CENTER FOR
PARALLEL OPTIMIZATION

OPTIMAL RESOURCE ALLOCATION AND BINDING OF NON-PIPELINED DESIGNS

by

Renato De Leone and Rajiv Jain

Computer Sciences Technical Report #972

October 1990

Optimal Resource Allocation and Binding of Non-Pipelined Designs

Renato De Leone
Department of Computer Sciences
1210 W. Dayton Street
University of Wisconsin
Madison, WI 53706
(608)262-5083
deleone@cs.wisc.edu

Rajiv Jain
Department of Electrical and Computer Engineering
1415 Johnson Drive
University of Wisconsin
Madison, WI 53706
(608)262-3610
rajiv@ece.wisc.edu

Optimal Resource Allocation and Binding of Non-Pipelined Designs

Abstract

In this paper we give an integer linear program (ILP) formulation of the resource allocation and binding problem of high-level synthesis. Given a behavioral specification and a time-step schedule for operations, the formulation minimizes the number of wiring nets and multiplexers used in the design. This is the first time that an ILP model for minimizing multiplexers and wiring nets has been mathematically formulated and optimally solved. The model handles chaining, multi-cycle operations and tradeoffs chip area between wiring and resources.

1 Introduction

Automatic generation of RTL (register-transfer level) designs [1] from a behavioral description is known as high-level synthesis. Data path synthesis produces an RTL data path which meets the specified constraints and achieves a design goal. To synthesize a desired RTL design several conflicting and interacting sub-problems have to be solved. These sub-problems are scheduling, module allocation, module selection, module binding, and multiplexer and register allocation. In order to synthesize an optimal design (optimality is defined in terms of area, delay or other such measure) all these sub-problems have to be solved concurrently. Some of these sub-problems, such as scheduling, are known to be NP-complete, and the combined problem of solving all sub-problems concurrently is thought to be at least NP-hard. Furthermore, the design constraints imposed by the designer, such as area, power, and speed, makes the synthesis problem even more difficult.

In this paper we outline a new and novel procedure for performing module binding and register and multiplexer allocation and binding with an objective of minimizing the number of nets required for interconnecting the design. We assume that scheduling and module allocation have been performed a priori. The inputs to the system we propose are the data flow graph, the design library, the scheduling information (i.e. the mapping of operations to time steps in which they are executed) and the number of modules of each operation type (such as number of adders and multipliers) to be used in the design [2]. Any standard scheduling program such as MAHA [16], HAL [17], SLICER [14], ISYN [13] and ALPS [10] produces this required scheduling and module allocation information.

1.1 Problem Statement and Previous Work

Module binding is the task of assigning modules to data flow operation nodes. In executing two addition operations in the same time step using two adders, the decision as to which addition operation is performed in which adder module is module binding. The task of optimal module binding is dependent upon scheduling, module allocation, interconnect costs and delays.

[1] An RTL design consists of a data path (modules and registers interconnected via multiplexers, busses and wires) and a controller (PLA, microprogrammed or hardwired) which provides the control signals for sequencing of events in the data path.

[2] Information on resource count need not be specified, as we will see later in the paper.

The task of *register and multiplexer allocation and binding* is to assign data values to (from) registers and route them through multiplexers or busses from (to) the modules. Several programs which perform module binding and register and multiplexer allocation with different allocation philosophies exist [1] [9] [11] [15] [19]. Most of theses systems try to minimize the number of registers and multiplexers, with the exception of [15] which tries to minimize wiring. Furthermore, these systems employ heuristics to solve the proposed problem. There are several drawbacks of this approach. The quality of the designs produced by these systems cannot be determined as there is no way of comparing against a known absolute minimum. The state-of-the-art synthesis systems are compared by example (such as: *For a fixed behavioral description, system X produces a design with one multiplexer more than system Y*).

While working on the WISRD (WIsconsin Synthesis Research and Development) project we were faced the issue of selecting a synthesis package which performed module, register and multiplexer allocation and binding. Naturally we wanted to have the best possible program which could perform this function. Unfortunately, on examining several existing packages we realized that selecting the best technique was not easy since comparisons between different programs were not possible owing to lack of a standard measure. So we thought of developing the package proposed here which is guaranteed to provide optimal results against which other systems can be evaluated. Another motivation for solving the problem as proposed here is the following presumption. For the purposes of design space exploration, designers like tools which give good results quickly. However, once the search space has been narrowed the designer is willing to accept a (slightly) larger design time with the guarantee of getting optimal results. Thus, existing synthesis heuristics can be used to get a first-cut solution and after the design measures (such as area and performance) have been determined the proposed method can be used for the final optimal RTL design.

In this paper we outline a method for solving the module binding and register and multiplexer allocation and binding tasks concurrently. First, we formulate the problem as a zero-one integer linear programming (ILP) problem and then use existing ILP solution techniques to solve the formulation. The ILP, however, is an NP-complete problem, implying that (in the worst case) the number of computations to be performed to get an optimal solution doubles with the addition of each decision variable. In 1983 Crowder et. al [4] demonstrated that optimal solutions to certain problems with 2750 binary variables can be obtained in less than an hour of IBM 370/168 CPU time. Their solution technique was a mixture of branch and bound and the cutting plane algorithm. Their results show that reasonable sized module binding and register and multiplexer allocation and binding problems can be quickly and optimally

solved. ILP techniques have been successfully used by Lee et. al [10] to model and solve the scheduling and module allocation problem with better computer runtime than several other state-of-the-art scheduling and module allocation systems.

The synthesis problem has been formulated as an ILP problem by Hafer and Parker in [5]. Hafer's formulation, however, does not minimize number of wiring nets or multiplexers used in the design with the result that the final design may actually require a large multiplexer and wiring area. Furthermore, this procedure entails enormous runtime and is not practical. By decomposing synthesis into two sub-problems, namely, (i) scheduling and module allocation, and (ii) module binding and register and multiplexer allocation and binding, we can approach each part independently and quickly solve them optimally. Solving each individual problem independently, however, does not necessarily lead to a globally optimal solution. The assumption of breaking synthesis into these two sub-problems has been used by almost all synthesis systems found in the literature, for example, [6], [8], [12], [14], [17], [18]. The notable exceptions are [5] in which all sub-components are solved concurrently, and [3] in which the synthesis problem is solved by partitioning it into two sub-problems different from the above breakdown, namely the scheduling, module allocation and module binding problem and, the register and multiplexer allocation. We believe the formulation presented in this paper to be the first attempt at solving the resource allocation and binding problem with the objective of minimizing wiring and multiplexer area.

There are several advantages to using the ILP approach. If the formulation is run to completion the results produced are optimal. During the process of finding the optimal solution the computer can print out the current best solution and inform the designer approximately how far is the solution from the optimal. Thus, if the designer should exhaust allocated run time for solving the problem and terminate the solution process before it runs to completion, the current best solution and the information as to how far it from the optimal is known. This information can be used by the designer to budget the design time versus the quality of designs produced. Furthermore, to reduce the solution time the designer can force the solution package to find a solution which is guaranteed to be within an ϵ percentage of the optimal solution. Thus, if the user is using the synthesis tools for budgeting, a quick and guaranteed estimate of the final design can be easily obtained.

Another advantage of the proposed technique is the ease of performing area tradeoffs between modules, registers and interconnect. In most of the other synthesis programs optimizing criteria are hardwired into the module binding and register and multiplexer allocation and binding programs. For example, [9] and [17] minimize register and multiplexer costs alone. Using the ILP formulation it is easy to modify the objective

function to the desired optimizing parameter without altering the underlying problem.

The paper is organized as follows. The basic model is developed in Section 2. In Section 3 we give an example with constraints. Some experimental results are also presented in this section. The area tradeoff model is presented in Section 4, and the paper concludes with comments on research directions.

2 Problem Formulation

In this section we outline the basic ILP model for resource allocation and binding. Our long-term objective is to employ heuristics such as relaxed Lagrangian method to get near-optimal solutions quickly. The model allows multi-cycle operations and chaining. Only the non-pipelined design-style model is presented in this paper. Future work includes extending the model to allows conditional branching and pipelined data paths.

The model assumes that scheduling has been completed a priori and the time-step information for every operation, the number of module of each type and the number of registers is predetermined. Tradeoffs between operator and register area and interconnect area will be done as part of future work. Before we develop the model for non-pipelined design we define some terms. We assume,

- n_1 modules of type $T_1 : M_1, \ldots, M_{n_1}$,
  n_2 modules of type $T_2 : M_{n_1+1}, \ldots, M_{n_1+n_2}$,
  $\ldots$,
  n_k modules of type $T_k : M_{n_1+n_2+\ldots+n_{k-1}+1}, \ldots, M_{n_1+n_2+\ldots+n_k}$,
  with $n = n_1 + n_2 + \ldots + n_k$ and $n_0 = 0$;
- m registers are required for the data path. Let the registers be labeled as $R_1, R_2, \ldots, R_m$;
- T is the number of time steps the input specification is partitioned into, and $t = 1, \ldots, T$;
- l_t operations $OP_1^{(t)}, OP_2^{(t)}, \ldots, OP_{l_t}^{(t)}$ must be executed at time t;
- r_t values $V_1^{(t)}, V_2^{(t)}, \ldots, V_{r_t}^{(t)}$ are available in registers in time step t.

Register Allocation and Binding: Let $y_{i,j,t} = 1$ if value $V_i^{(t)}$ is stored in register j in time step t and zero otherwise. The register allocation and binding is modeled by the set of following constraints. Let m be the total number of registers required for an RTL design. Every value produced in a time step and used in subsequent time steps must be stored in exactly one register. Thus, for value $i = 1, \ldots, r_t$ and time step $t = 1, \ldots T$,

$$\sum_{j=1}^{m} y_{i,j,t} = 1 \tag{2.1}$$

A register cannot hold more than one value across different time steps. For every register $j = 1, \ldots, m$ and every time step $t = 1, \ldots T$

$$\sum_{i=1}^{r_t} y_{i,j,t} \leq 1 \tag{2.2}$$

Module Allocation and Binding: Let $x_{i,j,t} = 1$ if operation $OP_i^{(t)}$ is executed on module j in time step t and zero otherwise. The module allocation and binding problem can be formulated as follows. Every operation must be executed on a module which can perform the operation. Suppose operation $OP_i^{(t)}$ is of type s, then

$$\sum_{j=n_{s-1}+1}^{n_s} x_{i,j,t} = 1 \tag{2.3}$$

Each module cannot execute more than one operation in any time step. Thus, for every module $j = 1, \ldots, n$ and in every time step $t = 1, \ldots, T$,

$$\sum_{i=1}^{l_t} x_{i,j,t} \leq 1 \tag{2.4}$$

Before we develop the interconnect model we will explain the overall objective of the formulation.

Objective function: The objective of the problem is to perform register and module allocation and binding such that the overall cost (area) of the RTL design is minimized. That is

$$\text{minimize } (ma_r + \sum_{i=1}^{k}(n_i \times a_i) + mux\ area + wiring\ area)$$

where, a_r is the register area and a_i is the area of the module implementing operation i.

The number of of modules of each type and registers to be used in the implementation are determined by the max-cut of the input description and scheduling of the input description respectively. Thus, $ma_r + \sum_{i=1}^{k}(n_i \times a_i)$ is fixed a priori and the only quantity we can minimize is wiring and multiplexer area. Given that the number of modules is fixed, we assume that wiring area is directly proportional to the number of nets in the design [7]. The number of nets in an RTL design consists of the connections between modules and registers. Large number of connections between resources (register and modules) require large number of multiplexers at the input of each resource resulting in large quantities of multiplexers and large number of wiring nets. By minimizing the number of nets, we can reduce the number of multiplexers required in the design. Hence for the purposes of area minimization we concentrate on the reduction of the number of nets required in the design. Thus, we model register and module allocation and binding with the aim of "minimizing nets". A similar objective has been adopted in [15].

Let $z_{i,j,t} = 1$ if the result of operation $OP_i^{(t)}$ is sent to module M_j (if $j \leq m$) or stored in register R_{j-m} (if $j > n$) at time t. The output of a module must to go to a register or another module. Thus, for every operation $i = 1, \ldots, l_t$ (l_t is the number of operations which are to be performed in time step t) and in every time step $t = 1, \ldots, T$,

$$\sum_{\substack{j=1 \\ j \neq i}}^{n+m} z_{i,j,t} = 1 \tag{2.5}$$

If at time t operation $OP_i^{(t)}$ produces a value V_{i_3} which is to be used in some successive time step t' then the value must be stored in a register. The operation which needs this value V_{i_3} in a subsequent time step must retrieve the value from the register the value is stored in. This is modeled as,

$$z_{i,n+j,t} = y_{i_3,j,t'}, \quad \forall\, j = 1, \ldots, m \tag{2.6}$$

If at time t operation $OP_i^{(t)}$ is executed and a value V_{i_3} is produced and is needed at some successive time $t' > t + 1$, then the value V_{i_3} must be stored in a register over several time steps.

$$z_{i,n+j,t} + \sum_{q=1}^{l_t} z_{q,n+j,t} \leq 1, \quad \forall\, j = 1, \ldots, m, \quad \forall\, t = t+1, \ldots, t' \tag{2.7}$$

Equation 2.7 ensures that register j is not used to store any other value in time steps t through t'. Multi-cycle operations can be similarly handled.

Define $w_{i,j,t} = 1$ if there exists a transfer of value from resource i to resource j in time step t. If $OP_i^{(t)}$ (executed at time t) requires a value $V_{i_1}^{(t)}$ then there must be a net from the register storing the value $V_{i_1}^{(t)}$ and the resource executing $OP_i^{(t)}$. This is modeled as,

$$w_{j,q,t} \geq x_{i,q,t} + y_{i_1,j-n,t} - 1, \quad \forall\, q = n+1, \ldots, n \text{ and } j = 1, \ldots, n+m \tag{2.8}$$

Constraints for every value required by an operation $OP_i^{(t)}$ can be similarly written.

If $OP_i^{(t)}$ (executed at time t) produces a value $V_{i_1}^{(t)}$ which is to be stored in a register or consumed by another resource, then there must be a net from the resource executing operation $OP_i^{(t)}$ and the resource storing (or consuming) $V_{i_1}^{(t)}$. This is modeled as,

$$w_{q,j,t} \geq x_{i,q,t} + z_{i,j,t} - 1, \quad \forall\, q = 1, \ldots, n \text{ and } j = 1, \ldots, n+m \tag{2.9}$$

Let $\sigma_{i,j} = \sum_{t=1}^{T} w_{i,j,t}$ ($\sigma_{i,j}$ is a dummy variable used for explanation only). $\sigma_{i,j}$ gives us the number of times a value is passed from resource i to resource j over all time-steps. A large $\sigma_{i,j}$ implies that in several time-steps resource i passes a value to resource j, and a small $\sigma_{i,j}$ means that the net from resource i to j is sparingly used. To minimize the net count, our objective is to increase few $\sigma_{i,j}$ to large values and to make as many $\sigma_{i,j}$'s as possible go to zero. Let $w'_{i,j} = 1$ if $\sigma_{i,j} \geq 1$. Our objective function then is

$$minimize \sum_{i=1}^{m+n} \sum_{j=1}^{m+n} c_{i,j} w'_{i,j} \tag{2.10}$$

where $c_{i,j}$ is a the cost of connecting resource i to resource j, and is usually assumed to be the area of a wire.

The condition $w'_{i,j} = 1$ if $\sum_{t=1}^{T} w_{i,j,t} \geq 1$ is modeled by the following constraint

$$w'_{i,j} \geq w_{i,j,t} \quad \forall\, t = 1, \ldots, T \quad \forall\, i = 1, \ldots, n \text{ and } j = 1, \ldots, n+m \tag{2.11}$$

From this constraint we observe that the variable $w_{i,j,t}$ is a dummy variable used for illustrating the model alone and may be eliminated from the final formulation.

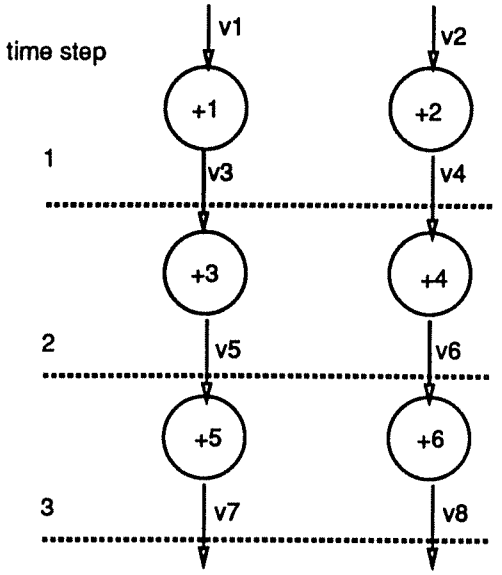


Figure 3.1: Example 1

3 An Example and Results

We will use the example in Figure 3.1 to illustrate the constraint equations. We have successfully experimented with data flow graphs of higher complexity than the one given in Figure 3.1.

The scheduling information provides us with the following information. $k = 1$ (that is there is one type of operation) and number of adders is two ($n_1 = 2$). $m = 2$ and $T = 3$. Operations are labeled as follows: +1 is $OP_1^{(1)}$, +2 is $OP_2^{(1)}$, +3 is $OP_1^{(2)}$, +4 is $OP_2^{(2)}$, +5 is $OP_1^{(3)}$, and finally, +6 is $OP_2^{(3)}$. Values are labeled as follows: v1 is $V_1^{(1)}$, v2 is $V_2^{(1)}$, v3 is $V_1^{(2)}$, v4 is $V_2^{(2)}$, v5 is $V_1^{(3)}$, v6 is $V_2^{(3)}$, v7 is $V_1^{(4)}$, and finally v8 is $V_2^{(4)}$.

To illustrate the formulation we will write down a subset of constraints for the example and use the simpler notation of the preceding paragraph in our presentation. Values $v1$ and $v2$ are stored in some input registers. By the convention adopted in literature [9], [15], [17] these input registers are read-only buffers and cannot be reused. [3] Thus, constraints 2.1 and 2.2 for these two input values are not required. Constraint 2.1 for

[3]This convention can be easily overruled in our formulation.

value $v3$ and $v4$ are:

$$\begin{aligned} y_{v3,3,2} + y_{v3,4,2} &= 1 \\ y_{v4,3,2} + y_{v4,4,2} &= 1 \end{aligned}$$

Note that registers are numbered 3 and 4. These equations ensure that values v3 and v4 are stored in a register. Similar constraint for values v5, v6, v7 and v8 can be written. Constraint 2.2 for value $v3$ and $v4$ are:

$$\begin{aligned} y_{v3,3,2} + y_{v4,3,2} &\leq 1 \\ y_{v3,4,2} + y_{v4,4,2} &\leq 1 \end{aligned}$$

Constraints for registers R1 and R2 for time step 3 can be similarly written. Now to model module binding. To ensure that operations +1 and +2 are executed in module 1 or 2, we have the constraints (cf. Constraint 2.3),

$$\begin{aligned} x_{+1,1,1} + x_{+1,2,1} &= 1 \\ x_{+2,1,1} + x_{+2,2,1} &= 1 \end{aligned}$$

Constraints for operations +3, +4, +5 and +6 can also be written. Furthermore, modules 1 and 2 may not execute more than one operation each in time step 1 is given by (cf. Constraint 2.4),

$$\begin{aligned} x_{+1,1,1} + x_{+2,1,1} &\leq 1 \\ x_{+1,2,1} + x_{+2,2,1} &\leq 1 \end{aligned}$$

Constraint 2.4 for modules 1 and 2 for time steps 2 and 3 can be written in a similar fashion.

Finally we write down the constraints for interconnect model. To model a connection between the module executing operations +1 and +2 and the registers which will store these values, we have the following constraints (cf. Constraint 2.5),

$$\begin{aligned} z_{+1,3,1} + z_{+1,4,1} &= 1 \\ z_{+2,3,1} + z_{+2,4,1} &= 1 \end{aligned}$$

Constraint 2.6 ensures that if a value stored in (or produced by) resource i is needed resource module j, then we must connect the two resources. For example, the result of +1 v3 which is stored in either R1 or R2 is needed by operation +3 in the next time step. Thus,

$$\begin{aligned} z_{+1,3,1} &= y_{v3,3,2} \\ z_{+1,4,1} &= y_{v3,4,2} \end{aligned}$$

Since we are not storing any value over more than one time step Constraint 2.7 is not required.

Lets us examine Constraint 2.8 in light of the v3 value transfer. Value v3 can be stored in resource 3 (R1) or 4 (R2), and operation +3 could be executed in resource 1 (M1) or 2 (M2). Now if v3 is stored in R1 and +3 is executed in M1 then there must exist a net from R1 to M1. This net is required in time step 2 and the corresponding constraint is given as,

$$w_{3,1,2} \geq x_{+3,1,2} + y_{v3,3,2} - 1$$

Similarly if v3 is stored in R2 and +3 is executed in M1 then we have the following constraint,

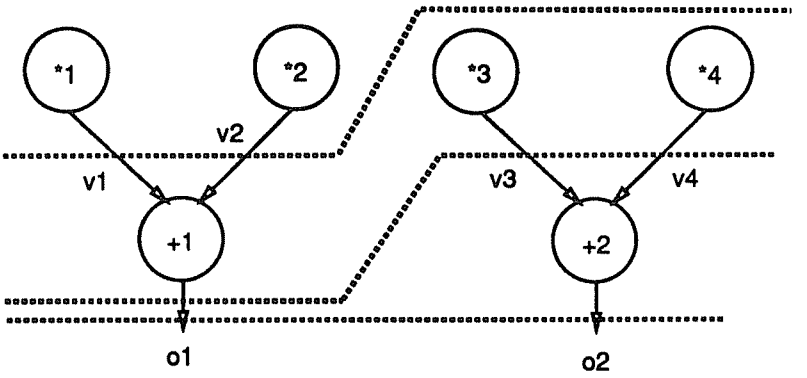
$$w_{4,1,2} \geq x_{+3,1,2} + y_{v3,4,2} - 1$$

Similar constraints for other value transfers can be written. To illustrate Constraint 2.9 we consider the value transfer from the module executing operation +1 to the register storing v3 in time step 1. This is given by the following four constraints. First constraint ensures a connection between M1 and R1 if +1 is performed in M1 and v3 is stored in R1; second ensures a connection between M1 and R2 if +1 is performed in M1 and v3 is stored in R2; third ensures a connection between M2 and R1 if +1 is performed in M2 and v3 is stored in R1; and finally, fourth constraint ensures a connection between M2 and R2 if +1 is performed in M2 and v3 is stored in R2.

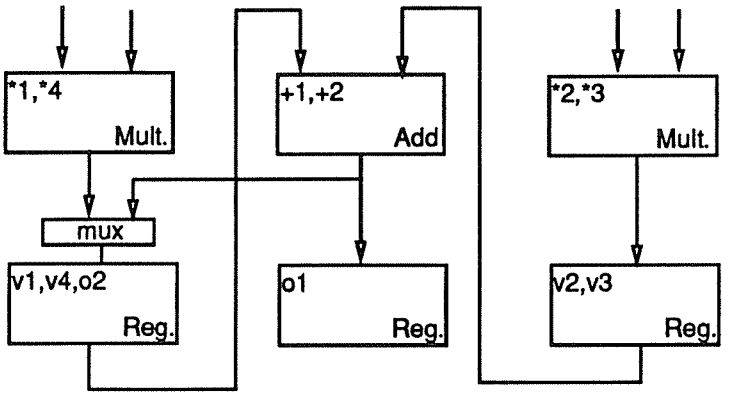
$$\begin{aligned} w_{1,3,1} &\geq x_{+1,1,1} + z_{+1,3,1} - 1 \\ w_{1,4,1} &\geq x_{+1,1,1} + z_{+1,4,1} - 1 \\ w_{2,3,1} &\geq x_{+1,2,1} + z_{+1,3,1} - 1 \\ w_{2,4,1} &\geq x_{+1,2,1} + z_{+1,4,1} - 1 \end{aligned}$$

Constraint 2.11 and objective function are easy to write.

To verify the formulation we synthesized several RTL designs. Two of these results are given in Figures 3.2 and 3.3. The schedule in Figure 3.3 is taken from [10]. The edges feeding from the input registers have not been shown for clarity. Both problems were formulated and solved using GAMS [2].

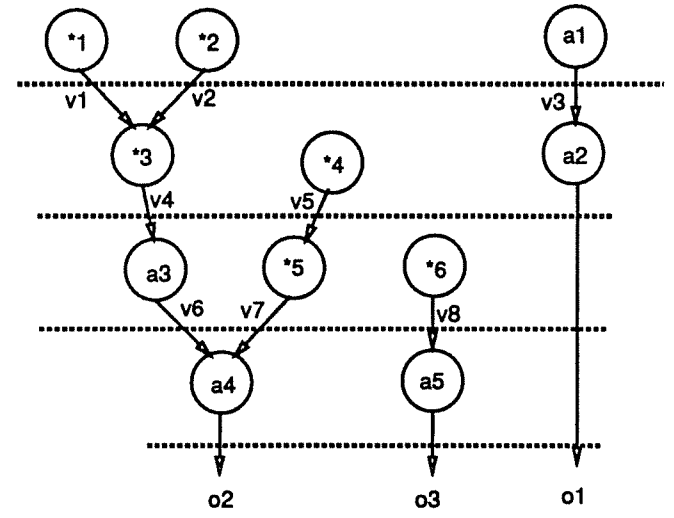


(a) Scheduled complex multiplication

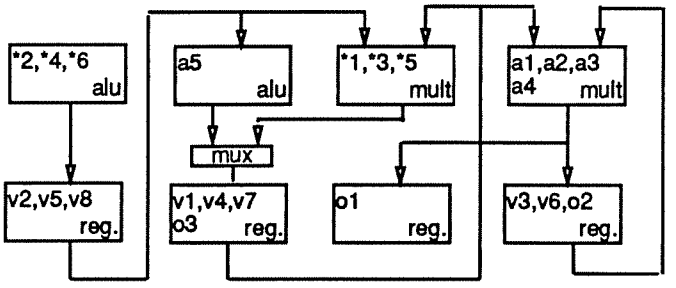


(b) Synthesized RTL design

Figure 3.2: Example 2: Complex multiplication



(a) Scheduled data flow graph



(b) Synthesized RTL design

Figure 3.3: Example 3

4 Area Tradeoffs

As we mentioned earlier, the final implementation is not constrained to have the scheduler specified quantities of modules and registers. Module and register quantities need not be specified at the start of the solution. We can perform area tradeoff between the interconnection, module and register area. The advantages of these tradeoffs are outlined in [9]. Thus, the input specification to the program consists of scheduling information and the cost (area) of each module type, register and interconnect.

For tradeoff studies we can assume a large supply of each resource type for binding. This would, however, result in a large number of binary decision variables, which in turn would result in large solution time. To compute a more realistic upper-bound on the resource quantities we first determine the lower-bound number of resources (modules and registers) from the scheduling information to which we add an increment based on the area of each resource. This increment is computed as follows. Let a_i be the area of module type i and a_{reg} be the area of a register. We then compute an upper-bound on the number of resources of each type $1 \leq j \leq k$ as follows:

$$n_j = \lfloor a_{max}/a_j \rfloor \, c + \text{lower-bound of module type j}$$

where $a_{max} = \max_{1 \leq i \leq k}(a_i)$ and $c \geq 0$ is an integer user settable parameter. $c = 0$ forces the program not to make any tradeoffs and to use the lower-bound quantities as determined by the scheduler for the final RTL design. A large value of c would indicate a large supply of resources to tradeoff. Practically, the value of c would range between one and three. Justification for such an estimate stems from the fact that if a multiplier costs five times as much as an adder, then we should tradeoff five adders with one multiplier. The upper-bound on register count is computed identically.

Define $u_j = 1$ if resource j is used in any time step and zero otherwise. $1 \leq j \leq n$ defines a module usage and $n+1 \leq j \leq n+m$ defines a register usage. For every time step $1 \leq t \leq T$ and for every operation $OP_i^{(t)}$ executed in time step t

$$u_j \geq x_{i,j,t} \tag{4.12}$$

Similarly for registers we have that for every time step $1 \leq t \leq T$ and for every value $V_i^{(t)}$ stored in time step t

$$u_{n+j} \geq y_{i,j,t} \tag{4.13}$$

The objective function then becomes:

$$minimize \sum_{i=1}^{m+n} \sum_{j=1}^{m+n} c_{i,j} w'_{i,j} + \sum_{s=1}^{k} a_s \left(\sum_{j=n_s+1}^{n_s} u_j \right) + \sum_{j=n+1}^{n+m} a_{reg} u_j \tag{4.14}$$

5 Conclusion

In this paper we have formulated the resource allocation and binding problem for non-pipelined designs. The formulation handles chaining, multi-cycle operations and makes tradeoffs between wiring area and module and register area. Currently we are working on adding conditional branching capability, optimization with commutative operations, and pipelined designs. We are also in the process of automating generation of constraints from scheduler information. The above formulation has large number of binary variables and may entail enormous computer runtime for optimal solution. Formulation of the problem is only the first step towards our goal. The next step is to develop heuristics (using methods such as Lagrangian relaxation) which will give near-optimal solutions quickly (in order of seconds for reasonable sized problems).

References

[1] F. Brewer and D. Gajski. Chippe: A System for Constraint Driven Behavioral Synthesis. *IEEE Transactions on Computer-Aided-Design*, 9(7), July 1990.

[2] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User Guide.* The Scientific Press, Redwood City, CA, 1988.

[3] R. Cloutier and D. Thomas. The Combination of Scheduling, Allocation and Mapping in a Single Algorithm. In *Proceedings of the 27th Design Automation Conference.* ACM/IEEE, June 1990.

[4] H. Crowder, E. L. Johnson, and M. Padberg. Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, 31(5), September-October 1983.

[5] L. Hafer and A. Parker. A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic. *IEEE Transactions on Computer-Aided-Design*, 2(1), January 1983.

[6] B. S. Haroun and M. I. Elmasry. Automatic Synthesis for DSP Silicon Compilers. *IEEE Transactions on Computer-Aided-Design*, 8(4), April 1989.

[7] W. R. Heller, C. G. Hsi, and W. F. Mikhail. Wireability - Designing Wiring Space for Chips and Chip Packages. *IEEE Design and Test*, 1(3), August 1984.

[8] R. Jain, K. Kucukcakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proceedings of the 26th Design Automation Conference.* ACM/IEEE, June 1989.

[9] K. Kucukcakar and A. C. Parker. Data Path Tradeoffs Using MABAL. In *Proceedings of the 27th Design Automation Conference.* ACM/IEEE, June 1990.

[10] J-H. Lee, Y-C. Hsu, and Y-L. Lin. A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis. In *Proceedings of the International Conference on Computer-Aided-Design.* ACM/IEEE, November 1989.

[11] M. C. McFarland. Allocating Registers, Processors, and Connections. Technical report, Department of Electrical Engineering, Carnegie-Mellon University, August 1981.

[12] M. C. McFarland, A. C. Parker, and R. Camposano. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2), February 1990.

[13] J. A. Nestor and D. E. Thomas. Behavioral Synthesis with Interfaces. In *Proceedings of the International Conference on Computer-Aided-Design.* IEEE/ACM, November 1986.

[14] B. M. Pangrle and D. D. Gajski. Design Tools for Intelligent Silicon Compilation. *IEEE Transactions on Computer-Aided-Design*, 6(6), November 1987.

[15] N. Park and F. J. Kurdahi. Module Assignment and Interconnect Sharing in Register-Transfer Synthesis of Pipelined Designs. In *Proceedings of the International Conference on Computer-Aided-Design.* ACM/IEEE, November 1989.

[16] A. C. Parker, J. Pizarro, and M. J. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23rd Design Automation Conference.* ACM/IEEE, June 1986.

[17] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided-Design*, 8(6), June 1989.

[18] D. E. Thomas, E. M. Dirkes, R. A. Walker, J. V. Rajan, J. A. Nestor, and R. L. Blackburn. The System Architect's Workbench. In *Proceedings of the 25th Design Automation Conference.* ACM/IEEE, June 1988.

[19] C. J. Tseng and D. P. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided-Design*, 5(3), July 1986.