhz/ACR, PE3 is an AMD K6-IIIE+ 550Mhz/ACR, PE4 is an Analog Devices 21065L-60 MHz, PE5 is an IBM PowerPC 405GP-266Mhz, PE6 is an IBM PowerPC 750CX-500MHz, PE7 is an IDR32334-100MHz, PE8 is an IDT79RC32364, PE9 is an IDT79RC32V334,

PE10 is an IDT79RC64575, PE11 is a NEC VR5432-167MHz, PE12 is a ST20C2 50 MHz, and PE13 is a TI TMS320C6203-30MHz. The communication resource library is demonstrated in Table 4.8.

Table 4.7: PE library for the Telecom system.

Process	PE0 (33)	PE1 (77)	PE2 (99)	PE3 (125.6)	PE4 (10)	PE5 (65)	PE6 (210)
src/sink	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
ac	3.1e-5	2.8e-6	2.2e-6	2e-6	7.7e-6	1.5e-6	1.2e-5
ce	0.00068	0.0001	8.3e-5	7.6e-5	0.0012	0.00012	0.0012
fpba 2,3	0.0061	0.00083	0.00066	0.0006	0.0063	0.00049	0.004
fft	0.00013	0.00014	0.00011	0.0001	0.0007	6.7e-5	0.0008
fpba	0.004	0.00056	0.00044	0.0004	0.0035	0.00029	0.0027
gsm	0.0031	0.00061	0.0005	0.00046	0.0036	0.00031	0.0049

Process	PE7 (16)	PE8 (12.5)	PE9 (24)	PE10 (52.1)	PE11 (30)	PE12 (15)	PE13 (111.2)
src/sink	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
ac	1.5e-5	8.3e-6	6.5e-6	2.7e-6	1.1e-5	1.1e-6	7.6e-7
ce	0.0015	0.00073	0.00058	0.00036	0.0011	0.00017	3.8e-6
fpba 2,3	–	0.0038	–	0.00014	0.0043	0.0015	0.00018
fft	150e-6	0.00018	–	0.00039	0.0027	0.001	0.00011
fpba	0.004	0.00056	0.00044	0.0004	0.0035	0.00029	0.0027
gsm	0.0027	0.002	0.001	0.0012	0.003	0.00082	6.3e-5

Table 4.8: Communication resource library.

Communication Resource	Contact Price	rate(time per bit)	No. of contacts
VME	180	2.27e-9	4
USB 2.0	14.19	2.08e-3	4
USB	6.38	83.3e-3	4
PCI-32-33	10.56	947e-12	4
CAN	6.05	1e-6	4
Fire wire	14.81	2.5e-9	4

Table 4.9 and Figure 4.6 demonstrate solution quality improvement by genetic algorithm evolution for a particular co-synthesis algorithm run. For this specific run of the co-synthesis algorithm there are 100 generations (iterations) and every generation has a pool of solution, which consists of 1000 solutions. Here 722nd solution is ranked

Table 4.9: Solution quality improvement by GA evolution.

Rank	1st Generation			999th Generation		
	Solution	**Area**	**Soft DL viol**	**Solution**	**Area**	**Soft DL viol**
10	722	297	0.0692	976	297	0.0272
100	369	297	0.0365	356	297	0.0170
200	406	297	0.0297	381	297	0.0115
300	530	284	0.0326	803	284	0.0163
400	813	297	0.0261	814	297	0.0177
500	903	297	0.0252	972	297	0.0093
600	892	297	0.0194	832	297	0.0119
700	102	297	0.0131	190	297	0.0108
800	356	297	0.0158	795	297	0.0072
900	643	297	0.0175	181	297	0.0064
970	211	297	0.0058	536	297	0.0057
990	63	284	0.0055	510	284	0.0049
1000	290	284	0.0052	0	284	0.0029

as 10th for the first generation, this solution requires resource cost of 297 , and it violates the soft deadlines by 0.0692. The amount of soft deadline violation is calculated as the sum of the times by which every copy of each task misses its soft deadline. For the final (999th) generation, the solution (solution 976) that is ranked 10th requires the same area cost, but it has a smaller deadline violation of 0.0272 time units. The best ranked solution for generation 1 requires resource cost of 297, and violates the soft deadlines by 0.0052 time units. By GA evolution technique, quality of the solutions improves a lot. It can be observed that for the same resource cost as generation 1, best solution for the final generation misses the soft deadlines only by 0.0029 time units. The solution quality improvement by GA evolution for this co-synthesis run is plotted in Figure 4.6. We can observe from the periods of tasks presented in Figure 4.5 that each of the task 0, task 1, task 2, task 3 and task 4 have one copy, task 5 has three copies and each of the task 6, task 7 and task 8 have two copies within the hyperperiod of 0.001 time units. Priority list for the best ranked solution of a co-synthesis algorithm run, for the telecom system is presented in Table 4.10. The order of the processes in this list is decided based on the priority levels of the processes. The calculation of priority level is already explained in section 3.6.1. For this particular solution process 6, 1st copy has the highest priority, process 7, 1st copy has the second highest so on and so forth .

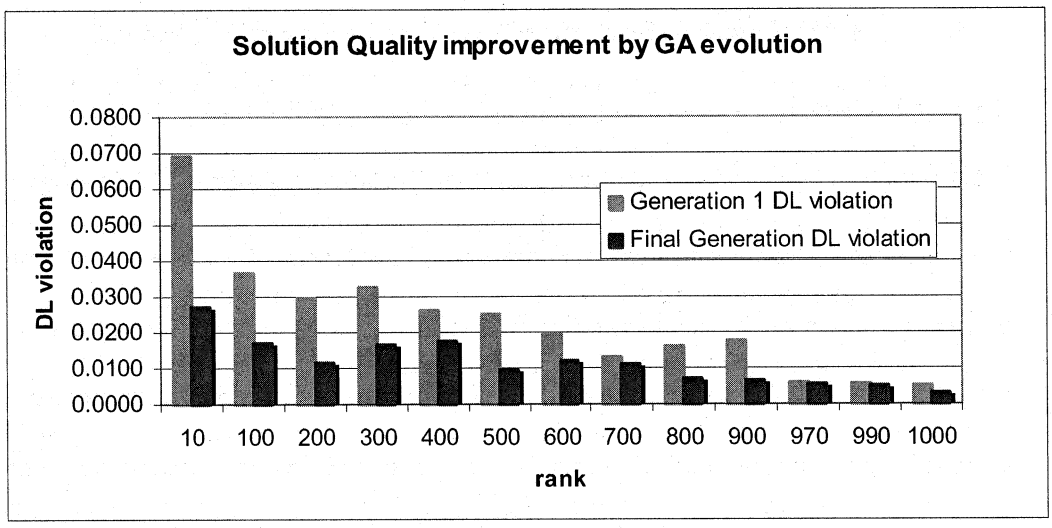


Figure 4.6: Generation 1 and Final Generation Soft Deadline Violation.

Based on this priority list, processes are scheduled on three allocated PEs, PE1, PE5 and PE13. The schedule lists for these PEs are provided in Figure 4.7. The processes are ready to execute only after required data from all the parent processes have been received. Therefore, earliest possible start time of a process depends on the completion time of the parent processes. It also depends on the availability of the communication resource. For example in Figure 4.5, the parent process of process 3, 1st copy (process 2, 1st copy) completes execution by 1.15e-5 time units, but the assigned communication resource to transfer data from the parent process is not available until 4.5 e-5 time units. The required data is transferred between 4.5e-5 time units and 5.25e-5 time units, and finally process 3, 1st copy is ready to execute at 5.25e-5

Table 4.10: Priority list.

6,0	7,0	8,0	9,0	27,0	28,0	29,0	14,0
15,0	16,0	17,0	32,0	33,0	34,0	0,0	1,0
2,0	3,0	32,1	33,1	35,0	40,0	41,0	44,0
45,0	36,0	37,0	34,1	42,0	46,0	38,0	11,0
19,0	22,0	23,0	10,0	24,0	18,0	4,0	12,0
20,0	25,0	30,0	32,2	33,2	35,1	36,1	37,1
39,0	40,1	41,1	43,0	44,1	45,1	47,0	5,0
13,0	21,0	26,0	31,0	42,1	46,1	38,1	39,1
43,1	47,1	34,2	35,2				

time units. A ready process can start execution on the PE when the PE is available to execute that process. A ready process may need to wait while other higher priority processes are being executed on the PE. As discussed earlier in section 3, a lower priority process may execute before a higher priority process if the higher priority process is not ready to execute. For example, we can observe process 41, 1st copy and process 45, 1st copy are ready to execute at 0 time unit, because these processes are not dependent on any other processes. These two processes are assigned on the same PE (PE1), higher priority process 3, 1st copy is assigned to. However, these processes are ready to execute before process 3, 1st copy, and they are allowed to be executed before the higher priority processes.

PE1

7,0	0->1e-5
28,0	1e-5->2e-5
33,0	2e-5->3e-5
41,0	3e-5->4e-5
45,0	4e-5->5e-5
3,0	5.25e-5->1.525e-4
23,0	1.525e-4->1.625e-4
24,0	1.625e-4->1.765e-4
25,0	1.765e-4->1.865e-4
19,0	1.987e-4->2.122e-4
33,1	3.333e-4->3.433e-4
20,0	3.558e-4->3.658e-4
41,1	5e-4->5.1e-4
45,1	5.1e-4->5.2e-4
11,0	5.34e-4->5.48e-4
12,0	5.48e-4->5.558e-4
33,2	6.666e-4->6.766e-4

PE5

1,0	0->1e-5
2,0	1e-5->1.15e-5
37,0	1.15e-5->2.15e-5
8,0	3.5e-5->3.65e-5
9,0	3.65e-5->0.0005265
29,0	0.0005265->0.0008165
30,0	0.0008165->0.0008265
37,1	0.0008265->0.0008365

PE13

15,0	0->1e-5
16,0	1e-5->1.077e-5
17,0	1.077e-5->0.00019077
34,0	0.00019077->0.000253
38,0	0.000253->0.00031677
10,0	0.0003167->0.00032057
34,1	0.0003458->0.00040883
42,0	0.0004088->0.00047183
46,0	0.0004718->0.00053483
11,0	0.0005348-0.00054383
42,1	0.0005348->0.0006068
46,1	0.0006068->0.0006698
34,2	0.0006792->0.0007422
38,1	0.000839->0.000902

Figure 4.7: Example of Schedule List.

The E3S benchmark for networking is demonstrated in Figure 4.8 and the PE library for this system is presented in Table 4.11. The E3S benchmark for office-automation is demonstrated in Figure 4.9 and the PE library for this system is presented in Table 4.12. The E3S benchmark for auto-industry is demonstrated in Figure 4.10 and the PE library for this system is presented in Table 4.13.

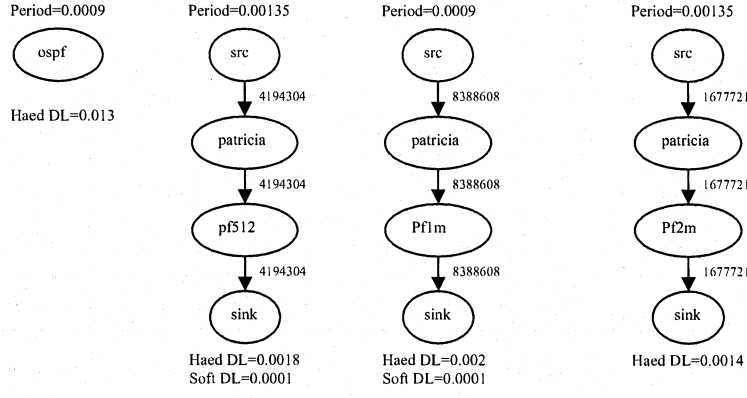


Figure 4.8: Networking E3S Benchmark.

Table 4.11: PE library for the Networking system.

Process	PE0 (33)	PE1 (88)	PE2 (77)	PE3 (99)	PE4 (125.6)	PE5 (65)	PE6 (16)	PE7 (250)
src/sink	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
ospf	0.0012	0.00014	0.00014	0.00011	9.8e-5	0.0018	0.0016	0.0013
patricia	0.0032	0.00059	0.00064	0.0005	0.00046	0.0045	0.0044	0.0036
pf512	0.00097	0.00044	0.00024	0.00012	0.00011	0.0012	0.0013	0.00065
pf1m	0.002	0.00086	0.00058	0.00024	0.00022	0.0023	0.0034-6	0.0012
pf2m	0.004	0.0018	0.0012	0.00069	0.00044	0.0046	0.0048	0.0025

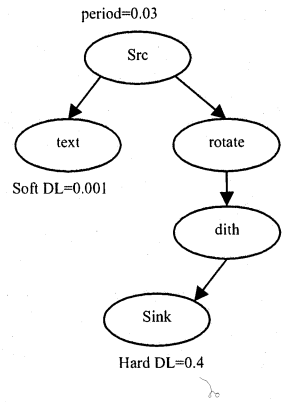


Figure 4.9: Office-Automation E3S Benchmark.

Table 4.12: PE library for the Office Automation system.

Process	PE0 (33)	PE1 (77)	PE2 (99)	PE3 (65)	PE4 (19)	PE5 (30)	PE6 (15)
src/sink	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
dith	0.029	0.0051	0.0039	0.0035	0.59	0.031	2.4
rotate	0.00061	0.0021	0.0012	0.0007	0.045	0.0083	0.27
text	0.0091	0.0028	0.0016	0.0016	0.29	0.017	1.1

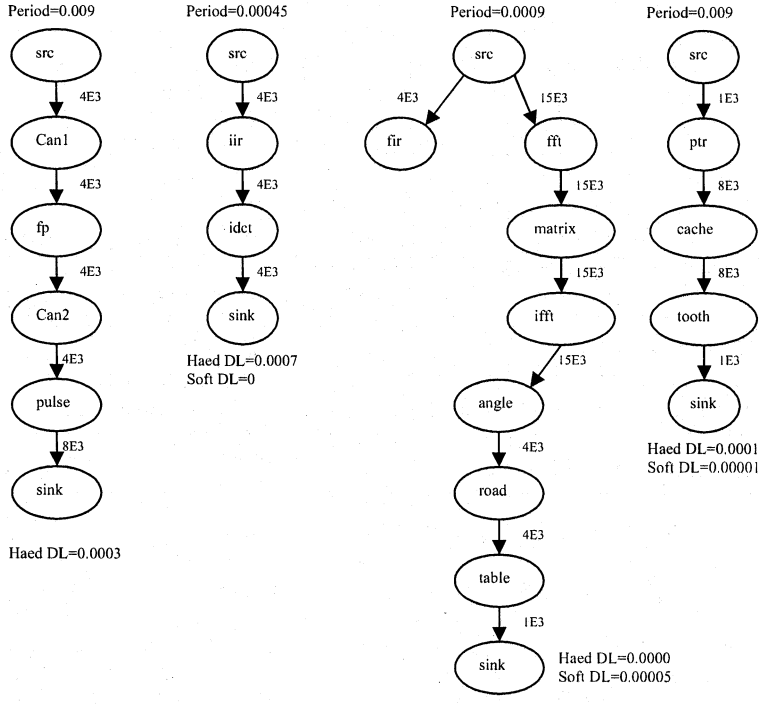


Figure 4.10: Auto-industry E3S Benchmark.

Table 4.13: PE library for the Auto-industry system.

Process	PE0 (33)	PE1 (77)	PE2 (99)	PE3 (126)	PE4 (65)	PE5 (216)	PE6 (45)	PE7 (30)
src/sink	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
can/angle	9e-6	1.5e-6	1.2e-6	1.1e-6	9.2e-7	1e-5	5.3e-7	2.8e-5
fp	2.3e-5	2.9e-6	2.3e-6	2.1e-6	1.8e-6	0.00013	8.9e-7	0.00016
pulse	4.5e-6	6.7e-7	4.8e-7	4.4e-7	3.2e-7	3.5e-6	2.1e-7	3.9e-6
iir	8e-5	1.2e-6	9.2e-6	8.4e-6	8.5e-6	5e-5	1.5e-6	8.3e-6
idct	0.00091	8.5e-5	6.7e-5	6.1e-5	5.7e-5	0.00076	2.6e-5	0.00084
fir	6.9e-5	9.7e-6	7.7e-6	7e-6	4.1e-6	4.8e-5	1.7e-6	3.8e-5
fft	0.014	0.0016	0.0011	0.001	0.00081	0.0084	0.00033	0.018
matrix	0.00067	0.0009	0.00064	0.00058	0.00031	0.022	0.00016	0.036
ifft	0.013	0.0015	0.001	0.00093	0.0007	0.0075	0.00032	0.016
road	3.2e-6	5e-7	3.6e-7	3.3e-7	4.1e-6	5e-5	1.4e-7	3.1e-6
table	3e-5	3.6e-6	2.8e-6	2.6e-6	2.9e-6	4.8e-5	1.9e-6	0.00013
ptr	0.00033	4.9e-5	3.9e-5	3.5e-5	3.7e-5	0.0004	1.6e-5	0.00049
cache	3.5e-6	4.8e-7	3.4e-7	3.1e-7	2.4e-7	2.7e-6	1.5e-7	3.8e-6
tooth	7.4e-5	8.5e-6	6.7e-6	6e-6	7.7e-6	5.3e-5	3.4e-4	8.4e-5

Table 4.14 compares the results produced by our algorithm for E3S benchmarks [43] with EMOGAC [8]. The soft deadline violation proportion is defined as the sum of the times by which every copy of each task misses its soft deadline, divided by the hyperperiod. The first column lists E3S benchmark examples for automotive, telecom, networking and office automation application. The second and third columns list the prices and deadline violation proportion presented by Dick [8]. The fourth and fifth columns show the prices of the solutions produced by our algorithm. It can be observed that our proposed method is able to achieve better deadline violation proportion for lower resource cost, as compared to the solutions produced by EMOGAC [8].

Table 4.14: Results - E3S benchmarks.

Example	EMOGAC		OUR	
	price	**Soft DL viol. prop.**	**price**	**Soft DL viol. prop.**
Automotive	169	2.08	139	1.17
	530	1.05	301	0.53
Telecom	291	4.58	228	3.70
	378	3.18	284	3.07
Networking	57	1.31	52	0.726
	70	1.23	76	0.64
Office Auto.	66	0.02	65	0.02

4.5 Multi-mode Applications

To test the multi-mode feature of our algorithm, we used Hou and Wolf examples from section 4.3 [17]. We considered a multi-mode system that has three possible modes, first mode is for an application with Hou and Wolf task 1 and task 2, second mode is for an application with Hou and Wolf task 1 and task 3, third mode is for an application with Hou and Wolf task 3 and task 4. The goal of the multi-mode feature is to find an architecture which is globally optimized for all three applications. To achieve this goal, while PE allocation and process assignment to PEs are performed for a particular mode, all other modes of the system are considered as well. The results for this multi-mode system are listed in Table 4.15 fourth column. The results for Oh and Ha's algorithm for the same system are listed in second column [29], and third column lists the results for Kim and Kim's algorithm [22]. From Table 4.15 we can conclude that our method has been able to achieve better or same quality results as compared to the other multi-mode algorithms presented in literature [22, 29].

Table 4.15: Results for the multi-mode feature.

Example	Oh and Ha	Kim and Kim	Proposed Method
H&W1,2 and H&W1,3 and H&W 3,4	220 (X,X,Link)	170 (X,Y,Link)	170 (X,Y,Link)

In this chapter we presented the experimental results for the proposed method. Experiments were conducted on random graphs from the literature [17, 33] and different industrial benchmarks [21, 43]. MPEG encoder application, telecommunication application, networking application, office automation application and automotive application have been used for co-synthesis [21, 43]. We compared the results with the results presented for various previous real-time co-synthesis techniques. The results for theses systems had better or same quality compared to those presented in the literature.

Chapter 5

Conclusions and Future Work

In this thesis, we presented a co-synthesis methodology for multiprocessor multi-task systems with real-time constraints. The proposed method determines a set of feasible solutions with optimized partitioning and real-time schedules for processes and data communication. The method is capable of producing acceptable solutions for critical systems with hard real-time deadlines, by employing process level prioritization and by meeting the process level deadlines.

A data flow graph was used to represent each of the tasks of a system. PE libraries of heterogeneous processing elements along with the imposed constraints for the target system are provided to the co-synthesis algorithm. A genetic algorithm was used for PE and communication resource allocation and assignment. The solutions for the first generation were produced by randomized allocation and assignment procedure.

Evolution of a new generation is performed by ranking, selection, crossover and mutation operation on the previous generation. Every solution of each generation of a co-synthesis run is scheduled and evaluated before the evolution of the next generation. The presented methodology employs efficient scheduling mechanism and allocation technique for PEs as well as for communication links. At the end of a run, the feasible solutions of the final generation that meet the imposed constraints are presented to the user. These solutions present the design space with trade-off points.

Experiments were conducted on random graphs from the literature [17, 33] and different industrial benchmarks [21, 43]. MPEG encoder application, telecommunication application, networking application, office automation application and automotive application have been used for co-synthesis [21, 43]. Timing and area constraints were imposed on these applications. Our algorithm produced low cost architectures for these fairly large and complex embedded systems, satisfying the schedulability condition in practical times. Moreover, the proposed scheduling methodology achieved better PE utilization as compared to the conventional non-preemptive scheduling technique. The results for theses systems had better or same quality compared to those presented in the literature. Moreover, previous co-synthesis algorithms presented in the literature do not provide allocation techniques and scheduling methods for the communication events [9, 23, 39], which is crucial for system optimization and accurate solution eval-

uation.

Our proposed co-synthesis method is accurate and efficient for finding a feasible optimized architecture. However, it can be improved in a number of ways. One of the enhancements in this approach could be for system-on-chip co-synthesis. It is possible to implement some embedded systems using a single integrated circuit (IC). This may reduce the cost and improve the performance.

Another interesting direction could be the co-synthesis of dynamically reconfigurable embedded systems. Dynamic reconfiguration of FPGAs may reduce the amount of hardware required in an embedded system. Recently the flexibility and reconfiguration speed of FPGAs have been improved. Therefore, option of reconfiguring the FPGAs while the embedded system is operating has become more practical.

Finally more non-functional requirements could be considered in addition to area and real time constraints. Reliability is a very critical metric for fault-tolerant systems. Reliability could be considered as one of the multi-objective optimization goal for the co-synthesis algorithm. Low power consumption can also be considered as a non-functional requirement.

Bibliography

[1] P. Areto, S. Juhasz, Z. Mann, A. Orban, and D. Papp, "Hardware-software partitioning in embedded system design". *Proceedings of the IEEE Int. Symposium on Intelligent Signal Processing*, pp. 197–202, Sept. 2003.

[2] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell, "Applying new scheduling theory to static priority pre-emptive scheduling". *Software Engineering Journal*, Vol. 8, pp. 284–292, 1993.

[3] J. Axelsson, "Architecture synthesis and partitioning of real-time systems: acomparison of three heuristic search strategies". *Proceedings of the Fifth International Workshop on Hardware/Software Codesign. (CODES/CASHE '97)*, Braunschweig, Germany, pp. 161–165, March 1997.

[4] S. Chakraverty, C. Ravikumar, and D. Choudhuri, "An evolutionary scheme for cosynthesis of real-time systems". *Design Automation Conference*, pp. 251 –256, Jan 2002, Bangalore, India.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment". *Journal of the ACM*, Vol. 20, pp. 46 –61, 1973.

[6] B. Dave and N. Jha, "Cohra: Hardware-software co-synthesis of hierarchical distributed embedded system architectures". *Proceedings of the 11th International Conference on VLSI Design: VLSI for Signal Processing*, pp. 347, 1998, Chennai, India.

[7] B. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of embedded systems". *Proceedings of the 34th Annual Conference on Design Automation*, pp. 703–708, June 1997, California.

[8] R. P. Dick. *Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems*. PhD thesis, Princton University, Nov. 2002.

[9] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems". *IEEE Transactions Computer-Aided Design*, Vol. 17, pp. 920–935, Oct. 1998.

[10] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "Hardware software partitioning of VHDL system specifications". *Proceedings of European Design Automation Conference,* , pp. 434–439, 1996, Geneva, Switzerland.

[11] R. Ernst, J. Henkel, and T. Benner, "Hardware software cosynthesis for micro-controllers". *IEEE Design and Test of Computers*, Vol.10, pp. 64–75, Dec. 1993.

[12] J. Fisher, "Trace scheduling: A technique for global microcode compaction". *IEEE transaction on Computers*, Vol. C3, No. 7, pp. 478 –490, July 1981.

[13] D. E. Goldberg. *Genetic algorithms in search, optimization*, and machine learning. *Addison-Wesley, Reading, MA*, 1989.

[14] R. K. Gupta and G. D. Micheli, "Hardware-software cosynthesis for digital systems". *IEEE Design and Test of Computers*, Vol. 10, pp. 29–41, Sept. 1993.

[15] L. Hafer and E. Hutchings, "Bringing up bozo". *IEEE Design and Test of Computers*, March 1990.

[16] H. Henkel and R. Ernst, "A hardware/software partitioner using a dynamically determined granularity". *Proceedings of the 34th Conference Design Automation*, page 691–696, 1997.

[17] J. Hou and W. Wolf, "Process partitioning for distributed embedded systems". *Fourth International Workshop on Hardware/Software Co-Design*, pp. 70-76, Mar. 1996, Pittsburgh.

[18] T. Hu, "Parallel sequencing and assembly line problems". *Operation Research*, Vol. 9, No. 6, pp. 841–848, 1961.

[19] A. Kalavade and E. Lee, "The extended partitioning problem: hardware/software mapping and implementation-bin selection". *Proceedings of the 34th 34th Design Automation Conference*, pp. 12–18, June 1997, Chapel Hill, NC.

[20] I. Karkowski and H. Corporaal, "Design space exploration algorithm for heterogeneous multi-processor embedded system design". *Design Automation Conference*, pp. 82–87, June 1998.

[21] G. N. Khan and U. Ahmed, "Hardware-software cosynthesis of multiprocessor embedded architectures". *Proceedings of the 4th IEEE Int. Symp. Embedded Computing, (AINA-07 workshops)*, pp. 804 – 810, May 2007.

[22] Y. Kim and T. Kim, "HW/SW partitioning techniques for multi-mode multi-task embedded applications". *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pp. 25 –30, 2006, Philadelphia, PA, USA .

[23] C. Lee and S. Ha, "Hardware-software cosynthesis of multitask MPSoCs with real-time constraints". *Proceedings of the 6th International Conference On ASIC*, Vol. 2, pp. 919 – 924, Oct. 2005.

[24] Y. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration". *32nd ACM/IEEE conference on Design Automation*, pp. 456–461, 1995.

[25] H. Liu and D. Wong, "Integrated partitioning and scheduling for hardware/software co-design". In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pp. 609–614, Oct 1998, Austin, TX.

[26] V. Mooney, III, and G. D. Micheli, "Real time analysis and priority scheduler generation for hardware-software systems with a synthesized run-time system". *Computer-Aided Design. Digest of Technical Papers*, Nov 1997, San Jose.

[27] J. Nestor and D. Thomas, "Behavioral synthesis with interfaces". *Design Automation Conference*, pp. 461–466, 1986.

[28] H. Oh and S. Ha, "A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling". *7th international workshop on Hardware/software codesign*, pp. 183–187, 1999.

[29] H. Oh and S. Ha, "Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. *Proceedings of the 10h Int. Symposium on Hardware/Software Codesign*, pp. 133–138, 2002.

[30] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's". *IEEE transaction on CAD/CAS*, Vol. 8, No. 6, pp. 661 –679, July 1989.

[31] D.-T. Peng, K. Shin, and T. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems". *IEEE Transactions on Software Engineering*, Vol. 23, No. 12, pp. 745 – 758, December 1997.

[32] R. Postman, J. Lis, A. Nicolau, and D. Gajski. "Percolation based scheduling", *Design Automation Conference*, pp. 444–449, 1990, Orlando.

[33] 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.

[34] K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 4, pp. 412–420, April 1995.

[35] K. Ramamritham and J. A. Stankovic, "Efficient scheduling algorithms for real-time multiprocessor systems". *IEEE Transactions on of Parallel and Distributed Systems*, Vol. 1, pp. 184–194, 1990.

[36] D. Shaha, R. S. Mitra, and A. Basu, "Hardware software partitioning using genetic algorithm". *Tenth Int. Conference on VLSI Design-97*, pp. 155–160, Jan. 1997, Hyderabad, India.

[37] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 175–187, Feb 1993.

[38] J. D. Ullman, "NP-Complete scheduling problems". *J. Comput. Syst. Sci.*, Vol. 10, pp. 384–393, 1975.

[39] W. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems". *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 5, pp. 218–229, June 1997.

[40] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph inhardware/software co-synthesis". *Design, Automation and Test in Europe*, pp. 620–625, March 2001, Munich, Germany.

[41] J. Xu, "Multiprocessor scheduling of processes with release times, deadlines, precedences, and exclusion relations". *IEEE Transactions on Software Engineering*, Vol. 19, No. 2, pp. 139–154, 1993.

[42] T. Y. Yen and W. H. Wolf, "Communication synthesis for distributed embedded systems". *Proceedings of the Int. Conf. Computer-Aided Design*, pp. 288–294, Nov. 1995, San Jose, California .

[43] E3S: The Embedded System Synthesis Benchmarks Suite. http:// ziyang.eecs.northwestern.edu/dickrp/ tools.html.