Module Selection and Data Format Conversion for Cost-Optimal DSP Synthesis[†]

Kazuhito Ito   Lori E. Lucke   Keshab K. Parhi

Department of Electrical Engineering
University of Minnesota
Minneapolis, MN 55455

Abstract

In high level synthesis each node of a synchronous data-flow graph (DFG) is scheduled to a specific time and allocated to a processor. In this paper we present new integer linear programming (ILP) models which generate a blocked schedule for a DFG with implicit retiming, pipelining, and unfolding while performing module selection and data format conversion. A blocked schedule is a schedule which overlaps multiple iterations of the DFG to guarantee a minimum number of processors. Component modules are selected from a library of processors to minimize cost. Furthermore, we include data format converters between processors of different data formats. In addition, we minimize the unfolding factor of the blocked schedule.

1 Introduction

In high-level synthesis a synchronous data-flow graph (DFG) is mapped onto a set of modules, registers, and interconnections [1]. An example of a DFG is shown in Fig. 1. The data-flow graph represents an iterative algorithm such as a digital signal processing algorithm. A DFG can be non-recursive or recursive. A recursive DFG contains feedback loops (or cycles) and therefore has an inherent lower bound on its iteration period called the *iteration bound* [2, 3]. High level synthesis consists of scheduling and resource allocation where the goal is to assign an operation in the DFG to an execution time on a particular processor. In this paper we consider time-constrained scheduling where required resources are minimized while satisfying the iteration period specification.

Finding an optimal schedule during synthesis of a DFG is an NP-complete problem [4]. Therefore, many heuristic schedulers have been proposed [1][5]-[8]. While these schedulers generate reasonable schedules in short CPU time, the optimality of the schedule may not be guaranteed. Integer linear programming (ILP) solutions have been proposed to solve the scheduling problem during high level VLSI synthesis of DSP algorithms [9]-[15]. The ILP model in [15] operates on the original DFG and generates a *blocked schedule* which automatically retimes [16], pipelines [17], and unfolds [3] the DFG. The ILP formulation is attractive because of the solution optimality and the ease of adding additional constraints to the scheduling problem.

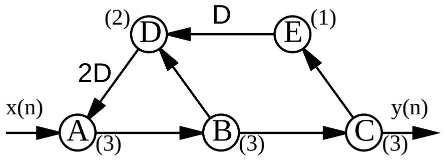


Fig. 1. Data-Flow Graph. (Node execution times are in parentheses.)

In this paper we extend the ILP model of [15] to solve the problem of module selection while scheduling. The objective of module selection is *automatic allocation of each operation to a library of processors* to synthesize a system using less silicon area and lower power. The module selection during scheduling has been addressed in [18]-[20] in the context of heuristic scheduling and in [14] for scheduling large-grain signal processing algorithms by ILP. In this paper we support fine-grain signal processing algorithms. One common way to build a library of components is to include bit-serial and bit-parallel units [21]. If both types of units are used and must communicate, then it is essential to include a serial to parallel (or parallel to serial) data format converter. Thus, we include support for cost and computational latency of *data format conversion* which has not been considered before.

This paper is organized as follows. In section 2, blocked scheduling and unfolding are discussed. Module selection and data format conversion are demonstrated in section 3. The time assignment ILP model with module selection and data format conversion is presented in section 4. Section 5 contains the processor allocation ILP model with support for unfolding factor minimization. In section 6 scheduling results of several benchmarks are presented.

2 Blocked schedules

Fig. 2(a) shows a critical path method (CPM) schedule [1, 5] of the DFG in Fig.1. In this schedule a single itera-

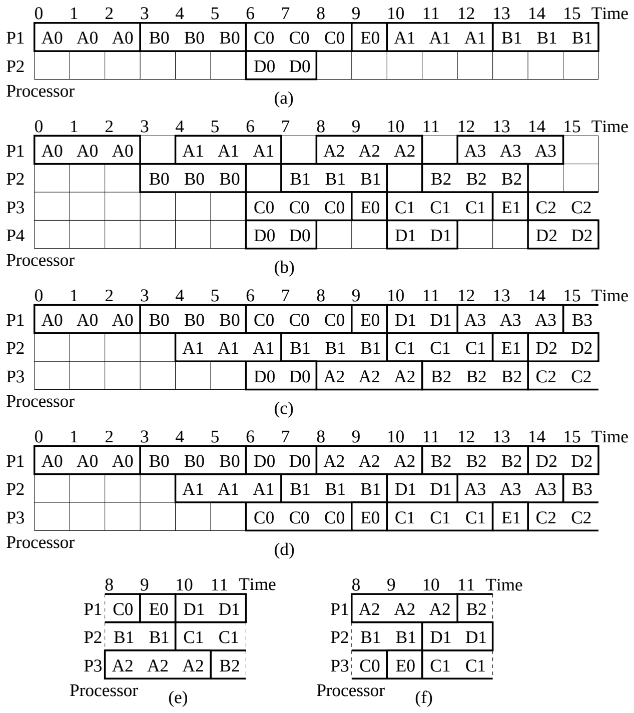
[†]This research was supported by the Advanced Research Projects Agency and monitored by Solid State Electronics Directorate of the Wright-Patterson AFB under contract number F33615-93-C-1309.

tion does not overlap subsequent iterations. The minimum possible iteration period for a CPM scheduler is equal to the critical path length of the DFG. There are many cases where it is not possible to reduce the critical path time to the iteration bound through DFG transformations such as retiming and pipelining [16, 22]. For example, no matter how the DFG in Fig. 1 is retimed, the critical path will always be greater than the iteration bound of 4 u.t. (units of time).

This limitation is overcome by schedulers which overlap multiple iterations [6]–[9][23]. These schedulers schedule a single iteration of the DFG but allow subsequent iterations to overlap the first. In an overlapped schedule, each node computation is folded into T_r equivalence classes and executed every T_r u.t. where T_r is the iteration period. This is sometimes referred to as loop unrolling or functional pipelining [6, 23]. An overlapping schedule automatically supports retiming and functional pipelining. The minimum possible iteration period for an overlapping scheduler is limited by the longest execution time of a single node or the iteration bound, whichever is largest. Moreover, the processor utilization may not be optimal in overlapped schedules as shown in Fig. 2(b).

Unfolding [3] and cyclo-static techniques [24] can be used to guarantee a rate-optimal and processor-optimal schedule even when there exists a node whose computation time exceeds the iteration bound. Both of these techniques require scheduling multiple iterations of the DFG. We call a multiple iteration schedule a *blocked schedule*. While an iteration is repeated in a non-blocked schedule, a block of iterations is repeated in a blocked schedule. An example of a blocked schedule for the DFG in Fig. 1 is shown in Fig. 2(c). In this case, a schedule of 12 u.t., representing three iterations of the iteration period of 4 u.t., is repeated in every processor. Here the processor utilization is optimized since the number of processors is reduced from 4 in Fig. 2(b) to 3 in Fig. 2(c). The blocked schedule can always achieve the iteration bound of the DFG with optimal processor utilization. An apparent disadvantage of unfolding is the need to schedule multiple executions of each node.

The blocked scheduler proposed in [15] generates an abstracted blocked schedule of the original DFG, like the ones shown in Fig. 2(e) and in Fig. 2(f), without explicitly unfolding the original DFG. The complete blocked schedule can be generated from the abstracted schedule by simply repeating the schedule while exchanging processor assignments such that an iteration of any node is executed in a single processor. For example, while expanding the abstracted schedule of Fig. 2(e) to generate the complete schedule in Fig. 2(c), the allocations of $P3$ and $P2$ must be exchanged so that $B2$ is completed in processor $P3$. Thus it is possible to generate the abstracted schedule by scheduling the original DFG without considering multiple iterations.



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time
P1 A0 A0 A0 B0 B0 B0 C0 C0 C0 E0 A1 A1 A1 B1 B1 B1
P2 D0 D0
Processor
(a)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time
P1 A0 A0 A0 A1 A1 A1 A2 A2 A2 A3 A3 A3
P2 B0 B0 B0 B1 B1 B1 B2 B2 B2
P3 C0 C0 C0 E0 C1 C1 C1 E1 C2 C2
P4 D0 D0 D1 D1 D2 D2
Processor
(b)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time
P1 A0 A0 A0 B0 B0 B0 C0 C0 C0 E0 D1 D1 A3 A3 A3 B3
P2 A1 A1 A1 B1 B1 B1 C1 C1 C1 E1 D2 D2
P3 D0 D0 A2 A2 A2 B2 B2 B2 C2 C2
Processor
(c)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time
P1 A0 A0 A0 B0 B0 B0 D0 D0 A2 A2 A2 B2 B2 B2 D2 D2
P2 A1 A1 A1 B1 B1 B1 D1 D1 A3 A3 A3 B3
P3 C0 C0 C0 E0 C1 C1 C1 E1 C2 C2
Processor
(d)
8 9 10 11 Time
P1 C0 E0 D1 D1
P2 B1 B1 C1 C1
P3 A2 A2 A2 B2
Processor
(e)
8 9 10 11 Time
P1 A2 A2 A2 B2
P2 B1 B1 D1 D1
P3 C0 E0 C1 C1
Processor
(f)

Fig. 2. Several schedules for the DFG in Fig. 1. (a) Critical path schedule with T_r = 10 u.t. (b) Overlapped schedule with T_r = 4 u.t. (c) Blocked schedule with T_r = 4 u.t. (d) Blocked schedule with minimized unfolding factors. (T_r = 4 u.t.) (e) Abstracted version of the blocked schedule of (c). (f) Abstracted version of the blocked schedule of (d).

Let the *unfolding factor of a processor* be defined as the length of one period of the blocked schedule of the processor divided by the iteration period. In the processor-optimal blocked schedule shown in Fig. 2(c), all the unfolding factor of each processor is 3. It is important to note that the unfolding factors of processors need not be identical. For example in the blocked schedule shown in Fig. 2(d) which is also processor-optimal, the unfolding factors of processors $P1$, $P2$, and $P3$ are 2, 2, and 1, respectively. From the viewpoint of control circuit cost, smaller unfolding factors are preferable since the cost of the control circuit could be proportional to the length of the iteration period and therefore proportional to the sum of the processor unfolding factors. Thus, the schedule in Fig. 2(d) is superior to the schedule in Fig. 2(c) since the sum of processor unfolding factors is 5 for Fig. 2(d) and 9 for Fig. 2(c).

Generating a blocked schedule by a single complicated ILP model requires a long solution time. In our approach, a blocked schedule is generated by two ILP models to improve the solution time without degrading the optimality. First the time assignment ILP model finds the start times for each node by folding the nodes into equivalent time partitions as in an overlapping scheduler. Module selection and data format conversion are also performed by this ILP model. Then the processor allocation ILP model finds

Table 1 Library of Processor Types (wordlength = 16)

type	processor	C	L	m	I	O
A_{bp}	Bit-parallel adder	1	1	53	*bp*	*bp*
A_{hp}	Half-word parallel adder	1	2	19	*hp*	*hp*
A_{ds}	4-bit digit-serial adder	1	4	6	*ds*	*ds*
M_{bp}	Bit-parallel multiplier	5	1	331	*bp*	*bp*
M_{hp}	Half-word parallel multiplier	6	2	173	*hp*	*hp*
M_{ds}	4-bit digit-serial multiplier	9	5	86	*ds*	*ds*

Table 2 Converter Types

type	conversion	C	L	m
$v_{bp,hp}$	*bp*→*hp*	0	1	3
$v_{bp,ds}$	*bp*→*ds*	0	3	4
$v_{hp,bp}$	*hp*→*bp*	1	1	3
$v_{hp,ds}$	*hp*→*ds*	0	2	3
$v_{ds,bp}$	*ds*→*bp*	3	3	4
$v_{ds,hp}$	*ds*→*hp*	2	2	3

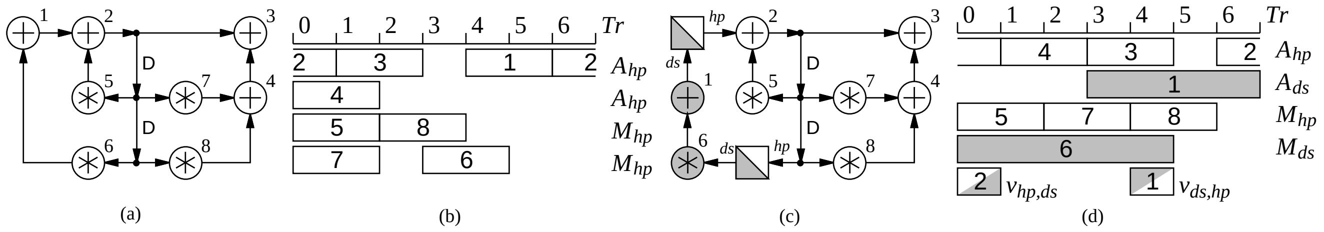


Fig. 3. Time-constrained schedule for a biquad filter. (a) Data-flow graph with a required iteration period of 7 u.t. (b) The schedule where only one type of processor for each operation type is available. (c) The assignment of nodes to processor types and insertion of converters when more than one types of processor for each operation type are available. (d) The schedule corresponding to (c).

the processor allocation for each of the nodes in the DFG. The time assignment is implicitly unfolded and the sum of unfolding factors of the processors is minimized.

3 Extension to support module selection and data format conversion

In the scheduling examples discussed in Fig. 2, we assume a node has a predetermined computation time. We extend the DFG as follows to support module selection. First consider a library of processors with varying processor types as shown in Table 1. These processors are derived assuming the use of 16 bit wordlength fixed point arithmetic. The *computational latency*, C, represents the time from an input to its associated output. The *pipeline period*, L, represents the time between successive operations. The *cost*, m, represents the cost in terms of area (i.e., the equivalent number of full adders) of each processor. The *input and output data formats*, I and O, represent the digit-size or the number of bits processed in every clock cycle in each processor. A 4-bit digit-serial architecture processes the data 4 bits at a time. These architectures may be derived using the techniques described in [21].

Each node in the DFG can be assigned to one of the processors in the library. For example, the DFG in Fig. 3(a) represents a biquad filter. Nodes 1, 2, 3, and 4 can be assigned to any of the adders, A_{bp}, A_{hp}, or A_{ds}, in Table 1. Similarly, nodes 5, 6, 7, and 8 can be assigned to any of the multipliers, M_{bp}, M_{hp}, or M_{ds}, in Table 1. Furthermore to support data format conversion, we include a library of data format converters which convert between all possible data formats listed in the library of processors. The library of processors in Table 1 requires data format converters as shown in Table 2. Each of the data format converters is classified according to its conversion type, its *conversion latency*, C, its *pipeline period*, L, and its *cost*, m. The conversion latency is the time between input of the first digit and the output of the first digit of converted data. For example, it is 0 for a bit-parallel to half-word parallel converter ($v_{bp,hp}$) since the first half-word is available as soon as a bit-parallel data is input. The conversion latency for a 4-bit digit-serial to bit-parallel parallel converter ($v_{ds,bp}$) is 3 since it takes 3 u.t. to input and store three digits and the converted data is output when the last digit is input.

When the processor library is limited to just two processors, A_{hp} and M_{hp} in Table 1, then a blocked schedule of the DFG of Fig. 3(a) can be obtained as shown in Fig. 3(b). This schedule requires two A_{hp} adders and two M_{hp} multipliers with a total cost of 384 units. When the processor library is expanded to include all the processors and data format converters in Tables 1 and 2, then nodes are assigned to processors and data format conversions are inserted as shown in Fig. 3(c). Fig. 3(d) shows the abstracted blocked schedule. Nodes 1 and 6 are assigned to slower and less expensive processors. Data format conversions, symbolized by a box in Fig. 3(c), are necessary between nodes 2 and 6, and 1 and 2. One converts from half-word parallel to 4-bit digit-serial and the other converts from 4-bit digit-serial to half-word parallel. The blocked schedule with module selection and data format conversion has a total cost of only 290 units compared to the original cost of 384.

4 ILP model for time assignment

We define a time assignment ILP model to derive the cost optimal architecture for the given DFG. The time assignment model assigns a start time to each node within the DFG so as to satisfy precedence constraints, while performing module selection and data format conversion.

The following terminology is used.

- The DFG is defined as (N, E). N is the set of nodes and E is the set of edges. W_e is the number of delays on edge $e \in E$.
- T_r is the given iteration period.
- *PROC* is the library of available processors.
- F_i denotes the subset of processors $F_i \subset PROC$, capable of executing node $i \in N$.
- Each processor, $t \in PROC$, has computational latency C_t, pipeline period L_t, and cost m_t.
- A binary variable $x_{i,j,t} = 1$ means that node i starts at time j on a processor of type t.
- *FORM* is the set of all the formats.
- $I(t)$ and $O(t)$ are respectively the input and output data formats of a processor of type t.
- *CONV* is the library of available converters.
- v_{qr} denotes a data format converter which converts data from format q to format r. Each data format converter, v, has conversion latency C_v, pipeline period L_v, and cost m_v.
- A binary variable $y_{i,j,v} = 1$ means that a data format converter of type v is used and the conversion for the output data of node i starts at time j.
- LB_i and UB_i are the lower bound and the upper bound of the time at which the computation of node i can start. LB_v^i and UB_v^i are the lower bound and the upper bound of the time at which a converter of type v could start converting the data output from node i. These bounds determine the scheduling range of node i and are calculated as in [7, 15], assuming nodes are executed on the fastest processor available.
- R_i (R_v^i) denotes the scheduling range of node i (converter v), which is the closed time interval $[LB_i, LB_i + 1, \ldots, UB_i]$ ($[LB_v^i, LB_v^i + 1, \ldots, UB_v^i]$). $R_i + k$ is defined to denote a closed time interval $[LB_i+k, \ldots, UB_i+k]$ where k is an integer.
- M_t and M_v are integer variables respectively indicating the number of processors of type t and the number of converters of type v.

We describe the ILP model as follows. The model minimizes the total cost of processors and converters (1) while satisfying the constraints (2)–(8).

$$\text{Minimize } COST = \sum_{t \in PROC} m_t M_t + \sum_{v \in CONV} m_v M_v \quad (1)$$

The node assignment constraint (2) ensures that node i has one start time and is assigned to one processor. The converter assignment constraint (3) ensures that a data format converter of type v_{qr} is used if an edge (a, b) exists and node a is assigned to a processor whose output data format is q and node b is assigned to a processor whose input data format is r.

In the precedence constraint from processor to processor (4), the data format conversion time is taken into account. If an edge $e = (a, b)$ exists, the computation of node b must start at least $C_{t_a} + C_{v_{O(ta),I(tb)}} - W_e T_r$ u.t. later than the computation of node a starts since the computation of node a takes C_{t_a} u.t. and the data format conversion takes $C_{v_{O(ta),I(tb)}}$ u.t. If $O(t_a) = I(t_b)$, no data format conversion is performed since $C_{v_{rr}} = 0$ for $r \in FORM$.

Inequalities (5) and (6) ensure the precedence constraints from processor to converter and from converter to processor, respectively. In the case that the output format of the converter and the input format of the processor are different, there is no need to constrain the precedence relation between them. In that case, inequality (6) is automatically satisfied.

Inequalities (7) and (8) are used to count the number of processors and the number of converters of each type. In an overlapped schedule with an iteration period of T_r, there are T_r time partitions. Each time unit, j_0, belongs to the time partition denoted by $j_0 - \left\lfloor \frac{j_0}{T_r} \right\rfloor T_r$, or $j_0 \bmod T_r$, and nodes assigned to a time belonging to the same time partition are executed concurrently. Such nodes must be assigned to different processors. The parameter k_1 in constraint (7) is used to fold a time into its time partition. The parameter p is used to handle structural pipelining. When a node is assigned to a processor whose pipeline period is longer than the iteration period, the processor must be counted multiple times, $\left\lfloor \frac{L_t - 1}{T_r} \right\rfloor$, since the node occupies the processor for more than one iteration period. This accounts for the second term in constraint (7). The same applies to (8).

5 ILP model for processor allocation

The processor allocation model allocates node computations to processors to support unfolding using the start times and module selection provided by the time assignment model. Allocating data format conversions to data format converters can be performed in the same way as allocating node computations to processors. Therefore, only the allocation of node computations to processors is considered here. The goal of the allocation is to minimize the unfolding factor while supporting blocked schedules.

5.1 Precise calculation of unfolding factor

As discussed in section 2, the processor allocation in Fig. 2(e) is preferable to that in Fig. 2(d) since the sum of the processor unfolding factors in Fig. 2(d) is less than that of Fig. 2(c). For the purpose of calculating unfolding factors, we only need to consider the allocation of nodes whose

$$\sum_{t\in F_i}\sum_{j\in R_i} x_{i,j,t} = 1 \qquad \forall i \in N. \tag{2}$$

$$\sum_{j\in R^a_{v_{qr}}} y_{a,j,v_{qr}} \geq \sum_{\substack{ta\in F_a\\ O(ta)=q}}\sum_{ja\in R_a} x_{a,ja,ta} + \sum_{\substack{tb\in F_b\\ I(tb)=r}}\sum_{jb\in R_b} x_{b,jb,tb} - 1 \qquad \forall q,r \in \mathit{FORM}, e=(a,b)\in E \tag{3}$$

$$\sum_{ta\in F_a}\sum_{\substack{ja\in R_a\\ ja\geq j-C_{t_a}-C_{v_{O(ta),f}}+1}} x_{a,ja,ta} + \sum_{\substack{tb\in F_b\\ I(tb)=f}}\sum_{\substack{jb\in R_b\\ jb\leq j-W_eT_r}} x_{b,jb,tb} \leq 1 \qquad \begin{array}{l}\forall f \in \mathit{FORM}, e=(a,b)\in E,\\ j\in (R_a + C_{t_a} + C_{v_{O(ta),f}} - 1)\cap (R_b + W_eT_r)\end{array} \tag{4}$$

$$\sum_{ta\in F_a}\sum_{\substack{ja\in R_a\\ ja\geq j-C_{t_a}+1}} x_{a,ja,ta} + \sum_{\substack{v_{qr}\\ r=f}}\sum_{\substack{j_1\in R^a_{v_{qr}}\\ j_1\leq j}} y_{a,j_1,v_{qr}} \leq 1 \qquad \forall f\in \mathit{FORM}, a\in N, j\in (R_a + \min C_{t_a} - 1)\cap(\cup R^a_{v_{qr}}) \tag{5}$$

$$\sum_{\substack{v_{qr}\\ r=f}}\sum_{\substack{j_1\in R^a_{v_{qr}}\\ j_1\geq j-C_{v_{qr}}+1}} y_{a,j_1,v_{qr}} + \sum_{\substack{tb\in F_b\\ I(tb)=f}}\sum_{\substack{jb\in R_b\\ jb\leq j+W_eT_r}} x_{b,jb,tb} \leq 1 \qquad \begin{array}{l}\forall f \in \mathit{FORM}, e=(a,b)\in E,\\ j\in (\cup(R^a_{v_{qr}} + C_{v_{qr}} - 1))\cap (R_b + W_eT_r)\end{array} \tag{6}$$

$$\sum_{i\in N}\left\{\sum_{k_1=\left\lfloor\frac{LB_i}{T_r}\right\rfloor}^{\left\lceil\frac{UB_i}{T_r}\right\rceil}\ \sum_{p=0}^{L_t-\left\lfloor\frac{L_t-1}{T_r}\right\rfloor T_r-1} x_{i,j+k_1T_r-p,t} + \left\lfloor\frac{L_t-1}{T_r}\right\rfloor\sum_{j_1\in R_i} x_{i,j_1,t}\right\} \leq M_t \qquad \forall j\in[0,T_r-1],\ t\in \mathit{PROC} \tag{7}$$

$$\sum_{i\in N}\left\{\sum_{k_1=\left\lfloor\frac{LB^i_v}{T_r}\right\rfloor}^{\left\lceil\frac{UB^i_v}{T_r}\right\rceil}\ \sum_{p=0}^{L_v-\left\lfloor\frac{L_v-1}{T_r}\right\rfloor T_r-1} y_{i,j+k_1T_r-p,v} + \left\lfloor\frac{L_v-1}{T_r}\right\rfloor\sum_{j_1\in R^i_v} y_{i,j_1,v}\right\} \leq M_v \quad \forall j\in[0,T_r-1],\ v\in \mathit{CONV} \tag{8}$$

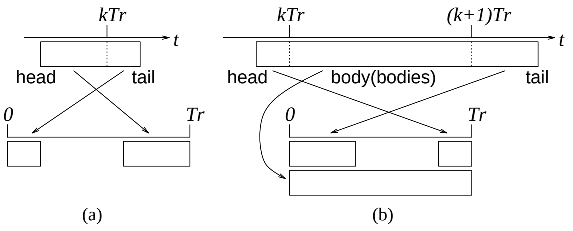


Fig. 4. Dividing computation of a node. (a) Division into head and tail. (b) Division into head, bodies, and tail.

computation crosses a multiple of the original iteration period T_r. For example, the computations of nodes B and C in Figs. 2(e) and (f) cross the iteration period.

First we calculate the unfolding factors assuming there are no nodes with a computation time (pipeline period) longer than the iteration period. Then we modify our calculations to include nodes with computation times longer than the iteration period. Let a computation of a node be divided at a multiple of the original iteration period as illustrated in Fig. 4(a). Let the first portion be called the *head* of the node and the second portion be called the *tail* of the node. The head is assigned to the end of an iteration cycle and the tail is assigned to the beginning of an iteration cycle as shown in Fig. 4(a).

The head and the tail of a node are allocated either to an identical processor or to two distinct processors. Nodes are divided into *node groups* depending on their allocation as follows.

[*Definition*: Node group]

A node group is the set of nodes such that either the head or tail of a node in a node group is allocated to a processor to which the tail or head of another node in the same node group is allocated.

There may exist at most one node in a node group whose head is allocated to a processor to which no tail is allocated. When this occurs we say *a head is allocated alone*.

The schedule is unfolded by a factor such that all the computations of nodes in the node group are executed one after another on an identical processor. In the processor allocation in Fig. 2(f), there are two node groups: one consists of node B and the other consists of node C. The head of node B is allocated alone to processor $P1$. Since the head and the tail of node C are allocated to the identical processor, $P3$, the schedule is not unfolded and the unfolding factor for $P3$ is 1. On the other hand, the head and tail of node B are allocated to distinct processors $P1$ and $P2$. The schedules of these processors must be unfolded twice so that the head and the tail of node B are executed consecutively on an identical processor.

In the processor allocation in Fig. 2(e), there is only one node group consisting of nodes B and C. The head of node B is allocated alone. In this case, the schedules of these processors must be unfolded 3 times as shown in Fig. 2(c) so that the head and the tail of node B are executed consecutively on an identical processor, the head of node C is executed on the same processor as the tail of node B, and the head and the tail of node C are executed consecutively on an identical processor. Consequently, the unfolding factor of a processor which executes the computations of nodes in a node group is equal to the number of nodes in the node group plus one.

We can calculate the unfolding factors more precisely as follows. Let a binary variable $g_{i1,i2} = 1$ if node $i1$ and node $i2$ are in the same node group, otherwise $g_{i1,i2} = 0$. By definition, $g_{i,i} = 1$ since node i is always in the same node group as node i. Then, $\sum_k g_{i,k}$ gives the number of nodes in the node group to which node i belongs. Let γ_i be defined as follows:

$$\gamma_i = \begin{cases} \sum_k g_{i,k} + 1 & \text{if head of node } i \text{ is allocated alone,} \\ \sum_k g_{i,k} & \text{otherwise.} \end{cases} \tag{9}$$

Let β_i be the unfolding factor for a processor on which the head of node i is allocated. Then β_i is calculated as

$$\beta_i = \max_{\substack{i1 \\ g_{i,i1}=1}} \gamma_{i1}. \tag{10}$$

The 'max' operation guarantees that the unfolding factors for all the nodes in each node group will be the same.

When the computation of a node is greater than the iteration period, the node must be divided into *bodies* as well as a head and tail as illustrated in Fig. 4(b). In this case, the unfolding factor is increased by the number of bodies of the node. Therefore, γ_i is redefined as

$$\gamma_i = \begin{cases} \sum_k (w_k + 1) g_{i,k} + 1 & \text{if head of node } i \text{ is allocated alone,} \\ \sum_k (w_k + 1) g_{i,k} & \text{otherwise} \end{cases} \tag{11}$$

where w_k is the number of bodies of node k. β_i can be calculated as in (10).

5.2 ILP model for processor allocation with unfolding factor minimization

First we identify those nodes, in the original set of nodes, whose computation times cross a multiple of the iteration period. Let $N1$ denote the set of such nodes. The $N1$ nodes are divided into heads, bodies, and tails as discussed above. Let $S2$, $S3$, and $S4$ denote the set of heads, the set of tails, and the set of bodies, respectively. The allocation model operates on the set of nodes $M = (N - N1) \cup S2 \cup S3 \cup S4$ where N is the original set of nodes.

The following terminology is used.

- A binary variable $y_{d,p} = 1$ if $d \in M$ is allocated to processor p, otherwise $y_{d,p} = 0$.
- j is a time step.
- t_d is the type of processor to which computation d is assigned.
- tl_i is the tail of node i.
- P^i is the processor to which the head of node i is allocated.
- w_i is the number of bodies of node i.
- T_d is the time at which computation d starts.
- c_d is the time duration of computation d.
- I_t is the set of computations which are assigned to a processor of type t.
- PS_t is the set of M_t processors of type t.
- K is a sufficiently large positive integer.

Since all the heads are allocated at the end of the iteration cycle, they would never be allocated to an identical processor. Therefore, we can fix the allocation of heads prior to the solution of the processor allocation ILP model. The parameter P^i denotes the processor to which the head of node i is allocated. This simplifies the computation of node groups, i.e., the values of $g_{i1,i2}$.

We minimize the sum of the unfolding factors (12) subject to the following constraints (13)–(21).

$$\text{Minimize } COST = \sum_{i \in N1} ((w_i + 1)\beta_i + \beta_i^t) \tag{12}$$

$$\sum_{p \in PS_{t_d}} y_{d,p} = 1 \quad \forall d \in (N - N1) \cup S2 \cup S3 \cup S4 \tag{13}$$

$$\sum_{\substack{d \in I_t \\ T_d \bmod T_r \le j < (T_d \bmod T_r) + c_d}} y_{d,p} \le 1 \tag{14}$$

$$\forall j \in [0, T_r - 1],\ p \in PS_t,\ t \in PROC$$

$$g_{i1,i2} \ge y_{tl_{i1},P^{i2}} \tag{15}$$

$$g_{i1,i2} \ge y_{tl_{i2},P^{i1}} \tag{16}$$

$$g_{i1,i2} \ge g_{i1,i} + y_{tl_i,P^{i2}} - 1 \forall i \in N1 \tag{17}$$

$$g_{i1,i2} \ge g_{i1,i} + y_{tl_{i2},P^i} - 1 \forall i \in N1 \tag{18}$$

$$\forall i1, i2 \in N1$$

$$\beta_i \ge \sum_{k \in N1} (w_k + 1) g_{i,k} + 1 - \sum_{k \in N1} y_{tl_k,P^i} \tag{19}$$

$$\forall i \in N1$$

$$\beta_i \ge \beta_k - K(1 - g_{i,k}) \quad \forall k \in N1 \tag{20}$$

$$\beta_i^t \ge \beta_i - 1 - K \sum_{k \in N1} y_{tl_k,P^i} \quad \forall i \in N1. \tag{21}$$

Constraint (13) ensures that each computation is allocated to one processor. Constraint (14) prevents more than one computation from being allocated to the same processor during the same time class. Constraints (15) to (18) compute the value of $g_{i1,i2}$. The right-hand side of constraint

Table 3 Solution Comparison for the EWF Example

	Our solutions				Solutions in [11]			
T_r	$*$	$+$	$*$p	$+$	$*$	$+$	$*$p	$+$
16	2	3	1	3	2	3	1	3
17	2	2	1	2	2	2	2	2
18	2	2	1	2			1	2
19	1	2	1	2	1	2		
28	1	1	1	1				

(19) is equal to γ_i, since the last term becomes 0 if the head of node i is allocated alone and 1 otherwise. Constraints (19) and (20) find the maximum β over all the nodes in the same node group. If there exists a node group where a head is allocated alone, then there must exist a tail which is allocated to a processor to which no head is allocated. β_i^t is the unfolding factor of the processor to which a tail would be allocated alone. It is computed by (21). The cost function (12) is minimized by minimizing the sum of β_i, $w_i\beta_i$, and β_i^t. These represent the unfolding factors of the head, body, and tail of a node respectively.

6 Results

We simulated several DFGs to prove the effectiveness of our models. All the ILP models were solved using the ILP solver GAMS/OSL [25] on a SparcStation 2. To show that our model is able to derive optimal solutions, it is applied to the scheduling of the 5th order elliptic wave filter (EWF) which has been used in [5][7]-[13]. In this case, a single processor type is assumed for each operation type, i.e., either nonpipelined multiplier and adder (symbolized as '$*$' and '$+$') or pipelined multiplier and adder ('$*$p' and '$+$'). The specification of these processors are the same as in [11]. The number of processors required in each case are equal to or better than those in [11]. Shown in Table 3 are the numbers of processors in the solution for each iteration period, T_r, which compares to the latency f as described in [11]. Though [11] shows the result of resource-constrained scheduling, our model derived the same results for most cases and a better result for one case. With 1 pipelined multiplier and 2 adders, the approach in [11] required 18 units for the iteration period while our approach requires 17 units of time for the iteration period for the same resource constraints.

We scheduled several benchmarks using our models for a given iteration period with the components shown in Tables 1 and 2. The 4th order lattice and Jaumann filter benchmarks have been used in [7], the 4 stage pipelined lattice filter benchmark has been used in [8], and the 16 point FIR filter benchmark has been used in [5]-[8]. Table 4 contains: the specified iteration period; the number of processors of each type obtained by our solution; the cost of each solution; and the CPU time in seconds required to calculate each solution. For example, in the case of the 4 stage pipelined lattice filter with a given iteration period $T_r = 3$, the synthesized architecture consists of two A_{bp} adders, seven A_{ds} adders, five M_{bp} multipliers, and six $v_{bp,ds}$ converters.

Table 4 Time Assignment Benchmarks

T_r	Architecture	Cost	CPU
4th Order Lattice Filter			
14	$3A_{bp}$, $2M_{bp}$	821	1.58
15	$2A_{bp}$, M_{bp}	437	3.15
16	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$	409	18.0
17	A_{bp}, M_{bp}	384	21.2
18	A_{hp}, A_{ds}, M_{hp}, $v_{hp,ds}$, $v_{ds,hp}$	223	11.9
5th Order Wave Elliptic Filter			
25	$3A_{bp}$, M_{bp}	490	3.16
26	$2A_{bp}$, M_{bp}	437	26.2
27	A_{bp}, $2A_{hp}$, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$	428	658
28	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$	409	417
4th Order Jaumann Filter			
16	$2A_{bp}$, M_{bp}	437	14.9
17	A_{bp}, M_{bp}	384	14.3
18	A_{bp}, M_{bp}	384	39.9
19	$2A_{hp}$, M_{hp}	211	24.7
20	$2A_{hp}$, M_{hp}	211	48.6
4 stage Pipelined Lattice Filter			
3	$2A_{bp}$, $7A_{ds}$, $5M_{bp}$, $6v_{bp,ds}$	1827	23.6
4	$2A_{hp}$, $7A_{ds}$, $4M_{bp}$, $2v_{bp,hp}$, $6v_{bp,ds}$, $v_{hp,bp}$, $v_{hp,ds}$	1440	58.2
5	$9A_{ds}$, $3M_{bp}$, $9v_{bp,ds}$, $2v_{ds,bp}$	1091	40.6
6	$8A_{ds}$, $2M_{bp}$, M_{hp}, $6v_{bp,ds}$, $v_{hp,ds}$, $v_{ds,bp}$, $v_{ds,hp}$	917	77.2
7	$7A_{ds}$, $2M_{bp}$, M_{ds}, $6v_{bp,ds}$, $v_{ds,bp}$	818	105
16 Point FIR Filter			
1	$60A_{ds}$, $8M_{bp}$, $24v_{bp,ds}$, $24v_{ds,bp}$	3200	3.53
2	$30A_{ds}$, $4M_{bp}$, $12v_{bp,ds}$, $12v_{ds,bp}$	1600	5.65
3	$20A_{ds}$, $3M_{bp}$, $8v_{bp,ds}$, $8v_{ds,bp}$	1177	7.85
4	$15_{A}ds$, $2M_{bp}$, $6v_{bp,ds}$, $6v_{ds,bp}$	800	7.25
5	$12A_{ds}$, M_{bp}, $3M_{ds}$, $3v_{bp,ds}$, $3v_{ds,bp}$	685	20.4

Table 5 contains the results of the processor allocation ILP model. This table shows the minimum unfolding factor necessary to achieve the processor allocation and the CPU time required to solve the ILP model. B is the sum of unfolding factors of all the processors and max β is the maximum unfolding factor. Bars ('—') in both of these two columns mean that the processor allocation is obvious since the number of processors and the number of converters are 1. A bar in the column of max β means there exists no node computation which crosses a multiple of the iteration period and therefore B is 0.

7 Conclusion

We have proposed two new ILP models for the time-constrained scheduling problem. Our models perform module selection and data format conversion while automatically retiming, pipelining, and unfolding the DFG in an implicit manner. We have run several benchmarks to prove the utility of these models. The ILP model is very attrac-

Table 5 Processor Allocation Benchmarks

DFG	T_r	B	max β	CPU Time
4th Order Lattice Filter	14	0	—	0.57
	15	0	—	0.86
	16	—	—	—
	17	—	—	—
	18	—	—	—
5th Order Wave Elliptic Filter	25	0	—	0.70
	26	0	—	0.57
	27	0	—	0.83
	28	—	—	—
4th Order Jaumann Filter	16	0	—	0.75
	17	—	—	—
	18	—	—	—
	19	0	—	0.57
	20	0	—	0.83
4 stage Pipelined Lattice Filter	3	30	4	2.83
	4	18	3	4.12
	5	57	4	25.36
	6	15	2	2.24
	7	31	3	24.31
16 Point FIR Filter	1	344	4	—
	2	126	5	804.71
	3	87	4	233.46
	4	45	3	9.03
	5	64	4	52.35

tive because we can easily add additional constraints to our models to impose new requirements such as minimum latency, minimum interprocessor communications, minimum registers, and low power.

References

[1] M. C. McFarland, A. C. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," *Proc. of the IEEE*, vol. 78, pp. 301–318, Feb. 1990.

[2] M. Renfors and Y. Neuvo, "The Maximum Sampling Rate of Digital Filters under Hardware Speed Constraints," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 196–202, Mar. 1981.

[3] K. K. Parhi and D. G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding," *IEEE Trans. Computers*, vol. C-40, pp. 178–195, Feb. 1991.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.

[5] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 661–679, June 1989.

[6] N. Park and A. C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," *IEEE Trans. Computer-Aided Design*, vol. 7, Mar. 1988.

[7] S. M. Heemstra de Groot, S. H. Gerez, and O. E. Herrmann, "Range-Chart-Guided Iterative Data-Flow Graph Scheduling," *IEEE Trans. Circuits Syst.-I: Fund. Theory & Appl.*, vol. CAS-39, pp. 351–364, May 1992.

[8] C.-Y. Wang and K. K. Parhi, "Resource Constrained Loop List Scheduler for DSP Algorithms," *Journal of VLSI Signal Processing, Special issue of VLSI Design Methodologies for Digital Signal Processing Systems*, to appear in 1994.

[9] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-10, pp. 464–475, Apr. 1991.

[10] C. H. Gebotys, "Synthesis of Throughput-Optimized Multichip Architectures," in *Proc. IEEE Custom Integrated Circuits Conf.*, San Diego, pp. 5.2.1–5.2.4, May 1993.

[11] C. H. Gebotys and R. J. Gebotys, "Optimal Mapping of DSP Applications to Architectures," in *Proc. 26th Hawaii Int. Conf. System Sciences*, pp. 116–123, 1993.

[12] C. H. Gebotys and M. I. Elmasry, "Global Optimization Approach for Architecture Synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-12, pp. 1266–1278, Sept. 1993.

[13] C.-T. Hwang and Y.-C. Hsu, "Zone Scheduling," *IEEE Trans. Computer-Aided Design*, vol. CAD-12, pp. 926–934, July 1993.

[14] A. Bachmann, M. Schöbinger, and L. Thiele, "Synthesis Methods for Domain Specific Multiprocessor Systems including Memory Design," in *VLSI Signal Processing, VI*, pp. 417–425, 1993.

[15] L. E. Lucke and K. K. Parhi, "Generalized ILP Scheduling and Allocation for High-Level DSP Synthesis," in *Proc. IEEE Custom Integrated Circuits Conf.*, San Diego, pp. 5.4.1–5.4.4, May 1993.

[16] C. E. Leiserson *et al.*, "Optimizing Synchronous Circuitry by Retiming," in *3rd Caltech Conf. on VLSI*, Pasadena, CA, pp. 87–116, Mar. 1983.

[17] J. Jump and S. Ahuja, "Effective Pipelining of Digital Systems," *IEEE Trans. Computers*, vol. C-27, pp. 855–865, Sept. 1978.

[18] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures," *IEEE Design & Test of Computers*, vol. 8, pp. 40–51, June 1991.

[19] M. Ishikawa and G. D. Micheli, "A module Selection Algorithm for High-Level Synthesis," in *Proc. of the IEEE Int. Symp. Circuits and Systems*, Singapore, pp. 1777–1780, June 1991.

[20] A. H. Timmer and J. A. G. Jess, "Execution Interval Analysis under Resource Constraints," in *Proc. of the IEEE Int. Conf. on Computer Aided Design*, pp. 454–459, 1993.

[21] K. K. Parhi, "A Systematic Approach for Design of Digit-Serial Processing Architectures," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 358–375, Apr. 1991.

[22] L. E. Lucke and K. K. Parhi, "Data-Flow Transformations for Critical Path Reduction in High-Level DSP Synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-12, pp. 1063–1068, July 1993.

[23] E. F. Girczyz, "Loop Winding—A Data Flow Approach to Functional Pipelining," in *Proc. of the IEEE Int. Symp. Circuits and Systems*, Philadelphia, pp. 382–385, May 1987.

[24] D. A. Schwartz and T. P. Barnwell, III, "Cyclo-static solutions: Optimal multiprocessor realizations of Recursive Algorithms," *VLSI Signal Processing II*, 1986.

[25] A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide, Release 2.25*. South San Francisco, CA: The Scientific Press, 1992.