Optimal Temporal Partitioning and Synthesis for Reconfigurable Architectures *

Meenakshi Kaul
mkaul@ececs.uc.edu

Ranga Vemuri
Ranga.Vemuri@UC.EDU

Laboratory for Digital Design Environments
Department of ECECS, University of Cincinnati,
P.O. Box 210030, Cincinnati, OH 45221–0030

Abstract

We develop a 0-1 non-linear programming (NLP) model for combined temporal partitioning and high-level synthesis from behavioral specifications destined to be implemented on reconfigurable processors. We present tight linearizations of the NLP model. We present effective variable selection heuristics for a branch and bound solution of the derived linear programming model. We show how tight linearizations combined with good variable selection techniques during branch and bound yield optimal results in relatively short execution times.

1 Introduction

Dynamically reconfigurable processors are becoming increasingly viable with the advent of modern field-programmable devices, especially the SRAM-based FPGAs. Execution of hardware computations using reconfigurable processors necessitates a temporal partitioning of the specification. Temporal partitioning divides the specification into a number of specification segments that are destined to be executed one after another on the target processor. While the processor is being reconfigured between executing the specification segments, results to be carried from one segment to a future segment must be stored in a memory. Reconfiguration time itself, along with the time it takes to save and restore the active data, is considered an overhead; it is desirable to minimize the number of reconfiguration steps, ie., the number of segments resulting from temporal partitioning, as well as the total amount of data to be stored and restored during the course of execution of the specification.

In addition to the traditional synthesis process, a temporal partitioning step must be undertaken to implement hardware computations using reconfigurable processors. While techniques exist to partition and synthesize behavior level descriptions for spatial partitioning on multiple chips, no previous attempts have been published for temporal partitioning combined with behavioral synthesis.

The paper presents a 0-1 non-linear programming (NLP) formulation of the combined problem of temporal partitioning and scheduling, function unit allocation and function unit binding steps in high-level synthesis. The objective of the formulation is to minimize the communication, that is, the total amount of data transferred among the partition segments, so that the reconfiguration costs are minimized.

We propose compact linearizations of the NLP formulation to transform it into a mixed integer 0-1 linear programming (LP) model. We show the effectiveness of our linearizations through experimentation. In addition, we develop efficient heuristics to select the best candidate variables upon which to branch-and-bound while solving the LP model. Again, we show the effectiveness of our heuristics through experimental results.

The paper is organized as follows. In Section 2 we discuss related work, and in Section 3 we provide an overview of our approach. The basic formulation is presented in Section 3 and its solution in Section 4. Experimental results are presented in Section 7 and Section 9.

2 Previous Work

Synthesis for reconfigurable architectures involves synthesis and partitioning. The partitioning can be both temporal and spatial. There has been significant research on spatial partitioning and synthesis, though the issue of temporal partitioning has been largely ignored. Early research [11, 12] in the synthesis domain solved the spatial partitioning problem independently from the scheduling and allocation subproblems.

The problem of simultaneous spatial partitioning and synthesis was first formulated as an IP by Gebotys in [1]. It produced synthesized designs which were 10% faster than previous research. However as mentioned by Gebotys in [2], the size of the model was too large and could solve only small examples. In both [1, 2] the dimension of the problem is reduced by not considering the binding problem. Though the model can handle pipelined functional units and and functional units whose latency is greater than one cycle, it cannot handle design explorations where two different types of functional units can implement the same operation. For example, we cannot explore the possibility of using a non-pipelined and a pipelined multiplier in the same design. Also heuristics were proposed to assign entire critical paths to partitions. This might lead to solutions that are not globally optimal. Both [1, 2] focus on synthesis for ASICs and hence attempt to minimize area. For reconfigurable processors based on FPGA technology, area (resources on the FPGA devices) should be treated as a constraint which must be satisfied by every temporal segment in the temporal partition.

*This work is supported in part by the US Air Force, Wright Laboratory, WPAFB, under contract number F33615-97-C-1043.

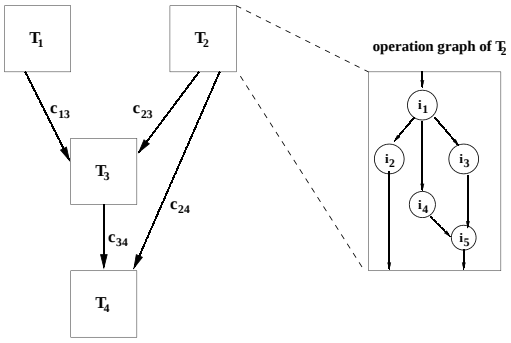

Figure 1: **Behavioral Specification**

The work of Niemann [3] presents an IP-based methodology for hardware software partitioning of codesign systems. MULTIPAR [4] also has a non-linear 0-1 model for spatial partitioning and synthesis, without involving binding. The linearization technique is however not the tightest for such a formulation, (see section 4 for details). To achieve faster runtime they solved the formulation by heuristic techniques, leading to suboptimal results.

Since a functional unit is not explicitly modeled in the above [1, 2, 4] formulations, they cannot determine the 'actual' area utilization of a partition. This factor though not critical for ASIC designs, is critical for reconfigurable processors based on FPGA technology. For example, in an optimal solution, a temporal segment may contain 1 multiplier and 5 adders and another temporal segment may contain 2 multipliers and 2 adders. Our formulation, which explicitly models binding of operations to functional units, can determine whether a functional unit is actually being used in any temporal segment. Moreover, we explicitly model the usage of each functional unit in each temporal segment and hence can explore the design space using 5 adders and 2 multipliers, although all these functional units may not simultaneously fit on the processor resources (function generators or CLBs in the FPGAs). Our model can automatically determine the above optimal solution, that simultaneously meets the resource constraints at the FPGA resource level as well as at the functional unit level.

3 System Specification

The behavior specification is captured in the form of a *Task Graph*, as shown in the Figure 1. The vertices in the graph denote a set of tasks, T. The dependencies among the tasks are represented by directed edges. Each task can be visualized as being composed of a number of operations which should stay together in one temporal partition. The edge labels in the task graph represent the amount of communication required if the two tasks connected by an edge are placed in different partitions. Let I be the set of all operations in the specification.

The cost metrics of the target FPGA, FPGA resource capacity (C) and temporary on-board memory size (M_s) are the inputs to our system. Typical resources for an FPGA are combinational logic blocks and function generators. We assume a component library consisting of various functional units which can execute the operations in the specification. The components in the library are characterized by cost metrics in terms of delay times and FPGA resource requirements.

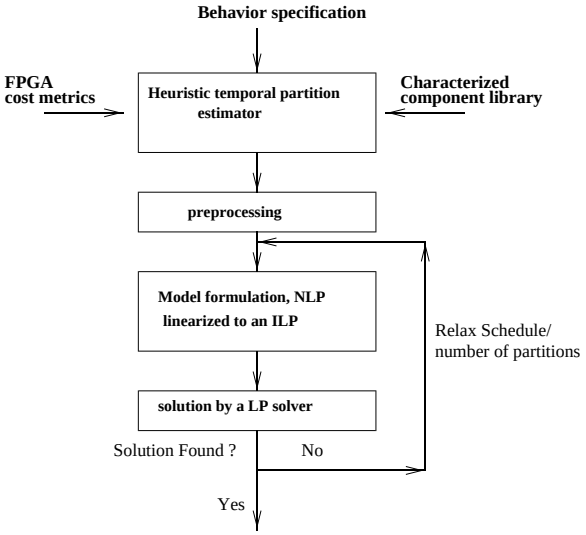

Figure 2: **Flow of the Temporal Partitioning and Synthesis system**

An outline of the temporal partitioning system is shown in Figure 2. The system proceeds by first heuristically estimating the number of segments (N), which becomes an upper bound on the number of temporal segments in the NLP formulation. It uses a fast, heuristic list scheduling technique to estimate the number of segments.

Before the problem can be formulated, we need to determine the ASAP and ALAP schedules for the operations. These will be used to set the mobility ranges for the operations in the formulation. This is done in a preprocessing step over the combined operation graph of the specification. From this schedule we can also determine the set of functional units F, which must be used for the design exploration.

While partitioning the specification, we honor the task boundaries in the specification. That is, a task cannot be split across two temporal segments. However, if two different tasks are together on the same segment, then they share control steps and functional units among them. If it is desired to permit splitting of tasks across segments, then each operation in the specification may be modeled as a task in our system. In this case, each task would have only one operation associated with it. The entire formulation developed in this paper will work correctly. Formally the system is defined as follows -

- $t_i \rightarrow t_j$, a directed edge between tasks, $t_i, t_j \in T$, exists in the task graph. t_i is at the tail of the directed edge and t_j is at the head, when the execution of task t_j depends on some output of t_i.
- $i_i \rightarrow i_j$, a directed edge between operations, $i_i, i_j \in I$ exists.
- $Bandwidth(t_i, t_j)$, number of data units to be com-

municated between tasks t_i and t_j.
• $Op(t)$, the set of all the operations in the operation graph of a task t.
• F, is the set of functional units required for the most parallel schedule of the operation graph.
• $Fu(i)$, the set of functional units from F, on which operation i can execute.
• $Fu^{-1}(k)$, the set of operations which can execute on functional unit k.
• $CS(i)$, the set of control steps over which operation i can be scheduled. Ranges from $ASAP(i) \ldots ALAP(i)+L$, where L is the user-specified relaxation over the maximum ALAP for the schedule. $ASAP(i)$ and $ALAP(i)$ are the As Soon As Possible and As Late As Possible control steps for operation i.
• $CS^{-1}(j)$, the set of operations which can be scheduled on control step, j.
• N upper bound on the number of partitions. The partitions are numbered $1 \ldots N$, the index of the partition specifies the order of execution of the partitions. Note that the generated optimal solution may have fewer than N partitions.
• M_s scratch memory available for storage between partitions.
• $FG(k)$, the number of function-generators used for functional unit k, obtained from the characterized component library.
• C, is the resource capacity of the FPGA. sectionNon-Linear 0-1 model

In this section we describe the variables, constraints and cost function used in the formulation of our model.

3.1 Variables

We have three sets of decision variables, which model the three important properties - y_{tp} models the partitioning at the task level, x_{ijk} models the synthesis subproblem at the operation level, $w_{pt_1t_2}$ models the communication cost incurred if two tasks connected to each other are not placed in the same partition. All are 0-1 variables.

$$y_{tp} = \begin{cases} 1 & \text{if task } t \in T \text{ is placed in partition p,} \\ & 1 \leq p \leq \text{N} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1 & \text{if operation } i \in I \text{ is placed in control} \\ & \text{step } j \in CS(i), \text{ and uses functional} \\ & \text{unit } k \in Fu(i) \\ 0 & \text{otherwise} \end{cases}$$

$$w_{pt_1t_2} = \begin{cases} 1 & \text{if task } t_1 \text{ is placed in any partition} \\ & 1 \cdots p-1 \text{ and } t_2 \text{ is placed in any of} \\ & p \cdots N \text{ and } t_1 \rightarrow t_2 \\ 0 & \text{otherwise} \end{cases}$$

As will be seen y_{tp} and x_{ijk} are the fundamental system modeling variables. All other variables are secondary and are constrained in terms of the fundamental variables.

3.2 Temporal Partitioning

The variables y_{tp} model the partitioning behavior of the system. Temporal partitioning has the following constraints.

Uniqueness Constraint: Each task should be placed in exactly one partition among the N temporal partitions.

$$\forall t \in T \quad : \sum_{p=1}^{N} y_{tp} = 1 \tag{1}$$

Temporal order Constraint: Because we are partitioning over time, a task t_1 on which another task t_2 is dependent cannot be placed in a later partition than the partition in which task t_2 is placed. It has to be placed either in the same partition as t_2, or in an earlier one.

$$\forall t_2 \in T, \ \forall t_1 \rightarrow t_2, \ \forall p_2, \ 1 \leq p_2 \leq N-1 \quad :$$

$$\sum_{p_2 < p_1 \leq N} y_{t_1p_1} + y_{t_2p_2} \leq 1 \tag{2}$$

Scratch Memory Constraint: The amount of intermediate data stored between partitions should be less than the scratch pad memory M_s. The variable $w_{pt_1t_2}$, if 1, signifies that t_1 and t_2 have a data dependency and are being placed across temporal partition p. Therefore the data being communicated between them, $Bandwidth(t_1, t_2)$, will have to be stored in the scratch memory of partition p. The sum of all the data being communicated across a partition should be less than the scratch pad memory.

$$\forall p, \ 2 \leq p \leq N \quad :$$

$$\sum_{t_2 \in T} \sum_{t_1 \rightarrow t_2} (w_{pt_1t_2} * Bandwidth(t_1, t_2)) \leq M_s \tag{3}$$

Notice in the Figure 3, how the variable $w_{pt_1t_2}$, models not only the communication among tasks which are on adjacent temporal partitions, but also on the non-adjacent ones. 3 tasks are to be placed optimally on 3 partitions. On the left of the figure are the original equations used to model the constraints for the example. The equations on the right of the figure show the variables which will be 1 in the mapping of tasks to partitions shown and the constraint which have to be satisfied. The variable $w_{pt_1t_2}$, is shown to range from 2 to N, because the data at partition 1 is actually the external input to the system. We can reasonably assume that there is enough memory for the external input at any time, since this is known a priori and is not a function of the partitioning of the system. $w_{pt_1t_2}$ are 0-1 non-linear terms constrained as -

$$\forall p, \ 2 \leq p \leq N, \ \forall t_2 \in T,$$

$$\forall t_1 \rightarrow t_2, \ \forall p_1, \ 1 \leq p_1 < p, \ \forall p_2, \ p \leq p_2 \leq N \quad :$$

$$w_{pt_1t_2} \geq y_{t_1p_1} * y_{t_2p_2} \tag{4}$$

$$\forall p, \ 2 \leq p \leq N, \ \forall t_2 \in T, \ \forall t_1 \rightarrow t_2 \quad :$$

$$\sum_{1 \leq p_1 < p} \sum_{p \leq p_2 \leq N} (y_{t_1p_1} * y_{t_2p_2}) - w_{pt_1t_2} = 0 \tag{5}$$

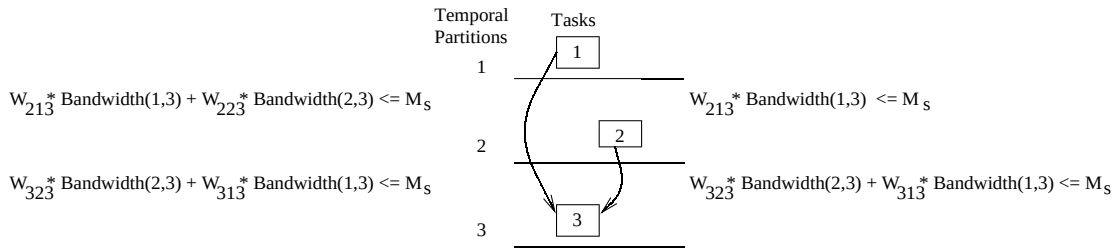

Figure 3: **The memory constraints to be satisfied if tasks are mapped to partitions as shown**

Equation (4) constrains $w_{pt_1t_2}$ to take a value 1, if any of the non-linear product terms in the associated equations are 1. Equation (5) constrains $w_{pt_1t_2}$ to a value zero, when all the associated product terms are 0. The equation alone (4) does not stop $w_{pt_1t_2}$, from taking a value 1 when all the product terms are 0 because $1 \geq 0$ is a valid solution to the constraint (4).

3.3 Synthesis

For the sake of clarity and ease of understanding we will not describe in the equations for synthesis, the formulations for pipelining, chaining, and latency of functional units. This formulation is easily extendible to incorporate those features, as described in [6] and [7]. We assume for the current model that the latency of each functional unit is one control step, and the result of an operation is available at the end of the control step.

Unique Operation Assignment Constraint: Each operation should be scheduled at one control step and on only one functional unit. Therefore only one variable x_{ijk} for an operation i, will be 1.

$$\forall i \quad : \sum_{k \in Fu(i)} \sum_{j \in CS(i)} x_{ijk} = 1 \tag{6}$$

Temporal Mapping Constraint: This constraint prevents more than one operation from being scheduled at the same control step on the same functional unit.

$$\forall j \quad : \sum_{k \in Fu(i)} \sum_{i \in CS^{-1}(j)} x_{ijk} \leq 1 \tag{7}$$

Dependency Constraint: To maintain the dependency relationship between operations, an operation i_1 whose output is necessary for operation i_2, should not be assigned a later control step than the control step to which i_2 is assigned.

$$\forall i_1 \rightarrow i_2, \;\; \forall j_2 \leq j_1, \;\; j_1 \in CS(i_1), \;\; j_2 \in CS(i_2) \quad :$$

$$\sum_{k_1 \in Fu(i_1)} x_{i_1 j_1 k_1} + \sum_{k_2 \in Fu(i_2)} x_{i_2 j_2 k_2} \leq 1 \tag{8}$$

3.4 Combining Partitioning and Synthesis

It is essential for partitioning for an FPGA to meet the area constraints of the FPGA. It is not necessary that all the functional units F, used in the design exploration are finally used in a partition. To determine whether a functional unit has been used in a partition we define the following decision variables -

$$u_{pk} = \begin{cases} 1 & \text{if functional unit } k \in F \text{ used in partition p, } 1 \leq p \leq N \text{ to perform some operation} \\ 0 & \text{otherwise} \end{cases}$$

$$o_{tk} = \begin{cases} 1 & \text{if task } t \in T \text{ uses functional unit } k \in F \text{ to perform some operation} \\ 0 & \text{otherwise} \end{cases}$$

These variables are constrained by,

$$\forall t \in T, \;\; \forall p, \;\; 1 \leq p \leq N, \;\; \forall k \in F \quad : u_{pk} \geq y_{tp} * o_{tk} \tag{9}$$

$$\forall p, \;\; 1 \leq p \leq N, \;\; \forall k \in F \quad : \sum_{t \in T} (y_{tp} * o_{tk}) - u_{pk} \geq 0 \tag{10}$$

The variable u_{pk}, defines the functional unit usage in a partition and the variable o_{tk} defines the functional unit usage in a task. These variables are also secondary and are defined in terms of the fundamental modeling variables y_{tp} and x_{ijk}. Equation (9), constrains u_{pk} to take a value 1, when any of its associated non-linear terms is 1. Equation (10) is needed to make sure, that at least one task on that particular partition uses the functional unit. The derivation for variable o_{tk} will be described in Section 4.

Resource Constraints We introduce resource constraints in terms of variables u_{pk}. Typical FPGA resources include function generators, combinational logic blocks (CLB) etc. Similar equations can be added if multiple resource types exist in the FPGA. α is a user defined logic-optimization factor in the range 0-1. Typical values of α using Synopsis FPGA components are in the range 0.6-0.8 [5].

$$\forall p, 1 \leq p \leq N \quad : \alpha * \sum_{k \in F} (u_{pk} * FG(k)) \leq C \tag{11}$$

Unique Control Step Constraint: We introduce constraints to make sure that each control step is

mapped uniquely to a temporal segment. We use a new derived variable c_{tj} to formulate this constraint.

$$c_{tj} = \begin{cases} 1 & \text{if task } t \in T \text{ has any operation} \\ & \text{mapped to control step } j \\ 0 & \text{otherwise} \end{cases}$$

$$\forall t \in T,\ \forall j,\ \forall i \in Op(t) \cap CS^{-1}(j)$$

$$c_{tj} \geq \sum_{k \in Fu(i)} x_{ijk} \tag{12}$$

$$\forall t_1 \in T,\ \forall t_2 \neq t_1,\ \forall j,\ \forall p_1, 1 \leq p_1 \leq N,$$
$$\forall p_2 \neq p_1, 1 \leq p_2 \leq N \quad :$$
$$c_{t_1 j} + y_{t_1 p_1} + c_{t_2 j} + y_{t_2 p_2} \leq 3 \tag{13}$$

If the operations of two distinct tasks use same control steps, then these tasks should be on the same partition.

In this paper, we have not considered flip-flop resource constraints. To consider flip-flop resources, the formulation must estimate the number of registers necessary to synthesize the design. It is straight-forward to add register optimization to our formulation on the lines proposed by Gebotys et al. [6].

3.5 Cost Function

Minimize the cost of data transfer between temporal partitions. This cost function will get an optimal solution using the least number of partitions and the least amount of inter-partition data transfer.

$$Minimize: \sum_{t_2 \in T} \sum_{t_1 \to t_2} \sum_{1 \leq p \leq N} (w_{pt_1t_2} * Bandwidth(t_1, t_2)) \tag{14}$$

4 Solving the 0-1 Non-linear Model

We can use various solution techniques in the mathematical programming field, for solving models which are non-linear in their objective function and constraints. The main approaches are (1) Linearization methods, (2) Enumeration methods, (3) Cutting plane methods. Refer to [8] for an interesting survey of all the approaches. Due to the existence of a good set of linearization techniques and the easy availability of LP codes for solving linear models we have chosen the linearization technique, though enumerative methods and cutting plane methods are viable alternatives.

Linearization of Equations (9-10) For each non-linear product term of the form $a * b$, we generate a new 0-1 variable c as, $c = a * b$, which can be written in Fortet's linearization method[8] [1] as

$$a + b - c \leq 1 \tag{15}$$

and

$$-a - b + 2 * c \leq 0 \tag{16}$$

Constraint (15) implies $c = 1$, when both a and b are 1. Constraint (16) implies $c = 0$, when either a or b become 0. We need both the constraints to get the correct solution.

However Glover and Wolsey [9] proposed an improvement, by defining c as a continuous real-valued variable (with an upper bound of 1), instead of, an integer variable by replacing equation (16) by the following two constraints, while retaining equation (15) intact -

$$a \geq c \tag{17}$$

and

$$b \geq c \tag{18}$$

Glover's linearization has been shown to be tighter than Fortet's. This has also been borne out by our experimentations. Equations (9) and (10) can be now replaced by the following compact linearized equations. Here z_{ptk} is a continuous real-valued variable bounded between 0 and 1 ($0 \leq z_{ptk} \leq 1$).

$$\forall t \in T,\ \forall p, 1 \leq p \leq N,\ \forall k \in F \quad : y_{tp} + o_{tk} - z_{ptk} \leq 1 \tag{19}$$

$$\forall t \in T,\ \forall p, 1 \leq p \leq N,\ \forall k \in F \quad : o_{tk} \geq z_{ptk} \tag{20}$$

$$\forall t \in T,\ \forall p, 1 \leq p \leq N,\ \forall k \in F \quad : y_{tp} \geq z_{ptk} \tag{21}$$

$$\forall t \in T,\ \forall p, 1 \leq p \leq N,\ \forall k \in F \quad : u_{pk} \geq z_{ptk} \tag{22}$$

$$\forall p, 1 \leq p \leq N,\ \forall k \in F \quad : \sum_{t \in T} z_{ptk} - u_{pk} \leq 0 \tag{23}$$

Constraint (19) implies $z_{ptk} = 1$, when both y_{tp} and o_{tk} are 1. Constraints (20) and (21) imply $z_{ptk} = 0$, when either y_{tp} or o_{tk} become 0. We need all three constraints to get the correct solution.

Linearization of Equations (4) - (5) This linearization can be done exactly as we have done for equations (9) and (10) by defining a new continuous variable for each distinct non-linear term. As will be shown shortly however in our final formulation we use lesser number of variables in linearizing $w_{pt_1t_2}$.

Formulating constraints for decision variable o_{tk} For any task t and functional unit k, variable o_{tk} is 1 if any of the x_{ijk} variables used to denote the synthesis variables of task t using functional unit k are 1. Consider a small example, where variables $x_{11k}, x_{12k}, x_{21k}, x_{22k}$ are the synthesis variables for task 1 and functional unit k. Then

$$\begin{aligned} o_{1k} &= x_{11k} \vee x_{12k} \vee x_{21k} \vee x_{22k} \\ \overline{o_{1k}} &= \overline{x_{11k} \vee x_{12k} \vee x_{21k} \vee x_{22k}} \\ &= \overline{x}_{11k} \wedge \overline{x}_{12k} \wedge \overline{x}_{21k} \wedge \overline{x}_{22k} \\ &= \overline{x}_{11k} * \overline{x}_{12k} * \overline{x}_{21k}, * \overline{x}_{22k} \end{aligned}$$

logical $\wedge$ and product $*$ being interchangeable for 0-1 terms. Using $\overline{a} = 1 - a$ and using the Glover's linearization we get -

$$x_{11k} + x_{12k} + x_{21k} + x_{22k} - o_{tk} \geq 0 \tag{24}$$

$$o_{tk} \geq x_{11k},\ o_{tk} \geq x_{12k},\ o_{tk} \geq x_{21k},\ o_{tk} \geq x_{22k} \tag{25}$$

[1] For each distinct product of variables, $\prod_{j \in N_k} x_j$, replace it by a new 0-1 variable x_{n+k} and add the constraints: $\sum_{j \in N_k} x_j - x_{n+k} \leq |N_k| - 1$, and $-\sum_j x_j + |N_k| * x_{n+k} \leq 0$

Graph No.	Characteristics			ILP Model		Run-Time
	N	A+M+S	L	Var	Const	
1	3	2+2+1	1	230	549	>7200
1	2	2+2+1	2	241	493	>7200
1	2	2+2+1	3	287	562	953.3
3	3	2+2+1	1	741	2239	>7200

Table 1: **Some Preliminary results**

Based on the above discussion we get the following constraints for o_{tk} -

$$\forall t \in T,\ \forall j \in CS(i),\ \forall k \in F,\ \forall i \in Op(t) \wedge Fu^{-1}(k) \quad :$$

$$o_{tk} \geq x_{ijk} \tag{26}$$

$$\forall t \in T,\ \forall k \in F \quad :$$

$$\sum_{i \in Op(t) \wedge Fu^{-1}(k)} \sum_{j \in CS(i)} x_{ijk} - o_{tk} \geq 0 \tag{27}$$

5 Preliminary Results

To verify our formulation by experimentation, we generated various random graphs. We ran some experiments with the current formulation to see how it performs. However as we see in Table 1, only one of the formulation solved in reasonable time, even though the graphs for which the experiments are done are not very large. Others were terminated when their run time became too large. Refer to Table 4, for the actual sizes of the graphs in this experiment. In the result tables, N denotes the number of partitions, A, M, S the number of adders, multipliers and subtracters respectively used in the design exploration. L, is the user specified latency margin bound. Var and Const denote the number of variables and constraints respectively that are generated for each graph by the ILP formulation. The run times are in seconds and all experiments have been run on an UltraSparc machine running at 175 Mhz. We use *lp_solve*, a public domain LP solver [10] for solving our ILP formulation.

6 Additional constraints that tighten the model further

When an ILP problem is solved by using LP techniques, we are solving the LP *relaxation* of the model. The LP relaxation is the same as the ILP, except the 0-1 constraints on the variables are dropped. The LP relaxation has a much larger feasible region, and the ILP's feasible region lies within it. An important method of reducing the solution time is to modify the formulation with constraints which cut away some of the LP feasible region, without changing the feasible region of the ILP. This is called *tightening* the LP model. After studying the model carefully we could identify some more constraints which cut off a large amount of non-optimal integer and non-integer solutions and helped in reducing the time required to solve the model.

- If a task t_1 is placed on some partition p, and $t_1 \rightarrow t_2$ exists then it means that t_2 definitely can only be placed in partition p or greater and therefore they cannot contribute to the scratch memory of any partitions less than or equal to p.

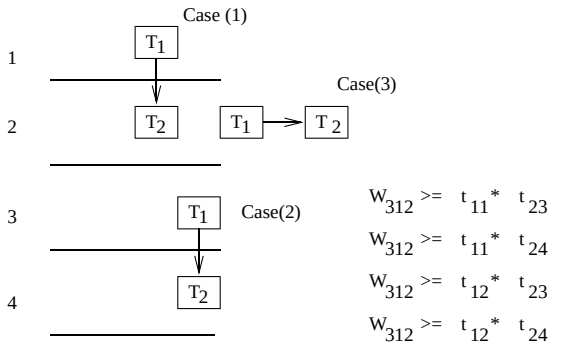

Figure 4: **Equations for variable w for 2 tasks and 4 partitions**

$$\forall t_2 \in T,\ \forall t_1 \rightarrow t_2,\ \forall p_1,\ 2 \leq p_1 \leq N \quad :$$

$$w_{p_1 t_1 t_2} + \sum_{p, p_1 \leq p \leq N} y_{t_1 p} \leq 1 \tag{28}$$

- If a task t_2 is placed on some partition p, and $t_1 \rightarrow t_2$ exists then it means that t_1 definitely can only be placed in partition p or lesser and therefore they cannot contribute to the scratch memory of any partitions greater than p.

$$\forall t_2 \in T,\ \forall t_1 \rightarrow t_2,\ \forall p_1,\ 2 \leq p_1 \leq N \quad :$$

$$w_{p_1 t_1 t_2} + \sum_{1 \leq p < p_1} y_{t_2 p} \leq 1 \tag{29}$$

- If $t_1 \rightarrow t_2$ exists, and both tasks are in the same partition, then they cannot contribute to the scratch memory of any of the partitions.

$$\forall t_2 \in T,\ \forall t_1 \rightarrow t_2,\ \forall p, 2 \leq p \leq N,$$
$$\forall p_1 \neq p, 2 \leq p_1 \leq N \quad :$$
$$y_{t_1 p} + y_{t_2 p} + w_{p_1 t_1 t_2} \leq 2 \tag{30}$$

In our formulation we have not introduced explicitly a new term for each distinct product term $y_{t_1 p_1} * y_{t_2 p_2}$, as we did for $o_{tk} * y_{tp}$. We have linearized $w_{pt_1 t_2}$ as follows -

$$\forall p, 2 \leq p \leq N,\ \forall t_2 \in T,\ \forall t_1 \rightarrow t_2 \quad :$$

$$w_{pt_1 t_2} \geq \sum_{1 \leq p_1 < p} y_{t_1 p_1} + \sum_{p \leq p_2 < N} y_{t_2 p_2} - 1 \tag{31}$$

Fewer new variables are introduced in the linearization because a single new variable reflects the constraints of several non-linear product terms. The main limitation in the above linearization is that though $w_{pt_1 t_2}$ will take a value 1 if one or more of its constituent products is 1, it will also

Graph No.	Characteristics			ILP Model		Run-Time
	N	A+M+S	L	Var	Const	
1	3	2+2+1	1	230	656	86.2
1	2	2+2+1	2	241	551	4670.4
1	2	2+2+1	3	287	620	9.7
3	3	2+2+1	1	741	2526	>9000

Table 2: **Results of tightening the Constraints**

take a value 1 even if none of the constituent products is 1. This is because there is no constraint which will limit the value of $w_{pt_1t_2}$ to 0, if all its constituent product terms are 0. If the new linear term is in a minimizing cost function, then such a solution will get cutoff after being generated as the solution, since the objective value of the cost function is greater in this case. This is true in our case as $w_{pt_1t_2}$ is part of the cost function. However as we shall see the new tightened inequalities will actually limit the solution space. $w_{pt_1t_2}$ will never be 1 if all of its constituent non-linear terms are 0.

Equations (28), (29) and (30) eliminate the problem mentioned above. We will show by an example how this is possible. In Figure 4 two tasks t_1 and t_2, $t_1 \rightarrow t_2$ are to be place in four partitions. The variable w_{312}, will have the equations shown. Consider the following three cases, where none of the 4 product terms shown in the figure are 1, how the variable w_{312} will still never be 1. (1) If $t_{11} = 1, t_{22} = 1$, the value $w_{312} = 1$ will be cut off by equation (29), (2) if $t_{13} = 1, t_{24} = 1$, the value $w_{312} = 1$ will be cut off by equation (28), (3) $t_{12} = 1, t_{22} = 1$, the value $w_{312} = 1$ will be cut off by equation (30).

- Another constraint observed, which reduced dramatically the solution time is that, if a task t uses a functional unit k and is placed in partition p, then the associated u_{pk} variable should also reflect this.

$$\forall t \in T, \ \ \forall k \in F, \ \ \forall p, \ \ 1 \leq p \leq N \quad : \\ o_{t_k} + y_{tp} - u_{pk} \leq 1 \qquad (32)$$

Equations (1), (2), (3), (6), (7), (8), (11), (12), (13), (19), (20), (21), (22), (23), (26), (27), (28), (29), (30), (31) and (32) are the constraints and Equation (14) is the cost function in our final model.

7 Experimental Results for Tightened Constraints

With the tightened constraints in place, we again ran the same series of experiments as in Table 1 and observed a significant improvement in the run times. Examples in rows 1, 2 and 3 of Table 2 solved, though the run times are very large for the sizes of graphs under consideration.

8 Solution by branch and bound

While solving a 0-1 LP by the branch and bound technique, an active node of the branching tree (initially the given mixed 0-1 problem), is chosen and its LP relaxation is solved, and a fractional variable, if any is chosen to branch on. If the variable is a 0-1 variable, one branch sets the variable to 0 and the other to 1. In this framework, there are two choices to be made at any time: the active node to be developed and the choice of the fractional variable to branch on. The variable choice can be very critical in keeping the size of the b-and-b tree small. We formulated the following heuristic to guide the selection of the branch and bound and has worked very well in practice. For the task graph, do a topological ordering of the tasks. For dependency $t_1 \rightarrow t_2$, t_1 will have a higher priority than t_2. The priorities range from 1..n, highest to lowest, and when the variable for tasks are generated in the ILP i.e., y_{tp}, the index t reflects this priority. While solving the model by an LP-solver, we always take the branch which sets the variable value to 1 first. To pick a variable to branch and bound we use the following heuristic -

- If there are variables y_{tp} still fractional, then pick the variable y_{tp} with the lowest t value and p value to branch on first.
- Once no fractional y_{tp} variables remain, pick any fractional u_{pk} variable to branch on. As a naive approach at this point once all the tasks are assigned to partitions we might have chosen to branch on any fractional x_{ijk} variable to continue the synthesis process, but our approach cuts off all solutions which are using some functional unit k which does not fit in the partition, very early in the branching process.

As you will observe in Section 9, this variable selection process has greatly reduced the run time of the experiments. This result emphasizes that careful study into the variable selection method must be done, rather than leave the variable selection to the solver (which randomly chooses a variable to branch on).

We choose to pick y_{tp} as a variable to branch on because once the task get assigned to partitions, the remaining problem is now just a scheduling-allocation problem whose linearization is quite tight and so produces less number of non-integer solutions. Observe that we are not forcing the tasks to lie on a particular partition, as in [2], where the variables in the critical path are forced into one partition, thus making the solution only locally optimal. Our method just guides the solution process to a quick solution (which may or may not be optimal), and this solution then acts as a lower bound on all other solutions undertaken after it in the solution process. This helps in eliminating many non-optimal branches of the solution tree. Since we are never forcing the value of any variable to some value, our solutions are always global optimal.

9 Experimental Results

We first explore the effect of different design parameters on an example behavioral specification. Table 3 shows the results of fixing the number of functional units and varying the latency and number of partitions. This graph called graph 1 has 5 tasks and 22 operations. 2 adders, 2 multipliers and 1 subtracter were used to schedule the design. The first row in Table 3

Graph No.	Characteristics					ILP Model		Results	
	Tasks	Opers	N	A+M+S	L	Var	Const	RunTime	Feasible
1	5	22	3	2+2+1	1	230	656	8.96	Yes
2	10	37	4	3+2+2	1	698	1992	51.13	Yes
3	10	45	3	2+2+2	1	741	2526	267.7	Yes
4	10	44	2	2+2+2	1	564	1421	240.64	Yes
4	10	44	3	2+2+2	0	635	1942	167.23	Yes
5	10	65	3	2+2+2	0	748	2472	.78	No
5	10	65	2	2+2+2	1	813	2032	310.45	Yes
6	10	72	3	2+2+2	0	1055	2900	882.27	Yes
6	10	72	2	2+2+2	1	1158	2465	1763.27	Yes

Table 4: **Temporal Partitioning Results for various Graphs**

N	A+M+S	L	Var	Const	RunTime	Feasible
3	2+2+1	0	183	583	1.72	No
3	2+2+1	1	230	656	8.96	Yes
2	2+2+1	2	241	551	9.91	Yes
2	2+2+1	3	287	620	8.86	Yes

Table 3: **Results of variation of Latency and number of Partitions for Graph 1**

shows that with no relaxation in latency, the design could not be feasibly partitioned onto 3 partitions. So the latency bound was relaxed by 1, and the design was optimally partitioned and synthesized onto 3 partitions. The latency bound was relaxed by 2 and the design fit onto 2 partitions. It was further relaxed to 3, and it fit optimally onto a single partition though 2 partitions were used in the design space exploration. In all these examples, the execution time for getting the optimal results is very small.

For any ILP formulation, being solved by branch and bound using LP relaxation of the model, it is very important to have tight linear relaxations and good variable selection methods, so that a large amount of non-integer solutions are cutoff from the solution space. We ran a series of experiments to substantiate the variable selection technique and the tightened constraints we have imposed on our model. In Table 4, the results of running the experiments on larger graphs are shown. The columns Tasks and Opers give the size of the specification in terms of the number of tasks and operations in them. Medium sized graphs of upto 72 operations, can be optimally partitioned in very small execution times.

10 Conclusion

To make our model an effective tool for temporal partitioning and synthesis, we need to add constraints to model the registers and buses used in the design. Note however that the number of variables (which largely influence the solution time) will not increase, as the current variable set is enough to model the additional constraints. In this paper we have presented a novel technique to perform temporal partitioning and synthesis optimally. Good linearization techniques and careful variable selection procedures were very helpful in solving the models in a short time. The effectiveness was demonstrated by the results.

References

[1] C. H. Gebotys, "Optimal Synthesis of Multichip Architectures", *IEEE ICCAD, p238-241, Nov. 1992.*

[2] C. H. Gebotys, "An Optimal methodology of Synthesis of DSP Multichip Architectures", *Journal of VLSI Signal Processing, 11, p9-19, 1995.*

[3] R. Niemann and P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming", *Proceedings of the ED&TC, 1996.*

[4] Y. Chen, Y. Hsu and C. King, "MULTIPAR: Behavioral Partitioning for Synthesizing Application-Specific Multiprocessor Architectures", *Proceedings of the EuroDac, p14-18, 1992.*

[5] M. Vootukuru, Ranga Vemuri and Nand Kumar, "Resource Constrained RTL Partitioning for Synthesis of Multi-FPGA Designs", *Proceedings of the 10th International Conference on VLSI Design, IEEE Press, 12 pages, 140-144, January 1997.*

[6] C.H. Gebotys and M. I. Elmasry, "Optimal VLSI architectural synthesis Area, Performance and Testability", *Kluwer Academic Publishers.*

[7] B. Landwehr, P. Marwedel and R. Domer, "OSCAR:Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming", *Proceedings of the EuroDac, p90-95, 1994.*

[8] P. Hansen, B. Jaumard and V. Mathon, "Constrained Nonlinear 0-1 programming", *ORSA Journal of Computing, Vol. 5, No. 2, 1993.*

[9] F. Glover and E. Woolsey, "Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program", *Operations Research 21:1, 156-161, 1974.*

[10] M. Berkelaar, *lp_solve()* Eindhoven University of Technology.

[11] K. Kucukcakar and A. Parker, "CHOP: A Constraint-Driven-System-Level Partitioner", *DAC, p514-519, 1991.*

[12] R. Gupta and G. DeMicheli, "Partitioning of Functional Models of Synchronous Digital Systems", *Proc. of ICCAD-90, p216-219, 1990.*