A Formal Approach to the Scheduling Problem in High Level Synthesis

Cheng-Tsung Hwang, Jiahn-Hurng Lee, and Yu-Chin Hsu, *Member, IEEE*

***Abstract*—This paper presents an integer linear programming (ILP) model for the scheduling problem in high level synthesis. In addition to *time-constrained scheduling* and *resource-constrained scheduling*, a new scheduling problem called *feasible scheduling* is constructed, which provides a paradigm for exploring the solution space. Extensive consideration is given to the following applications:**

- **scheduling with:**
  - **chaining;**
  - **multicycle operations by nonpipelined function units;**
  - **multicycle operations by pipelined function units;**
- **functional pipelining;**
- **loop folding;**
- **mutually exclusive operations;**
- **scheduling under bus constraint;**
- **minimizing lifetimes of variables.**

The complexity of the number of variables in the formulation is $O(s \cdot n)$ where s and n are the number of control steps and operations, respectively. Since we use the as soon as possible (ASAP), as late as possible (ALAP), and list scheduling techniques to reduce the solution space, the formulation becomes very efficient. A solution to a practical problem, such as the fifth-order filter, can be found optimally in a few seconds.

I. Introduction

RECENTLY, automatic data-path synthesis of a digital system from a behavioral description has gained much attention in the CAD research community [1]–[25]. The synthesis task starts with a behavioral description of a digital system and a set of time and/or resource constraints. The goal is to produce a structure of the digital system that satisfies the constraints. It includes four subtasks. The first subtask is to describe the behavior of the digital system using a hardware description language (HDL). This step is usually followed by a translation of the description into a graph-based representation called the control data flow graph (CDFG). The next subtask is operation scheduling, where each operation in the CDFG is assigned to a control step. The third subtask allocates the resources for the digital system. Here, function units are assigned to execute the operations, storage units are assigned to store the values, and wires are allocated to interconnect them using the data transfer information derived from the CDFG. At this point, a data path is completed. Finally, based on the schedule graph and the data path, a control unit is synthesized to synchronize the executions of the operations.

Among the above steps, operation scheduling and hardware allocation are the two major subtasks. These two subtasks are interdependent. In order to have an optimal design, a system should perform both subtasks simultaneously [1], [2]. However, due to the time complexity, many systems perform them separately [4]–[13] or introduce iteration loops between the two subtasks [3], [14]–[16].

Roughly speaking, operation scheduling determines the cost-speed tradeoffs of the design. If the design is subject to a speed constraint, the scheduling algorithm will attempt to parallelize the operations to meet the timing constraint. Conversely, if there is a limit on the cost (area or resources), the scheduler will serialize operations to meet the resource constraint. Once the operations are scheduled, the number and types of function units, the lifetimes of variables, and the timing constraints are fixed. Thus a good scheduler is very important to an automated data-path synthesis system [3], [12], [14], [15], [17]–[20]. According to Gajski [17], it is "perhaps the most important step during the structure synthesis."

We address in this paper three scheduling problems, each with different requirements.

(P1) Time-Constrained Scheduling: Given constraints on the maximum number of time steps, find the cheapest schedule which satisfies the constraints.

(P2) Resource-Constrained Scheduling: Given constraints on the resources, find the fastest scheduling which satisfies the constraints.

(P3) Feasible-Constrained Scheduling: Given constraints on the resources and the time steps, decide if there exists a schedule which satisfies the constraints. Output the solution if it exists.

Instead of giving heuristic algorithms to schedule the operations of the CDFG as most systems do, we begin with a mathematic description of the scheduling objectives and constraints which can be translated easily into integer linear programming (ILP) formulations.

We also extend the formulations to various requirements which are encountered in the real world. The complexity of the number of variables in our formulation is $O(s \cdot n)$ where s is the number of control steps and n is the number of operations. While formulating the equations, we try to reduce the solution space as much as possible. Experiments show that optimal solutions for a practical example such as the fifth-order filter can be obtained in a very short time.

This paper is organized as follows: Section II reviews previous work and related research. Section III gives the approaches and formulations of the three scheduling problems. Various extensions are introduced in Section IV. Section V shows the experimental results. Finally, concluding remarks are made in Section VI.

Manuscript received April 7, 1989; revised January 19, 1990. This work was supported in part by the National Science Council, Republic of China, under Grants NSC78-0404-E007-13 and NSC79-0404-E007-24. This paper was recommended by Associate Editor R. K. Brayton.

The authors are with the Department of Computer Science, Tsing Hua University, Hsin-Chu, Taiwan 30043, Republic of China.

IEEE Log Number 9042077.

0278-0070/91/0400-0464$01.00 © 1991 IEEE

II. Previous Work and Related Research

Many systems with different scheduling techniques have been reported. McFarland *et al.* [2] give a good tutorial on the high level synthesis problem, where they show how the synthesis task can be decomposed into a number of distinct but not independent subtasks and give a survey on the techniques for solving these subtasks. A recent survey of the synthesis task is by Paulin [3], where the concentration is on scheduling techniques.

In this section, some of the basic scheduling techniques are discussed. We also address some considerations which are peculiar to scheduling as a part of data-path synthesis. The integer programming technique, which is the technique used in this paper, is discussed in the last subsection.

2.1. Basic Scheduling Techniques

The simplest scheduling technique is *as soon as possible* (ASAP) scheduling [4], [5] where the operations in the CDFG are scheduled step by step from the first control step to the last. An operation is called *ready operation* if all of its predecessors are scheduled. This procedure repeatedly schedules ready operations to the next control step until all the operations are scheduled.

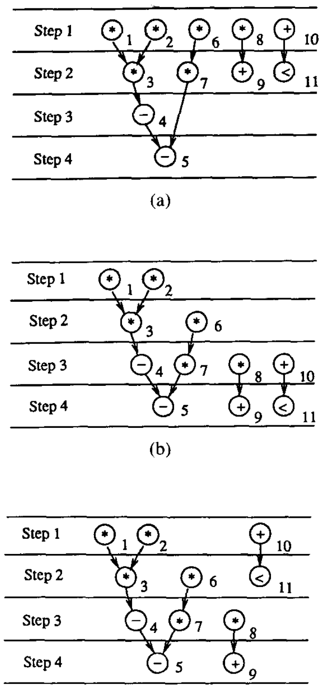
As late as possible (ALAP) scheduling [8] performs a very similar procedure as ASAP. In contrast to ASAP, ALAP scheduling schedules the operations from the last control step toward the first. An operation is scheduled to the next control step as all its successors are scheduled. Fig. 1 gives an example of ASAP and ALAP scheduling.

Since it is not practical to assign too many operations of the same type into a control step due to the constraint on the number of function units, a variation of ASAP is to delay the ready operations when their number exceeds the number of function units. Selection of the operations to be delayed is arbitrary. This technique is called *ASAP with conditional post-ponement* [6]–[8].

The *list scheduling* technique [9]–[13], which was originally used in microcode compaction [9], has been adopted by many high level synthesis systems. Similar to ASAP, the operations in the CDFG are assigned to control steps from the first control step to the last. The ready operations are given a priority according to heuristic rules and are scheduled into the next control step according to this predefined priority. When the number of scheduled operations exceeds the number of resources, the remaining operations are delayed.

The third type of scheduling is "global" in the way it selects the next operation to be scheduled and in the way it decides the control step in which to put it. There are two variations—freedom-based scheduling and force-directed scheduling. In *freedom-based scheduling* [12], [14], [15], the operations on the critical path are scheduled first. The operations not on the critical path are assigned one at a time according to their degree of freedom. In*force-directed scheduling* [3], [21], "force" values are calculated for all operations at all feasible control steps. The pairing of operation and control step that has the most attractive force is selected and assigned. After the assignment, the forces of the unscheduled operations are re-evaluated. Assignment and evaluation are iterated until all the operations are assigned.

Among the above scheduling techniques, list scheduling requires that the number of function units be specified, while force-directed scheduling requires that the maximum number of control steps be specified. They correspond to resource-constrained and time-constrained scheduling, respectively.



(a)

(b)

(c)

Fig. 1. (a) ASAP scheduling. (b) ALAP scheduling. (c) Result.

2.2 Considerations in Data-Path Synthesis

In the real world, many variations on how the operations are implemented and the different structures of the data flow graph have to be considered during the scheduling phase. We can join several operations with data dependencies in one cycle (chaining) [3], [4], [12], [15] or execute an operation which crosses more than one cycle (multicycle operation). A multicycle operation can be performed by either a pipelined [12] or nonpipelined function unit.

The data flow graph may contain mutually exclusive operations. Mutually exclusive operations occur when there are multiple branches in the CDFG and only one branch occurs at a time. In this case, we want to find a schedule in which the resources are efficiently utilized.

With very little additional hardware, the throughput of a data can be improved by pipelining the path [14], [22]. A pipelined data path (also called functional pipelining) is a data path in which operations of different instances can be performed concurrently. Sehwa [22] was the first system for synthesizing a pipelined data path. It uses a modified list scheduling technique to schedule the operations. A graph partitioning technique for scheduling a pipelined data path was presented in [23]. The force-directed scheduling algorithm has also been adapted to solve the same problem in [3], [21].

Spaid [25] incorporates structure transformations and retiming to modify the CDFG in order to reduce the critical path. After reducing the critical path of the CDFG, a greedy scheduling technique is used to schedule the operations. A better result can be obtained if retiming and scheduling are performed at the same time.

The idea of loop folding was proposed in [24]. The goal of loop folding is to reduce the running time of a loop by overlapping the execution of different loop iterations. In that paper, operations in one loop iteration are folded to the next loop iteration iteratively until the loop length is unlikely to be reduced. The list scheduling technique is used to schedule the overlapped loop iterations.

2.3. *ILP Approaches*

Since the above scheduling methods assign operations to control steps one at a time, their results depend strongly on the order of the assignments. We state the scheduling problem by a mathematic description and then solve it using an ILP method. Although, strictly speaking, it is not new, it is the first one with a realistic approach. In the following paragraphs, we survey some similar approaches which were used to describe or solve the synthesis of digital logic systems.

An integer programming model for synthesizing a digital logic at the register-transfer level (RTL) was formulated in [1]. The model gives detailed specifications for a data-path synthesis. All of the characteristics such as variable storage, operation precedence, resource sharing, and control structures are included in this model. Due to the complexity of the formulation, only a small problem can be solved.

An integer programming approach was proposed for microcode scheduling in CATHEDRAL-II [13], which is a synthesis system for multiprocessor DSP systems. After a customized data path has been synthesized and the high level operations are mapped onto a set of RTL operations, the microcode scheduling is performed. The model contains data precedence, resource conflict, and controller pipelining constraints. Since excessive CPU time is required to solve large problems, the model is replaced by a graph-based heuristic scheduling algorithm.

After extensively studying these two papers, we have found there are places where we can reduce the solution space of the scheduling problem. First, a formulation with linear cost function can be solved much more easily than that of a nonlinear cost function. By carefully arranging the data dependency relationships in the formulation, we found it is possible to formulate the cost function as linear. Second, the search space for each operation can be reduced by restricting the range of control steps for each operation. This can be done by using both ASAP and ALAP scheduling. Third, by introducing a lower limit and an upper limit on the number of function units of each type, we can prevent many unnecessary searches.

By taking these considerations into our formulation and using an ILP package, optimal solutions for a practical sized problem like the fifth-order filter can be achieved within a few seconds. Moreover, we believe that the detailed formulations of the scheduling problem will lead to a deeper understanding of the problem. Better algorithms or heuristics for near optimal solutions can be expected.

III. ILP Formulations for the Scheduling Problems

In this section, we will present ILP formulations for time-constrained scheduling, resource-constrained scheduling, and feasible scheduling problems. For simplicity of explanation, two assumptions are made.

1) Each operation is assumed to have a one-cycle propagation delay.

2) Only nonpipelined data paths are considered.

Other general considerations will be discussed in the next section.

ASAP, ALAP, and list scheduling are used to trim the solution space in our formulation. ASAP and ALAP determine, respectively, the earliest possible time and latest possible time of an operation. List scheduling sets an upper limit on the number of control steps for resource-constrained scheduling.

The notations used in our formulations are defined as follows: suppose the data flow graph, $G(V, E)$, contains n $(|V|)$ operations, e $(|E|)$ data dependencies, and is going to be scheduled into s steps. Each of the operations is labeled as o_i, where $1 \le i \le n$. A precedence relation between two operations o_i and o_j is denoted by $o_i \rightarrow o_j$, where o_i is the immediate predecessor of o_j. The earliest possible time (ASAP) and the latest possible time (ALAP) of o_i are S_i and L_i, respectively. The cost of a function unit of type t_k (FU_{t_k}) is c_{t_k}, and there are m types of function units available. A relation between an operation o_i and a function unit FU_{t_k} is denoted by $o_i \in FU_{t_k}$, if FU_{t_k} can perform the function of o_i.

3.1. *Time-Constrained Scheduling*

A time-constrained scheduling problem can be defined as follows. *Given the maximum number of control steps, find a minimal cost schedule that satisfies the given set of constraints.* Here the cost of a data path may be the costs of function units, interconnections, and registers. For simplicity of the formulation, only the cost of function units is considered. The others are considered in the next section.

It is obvious that the cost of function units is minimized if all the function units are *fully utilized* in a system. In other words, operations of the same type should be evenly distributed among all control steps. This is achieved in our model by *minimizing the maximal number* of operations of the same type in each control step.

Our approach to time-constrained scheduling includes three substeps:

1) ASAP: determine the earliest possible time for each operation;
2) ALAP: determine the latest possible time for each operation;
3) ILP: minimize the cost of resources.

The variables used in the formulation are the following.

1) M_{t_k} are integer variables which denote the number of function units of type t_k needed.

2) $x_{i,j}$ are 0–1 integer variables associated with o_i. $x_{i,j} = 1$ if o_i is scheduled into step j; otherwise, $x_{i,j} = 0$.

Now the problem can be formulated as minimizing

$$\sum_{k=1}^{m} (c_{t_k} * M_{t_k}) \tag{1}$$

subject to

$$\sum_{o_i \in FU_{t_k}} x_{i,j} - M_{t_k} \le 0, \qquad \text{for } 1 \le j \le s,\ 1 \le k \le m; \tag{2}$$

$$\sum_{j=S_i}^{L_i} x_{i,j} = 1, \qquad \text{for } 1 \le i \le n; \tag{3}$$

$$\sum_{j=S_i}^{L_i} (j * x_{i,j}) - \sum_{j=S_k}^{L_k} (j * x_{k,j}) \le -1, \qquad \text{for all } o_i \rightarrow o_k. \tag{4}$$

TABLE I
VARIABLES DISTRIBUTION OF THE SCHEDULING EXAMPLE

FU Type	Multiplier (*)						ALU (+, −, <)				
Operation	o_1	o_2	o_3	o_6	o_7	o_8	o_4	o_5	o_9	o_{10}	o_{11}
Step 1	$x_{1,1}$	$x_{2,1}$		$x_{6,1}$		$x_{8,1}$				$x_{10,1}$	
Step 2			$x_{3,2}$	$x_{6,2}$	$x_{7,2}$	$x_{8,2}$			$x_{9,2}$	$x_{10,2}$	$x_{11,2}$
Step 3					$x_{7,3}$	$x_{8,3}$	$x_{4,3}$		$x_{9,3}$	$x_{10,3}$	$x_{11,3}$
Step 4								$x_{5,4}$	$x_{9,4}$		$x_{11,4}$

The objective function in (1) states that we are going to minimize the total cost of function units. Constraint (2) states that no schedule should have a control step containing more than M_{t_k} function units of type t_k. It is clear that o_i can only be scheduled into a step between S_i and L_i, which is reflected in (3). Constraint (4) ensures that the precedence relations of the data flow graph (DFG) will be preserved. Let us illustrate the above formulation using the example below.

Consider the data flow graph in Fig. 1, which is going to be scheduled into four control steps. The ASAP and ALAP schedules are shown in Fig. 1(a) and (b), respectively. The distribution of variables is shown in Table I. Here, the horizontal rows represent the control steps, and each column represents an operation. A variable in the table means that the operation could be assigned to the step. The available function units are multipliers (FU_{t_1}) and ALU's (FU_{t_2}) which are capable of performing addition, subtraction, and comparison. A multiplier costs 5 ($c_{t_1} = 5$) while an ALU costs 1 ($c_{t_2} = 1$). The following are the integer programming formulations.

Minimize $5 * M_{t_1} + M_{t_2}$ subject to

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} - M_{t_1} \leq 0;$$

$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} - M_{t_1} \leq 0;$$

$$x_{7,3} + x_{8,3} - M_{t_1} \leq 0;$$

$$x_{10,1} - M_{t_2} \leq 0;$$

$$x_{9,2} + x_{10,2} + x_{11,2} - M_{t_2} \leq 0;$$

$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} - M_{t_2} \leq 0;$$

$$x_{5,4} + x_{9,4} + x_{11,4} - M_{t_2} \leq 0;$$

$$x_{1,1} = 1;$$

$$x_{2,1} = 1;$$

$$x_{3,2} = 1;$$

$$x_{4,3} = 1;$$

$$x_{5,4} = 1;$$

$$x_{6,1} + x_{6,2} = 1;$$

$$x_{7,2} + x_{7,3} = 1;$$

$$x_{8,1} + x_{8,2} + x_{8,3} = 1;$$

$$x_{9,2} + x_{9,3} + x_{9,4} = 1;$$

$$x_{10,1} + x_{10,2} + x_{10,3} = 1;$$

$$x_{11,2} + x_{11,3} + x_{11,4} = 1;$$

$$x_{6,1} + 2x_{6,2} - 2x_{7,2} - 3x_{7,3} \leq -1;$$

$$x_{8,1} + 2x_{8,2} + 3x_{8,3} - 2x_{9,2} - 3x_{9,3} - 4x_{9,4} \leq -1;$$

$$x_{10,1} + 2x_{10,2} + 3x_{10,3} - 2x_{11,2} - 3x_{11,3} - 4x_{11,4} \leq -1.$$

The solution obtained by this formulation is optimal when the variables, $x_{1,1}$, $x_{2,1}$, $x_{3,2}$, $x_{4,3}$, $x_{5,4}$, $x_{6,2}$, $x_{7,3}$, $x_{8,3}$, $x_{9,4}$, $x_{10,1}$, and $x_{11,2}$ are set to 1. In this case, two multipliers and two ALU's are used. The scheduling result is shown in Fig. 1(c).

3.2. Resource-Constrained Scheduling

A resource-constrained scheduling problem can be formally stated as follows: *given the maximum number of resources, find the fastest schedule that satisfies the given set of constraints.* In general, the resources given are the number of function units, such as adders, multipliers, ALU's, and buses. Although registers and interconnections also contribute to the total area, they are difficult to specify as resource constraints.

Resource-constrained scheduling includes four substeps:

1) list scheduling: determine the upper limit on the number of control steps;
2) modified ASAP: determine the earliest possible time for each operation;
3) modified ALAP: determine the latest possible time for each operation;
4) ILP: minimize the number of control steps needed for the data path.

ASAP and ALAP scheduling can be modified so that a tighter range for each operation is obtained by taking the resource constraints into account. Assume there are p_i operations which are executed by the same type of function unit as o_i and proceed to o_i, and the available number of function units for o_i is n_i. Then, o_i cannot be scheduled before step $\lceil p_i/n_i \rceil$. It also follows that any successor of o_i cannot be scheduled before $\lceil p_i/n_i \rceil + 1$. Modified ASAP and ALAP are particularly useful when the number of function units or buses is small.

The variables used in the formulation are the following.

1) C_{step} is an integer variable which is the total number of control steps required.
2) $x_{i,j}$ are 0-1 integer variables associated with o_i, $x_{i,j} = 1$, if o_i is scheduled into step j; otherwise, $x_{i,j} = 0$.

A resource-constrained scheduling problem is formulated as minimizing

$$C_{\text{step}} \tag{1.1}$$

subject to

$$\sum_{o_i \in FU_{t_k}} x_{i,j} \leq M_{t_k}, \quad \text{for } 1 \leq j \leq s,\ 1 \leq k \leq m \tag{2.1}$$

$$\sum_{j=S_i}^{L_i} x_{i,j} = 1, \qquad \text{for } 1 \leq i \leq n; \tag{3}$$

$$\sum_{j=S_i}^{L_i} (j * x_{i,j}) - \sum_{j=S_k}^{L_k} (j * x_{k,j}) \leq -1, \qquad \text{for all } o_i \rightarrow o_k \tag{4}$$

$$\sum_{j=S_i}^{L_i} (j * x_{i,j}) - C_{\text{step}} \leq 0, \qquad \text{for all } o_i \text{ without successors.} \tag{5}$$

The objective function in (1.1) states that we are going to minimize the total number of control steps. Constraint (2.1) states that no schedule should have a control step containing more than M_{t_k} function units of type t_k. Note that the M_{t_k} in (2.1) is a constant. Constraints (3) and (4) are the same as those in time-constrained scheduling. No operations should be scheduled after C_{step}, as described in constraint (5).

Once again, we use the data flow in Fig. 1 to demonstrate this formulation. Assume the maximum number of multipliers and ALU's are all set at 2. Under these constraints, list scheduling is first used to decide the upper limit on the number of time steps (which is 4). Applying this limit, we then use ASAP and ALAP scheduling to obtain the range for each operation. Our formulation is to minimize C_{step} under certain constraints. Since the constraints for (3) and (4) are the same as in the previous example, we list only the constraints (2.1) and (5):

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \leq 2;$$

$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \leq 2;$$

$$x_{7,3} + x_{8,3} \leq 2;$$

$$x_{10,1} \leq 2;$$

$$x_{9,2} + x_{10,2} + x_{11,2} \leq 2;$$

$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \leq 2;$$

$$x_{5,4} + x_{9,4} + x_{11,4} \leq 2;$$

$$4x_{5,4} - C_{\text{step}} \leq 0;$$

$$2x_{9,2} + 3x_{9,3} + 4x_{9,4} - C_{\text{step}} \leq 0; \text{ and}$$

$$2x_{11,2} + 3x_{11,3} - 4x_{11,4} - C_{\text{step}} \leq 0.$$

This formulation will obtain the same result as previous example. At the mean time, the minimum number of control steps ($C_{\text{step}} = 4$) is obtained.

3.3. *Feasible Scheduling*

In this subsection, we combine the previous two formulations into a third scheduling problem. This problem does not ask for an optimum but asks whether a feasible solution exists. The formal definition of a feasible scheduling problem is as follows: *given a fixed amount of resources and a specified number of time steps, decide if there is a schedule which satisfies all the constraints. Output the solution if it exists.*

The formulation for the problem includes no objective function but does have a set of constraints:

$$\sum_{o_i \in FU_{t_k}} x_{i,j} \leq M_{t_k}, \qquad \text{for } 1 \leq j \leq s,\ 1 \leq k \leq m; \tag{2.1}$$

$$\sum_{j=S_i}^{L_i} x_{i,j} = 1, \qquad \text{for } 1 \leq i \leq n; \tag{3}$$

$$\sum_{j=S_i}^{L_i} (j * x_{i,j}) - \sum_{j=S_k}^{L_k} (j * x_{k,j}) \leq -1, \qquad \text{for all } o_i \rightarrow o_k. \tag{4}$$

By solving feasible scheduling problems, the solution for previous scheduling problems can be constructed. The advantages of using this approach are the following.

1) The formulation is a 0-1 ILP problem, and good heuristics exist to solve this type of problem. Also, the time required to find a solution by feasible scheduling is a lot less than by optimizing scheduling since we only need to search part of the solution space.

2) The number of function units and the number of time steps can be estimated by other fast heuristics.

3) Since we are, in fact, deciding a set of values for all variables in solving an ILP formulation, the time complexity is increased with the number of variables in the formulation. From this point of view, the range for each operation is smaller, due to the constraints on the number of function units and the time. This corresponds to a smaller solution space.

4) This approach allows a user or an expert system to control the speed-time tradeoff. Thus we can generate a set of optimal solutions and leave the selection of the best time/area implementation to the user.

Based on the above arguments, feasible scheduling seems to provide a general paradigm for solving a scheduling problem. Therefore, in the following section, we will not specify the kind of scheduling problem unless it is necessary.

3.4. *Complexity of the Scheduling Problem*

The complexity of feasible scheduling is analyzed in terms of the number of variables and equations. In feasible scheduling, the number of resources and the number of control steps have been fixed. Thus the only unknowns are 0-1 variables $x_{i,j}$. The exact number of $x_{i,j}$ is $\Sigma_{i=1}^{n} (L_i - S_i + 1)$ which is bounded by $s \cdot n$. Note that, due to constraint (3), only n variables will have a value of 1. Thus once a $x_{i,j}$ is decided to be 1, the remaining $L_i - S_i$ variables are implicitly set at 0. Therefore, the problem is easier to solve than it might appear.

The number of equations required for constraints (2.1), (3), and (4) is $(s \cdot m)$, n and e respectively, where e is the number of edges in the DFG.

In all, the number of variables in our formulation grows as $O(s \cdot n)$, and the number of equations is as $O(s \cdot m + n + e)$.

IV. Generalizations and Complexity Analysis

For practical applications, we have generalized the ILP formulations along with the complexity in the number of equations and variables to the following variations:

1) scheduling with:
   a) chaining,
   b) multicycle operations by nonpipelined function units,
   c) multicycle operations by pipelined function units,
2) functional pipelining;
3) loop folding,

4) mutually exclusive operations;
5) scheduling under bus constraint,
6) minimizing lifetimes of variables.

For the sake of notational convenience, we define a new integer variable—time variable (T_i)—which is the control step by which operation o_i is scheduled. It is easy to verify that T_i is equal to $\Sigma_{j=S_i}^{L_i} (j \cdot x_{i,j})$. By using this notation, constraint (4) is simplified to $T_i - T_j \leq -1$.

4.1. Chaining and Multicycle Operations

Fig. 2 shows a DFG where a multiplication ($*$) requires 100 ns and an additional (+) requires 40 ns. Let the cycle time be 120 ns. For a regular design, the graph must be scheduled into two control steps as shown in Fig. 2(a). By chaining two additions in one cycle, the graph can be scheduled in one 120-ns cycle (Fig. 2(b)).

If we reduce the clock cycle time to 60 ns, then a multiplication has to cross two cycles. In this cse, the multiplication is a *multicycle operation*. In Fig. 2(c), we need only one adder, one multiplier, and two 60-ns cycles to implement the graph. The multiplication can be executed by either a pipelined function unit or a nonpipelined function unit. The first and third examples need 2 control words, and the second needs only one. The time, resources, and control words required by the three examples are summarized in Fig. 2(d).

4.1.1) Chaining: In some cases, we can chain several operations in one cycle if their total running time is less than the cycle time. We call it scheduling with chaining. To formulate a chaining problem, we define a new precedence relation, $\Rightarrow$, between two operations. There exists a relation $\Rightarrow$ between o_i and o_j, denoted by $o_i \Rightarrow o_j$, if o_j is the nearest successor of o_i such that the running time from o_i to the end of o_j is greater than the cycle time. For example, assume all the operations in the DFG of Fig. 3 require 40 ns and the cycle time is 100 ns. We have the following relations: $o_1 \Rightarrow o_3$, $o_1 \Rightarrow o_6$, $o_2 \Rightarrow o_4$, $o_2 \Rightarrow o_7$, and $o_3 \Rightarrow o_5$.

To perform a scheduling with chaining, constraint (4) is modified as

$$T_i - T_j \leq 0, \qquad \text{for } o_i \rightarrow o_j \tag{4.1}$$

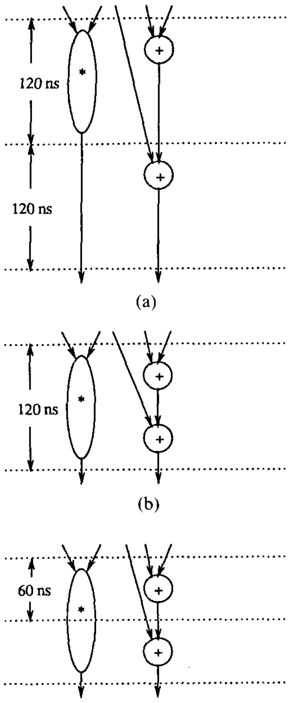
and additional constraints are included in the formulation

$$T_i - T_j \leq 1, \qquad \text{for } o_i \Rightarrow o_j. \tag{6}$$

Constraint (4.1) states that if o_i immediately precedes o_j in the DFG, then o_i should be scheduled before or at the same step as o_j. Constraint (6) states if $o_i \Rightarrow o_j$, then o_i should be scheduled before o_j.

a) Complexity analysis: Suppose that, at most, c operations are chained into a control step and the number of fan ins for an operation is k. Given an operation o_j, there are, at most, k^c-operations which satisfy the relation $o_i \Rightarrow o_j$. Thus the total number of equations introduced by (6) is determined by $k^c \cdot n$. In practical examples, k is smaller than 2 and c is a small number, so the constraint has $O(n)$ complexity.

4.1.2) Multicycle Operations with Nonpipelined Implementation: A multicycle operation can be performed either by a nonpipelined or a pipelined function unit. The difference between them is the existence of latches between the cycles. For nonpipelined implementation, once the operation is assigned, the function unit cannot be shared by other operations until the operation is completed.



(a)

(b)

(c)

result type	time	+	*	control
Regular	240 ns	1	1	2
Chaining	120 ns	2	1	1
Multicycling	120 ns	1	1	2

(d)

Fig. 2. (a) Regular clock cycle. (b) Chaining. (c) Multicycling. (d) Comparison.

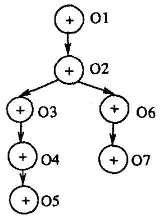


Fig. 3. Example of chaining.

Let $Y_{k,j}$ be the number of *initiations* of type t_k operations at step j, i.e., $Y_{k,j} = \Sigma_{o_i \in FU_{t_k}} x_{i,j}$. For a set of operations executed by a nonpipelined function unit (i.e., $o_i \in FU_{t_k}$) with propagation delay d_k, the number of function units used in control step j includes all the initiations between step $j - d_k + 1$ and step j. In other words, the number of function units used at step j is $\Sigma_{p=0}^{d_k-1} Y_{k,j-p}$. Therefore, constraint (2) is modified as

$$\sum_{p=0}^{d_k-1} \sum_{o_i \in FU_{t_k}} x_{i,j-p} \leq M_{t_k}, \qquad \text{for } 1 \leq j \leq s,\ 1 \leq k \leq m \tag{2.2}$$

and constraints (4) and (5) are changed to

$$T_i - T_j \leq -d_{o_i}, \qquad \text{for } o_i \rightarrow o_j, \text{ and} \tag{4.2}$$

$$T_i - C_{\text{step}} \leq 1 - d_{o_i}, \qquad \text{for all } o_i \text{ without successors} \tag{5.1}$$

where d_{o_i} is the delay time of o_i ($d_{o_i} = d_k$).

Constraint (2.2) states that the number of scheduled type t_k operations between control step $j - d_k + 1$ and step j shall not exceed M_{t_k}. Constraint (4.2) ensures the data dependencies. Constraint (5.1) requires that the initiations of o_i be $d_{o_i} - 1$ cycles earlier than the total execution time.

a) Complexity analysis: Since there is a one to one correspondence between the new constraints and the original, the complexity is the same as that of the basic scheduling problem. The complexities of the number of variables and equations are $O(s \cdot n)$ and $O(s \cdot m + n + e)$, respectively.

4.1.3) Multicycle Operations with Pipelined Implementation: For a pipelined functional unit, new input data can be initiated while previous data are still computed in the function unit. The time between two successive initiations is called *latency*. A pipelined function unit with a delay d and a fixed latency l can perform a new operation every l cycles, where $l \leq d$.

Before the derivation of the formulation, we begin with an example in which 12 operations are scheduled into 8 steps (Fig. 4(a)). Suppose that the 12 operations are to be performed by the same type function unit with $d = 8$ and $l = 3$. Since a pipelined function unit can be shared by the operations of any two steps s_i, s_j, where $|s_i - s_j|$ is an integer multiple of l, we can group the 8 control steps into 3 clusters, $c_1 = \{s_8, s_5, s_2\}$, $c_2 = \{s_7, s_4, s_1\}$ and $c_3 = \{s_6, s_3\}$. Here, a function unit can be shared by the operations in different steps within a cluster but cannot be shared by those in different clusters. Therefore, the number of function units needed is the total of the function units required by the three clusters; the number of function units required by a cluster is the maximum number of function units of the steps within this cluster. For the example in Fig. 4(a), the number of function units used at s_8 equals max $(3, 0, 1)$ + max $(0, 3, 1)$ + max $(3, 1) = 9$. A better schedule (Fig. 4(b)) needs only 4 function units (max $(3, 3, 3)$ + max $(1, 1, 1)$ + max $(0, 0) = 4$).

Therefore, for a pipelined function unit of type t_k, with a propagation delay d_k and a fixed latency l_k, constraint (2) is modified as

$$\sum_{p=0}^{l_k - 1} \max_{p'=0}^{\lfloor (d_k - 1 - p)/l_k \rfloor} \left(\sum_{o_i \in FU_{t_k}} x_{i,j-p'l_k} - p \right) \leq M_{t_k},$$
$$\text{for } 1 \leq j \leq s,\ 1 \leq k \leq m. \tag{2.3}$$

The number of function units required by all the clusters is determined by the outer summation, while the maximum function and the inner summation determine the function units required by each cluster.

Since a constraint with a maximum function is not linear, it can be replaced by the following constraints:

$$\sum_{o_i \in FU_{t_k}} x_{i,j-p'l_k} - Z_{k,j} \leq 0,$$
$$\text{for } 1 \leq j \leq s,\ 1 \leq k \leq m,\ 0 \leq p' \leq \left\lfloor \frac{d_k}{l_k} \right\rfloor \tag{2.3a}$$

and

$$\sum_{p=0}^{l_k - 1} Z_{k,j-p} \leq M_{t_k}, \qquad \text{for } 1 \leq j \leq s,\ 1 \leq k \leq m. \tag{2.3b}$$

1	O1
2	O2
3	O3
4	O4 O5 O6
5	
6	O7 O8 O9
7	
8	O10 O11 O12

(a)

1	O1
2	O2 O3 O4
3	
4	O5
5	O6 O7 O8
6	
7	O9
8	O10 O11 O12

(b)

Fig. 4. (a) A bad schedule. (b) A better schedule.

Constraint (2.3a) gives the maximum number of function units required by each cluster, while constraint (2.3b) asserts that the total of all the function units used at nonsharable control steps will not exceed M_{t_k}.

a) Complexity analysis: In (2.3a) we have introduced an integer variable ($Z_{j,k}$) and a group of equations for each control step and each type of pipelined function unit. Suppose there are m' among the m types of function units which are pipelined, and the delay time and latency for FU_{t_k} are d_k and l_k, respectively. Constraint (2.3a) requires $s \cdot m'$ integer variables and $s \cdot \Sigma_{k=1}^{m'} (d_k/l_k)$ equations; both are bounded by $O(s \cdot m)$.

4.2. Functional Pipelining (Pipelined Data Path)

A pipelined data path allows the execution of multiple tasks concurrently. Two consecutive tasks can be initiated with a certain interval, which is called the *latency* of the pipelined data path.

For a given latency l, the operations in control steps $j + pl$ ($p = 0, 1, 2, \cdots$) are executed simultaneously and cannot share the same function units. Consequently, constraint (2) is modified as

$$\sum_{p=0}^{\lfloor (s-j)/l \rfloor} \sum_{o_i \in FU_{t_k}} x_{i,j+pl} \leq M_{t_k}, \qquad \text{for } 1 \leq j \leq l,\ 1 \leq k \leq m \tag{2.4}$$

In [22], a theorem for pipelined data path is available, stated as the following.

Theorem 1: Given a DFG, the necessary and sufficient number of function units of each type (M_{t_k}) to realize a pipelined data path with a fixed latency l is $\lceil N_{t_k}/l \rceil$, where N_{t_k} is the maximum number of operations which must be performed by type t_k function units during a single iteration. □

We can state the theorem in other words, omitting the proof.

Theorem 2: Given a DFG and the number of function units available for each type (M_{t_k}), we can realize an optimal pipelined data path with latency $l = \max_{k=1}^{m} \lceil N_{t_k}/M_{t_k} \rceil$. □

The above theorems are very valuable for exploring the solution space. Suppose we are going to generate a table of optimal implementations. We can start with a minimum latency $l = 1$ and work towards a large one. For each latency l, we calculate the minimum number of resources for each type (M_{t_k}) according to theorem 1. If the number of resources for latency l is equal to those for latency $l - 1$, the solution for latency $l - 1$ is also the optimal solution for latency l. If the delay time is important in the design, the latency l and the number of resources M_{t_k} can be used to generate an ILP formulation which aims at minimizing the delay time. Thus we can obtain a set of implementations that are all optimal in terms of latency, number of resources, and delay time.

a) Complexity analysis: The number of equations required for constraint (2.4) is $l \cdot m$, which is less than that in constraint (2) of the basic scheduling problem ($= s \cdot m$), while the number of variables does not change in the formulation.

4.3. Loop Folding

The concept of loop folding is very similar to that of functional pipelining. The difference is that in loop folding, there are data dependencies between loop iterations, while in function pipelining, there is no data dependency between different instances. Thus the latency of a pipelined data path can be arbitrarily small, provided that the resources are unlimited. In the case of loop folding, the latency (or loop length [24]) depends on the number of resources given and the structure of DFG.

In the example of Fig. 5(a), suppose there exists a 1-*deg* [24] data dependency between o_i and o_j. In other words, the value generated by o_i will be used by o_j in the next iteration. If there is a path of length L from o_j to o_i in the DFG (Fig. 5(b)), the lower bound of the loop length would be $L + 1$, i.e., it is impossible to fold the loop into a schedule with a loop length less than $L + 1$.

Let $o_i \xrightarrow{\deg = d} o_j$ denote a d-deg [24] data dependency between o_i ($o_i \in FU_{t_k}$) and o_j and T_j^d be the time where o_j is executed at d iterations later. Suppose the loop length after folding is known to be l; we have $T_j^d = T_j + d \cdot l$. Therefore, a new constraint

$$T_i - T_j^d \le -d_{o_i}, \qquad \text{for all } o_i \xrightarrow{\deg = d} o_j \qquad (7)$$

or equivalently

$$T_i - T_j \le d \cdot l - d_{o_i}, \qquad \text{for all } o_i \xrightarrow{\deg = d} o_j \qquad (8)$$

is introduced into the previous formulations to enforce the data dependency between different loop iterations. The remaining constraints are the same as those for functional pipelining.

A loop folding problem is closely related to a retiming problem in synchronous circuits [26]. Retiming relocates the positions of the separating registers in the CDFG to obtain a shorter critical path, and hence, higher throughput. Algorithms [26] for retiming have been proposed to optimize synchronous circuits. In [25], after retiming on the CDFG, scheduling is performed on the modified DFG. The separation of retiming and scheduling will produce suboptimal design. Our formulation for loop folding performs both retiming and scheduling at the same time.

Complexity analysis: For a single ILP formulation, the number of equations added is equal to the number of d-deg data dependencies, which has worst-case complexity $O(e)$. There are no additional variables.

4.4. Mutually Exclusive Operations

As in the case of structured programming, the relationships among a set of operations, $\boldsymbol{O}$, can be represented as a tree where the internal nodes are of two types, XOR and AND, and the leaves are the operations. Let a node have n subtrees and the number of function units needed for each subtree be $N_{FU}^1, N_{FU}^2, \cdots, N_{FU}^n$, respectively. N_{FU} can be defined as follows:

$$N_{FU} = \begin{cases} \max\limits_{i=1}^{n} N_{FU}^i, & \text{if the node is an XOR node} \\ \sum\limits_{i=1}^{n} N_{FU}^i, & \text{if the node is an AND node} \\ x_{i,j}, & \text{if the node is a leaf.} \end{cases}$$

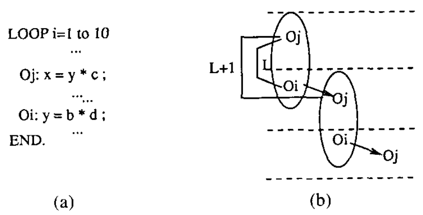


Fig. 5. (a) 1-deg data dependency between p_i and o_j. (b) The longest path from o_j to o_i is L.

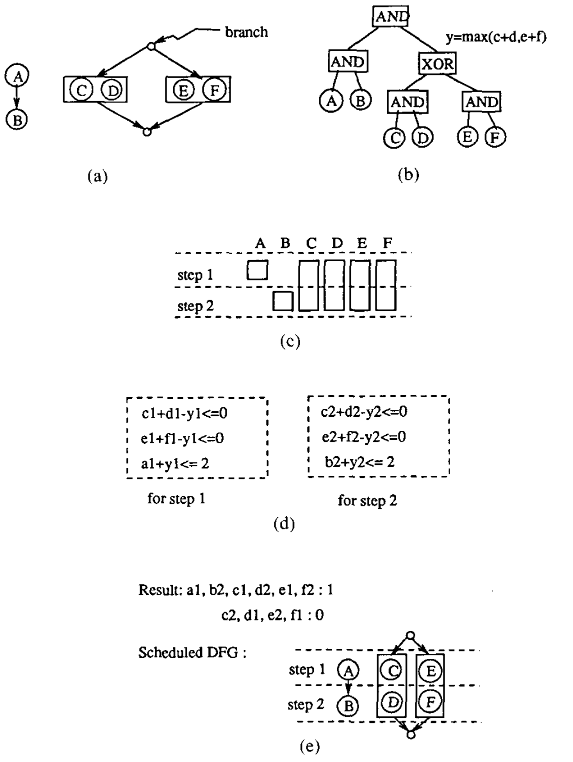


Fig. 6. (a) DFG. (b) A tree representation. (c) Distribution graph. (d) Part of the constraint. (e) Result and scheduled DFG.

Let $N_{FU_{t_k},j}(\boldsymbol{O})$ be the number of function units of type t_k required at control step j, which is defined in the above function. Constraint (2) is changed to

$$N_{FU_{t_k},j}(\boldsymbol{O}) \le M_{t_k}, \qquad \text{for } 1 \le j \le s,\ 1 \le k \le m. \qquad (2.5)$$

As an example, the DFG in Fig. 6(a) can be represented as a tree in Fig. 6(b). Suppose that two function units are given and the upper limit of the number of control steps is 2. The distribution graph is shown in Fig. 6(c). Fig. 6(d) shows part of the constraints. Here, $y1$ and $y2$ are introduced to satisfy $y1 \ge \max(c1 + d1, e1 + f1)$ and $y2 \ge \max(c2 + d2, e2 + f2)$. By solving the ILP formulation, an optimal schedule is obtained as shown in Fig. 6(e).

a) Complexity analysis: Suppose that there are N_{XOR} nodes in the execution tree and each node has N_i branches. In considering any control step of the DFG, we need to introduce N_{XOR}

integer variables for each XOR node and $\Sigma_{i=1}^{N_{\text{XOR}}} N_i$ equations for each branch. In the worst case, $N_{\text{XOR}} = \Sigma_{i=1}^{N_{\text{XOR}}} n_i = n$. Thus the complexity of constraint (2.5) is $O(s \cdot n)$ for both variables and equations.

4.5. Scheduling Under Bus Constraint

In the previous sections, we have focused on the minimization of the cost of function units. However, as the complexity of VLSI grows, the area for routing becomes important. Interconnection cost becomes a dominant factor of the cost function. In this subsection, we extend the formulation to minimize the cost of connections. Two models are considered for minimizing the number of buses. In the first model, the number of buses is calculated as twice the maximum number of operations among all the control steps. The second model makes a more sophisticated calculation by considering the broadcasting capability of a bus.

4.5.1) A Simple Model for Bus Minimization: Consider the example in Fig. 7. Although the number of function units required in Fig. 7(a) and (b) is the same, the latter needs 4 buses while the former needs 6 buses to connect the data path. Therefore, we prefer a Fig. 7(b) schedule. Since each function unit has two input buses, the number of buses can be estimated by multiplying the maximum number of operations among all the control steps by 2:

$$2 \cdot \sum_{i=1}^{n} x_{i,j} \le N_{\text{bus}}, \qquad \text{for } 1 \le j \le s. \tag{9}$$

a) Complexity analysis: Constraint (9) generates s equations, and no additional variables are introduced.

4.5.2) Bus with Broadcasting: In a bused architecture, when more than one operation which share a common input variable are scheduled into the same control step, the number of buses needed for that variable is only one (via broadcasting). Thus the number of buses required at a control step equals the number of distinct input variables of all the operations assigned to this step. Suppose the input variables at control step j are $v_1, v_2, \cdots, v_{|v|}$. We introduce a 0–1 integer variable $y_{r,j}$ for v_r ($1 \le r \le |v|$) at step j, where $y_{r,j}$ is 1 if v_r is accessed at this step; otherwise, 0. We have the constraint that the number of $y_{r,j}$ which are assigned to be 1 is less than the number of buses, i.e.,

$$\sum_{r=1}^{|v|} y_{r,j} \le N_{\text{bus}}, \qquad \text{for } 1 \le j \le s. \tag{10}$$

Since the transfer of variables during a control step ($y_{r,j}$) is directly related to the assignment of operations to a control step ($x_{i,j}$), we have to define the relationship between them. Let v_r be a shared input of a group of r_i operations, $o_{r_1}, o_{r_2}, \cdots$, and o_{r_t}. The value of $y_{r,j}$ is defined as follows: if $x_{r_1,j} = x_{r_2,j} = \cdots = x_{r_t,j} = 0$, then $y_{r,j}$ is given a value 0; otherwise, 1: i.e., $y_{r,j} = \textbf{OR}\,(x_{r_1,j}, x_{r_2,j} \cdots, x_{r_t,j})$. The following constraint is included to satisfy the definition of $y_{r,j}$:

$$\sum_{i=1}^{r_t} r_{r_i,j} - r_t \cdot y_{r,j} \le 0, \qquad \text{for } 1 \le r \le |v|, 1 \le j \le s. \tag{11}$$

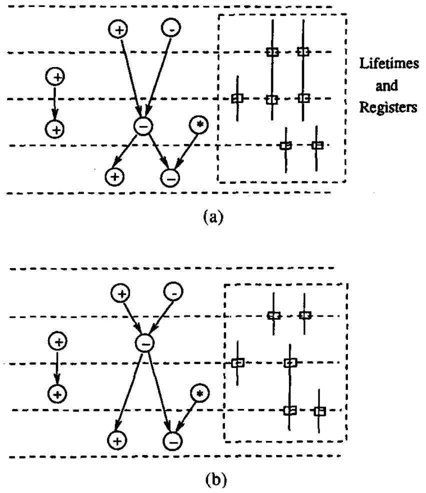


Fig. 7. (a) Bad schedule. ($SLK = 7$). (b) Better schedule ($SLK = 6$).

Note that when $r_t = 1$, $y_{r,j}$ in (10) can be directly replaced by $x_{i,j}$. The correspondent equation in (11) is omitted.

a) Complexity analysis: Constraint (11) requires a 0–1 variable and an equation for each variable at each control step. Thus the number of equations and variables is bounded by $s \cdot |v|$ where $|v|$ is the number of variables in DFG. Since $|v|$ is less than $2 \cdot n$ in a DFG, the complexity is $O(s \cdot n)$.

4.6. Minimizing Lifetimes of Variables

The *lifetime* of a variable is defined as the duration from the control step where it is defined to the step where it is last used. A variable must be assigned to a register during its lifetime; several variables can share the same register, provided that their lifetimes do not overlap. Thus a reduction in the lifetimes of variables has the potential of reducing the number of registers required. For example, both the two scheduled DFG's in Fig. 7 need the same number of operators and control steps; however, the schedule in Fig. 7(a) needs one more register than that in Fig. 7(b).

In addition to minimizing the function unit cost and the total number of steps, we can also take the lifetimes of variables into consideration. Let SLK_i be the longest duration that the output of o_i must be kept, i.e., $SLK_i = \max_{o_i \to o_j} (T_j - T_i - d_{o_i})$. In other words, SLK_i is the lifetime interval of the value computed by o_i. The cost function is modified so as to minimize

$$c_1 * f_1 + c_2 * \sum_{i=1}^{n} (SLK_i) \tag{1.2}$$

where f_1 may be the objective functions in time-constraint, resource-constraint or feasible scheduling ($f_1 = 0$). Since f_1 is our primary objective, c_1 is given a higher weight than c_2.

Finally, the lifetime of a variable is defined by including the following constraint:

$$T_j - T_i - d_{o_i} - SLK_i \le 0, \qquad \text{for all } o_i \to o_j. \tag{12}$$

TABLE II
NONPIPELINED DATA PATH WITH NONPIPELINED MULTIPLIER

System	HAL (FDS)				FDLS			ALPS				
Adders	3	3	2	2	3	2	2	3	2	2	2	1
Multipliers	3	2	2	1	3	2	1	3	2	2	1	1
Buses	—	—	—	—	—	—	—	6	6	4	4	4
Cycles	17	18	19	21	17	18	21	17	18	20	21	28

TABLE III
NONPIPELINED DATA PATH WITH PIPELINED MULTIPLIER

System	FDS and FDLS			ALPS				
Adders	3	3	2	3	3	2	2	1
Multipliers	2	1	1	2	1	1	1	1
Buses	—	—	—	6	6	6	4	4
Cycles	17	18	19	17	18	19	20	28

TABLE IV
FIFTH-ORDER FILTER WITH PIPELINED DATA PATH

	Nonpipelined Multiplier											
Latency	1	2	3	4	5	6	7	8	9	13	16	26
Adders	26	13	9	7	6	5	4	4	3	2	2	1
Multipliers	16	8	6	4	4	3	3	2	2	2	1	1
Delay	17	17	17	18	19	19	18	20	21	23	21	33
	Pipelined Multiplier											
Latency	1	2	3	4	5	6	7	8	9	13	—	26
Adders	26	13	9	7	6	5	4	4	3	2	—	1
Multipliers	8	4	3	2	2	2	2	1	1	1	—	1
Delay	17	17	18	19	19	17	18	20	22	23	—	33
	Result of [21]											
Latency	—	2	3	4	5	6	7	8	9	10	—	—
Adders	—	13	10	7	6	5	5	6	4	4	—	—
Multipliers	—	4	4	2	3	2	2	2	2	2	—	—

TABLE V
LOOP FOLDING WITH NONPIPELINED MULTIPLIER

System	Spaid		ALPS		
Adders	3	2	3	2	2
Multipliers	2	1	2(3)†	2	1
Buses	6	6	6	6	6
Sample period	17	21	16	17	19
Delay	—	—	18	19	21

†If a self-timed design is required.

TABLE VI
LOOP FOLDING WITH PIPELINED MULTIPLIER

System	Spaid							ALPS			
Adders	4	3	2	2	2	1	2	3	2	1	1
Multipliers	2	2	1	1	1	1	1	1(2)†	1	1	1
Buses	7	6	6	5	4	4	2	6	6	4	2
Sample period	16	17	18	19	21	29	39	16	17	28	34
Delay	—	—	—	—	—	—	—	18	19	28	34

†If a self-timed design is required.

a) Complexity analysis: For a given DFG $G(V, E)$, where the nodes are operations and the edges are the data precedence between operations, the above formulations require n integer variables each for an SLK_i, $1 \leq i \leq n$, and e equations each for a data dependency.

V. EXPERIMENTAL RESULTS

The system called ALPS has been implemented and tested. The programs for list scheduling, ASAP, ALAP, and ILP formulations are written in C on a VAX 11/8550 running ULTRIX, and the ILP formulation is solved using the LINDO [27] package on a VAX 11/8800 running VMS. LINDO starts with an optimal linear programming solution and produces an optimal integer solution using the branch-and-bound method. The fifth-order wave filter which was borrowed from [8] is given to illustrate various requirements. It contains 26 additions and 8 multiplications. As most systems do, we suppose a multiplication takes 2 cycles while an addition takes 1 cycle to complete. The critical path length is 17 cycles. The runtime for the various experiments of the example depends on the number of 0-1 variables and is within tens of seconds.

5.1. Nonpipelined Data Path

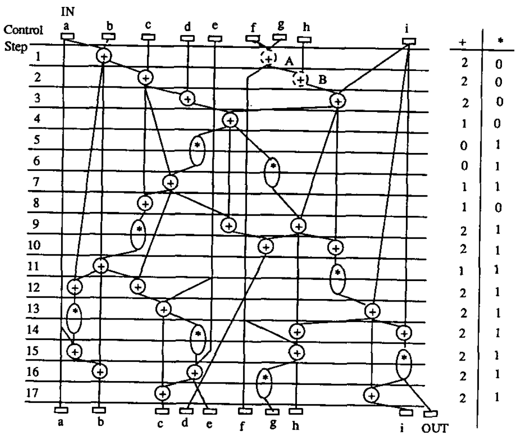
Tables II and III show the results with a nonpipelined data path. The multiplier can be nonpipelined (Table II) or pipelined (Table III). We also take into account the cost of buses. All the results are optimal.

5.2. Functional Pipelining (Pipelined Data Path)

The DFG of the fifth-order filter is used to test functional pipelining. (We assume that there are no data dependencies between iterations; i.e., the outputs of the DFG will not feed back into the inputs.) We have achieved the minimal number of resources for each latency and we have also minimized the delay time. The results are shown in the first and second parts of Table IV for nonpipelined and pipelined multipliers; respectively. The third part of Table IV shows the results of [21], where the maximum delay is set at 10 cycles. Note that in their implementation, the cycle time is longer so that a multiplication or two additions can be executed within a single control step.

5.3. Loop Folding

The critical path length of the fifth filter can be reduced to 16 cycles after loop folding or retiming [25], [26] while preserving the interiteration data precedences. Tables V and VI show the minimal sample period (= loop length) and delays using a nonpipelined multiplier and a pipelined multiplier. Here, delay means the number of control steps required for the entire DFG to be executed. Although delay time is ignored by other systems, we are concerned with it and try to minimize it for two reasons: first, with respect to the sample period, which corresponds to the throughput of the system, the delay time is directly related to the turn around time, which is one of the most



Control Step	+	*
1	2	0
2	2	0
3	2	0
4	1	0
5	0	1
6	0	1
7	1	1
8	1	0
9	2	1
10	2	1
11	1	1
12	2	1
13	2	1
14	2	1
15	2	1
16	2	1
17	2	1

□ Nodes A and B are folded to next iteration
□ Loop length = 17 control steps
□ Longest delay = 19 (In -> g -> B)
□ IN --> OUT = 16 control steps
□ Using 2 Adders, 1 Multiplier and 6 Busses

Fig. 8. A schedule after loop folding.

important performance criteria. Second, a longer delay increases the lifetimes of the variables. Thus minimizing the delay time will potentially reduce the register cost. Note that Spaid first retimed the DFG, then, performed a scheduling to find the loop length (called clock cycle in Spaid). Our loop folding technique performs retiming and scheduling simultaneously, which makes a better solution possible. Our scheduler is also able to make a scheduling under the self-timed [25] requirement. Fig. 8 shows the scheduled DFG of 1 multiplier and 2 adders under the self-timed requirement.

VI. Conclusion

We have proposed an approach to the scneduling problems in high level synthesis. Our approach includes list scheduling, ASAP, ALAP, and ILP. The ILP formulation is very efficient, and its complexity in the number of variables is $O(s \cdot n)$ where s and n are the number of control steps and operations, respectively. With the feasible scheduling formulation, we can explore the solution space more efficiently. For the problem of the fifth-order filter, optimal solution is obtained in a few seconds. We have also generalized the formulations to include practical requirements such as chaining, multicycling operations, structure and functional pipelining, loop folding, mutually exclusive operations, and the minimization of the cost of buses and registers.

Acknowledgment

The authors wish to thank Professor Y. L. Lin and F. S. Tsai for their helpful discussions on this work.

References

[1] L. Hafer and A. C. Parker, "A formal method for the specification, analysis, and design of register-transfer level digital logic," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 4-18, Jan. 1983.

[2] M. C. McFarland, A. C. Parker, and R. Camposano, "Tutorial on high-level synthesis," in *Proc. 25th Design Automation Conf.*, June 1988, pp. 330-336.

[3] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661-679, June 1989.

[4] C. Tseng and D. P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 379-395, July 1986.

[5] C. H. Gebotys and M. I. Elmasry, "A VLSI methodology with testability constraints," in *Proc. 1987 Canadian Conf. VLSI*, Winnipeg, Oct. 1987.

[6] P. Marwedel, "A new synthesis algorithm for the MIMOLA software system," in *Proc. 23rd Design Automation Conf.*, July 1986, pp. 271-277.

[7] H. Trickey, "Flamel: A high-level hardware compiler," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 259-269, Mar. 1987.

[8] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985, pp. 258-264.

[9] S. Davidson *et al.*, "Some experiments in local microcode compaction for horizontal machines," *IEEE Trans. Comput.*, pp. 460-477, July 1981.

[10] C. Y. Hitchcock and D. E. Thomas, "A method of automatic data path synthesis," in *Proc. 20th Design Automation Conf.*, July 1983, pp. 484-489.

[11] J. Nestor and D. E. Thomas, "Behavioral synthesis with interface," in *Proc. ICCAD-87*, Nov. 1986, pp. 112-115.

[12] B. M. Pangrle and D. D. Gajski, "State synthesis and connectivity binding for microarchitecture compilation," in *Proc. ICCAD-86*, Nov. 1986, pp. 210-213.

[13] H. DeMan, J. Rabaey, P. Six, and L. Claesen, "Cathedral-II: A silicon compiler for digital signal processing," *IEEE Design Test*, pp. 13-25, Dec. 1986.

[14] E. F. Girczyc and J. P. Knight, "An ADA to standard cell hardware compiler based on graph grammers and scheduling," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1984, pp. 726-731.

[15] A. C. Parker, J. Pizarro, and M. J. Mlinarr, "MAHA: A program for data path synthesis," in *Proc. 23rd Design Automation Conf.*, July 1986, pp. 461-466.

[16] R. Camposano, "Structural synthesis in Yorktown silicon compiler," in *VLSI'87*, C. H. Sequin, ed. New York: Elsevier, 1988, pp. 61-72.

[17] D. D. Gajski, N. D. Dutt and B. M. Pangrle, "Silicon compilation (tutorial)," in *Proc. Custom Integrated Circuits Conf.*, Rochester, NY, May 1986, pp. 102-110.

[18] N. Park and A. C. Parker, "Synthesis of optimal clocking schemes," in *Proc. 22nd Design Automation Conf.*, July 1985, pp. 489-495.

[19] G. Goosens *et al.*, "An efficient microcode compiler for custom multiprocessor DSP systems," in *Proc. ICCAD-87*, Nov. 1987, pp. 24-27.

[20] K. S. Hwang *et al.*, "Constrained conditional resource sharing in pipeline synthesis," in *Proc. ICCAD-88*, Nov. 1988, pp. 52-55.

[21] K. S. Hwang, A. E. Casavant, C.-T. Chang, and M. A. d'Abreu, "Scheduling and hardware sharing in pipelined data paths," in *Proc. ICCAD-89*, Nov. 1989, pp. 24-27.

[22] N. Park and A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. Computer-Aided Design*, Mar. 1988, pp. 356-370.

[23] S. Devadas and A. R. Newton, "Data path synthesis from behavioral description: An algorithmic approach," in *Proc. Int. Symp. on Circuits and Systems*, Philadelphia, May 1987, pp. 398-401.

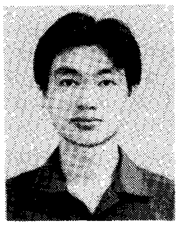
[24] G. Goossens, J. Vandewalle, and H. De Man, "Loop optimization in register-transfer scheduling for DSP-systems," in *Proc. 26th Design Automation Conf.*, June 1989, pp. 826-831.

[25] B. S. Haroun and M. I. Elmasry, "Architectural synthesis for DSP silicon compiler," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 431-447, April 1989.

[26] F. Rose, C. Leiserson, and J. Saxe, "Optimizing synthesis circuitry by retiming," in *Proc. Caltech Conf. on VLSI*, 1983, pp. 41-67.

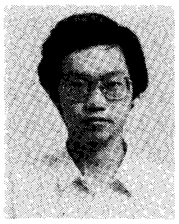
[27] "LINDO: Linear INteractive and discrete optimizer for linear, integer, and quadratic programming problems," LINDO Systems, Inc.

[28] C. Gebotys and M. I. Elmasry, "VLSI design synthesis with testability," in *Proc. 25th Design Automation Conf.*, June 1988, pp. 16-21.



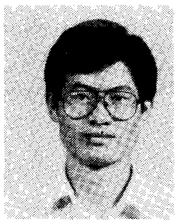
Cheng-Tsung Hwang received the B.S. degree in computer science from Tung-Hai University, Taichung, Taiwan, in 1987. He is currently working toward the Ph.D. degree in the Department of Computer Science at Tsing Hua University, Hsin-chu, Taiwan.

His research interests include silicon compilation and optimization in VLSI design.



Jiahn-Hurng Lee received the B.S. degree in computer science from Tung-Hai University, Taichung, Taiwan, in 1987, and the M.S. degree in computer science from Tsing Hua University, Hsin-chu, Taiwan, 1989.

His research interests include silicon compilation and optimization in VLSI design.



Yu-Chin Hsu (S'85-M'87) received the B.S. degree in computer science and information engineering from Taiwan University, Taipei, Taiwan, in 1981, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1986 and 1987, respectively.

He is currently an Associate Professor of Computer Science at Tsing Hua University, Hsin-chu, Taiwan. His research interests include most aspects of computer-aided design for VLSI.

Dr. Hsu received the Outstanding Young Author Award from the IEEE Circuits and Systems Society in 1990.