Synthesis of Control Circuits in Folded Pipelined DSP Architectures

Keshab K. Parhi, *Senior Member, IEEE,* Ching-Yi Wang, *Student Member, IEEE,* and Andrew P. Brown

Abstract—**This paper presents a *systematic folding transformation* technique to fold any arbitrary signal processing algorithm data-flow graph to a hardware data-flow architecture, for a specified folding set and specified technology constraints. The folding set specifies the processor in which and the time partition at which the task is executed. The folding set is typically obtained by performing scheduling and resource allocation for the algorithm data-flow graph and the specified iteration period. The technology constraints imposed on the hardware architecture (i.e., the level of pipelining and the implementation style of each processor) are also assumed to be known. The folding technique is used to derive the control circuitry of the hardware architecture (including registers, switches, and interconnections). We derive conditions for the validity of a specified folding set, and present approaches to generate the dedicated architecture using systematic folding of tasks to operators. We propose automatic *retiming* and *pipelining* of algorithms described by data-flow graphs *for folding*. The folding algorithm is applied after preprocessing the data-flow graph (DFG) for automated pipelining and retiming. Our folding algorithm can accommodate single or multiple implementation styles and single or multiple computation clocks, and applies to folding of regular and irregular data-flow graphs.**

I. Introduction

DIFFERENT real-time digital signal processing (DSP) applications require different implementation styles. For example, video, radar, and image processing applications require very high sample rates, while speech and communications applications require low or moderate sample rates. The wide range of real-time sample rate requirements for DSP applications encourages study of a family of circuits using many possible implementation styles.

The system sample rate requirement does not always dictate the target architecture or the implementation style. A low sample rate application with a functionally complex algorithm and a high sample rate application with a simple algorithm may need to be clocked at about the same rate (and may require comparable number of processors). As a simple example, consider the two systems: the first system implements a ten-tap finite impulse response (FIR) filter and requires an iteration period (or sample period) of ten units of time, and the second system implements a two-tap FIR filter and requires a sample or iteration period of two units of time. If we assume that each multiply–add operation (with no internal pipelining) requires one unit of time, then we can *fold* all ten operations to a single multiply–add operator for the first system and fold the two operations to a single multiply–add operator in the second system. By adding one more internal pipeline stage in the multiply–add operator (i.e., by using a two-stage pipelined multiplier), we can reduce the iteration period in both systems by a factor of 2. The process of executing many algorithm operations in one hardware operator is referred to as folding.

Systematic folding of regular algorithms to systolic arrays has been studied in [1]–[6]; synthesis of systolic architectures is carried out using the appropriate iteration vector, a linear processor space, and a linear scheduling vector. This folding of regular data-flow graphs is reviewed in Section III. Folding of any arbitrary irregular algorithm to a hardware architecture for any arbitrarily specified processor and scheduling functions (or equivalently a folding set) is an important problem. Although folding of arbitrary data-flow signal processing programs has been addressed for simple cases [7], [8], a systematic folding technique has so far not been studied, to the best of our knowledge. In this paper, we propose systematic techniques for synthesizing a hardware architecture for a specified irregular algorithm data-flow graph and a folding set. Our algorithms determine the entire control circuitry, including the switches, interconnections, and registers, for the hardware data-flow architecture.

To design a DSP circuit for a real-time application, one needs to consider a wide range of implementation styles. In some DSP applications, different processors may require different implementation styles, i.e., some operators may be implemented using bit-serial [9]–[12] while others may be implemented using bit-parallel or digit-serial [12]–[15] style. Different processors in a system may require different levels of pipelining. This might sometimes reduce the number of latches used in the system, at the expense of an increase in the number of computation clocks. These examples indicate that both the technology constraints (i.e., level of pipelining, implementation styles of processors, and the number of computation

Manuscript received March 5, 1991; revised August 22, 1991. This work was supported by the Army Research Office under Contract DAAL03-90-G-0063, the Office of Naval Research under Contract N00014-91-J-1008, and Texas Instruments Incorporated.

K. K. Parhi and C.-Y. Wang are with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455.

A. P. Brown is with 3M Corporation, St. Paul, MN 55144.

IEEE Log Number 9104045.

clocks) and the iteration rate constraints dictate the dedicated target architecture for a specified algorithm.

The iteration rate requirements and the technology constraints are used to perform scheduling and resource allocation [16]–[28]. Scheduling and allocation determine the folding set; the folding set specifies the processor which executes the task and the time partition at which the task is executed. The *time partition* is defined to be the time at which a task is executed modulo the iteration period. For example, if the iteration period is 16 units, then there are 16 time partitions (0 through 15). If a task is executed at cycle 18, then the time partition for that task is 2 (which is 18 modulo 16). In this paper, we address the synthesis of control circuits in folded, pipelined, DSP data paths for the case where the exact scheduling time of a task is unknown but its time partition is known [29]. Of course, when the exact scheduling times are known, the simpler problem reduces to the connection binding problem in high-level synthesis of DSP systems. Our algorithms can also accommodate single or multiple implementation styles, and single or multiple computation clocks.

Often a specified folding set may not be valid. Thus it is important to verify the validity of a specified folding set. While this verification is not so important for algorithms without feedback, it is more important for algorithms with feedback. We derive conditions to verify the validity of the folding sets in algorithms containing feedback. A new application of retiming and pipelining approaches is formulated for the folding problem. Automated retiming and pipelining have been used for clock period minimization in data-flow graphs [30]. New constraints are formulated for retiming and pipelining. The algorithm data-flow graph is preprocessed through the retiming and pipelining algorithm prior to folding. This preprocessing can easily detect an infeasible folding set.

In addition to connection binding of DSP circuits, the folding transformation technique can also fold bit-parallel or digit-serial architectures to bit-serial architectures (such folding operations require folding of switches). Previously we had proposed a systematic unfolding transformation technique to unfold any bit-serial architecture to digit-serial architectures [14], [15], [31]. The folding transformation can perform the reverse operation. It is important to note that while the digit-serial architecture obtained using the unfolding algorithm is unique, the folded bit-serial architecture obtained from the digit-serial architecture is not unique. A family of bit-serial architectures can be obtained from the same digit-serial architecture using different folding sets. Of course, it is always possible to get back the original bit-serial architecture using the appropriate folding set (as noted in [14]). Furthermore, all the folded bit-serial architectures are related in the sense that their unfolded versions are pipelined and retimed versions of each other.

The organization of this paper is as follows. Section II describes the data-flow graph model. Section III reviews the folding of regular data-flow graphs using linear algebra techniques. Sections IV and V introduce algorithms for control circuit generation for circuits with uniform implementation styles using single clock and multiple clocks, respectively. Section IV also presents the formulation of retiming and pipelining for folding. Sections VI and VII address algorithms for control circuit generation for circuits employing nonuniform implementation styles with single clock or multiple clocks, respectively.

II. Data-Flow Graph Model

The synchronous algorithm data-flow graph (DFG) model considered in this paper is identical to that in [32]. The tasks of the DSP algorithm are assumed to be executed repetitively. Each node in the algorithm DFG represents an algorithm operation or task, and any arc $U \rightarrow V$ with i delays (where i is any nonnegative integer) implies that the result of lth iteration of U is used to execute the $(l + i)$th iteration of V. The arcs with delays dictate the inter-iteration precedence constraints, whereas the arcs without delays represent the intra-iteration precedence constraints.

The folded hardware architecture is also described by a hardware DFG. In the hardware DFG, H_u denotes the hardware operator that executes the operation U. All operations processed by H_u collectively form an ordered set S_u, where the ordering of the elements represents a unique execution sequence of the tasks in the folded operator. Some operations in the set S_u can be null operations denoted as ϕ. If N_u represents the number of operations in S_u, then the ordering of the operations ranges from 0 to $N_u - 1$ and each execution order number corresponds directly to a time partition. Let P_u denote the level of pipelining of the hardware operator H_u. Our objective is to fold the original algorithm DFG for a specified folding set to obtain the hardware architecture DFG. We are primarily interested in obtaining the interprocessor communication links and the storage units required in these links. A delay or register in the hardware DFG represents a storage unit.

III. Folding of Regular Data-Flow Graphs

In this section, we review the folding of regular DFG's to systolic architectures using algebraic techniques [1]–[6]. The approach illustrated in this section is only different from previously published results in the sense that some of the edges in the algorithm DFG can contain delays. The new schedule inequalities are derived and made use of. Although not the main objective of this paper, the folding of regular DFG's is reviewed to motivate the folding of irregular DFG's. Another objective of considering the folding of regular DFG's is to show that the proposed folding algorithms for irregular DFG's apply to the folding of regular DFG's in obvious ways (more on this in examples 1 and 2 in Section IV).

A DFG is regular if the existence of an edge $\boldsymbol{I}_1 \rightarrow \boldsymbol{I}_1 + \boldsymbol{e}$ with i delays implies the existence of the edge $\boldsymbol{I}_1 + \boldsymbol{I}' \rightarrow \boldsymbol{I}_1 + \boldsymbol{I}' + \boldsymbol{e}$ with the same number of i delays for all

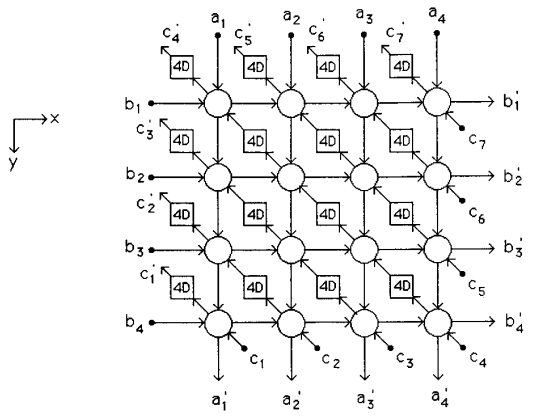

Fig. 1. A regular data-flow graph.

indexes $\boldsymbol{I}'$ permissible within the index set of the DFG. The notation $\boldsymbol{I}_1$ represents an index location. Fig. 1 shows an example of a two-dimensional regular DFG. The coordinates are represented by an index vector $\boldsymbol{I} = [x, y]^T$. In this DFG, there are three fundamental edges: $\boldsymbol{e}_1 = [0 \quad 1]^T$ with 0 delays, $\boldsymbol{e}_2 = [1 \quad 0]^T$ with 0 delays, and $\boldsymbol{e}_3 = [-1 \quad -1]^T$ with 4 delays. The edges $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, respectively, represent the edges in the y and x directions, and the edge $\boldsymbol{e}_3$ represents the diagonal edge (with negative direction).

The folding operation is described by an iteration vector $\boldsymbol{d}$. In other words, all operations at index locations with index difference $\boldsymbol{d}$ are executed by the same processor. For example, an iteration vector $[0 \quad 1]^T$ implies the folding of all operations located in parallel to the y axis (i.e., all operations with the same x index value but different y index values) into the same processor. The synthesized hardware architecture is also described by a regular data-flow graph with processors located along one dimension. A processor space, denoted by $\boldsymbol{P}^T$, describes the mapping of the task to a processor. A third notation needs to be introduced: the scheduling vector, denoted as $\boldsymbol{S}^T$. The task at index $\boldsymbol{I}$ is executed by the processor located at $\boldsymbol{P}^T\boldsymbol{I}$ at time unit $S^T I$.

Simple algebraic equations can be derived for consistency in choosing the parameters $\boldsymbol{P}^T$ and $\boldsymbol{S}^T$ for given $\boldsymbol{d}$. First, consider the folding of two tasks, displaced by the iteration vector, to the same processor. The tasks at index locations $\boldsymbol{I}$ and $\boldsymbol{I} + \boldsymbol{d}$ are executed at time units $\boldsymbol{S}^T\boldsymbol{I}$ and $\boldsymbol{S}^T\boldsymbol{I} + \boldsymbol{S}^T\boldsymbol{d}$. Since these two tasks cannot be executed by the same processor simultaneously, the constraint $\boldsymbol{S}^T\boldsymbol{d} \neq 0$ must be satisfied. Furthermore, since a processor executes these two tasks with time interval of $|\boldsymbol{S}^T\boldsymbol{d}|$, the hardware utilization efficiency (HUE) of the architecture is given by $1/|\boldsymbol{S}^T\boldsymbol{d}|$. Since the tasks located at $\boldsymbol{I}$ and $\boldsymbol{I} + \boldsymbol{d}$ are executed by the same processor, we must satisfy $\boldsymbol{P}^T\boldsymbol{I} = \boldsymbol{P}^T(\boldsymbol{I} + \boldsymbol{d})$, or equivalently, $\boldsymbol{P}^T\boldsymbol{d} = 0$. In words, the processor space vector is orthogonal to the iteration vector.

Now we can describe the mapping of the edges in the algorithm regular DFG to the hardware regular DFG. An edge $\boldsymbol{e} = \boldsymbol{I}_2 - \boldsymbol{I}_1$ in the algorithm DFG implies an edge from $\boldsymbol{P}^T\boldsymbol{I}_2$ to $\boldsymbol{P}^T\boldsymbol{I}_1$, i.e., in an edge in the direction $\boldsymbol{P}^T\boldsymbol{e}$ in the hardware DFG.

To calculate the number of storage units associated with the edge in the hardware DFG, we calculate the difference in the execution time units of two tasks connected by the edge in the algorithm DFG. For simplicity, assume each task can be executed with one time unit. Consider the edge from $\boldsymbol{I}$ to $\boldsymbol{I} + \boldsymbol{e}$ with i delays. The first task at $\boldsymbol{I}$ is executed at time instance $\boldsymbol{S}^T\boldsymbol{I}$ by the processor $\boldsymbol{P}^T\boldsymbol{I}$. The first task at $\boldsymbol{I} + \boldsymbol{e}$ is executed by the processor at $\boldsymbol{P}^T(\boldsymbol{I} + \boldsymbol{e})$ at time unit $\boldsymbol{S}^T(\boldsymbol{I} + \boldsymbol{e})$. Since two executions in any processor are separated by $|\boldsymbol{S}^T\boldsymbol{d}|$, the two iterations of the task are separated by $N'|\boldsymbol{S}^T\boldsymbol{d}|$, where N' is the number of nodes or tasks in the DFG which are mapped to the same processor. Therefore, the $(i + 1)$st iteration of the task at $\boldsymbol{I} + \boldsymbol{e}$ is executed at time unit $\boldsymbol{S}^T(\boldsymbol{I} + \boldsymbol{e}) + iN'\boldsymbol{S}^T\boldsymbol{d}$. The result of the first task of $\boldsymbol{P}^T\boldsymbol{I}$ needs to be held until the $(i + 1)$st task of $\boldsymbol{P}^T(\boldsymbol{I} + \boldsymbol{e})$. Thus the number of storage units for the edge in the hardware DFG is given by

$$\boldsymbol{S}^T(\boldsymbol{I} + \boldsymbol{e}) + iN'\boldsymbol{S}^T\boldsymbol{d} - \boldsymbol{S}^T\boldsymbol{I} = \boldsymbol{S}^T\boldsymbol{e} + iN'\boldsymbol{S}^T\boldsymbol{d}.$$

To ensure proper pipelining, the number of storage units must be greater than or equal to 1. In inequality form, we can write

$$\boldsymbol{S}^T\boldsymbol{e} + iN'\boldsymbol{S}^T\boldsymbol{d} \geq 1.$$

These inequalities are referred to as *schedule inequalities*. In solving these inequalities, the components of the schedule vector $\boldsymbol{S}^T$ are chosen to be as small as possible to minimize $\boldsymbol{S}^T\boldsymbol{d}$ (which in turn maximizes the HUE). This completes the theory of folding of regular DFG's.

We now illustrate the use of the folding algorithm described above using the DFG in Fig. 1. In this example, $N' = 4$. Let $\boldsymbol{S}^T = [s_1, s_2]$, and let $\boldsymbol{d}^T = [d_1 \quad d_2]$. The schedule inequalities for the three edges can be expressed by

$$\text{edge } (1 \quad 0)^T \text{ with 0 delays: } s_1 \geq 1$$

$$\text{edge } (0 \quad 1)^T \text{ with 0 delays: } s_2 \geq 1$$

$$\text{edge } (-1 \quad -1)^T \text{ with 4 delays:}$$

$$-s_1 - s_2 + 16(s_1d_1 + s_2d_2) \geq 1.$$

We illustrate folding of the DFG in Fig. 1 with two different folding sets (or, equivalently, for two different iteration vectors). Let $\boldsymbol{d}^T = [0 \quad 1]$, i.e., all tasks located on a line parallel to the y axis are mapped to the same processor. The processor space should be orthogonal to the iteration vector space. Choose $\boldsymbol{P}^T = [1 \quad 0]$. Solving the above inequalities for $d_1 = 0$ and $d_2 = 1$ yields $s_1 = 1$ and $s_2 = 1$. The HUE, $\boldsymbol{S}^T\boldsymbol{d}$, is 1. This architecture, therefore, is fully hardware efficient. The edges in the algorithm DFG are then mapped to the hardware DFG. For the edge $\boldsymbol{e}_2 = [1 \quad 0]^T$ with $i_2 = 0$ delays, $\boldsymbol{P}^T\boldsymbol{e}_2 = 1$ and $\boldsymbol{S}^T\boldsymbol{e}_2 + i_2\boldsymbol{S}^T\boldsymbol{d} = 1$. Therefore, in the hardware DFG, we introduce an edge between two consecutive processors in the positive direction with one delay. Similarly for the edge $\boldsymbol{e}_1 = [0 \quad 1]^T$ with $i_1 = 0$ delays, $\boldsymbol{P}^T\boldsymbol{e}_1 = 0$ and $\boldsymbol{S}^T\boldsymbol{e}_1$

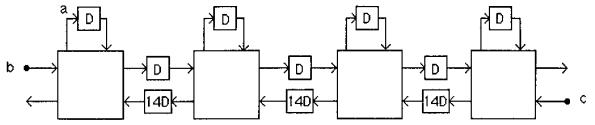

Fig. 2. Folded systolic array for the iteration vector $\boldsymbol{d}^T = [0, 1]$.

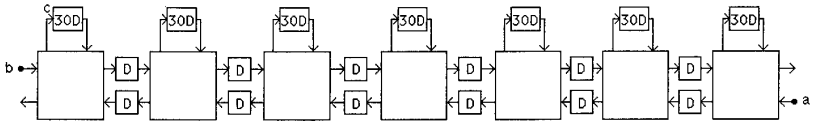

Fig. 3. Folded systolic array for the iteration vector $\boldsymbol{d}^T = [1, 1]$.

$+ i_1 \boldsymbol{S}^T\boldsymbol{d} = 1$. This implies a self loop in each processor with 1 delay. For the edge $[-1 \quad -1]$, the hardware DFG has an edge in the direction -1 with 14 storage units. The complete synthesized hardware architecture is shown in Fig. 2.

Consider another folding set for the algorithm DFG in Fig. 1. Let the iteration vector be given by $\boldsymbol{d} = [1 \quad 1]^T$. Choose the processor space given by $\boldsymbol{P}^T = [1 \quad -1]^T$ to satisfy $\boldsymbol{P}^T\boldsymbol{d} = 0$. The scheduling vector components can be obtained by solving the inequalities for $d_1 = d_2 = 1$, and are given by $s_1 = s_2 = 1$. The quantity $\boldsymbol{S}^T\boldsymbol{d}$ is 2, and this hardware architecture is utilized only half the time (since the HUE $= 1/2$). The edges are similarly mapped to obtain the final architecture in Fig. 3. A null operation is carried out between any two useful operations by the architecture in Fig. 3. The reader can verify the calculation of the number of storage units.

IV. Folded Circuits with Single Implementation Style and Single Clock

In this section and in the remainder of the paper, we consider folding of algorithms described by irregular DFG's. The nice algebraic techniques (which were applied to fold regular DFG's) can no longer be applied to these cases. This is because the folding set, the processor space, and the scheduling time can no longer be expressed as linear combinations of index coordinates of the tasks. The folding of these irregular algorithms is now considered.

This section presents the folding transformation technique for circuits using single implementation style and single clock. First we present the folding of algorithm DFG's with no feedback. Then we perform folding of algorithms containing feedback. We assume that data on each system input will be valid for N clock cycles before changing (where N operations are folded to the operator that processes the system input). For example, if an operator processing a system input folds three operations, then we assume that the input $x(0)$ is valid from 0 to $3T$, $x(1)$ is valid from $3T$ to $6T$, etc., where T is the clock period of the hardware operator. In other words, the system input is N-slow.

Before we consider folding, we illustrate the representation of folding sets. Consider a DFG with two types of operations, add and multiply. Let $S_A = \{A2, A1, A4, A3\}$ and $S_M = \{M1, M3, \phi, M2\}$ represent the (ordered) folding sets for the add and multiply, respectively. In this representation, the multiply operator implements one null operation every $\{(4l + 2)\}$nd cycle. Equivalently, we can also write $A1 = \{S_A|1\}$, $A_2 = \{S_A|0\}$, etc. This notation of folding set representation is used in the remainder of the paper.

A. Folding of Nonrecursive Data-Flow Graphs

Consider an arc $U \rightarrow V$ with i delays as shown in the DFG in Fig. 4. Let U and V, respectively, be executed by operators H_u and H_v with folding order u and v. Folding of the arc $U \rightarrow V$ results in a communication link between H_u and H_v. The input to each operator is a N-way switch and the hardware operator H_u connects to one switch input of H_v with the switching instance $(Nl + v)$.

Let the lth iteration of the task of U (i.e., U_l) start in cycle $[N_u l + u]$, where u is the execution, or folding order number of U within the set S_u. The operation of U_l is then completed in cycle $[N_u l + P_u + u]$ (where P_u is the level of pipelining of H_u). The result of U_l is used for the execution of the $(l + i)$th iteration of the task of V, V_{l+i}, which starts in cycle $[N_v(l + i) + v]$, where v is the execution order number of V within the set S_v. When this arc is subjected to the folding algorithm, the output of H_u needs to be stored for $[N_v(l + i) - N_u l - P_u + v - u]$ cycles. In order for the number of delays to be iteration independent (i.e., independent of l), N_u must be equal to N_v. If we let $N = N_u = N_v$, then the number of required storage units, or *folded arc delays*, must be

$$D_F(U \rightarrow V) = Ni - P_u + v - u.$$

If the number of folded arc delays is negative, then add multiples of N until it becomes nonnegative; this effectively implies the pipelining of the algorithm DFG (where the arc $U \rightarrow V$ now has sufficient delays to make folded arc delays nonegative). Note that adding $i'N$ delays to an arc $U \rightarrow V$ to make $D_F(U \rightarrow V)$ nonnegative is equivalent to adding i' delays to the arc $U \rightarrow V$ in the algorithm DFG.

If the number of folded arc delays is greater than N, then it is possible to reduce the number of latches by using a second slower clock for control circuit implementation.

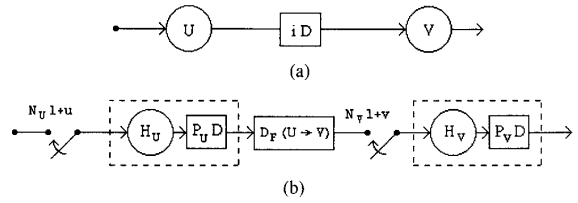

Fig. 4. (a) An arc $U \rightarrow V$ in the algorithm DFG. (b) The associated folded arc in the hardware architecture DFG with $D_F(U \rightarrow V)$ folded arc delays.

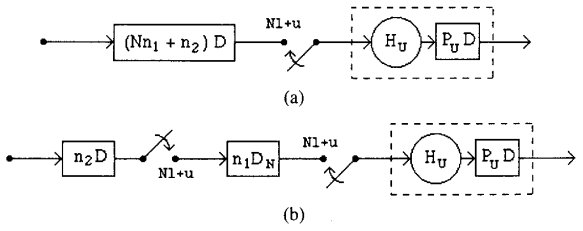

Fig. 5. Reduction of the number of latches in a hardware DFG. ($Nn_1 + n_2$) delays in (a) are reduced to n_2 fast delays and n_1N-slow delays in (b), denoted by D_N.

For example, if the number of latches is ($Nn_1 + n_2$) (where n_2 lies between 0 and ($N - 1$)), then we can reduce the number of latches to ($n_1 + n_2$) as shown in Fig. 5. We notice that the hardware operator H_u processes the signal every ($Nl + u$)th cycle. There is no need to store the variables not sampled by the hardware operator H_u. Thus we can use n_2 delays (clocked with period T, the computation clock period) and n_1 slow delays (clocked with period NT (see Fig. 5)). Such a realization may not always be more efficient (since overhead due to routing another clock can be larger than savings in latches), and should be considered only if appropriate.

Example 1: Folding of regular DFG's: Consider the folding of the DFG of Fig. 1 for the iteration vector $[0 \quad 1]^T$. The folded arc delays can be calculated as

$$D_F([1 \quad 0]^T) = 4(0) - 1 + 1 - 0 = 0$$

$$D_F([0 \quad 1]^T) = 4(0) - 1 + 1 - 0 = 0$$

$$D_F([-1 \quad -1]^T) = 4(4) - 1 + 0 - 2 = 13.$$

These delay calculations are exactly identical to the same in Fig. 2 if we include one operator or functional unit pipelining delay. □

Example 2: Folding of regular DFG's: Consider the folding of the DFG in Fig. 1 for the iteration vector $[1 \quad 1]^T$. In this case, the value of N is 8 (since four useful and four null operations are executed by each processor). In general, $N = N'S^Td$ using the notation in Section III. The folded arc delays are calculated as

$$D_F([1 \quad 0]^T) = 8(0) - 1 + 1 - 0 = 0$$

$$D_F([0 \quad 1]^T) = 8(0) - 1 + 1 - 0 = 0$$

$$D_F([-1 \quad -1]^T) = 8(4) - 1 + 0 - 2 = 29.$$

Once again if we include the one functional unit pipelining delay, the delay calculations in this example and in Fig. 3 match exactly. Thus we have illustrated the use of our algorithm for folding of regular DFG algorithms. □

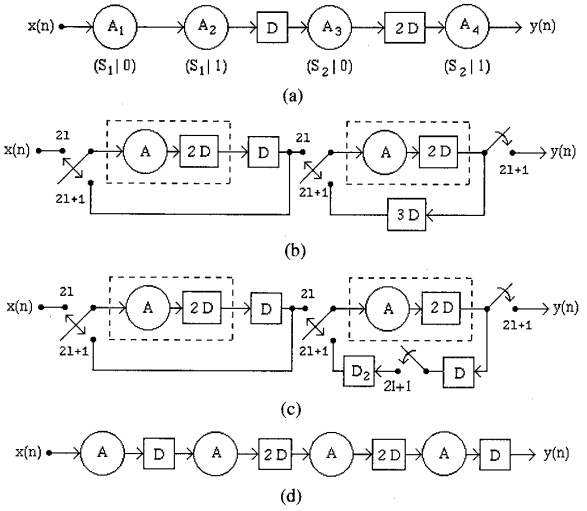

Fig. 6. (a) An algorithm DFG containing no feedback. (b) The folded hardware architecture DFG for the folding set $S_1 = (A_1, A_2)$, $S_2 = (A_3, A_4)$. (c) The architecture DFG of (b) redrawn in (c) using the latch reduction principle; D_2 is a 2-slow latch. (d) The two-unfolded version of the architecture DFG of (b) is an automatically pipelined version of (a).

Example 3: Consider the algorithm flow graph and the folding set specified in Fig. 6(a). All the tasks in this flow graph are of one type (specified as type A); the task A_i is the ith task of type A. We assume the use of a two-stage pipelined hardware "A" operator in the folded architecture. To fold this, we need to calculate the folded arc delays:

$$D_F(A_1 \rightarrow A_2) = 2(0) - 2 + 1 - 0$$

$$= -1 \rightarrow -1 + 2 = 1$$

$$D_F(A_2 \rightarrow A_3) = 2(1) - 2 + 0 - 1$$

$$= -1 \rightarrow -1 + 2 = 1$$

$$D_F(A_3 \rightarrow A_4) = 2(2) - 2 + 1 - 0 = 3.$$

The folded architecture is described by the DFG in Fig. 6(b). If a second slower chock was available for control circuit implementation, the three registers for the arc ($A_3 \rightarrow A_4$) can be realized using one faster register (clocked with period T) and a slow register (clocked with period $2T$) as shown in Fig. 6(c). In this example, two delays were added to the folded arc delays $D_F(A_1 \rightarrow A_2)$ and $D_F(A_2 \rightarrow A_3)$ to make these nonnegative; this is effectively equivalent to adding one delay to each of these arcs in the algorithm DFG. Indeed systematic unfolding of the hardware DFG of Fig. 6(b) leads to the DFG in Fig. 6(d), which is a pipelined version of Fig. 6(a) (for details on the systematic unfolding algorithm, see [14], [15], [31]). Note that the tasks in Fig. 6(d) have not been indexed. One can fold the algorithm DFG in Fig. 6(d) to obtain the folded hardware DFG of Fig. 6(b) or Fig. 6(c); in this case all folded arc delays will be automatically nonnegative. □

Example 4: Consider the folding of the algorithm DFG of Fig. 6(a) for a different folding set $S_1 = [A_1, A_3]$ and $S_2 = [A_2, A_4]$. The folded arc delays are calculated as

$$D_F(A_1 \rightarrow A_2) = 2(0) - 2 + 0 - 0$$
$$= -2 \rightarrow -2 + 2 = 0$$
$$D_F(A_2 \rightarrow A_3) = 2(1) - 2 + 1 - 0 = 1$$
$$D_F(A_3 \rightarrow A_4) = 2(2) - 2 + 1 - 1 = 2.$$

The folded hardware DFG is shown in Fig. 7(a). Note that this folded architecture requires fewer latches than that in Fig. 6(b). The number of latches can be further reduced by one by replacing the two fast inter-processor delays by one slower delay (clocked with twice the period). Fig. 7(b) shows the pipelined flow graph, obtained either by adding the appropriate delays in the algorithm DFG based on the folded arc delay calculations, or by unfolding the hardware DFG of Fig. 7(a) by a factor of 2. □

The two folded architectures in Examples 3 and 4 differ in the number of registers (the architecture of Fig. 7(a) requires fewer latches). Thus, the scheduling and allocation process in synthesis selects more than one possible alternative; then the best alternative could be chosen by examining the properties of the folded architectures for all the alternatives.

An important characteristic in the folded hardware architecture is the *total path delay* in any path, which is the sum of the computation delays (due to pipelining of the operators) and the communication delays (due to the storage units associated with inter-processor communication). There is a relationship between the total path delay in a hardware DFG and the number of delays in that path in the pipelined version of the original algorithm DFG as stated below.

Definition 1: The total path delay (including computation and communication) for a path $U \rightarrow V \rightarrow W$ is defined as

$$D_T(U \rightarrow V \rightarrow W) = u + P_u + D_F(U \rightarrow V) + P_v + D_F(V \rightarrow W) - w$$

where u and w are the folding orders of the tasks U and W, respectively.

Property 1: The total path delay of any path in the hardware DFG is equal to the folding set cardinality (N) times the number of delays in the pipelined version of the original algorithm DFG.

Proof: We consider the pipelined version of the original algorithm DFG (since the folded arc delays calculated using this DFG is always nonnegative). Let N be the cardinality of the folding set, i.e., N operations are folded to one processor. Let $n(U \rightarrow V)$ denote the number of delays in the pipelined version of the original algorithm DFG, i.e., $n(U \rightarrow V) = i(U \rightarrow V) + i'(U \rightarrow V)$, where $i(\cdot)$ is the number of arc delays in the original DFG. $Ni'(\cdot)$ delays were added to make $D_F(U \rightarrow V)$ nonnegative. We need to show that

$$D_T(U \rightarrow V \rightarrow W) = N[n(U \rightarrow V) + n(V \rightarrow W)].$$

To prove this, we use the definition of total path delay and the definition of folded arc delays:

$$D_T(U \rightarrow V \rightarrow W)$$
$$= u + P_u + D_F(U \rightarrow V) + P_v + D_F(V \rightarrow W) - w,$$
$$= u + P_u + Nn(U \rightarrow V) - P_u + v - u + P_v + Nn(V \rightarrow W) - P_v + w - v - w$$
$$= N[n(U \rightarrow V) + n(V \rightarrow W)]. \quad \square$$

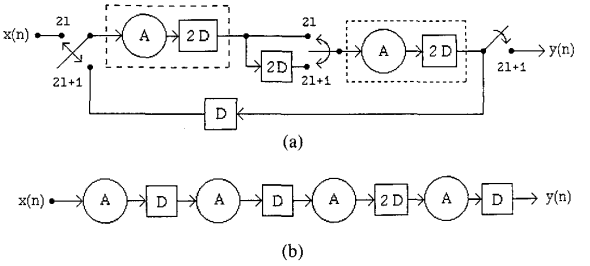

Fig. 7. (a) Another folded architecture DFG obtained by folding the algorithm DFG of Fig. 3(a) and the folding set $S_1 = (A_1, A_3)$ and $S_2 = (A_2, A_4)$. (b) The two-unfolded version of (a) is an automatically pipelined version of the algorithm DFG of Fig. 3(a).

Example 5: We can verify this property using the folded architectures in Fig. 6. The input-to-output total path delay in Fig. 6(b) is

$$0 + P + D_F(A_1 \rightarrow A_2) + P + D_F(A_2 \rightarrow A_3) + P + D_F(A_3 \rightarrow A_4) + P - 1$$
$$= 0 + 2 + 1 + 2 + 1 + 2 + 3 + 2 - 1 = 12$$

which is the same as the folding set cardinality, two, times the number of delays in the pipelined DFG version in Fig. 6(d) (which is 6). □

Often the DFG's may contain an arc which has one or more parallel paths. In such cases, the folding algorithm synchronizes the total path delay of all parallel paths.

Definition 2: The *synchronizing delay* of a path $U \rightarrow V \rightarrow W$, denoted as $[D_S(U \rightarrow V \rightarrow W)]$, is defined as the difference between the total path delay of the path $U \rightarrow V \rightarrow W$ and the folding set cardinality (N) times the number of delays in the path $U \rightarrow V \rightarrow W$ in the original algorithm DFG.

If the algorithm DFG contains parallel paths, then the synchronizing delays of all parallel paths are calculated. Additional delays are added to any one arc in all parallel paths such that the synchronizing delays of all parallel paths are the same (and equal to the maximum path synchronizing delay). This step guarantees simultaneous pipelining of all parallel paths.

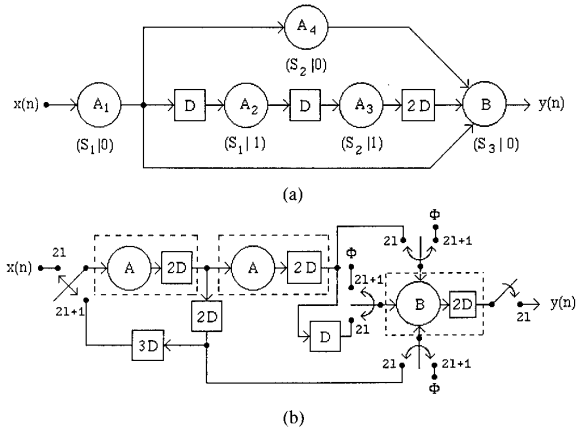

Fig. 8. (a) An algorithm DFG containing parallel paths. (b) The corresponding folded DFG; the number of latches used in this architecture can be reduced by two by using a second clock for control circuit design.

Example 6: Consider the algorithm DFG in Fig. 8. To fold this DFG, we first consider the folded arc delays:

$$D_F(A_1 \rightarrow A_2) = 2(1) - 2 + 1 - 0 = 1$$

$$D_F(A_2 \rightarrow A_3) = 2(1) - 2 + 1 - 1 = 0$$

$$D_F(A_3 \rightarrow B) = 2(2) - 2 + 0 - 1 = 1.$$

$$D_F(A_1 \rightarrow A_4) = 2(0) - 2 + 0 - 0$$

$$= -2 \rightarrow -2 + 2 = 0$$

$$D_F(A_4 \rightarrow B) = 2(0) - 2 + 0 - 0$$

$$= -2 \rightarrow -2 + 2 = 0$$

$$D_F(A_1 \rightarrow B) = 2(0) - 2 + 0 - 0$$

$$= -2 \rightarrow -2 + 2 = 0.$$

The total path delays for the paths $A_1 \rightarrow A_4 \rightarrow B$, $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow B$, and $A_1 \rightarrow B$ are, respectively, 4, 8, and 2. The synchronizing delays for these paths are, respectively, 4, 0, and 2. The maximum synchronizing delay is 4. Thus we need to add four delays to the folded arc delays for $A_1 \rightarrow A_2$, and two delays to the folded arc delays of $A_1 \rightarrow B$. So the modified numbers of delays for these two arcs are:

$$D_F(A_1 \rightarrow A_2) = 1 + 4 = 5$$

$$D_F(A_1 \rightarrow B) = 0 + 2 = 2.$$

Effectively, this process guarantees that two stages of pipelining have been introduced at the feedforward cutsets of the parallel paths. The folded DFG is shown in Fig. 8. We have assumed the computation time of the tasks A and B to be equal. □

We now address folding of operations with external inputs. If a system input is an input to multiple tasks, then some nodes may require delayed inputs (i.e., the input needs to be delayed using synchronizing delays). Let the system input be an input to the operation U. Then calculate the largest *synchronizing delay* of a path from the system input to the node U (let this delay be $D_S(p)$ be associated with path p). Then connect the system input to H_u with $D_S(p)$ delays and sampled with time instance $(Nl + u)$. The number of synchronizing delays could still be reduced, since we assumed that the input is valid for N clock cycles. If the number of synchronizing delays is less than N, then we can connect the input to H_u directly, otherwise we can connect the input to H_u with $(D_S(p) - N + 1)$ delays.

Example 7: Consider the example of a fourth-order FIR filter (implemented using broadcast input) shown Fig. 9(a). In this example, we assume use of identical multiply–add processors, and the tasks are mapped to two ordered sets $[MA1, MA2, MA3]$ and $[MA4, MA5, \phi]$. Note that a null operation ϕ was added to the second ordered set to ensure that the cardinality of both sets are equal. We assume a pipelining level of 3. The final folded version with all of the control circuitry is shown in Fig. 9(b). The reader can verify the synchronizing delay calculations for the inputs. □

So far we have addressed the folding of circuits without switches and have computed the number of storage units using folded arc and synchronizing delays. As stated earlier, the folding technique can also fold digit-serial architectures into bit-serial architectures. Since most digit-serial arithmetic processor architectures contain switches, we need to address folding of circuits containing switches. We first consider the simple algorithm DFG's, where each folded arc delay is nonnegative and all synchronization delays are zero (i.e., the algorithm DFG has already been pipelined). Let an input be switched to a node U at time instance $(Ll + m)$. Then this input should be switched to H_u in the hardware DFG with time instance $[N(Ll + m) + u]$ (recall u is the folding order of U).

Example 8: Consider folding of the algorithm DFG in Fig. 10(a). The reader can verify that both paths $A_1 \rightarrow A_4$ and $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4$ have equal synchronization delays. The folded architecture DFG is shown in Fig. 10(b). The switch instance $(3l + 1)$ in the algorithm DFG is converted to $[4(3l + 1) + 3]$ or $(12l + 7)$. Other switching instances have been converted in a similar way. □

1) Pipelining and Retiming for Folding: The calculation of the synchronizing delays in a manual way as described and used in the above examples is not very useful. Therefore, we propose preprocessing of the DFG using *automated pipelining and retiming for folding*. The objective of this step would be to insert the pipelining latches and to move existing latches to other arcs as appropriate, such that the pipelined version of the DFG leads to positive folded arc delays for all arcs. This preprocessing also automatically takes care of the path delay synchronization.

To automatically pipeline and retime for folding, we follow an approach similar to the retiming problem [30]. The retiming is a general approach to change the number of arc delays in a DFG. In [30], retiming was used to reduce the clock period. Use of pipelining was also

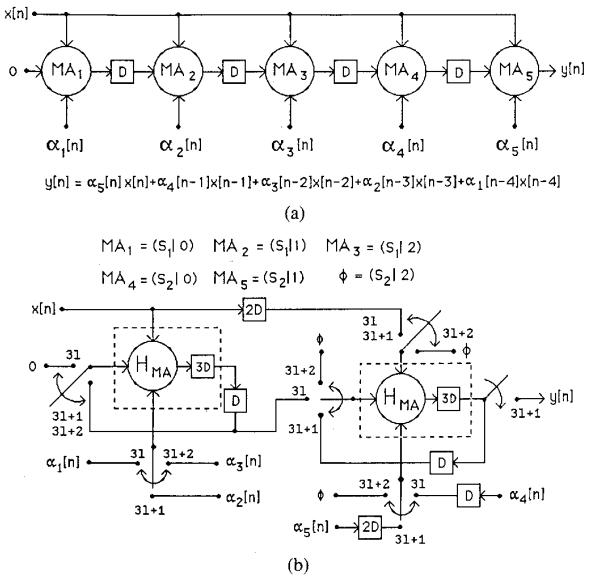

Fig. 9. (a) A fourth-order FIR filter implemented with multiply-add operators. (b) A complete folded architecture of the fourth-order filter using a uniform design style and a single clock. The folding sets are $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5, \Phi]$ with the following parameters: $P_1 = 3$, $N_1 = 3$, $P_2 = 3$, $N_2 = 3$. The set S_2 required a null operation to maintain the same cardinality as S_1. The notation $(S_1|2)$ of an operator implies that this operator is mapped to folded hardware unit 1 and its order is 2.

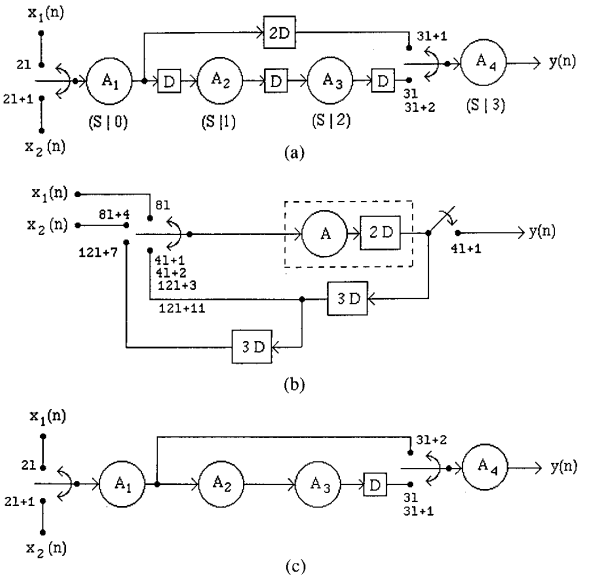

Fig. 10. (a) An algorithm DFG containing switches. (b) The corresponding folded architecture DFG. (c) A nonpipelined version of the algorithm DFG in (a) which would also yield the architecture DFG in (b).

achieved in retiming by considering the input and output nodes as separate host nodes. In contrast, our objective is to pipeline and retime for folding, such that the folded arc delays become positive.

We now derive the constraints for pipelining and retiming. Consider an edge $U \rightarrow V$ with i delays. The retiming and pipelining change the number of delays in this arc to i_r, given by

$$i_r(U \rightarrow V) = i(U \rightarrow V) + r(V) - r(U)$$

where $r(V)$ and $r(U)$ represent the retiming values of nodes V and U, respectively. Our objective is reduced to the calculation of the retiming values for all nodes in the DFG, such that the folded path delays of all arcs $U \rightarrow V$ for the pipelined and retimed DFG are nonnegative. Let $D_F(U \rightarrow V)$ denote the folded arc delays, for the arc $U \rightarrow V$, in the original DFG and $D_F'(U \rightarrow V)$ represent the folded arc delays in the retimed and pipelined DFG. Therefore, we need to satisfy $D_F'(U \rightarrow V) \geq 0$. This equation can be exploited to obtain constraints for pipelining and retiming for folding:

$$\begin{aligned} D_F'(U \rightarrow V) &= Ni_r(U \rightarrow V) - P_u + v - u \\ &= Ni(U \rightarrow V) - P_u + v - u + Nr(V) - Nr(U) \\ &= D_F(U \rightarrow V) + Nr(V) - Nr(U) \geq 0. \end{aligned}$$

This last inequality can be satisfied by the constraints

$$r(U) - r(V) \leq D_F(U \rightarrow V)/N.$$

However, since the retiming only considers insertion or movement of integer number of delays to or from any arc, the retiming variables $r(\cdot)$'s are constrained to be integers. This is achieved by modifying the right-hand side of the inequality, and by considering the positive and negative possible values of the folded arc delays. The integer retiming variables can be solved using

$$\begin{aligned} r(U) - r(V) &\leq \lfloor D_F(U \rightarrow V)/N \rfloor, \\ &\qquad \text{if } D_F(U \rightarrow V) \geq 0 \\ r(U) - r(V) &\leq -\lceil -D_F(U \rightarrow V)/N \rceil, \\ &\qquad \text{if } D_F(U \rightarrow V) < 0. \end{aligned}$$

In the above notation, $\lfloor x \rfloor$ is the floor function, which represents the largest integer less than or equal to x, and the ceiling function $\lceil y \rceil$ represents the least integer greater than or equal to y.

For any arbitrary DFG, the folded path delays can be calculated. These values can be used to solve the above set of constraints to obtain the retiming variables. As indicated in [30], these constraints can be solved by the Bellman–Ford shortest path algorithm (other algorithms can be used also). In this formulation, the number of constraint equations is exactly identical to the number of arcs or edges in the DFG. In the context of nonrecursive DFG's, *only* pipelining by inserting delays (and without moving arc delays) can also be achieved by using the set of inequalities:

$$\begin{aligned} r(U) - r(V) &\leq 0, \qquad \text{if } D_F(U \rightarrow V) \geq 0 \\ r(U) - r(V) &\leq -\lceil -D_F(U \rightarrow V)/N \rceil, \\ &\qquad \text{if } D_F(U \rightarrow V) < 0. \end{aligned}$$

The pipelined and retimed DFG is obtained by using the retiming variables and by calculating the delays in the retimed DFG. The retimed DFG can be easily folded. For simplicity, we add a constant number to all the retiming variables, such that the lowest retiming variable is zero. While considering the DFG's with switches, we first ignore the switches and replace them with ordinary edges. Then the DFG is pipelined and retimed. The retiming variables are modified so that the minimum value is zero. The new switch instance is obtained by adding the retiming variable of the node to the existing switching instances.

Example 10: Consider the DFG in Fig. 10(c). This DFG can be pipelined by using the constraints

$$r(A_1) - r(A_2) \leq -1$$

$$r(A_2) - r(A_3) \leq -1$$

$$r(A_3) - r(A_4) \leq 0$$

$$r(A_1) - r(A_4) \leq 0.$$

These constraints can be solved by using the shortest path algorithm using the constraint graph shown in Fig. 11. The retiming variables for nodes A_1 through A_4 are, respectively, obtained to be -2, -1, 0, and 0. By adding 2, these are modified to 0, 1, 2, and 2. Calculation of delays in the retimed DFG and new switching instances leads to the DFG in Fig. 10(a). □

B. Folding of Recursive Data-Flow Graphs

Any arbitrarily specified folding set is a valid set if the algorithm DFG contains no feedback. This is because the nonrecursive algorithm DFG can always be pipelined by adding delays. But in a recursive loop, the total number of loop delays cannot be altered (since this would change the functionality of the algorithm) and this imposes additional restrictions. Thus, if the folded arc delay $D_F(U \rightarrow V)$ for an arc $U \rightarrow V$ of a loop is negative and is made positive by adding Ni' delays, then the Ni' storage units must be subtracted from folded arc delays of one or more arcs in that loop; this effectively retimes [30] the original algorithm DFG. In other words, the unfolded algorithm DFG of the folded hardware DFG will be a retimed version of the original DFG. In general, for arbitrary DFG's, an unfolded algorithm DFG of a folded hardware DFG will be a pipelined and retimed version of the original DFG. Since the number of delays in a loop cannot be altered, property 2 must hold.

Property 2: The synchronizing delay of a loop is zero. In other words, the total path delay of the loop path must equal the number of loop delays times the folding set cardinality N. □

Property 2 should be satisfied for all loops to guarantee that a folding set is valid. In fact, preprocessing the DFG using pipelining and retiming as described in this section will determine the infeasibility of a folding set by indicating that no solution exists to the set of constraints.

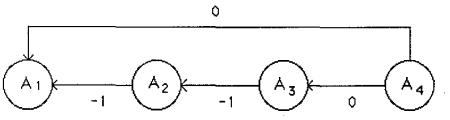

Fig. 11. Constraint graph for automatic pipelining and retiming of the DFG in Fig. 10(c).

Example 11: Consider the second-order IIR filter example as shown in Fig. 12(a). The four add operations are mapped to a single adder, and the four multiply operations are mapped to a single multiply operator. We assume that the adder is pipelined by one stage, and the multiplier is pipelined by two stages. A null operation was inserted to obtain a valid folding set. The complete folded architecture is shown in Fig. 12(b). The second-order filter is retimed as shown in Fig. 12(c), and the retimed DFG can be folded without requiring any null operations. The complete folded architecture corresponding to the retimed DFG is shown in Fig. 12(d). The reader can verify that the loops in Fig. 12(c) did not require any further retiming and only delays across one feedforward cutset were used to pipeline the DFG; this cutset contains the arcs $A_1 \rightarrow A_2$, $M_2 \rightarrow A_4$, and $M_4 \rightarrow A_4$. □

Example 12: Consider another algorithm DFG in Fig. 13(a). The retiming inequalities for folding are used to obtain the constraint graph in Fig. 13(b). Solving the graph in Fig. 13(b) for the shortest path yields the retiming variables for the nodes A_1, M_1, A_2, and M_2 to be 0, 1, 1, and 0, respectively. The retimed DFG is shown in Fig. 13(c). The retimed DFG is folded to obtain the hardware DFG in Fig. 13(d). □

Example 13: In this example, we illustrate the folding of circuits containing feedback and switches by folding a digit-serial adder (with digit-size three) shown in Fig. 14(a) to a bit-serial adder. Fig. 14(b) shows a folded bit-serial adder for the folding set $S = [A_1, A_2, A_3]$. Some switch simplifications reduce the DFG in Fig. 14(b) to 14(c). (See Fig. 16 for examples of switch transformations.) Fig. 15(a) shows another folded bit-serial adder for the digit-serial adder shown in Fig. 14(a). The hardware DFG in Fig. 15(a) is obtained by using the folding set $S = [A_2, A_3, A_1]$. Three unfolding of the bit-serial adder in Fig. 15(a) leads to the digit-serial adder in Fig. 15(b); the reader can verify that the DFG in Fig. 15(b) is a retimed and pipelined version of the DFG in Fig. 14(a). □

V. Folding Using Single Implementation Style and Multiple Clocks

Often DSP circuit designers can achieve full hardware utilization by using multiple computation clocks (while using a single implementation style). The use of multiple computation clocks may lead to a reduction in the number of pipelining latches of some operators, and can lead to overall improvement if the area saving due to latches is larger than the overhead associated with routing of the additional computation clock(s). We assume that the ratio of different clock rates is a rational number. We also as-

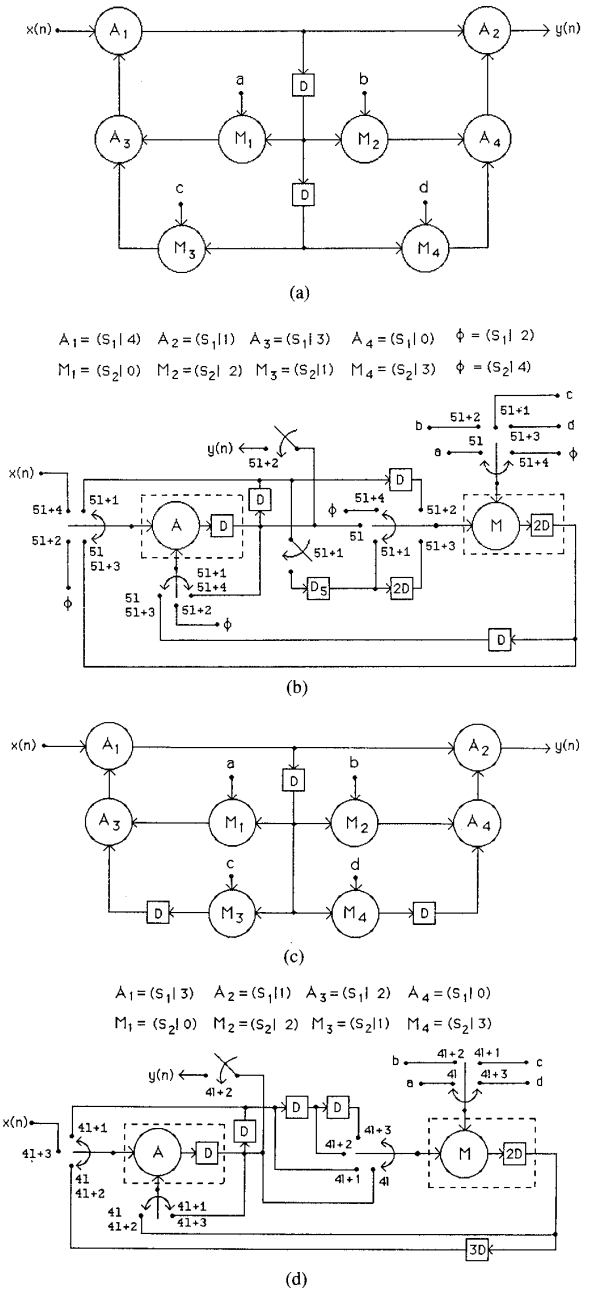

Fig. 12. (a) A second-order IIR filter implemented with four add operators and four multiply operators. We assume the computation time of the multiply is twice as long as the computation time of the add. (b) A complete folded architecture of the IIR filter with an added null operation in each ordered set in order to maintain the functionality of the filter. The 5-slow delay D_5 minimizes the number of registers. (c) A retimed version of the second-order IIR filter of (a). When folded, this retimed circuit does not require a null operation. (d) A complete folded architecture of the retimed IIR filter using a uniform design style and a single clock. This design does not use a null operation.

sume that the system input data are valid for N_u clock cycles for operator H_u and N_v clock cycles for operator H_v, if N_u and N_v represent the cardinalities of the folding sets S_u and S_v. Folding of such circuits is carried out in a

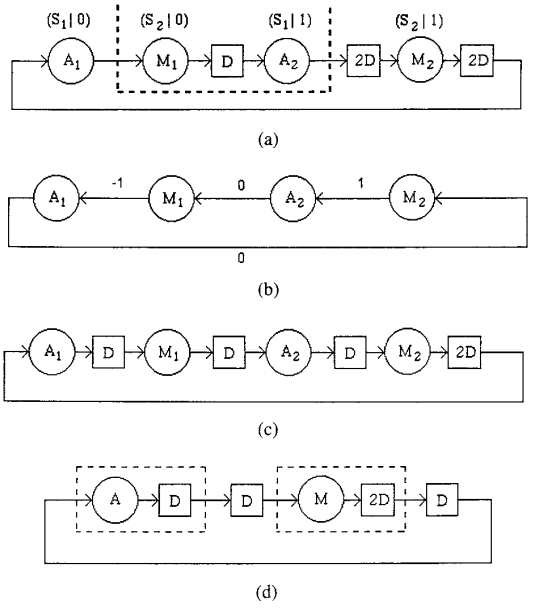

Fig. 13. (a) An algorithm DFG containing feedback. (b) The constraint graph for automatic retiming for folding of the DFG in (a). (c) The retimed DFG, where the retiming makes use of the delay transfer across the marked cutset in (a). (d) The folded architecture DFG.

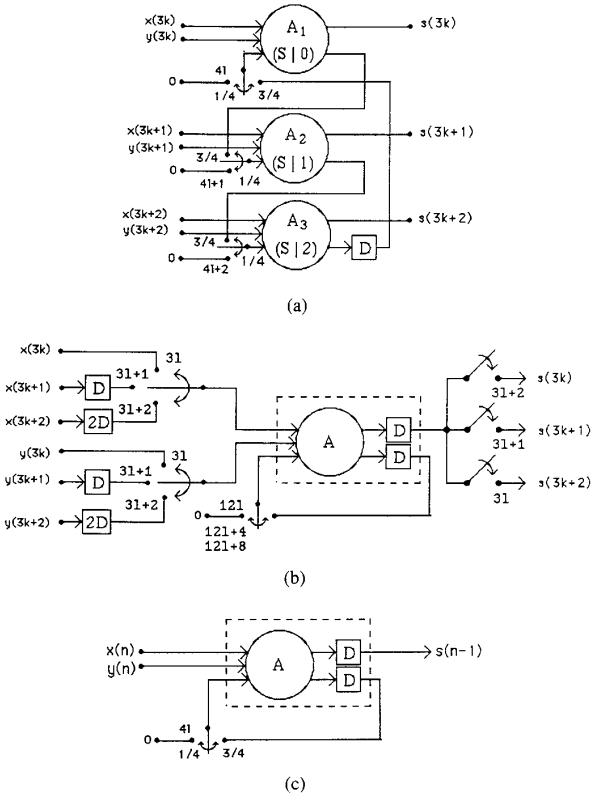

Fig. 14. (a) An algorithm DFG of a digit-serial binary adder for digit size of three and word length of four. (b) A folded bit-serial adder. (c) An equivalent bit-serial adder obtained from (b) using the switch transformation of Fig. 16.

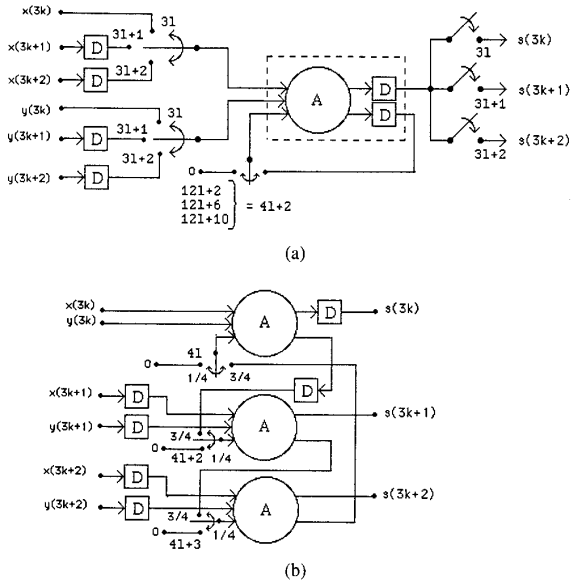

Fig. 15. (a) A folded architecture DFG of the algorithm DFG in Fig. 14(a) obtained using the folding set $S = (A_2, A_3, A_1)$. (b) The three-unfolded DFG of (a) is an automatically pipelined and retimed version of Fig. 14(a).

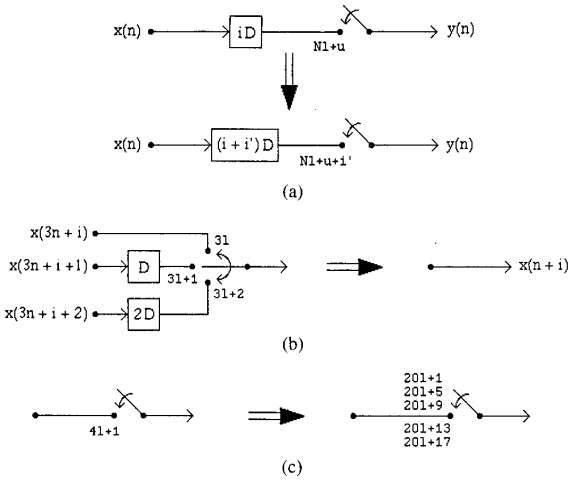

Fig. 16. Switch transformations: (a) additive switching instance transformation, (b) switch compression, and (c) switching instance expansion (multiplicative switching instance transformation).

manner similar to the single clock/single implementation style case, except for converting the clock cycle numbers at the clock boundaries.

Consider an arc between two nodes in the original DFG ($U \rightarrow V$) with i delays. The notation remains the same as in Section IV, except for the introduction of multiple clock cycles. Let r_u (r_v) represent the clock rate of the hardware operator H_u (H_v) (we assume r_u and r_v to be coprime). The execution of the lth iteration of U, U_l, starts in cycle $[N_u l + u]$ and is completed in cycle $[N_u l + P_u + u]$, with respect to the clock of H_u. The $(l + i)$th iteration of V, V_{l+i}, is executed in cycle $[N_v(l + i) + v]$, with respect to the clock of H_v. To obtain the number of registers needed for the appropriate data storage between the two folded operators, we take the difference between the input clock cycle of H_v and the output clock cycle of H_u. In this case, the clock cycles are referenced to two different rates; therefore our algorithm converts the output clock cycle of H_u in terms of clock cycle of H_v:

$$r_v \left\lceil \frac{(N_u l + P_u + u)}{r_u} \right\rceil$$

where $\lceil x \rceil$ is the ceiling function. The number of storage units in the arc $H_u \rightarrow H_v$ in the hardware DFG (as implemented from the H_v side and clocked with rate r_v) is given by

$$[N_v(l + i) + v] - r_v \left\lceil \frac{(N_u l + P_u + u)}{r_u} \right\rceil.$$

For the number of folded arc delays to be iteration independent, the following relation must be satisfied:

- N_u must be a multiple of r_u,
- $N_v/N_u = r_v/r_u$.

When the above relation is true, the number of folded arc delays, as referenced to the clock of H_v, required between the folded operators H_u and H_v to maintain the functionality of the original algorithm is given by

$$D_F(U \rightarrow V) = [N_v i + v] - r_v \left\lceil \frac{(u + P_u)}{r_u} \right\rceil.$$

It is also necessary to connect

$$r_u \left\lceil \frac{(P_u + u)}{r_u} \right\rceil - (P_u + u)$$

delays clocked at rate r_u to the output of H_u.

For designs using multiple clock, it is always necessary to convert the number of delays at the clock rate boundaries for calculations of total path delays and synchronizing delays also. All other procedures remain the same as in Section IV. Similar to the single clock case, for any folded operator H_v, we may be able to reduce the number of registers in the control circuits by utilizing N_v-*slow* latches (i.e., latches clocked with a rate N_v times slower than the clock rate of H_v). Let D_{uv} represent the number of folded arc delays referenced to the rate r_v on an arc $U \rightarrow V$. If $D_{uv} \geq N_v$, we can replace D_{uv} with $\lfloor D_{uv}/N_v \rfloor$ N_v-*slow* delays and $D_{uv} - N_v \lfloor D_{uv}/N_v \rfloor$ folded arc delays referenced to r_v, where the $\lfloor x \rfloor$ represents the floor function. The retiming formulation of Section IV also holds for this case.

Example 14: Consider the fourth-order FIR filter in Fig. 9(a). Let the folding sets be $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$. In this case, the ratio of the clock rates must be 3:2; therefore, the hardware multiply–add operators must be pipelined by three and two levels, respectively. The complete folded architecture is shown in Fig. 17. In this example, the second multiply–add operator is pipelined only by two stages (as opposed to three as in Fig. 9) and this leads to saving in the number of

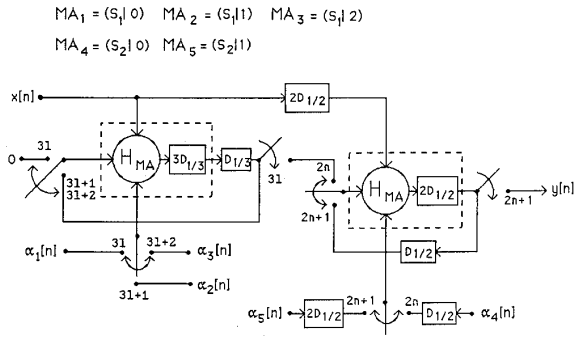

Fig. 17. A complete folded architecture of the fourth-order FIR filter in Fig. 9(a) using a single design style and multiple clocks. The indexes l and n distinguish the two clocks. The folding sets are $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$ with the parameters: $P_1 = 3$, $N_1 = 3$, $P_2 = 2$, $N_2 = 2$, $r_2 = 3:2$. Three times the clock period of latch $D_{1/3}$ is the same as two times the clock period of latch $D_{1/2}$.

pipelining latches. Three times the clock period of $D_{1/3}$ is same as two times the clock period of $D_{1/2}$. □

VI. Folded Circuits Using Multiple Styles and Single Clock

In this section, we extend the folding transformation technique to accommodate multiple operator implementation styles (i.e., some operators could be bit serial while others can be digit serial [12]–[15] or bit parallel, etc.). We assume use of a single computation clock. All notations remain the same as in Section IV, except we must introduce an additional variable for specifying the number of units of data processed by each operator in a clock cycle. Let d_u represent the number of data units processed by H_u every clock cycle, and similarly d_v represents the number of data units processed by H_v every clock cycle. For example, if we assume a unit of data to be 1 b and that $d_u = 2$ and $d_v = 3$, then H_u may represent a digit-serial operator with a digit size of 2, and H_v may represent a digit-serial operator with a digit size of 3 [14], [15]. Even though the two operators both use digit-serial architectures, they are considered to be nonuniform due to their different digit sizes (i.e., $d_u \neq d_v$). If we assume that a unit of data equals one word, then H_u would correspond to a word-parallel architecture with a block size of 2, and H_v would correspond to a word-parallel architecture with a block size of 3. The design methodology considered in this section is useful for low to moderate sample rate applications and can reduce the area of some hardware operators without degrading the circuit performance. We assume that the data on each system input are valid for one clock cycle (this will be clearer later).

Consider an arc $U \rightarrow V$ with i delays. Let $d_u \neq d_v$ and assume d_u and d_v to be coprime. If $d_u = d_v = d$, and both of these are greater than 1, then for digit-serial processing we can replace i delays by iw/d in the flow graph (where w is the word length) and use the folded path formula presented in Section IV.

Let the execution of the lth iteration of U, U_l, start in cycle $[N_u \lfloor l/d_u \rfloor + u]$ and be available to H_v in cycle $[N_u \lfloor l/d_u \rfloor + P_u + u + C_{uv}]$, where C_{uv} represents the latency of the d_u/d_v data format conversion. The $(l+i)$th iteration of V, V_{l+i}, is executed in cycle $[N_v \lfloor (l+i)/d_v \rfloor + v]$. The number of folded arc delays required for the arc between H_u and H_v is the difference of the input cycle of H_v and the output cycle of H_u:

$$D_F(U \rightarrow V) = N_v \left\lfloor \frac{l+i}{d_v} \right\rfloor + v - N_u \left\lfloor \frac{l}{d_u} \right\rfloor - P_u - u - C_{uv}.$$

To simplify the above, we need to calculate the delay (in number of cycles) through the d_u/d_v converter in terms of the iteration number l [33], [34]. The iteration number l is processed in time unit $d_u \lfloor l/d_u \rfloor$ and is output in $d_v \lfloor l/d_v \rfloor + d_v - 1$ (since $d_v - 1$ is the latency of the converter, i.e., the first output is available after the latency number of cycles) [34]. The input cycle period of the converter is d_u time units and the output cycle period is d_v time units. Thus

$$C_{uv} = d_v \left\lfloor \frac{l}{d_v} \right\rfloor - d_u \left\lfloor \frac{l}{d_u} \right\rfloor + d_v - 1.$$

Using this value of C_{uv} and assuming i to be a multiple of d_v (otherwise the number of registers would not be iteration independent), we get

$$D_F(U \rightarrow V) = (N_v - d_v) \left\lfloor \frac{l}{d_v} \right\rfloor - (N_u - d_u) \left\lfloor \frac{l}{d_u} \right\rfloor + \frac{N_v i}{d_v} + v - u - P_u - d_v + 1.$$

For the number of folded arc registers to be iteration independent, we must satisfy:

- $N_u = d_u$ and $N_v = d_v$, and
- the number of delays in the arc $U \rightarrow V$ "i" in the algorithm DFG must be a multiple of d_v.

Because of the first condition, the input and output samples are valid for one cycle only. Under these conditions, we get

$$D_F(U \rightarrow V) = i + v - u - P_u - d_v + 1.$$

The above number of delays requires further modification. This is because the folding order 0 can be both odd or even depending on the values of N_u and N_v. Therefore, for simplicity, we assume $u = 0$ in above expression, and the folding order is artificially reset to 0 by adding $d_u - u$ delays at the output of H_u. Thus, the number of folded arc delays is given by

$$D_F(U \rightarrow V) = i + v - u - P_u + d_u - d_v + 1.$$

Once again, one needs to adjust the number of total path and synchronizing delays (for a path containing a data boundary) by accounting for the data format converter delays. The above number of folded arc delays needs to be used in the retiming formulation.

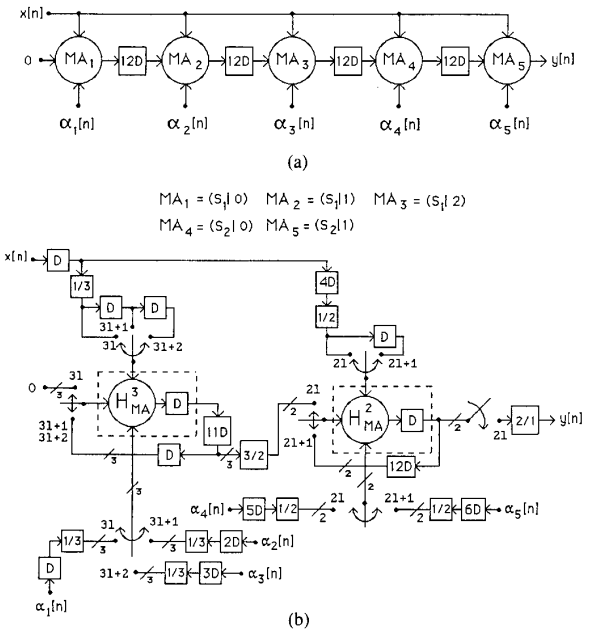

Fig. 18. (a) Algorithm DFG of Fig. 9(a), redrawn for a bit-serial implementation with a word length of 12. (b) A complete folded architecture of the fourth-order FIR filter of (a) using multiple design styles and single clock. The folding sets are $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$. The parameters are: $d_1 = 3$, $P_1 = 1$, $N_1 = 3$, $d_2 = 2$, $P_2 = 1$, $N_2 = 2$.

Example 15: Consider the folding of the FIR filter in Fig. 9(a) to two multiply-add processors, one three-digit serial and operator, and one two-digit serial operator. Assume a word length of 12. First we transform the word-level algorithm DFG of Fig. 9(a) to the bit-level algorithm DFG of Fig. 18(a) by replacing each word-level delay to 12 bit-level delays. Let the folding sets be $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$, where S_1 has a data size of three and S_2 has a data size of two. The complete folded architecture is shown in Fig. 18(b). The appropriate converter circuits have been placed in the required locations for data format conversions. The notations A^2 and A^3, respectively, represent two- and three unfolded digit-serial "A" processors [14], [15]. □

VII. Folded Circuits Using Multiple Styles and Multiple Clocks

This particular design methodology of multiple clocks and nonuniform implementation styles for the hardware operators is the most general case. Although this design style may not produce efficient architectures, we present this methodology because of its generality.

Consider an arc between two nodes in the original DFG $(U \rightarrow V)$ with i delays. The notation is the same as in previous sections. The lth iteration of U, U_l, starts in cycle $[N_u \lfloor l/d_u \rfloor + u]$ and is completed in cycle $[N_u \lfloor l/d_u \rfloor + P_u + u + C_{uv}]$. First, we connect $d_u - u$ delays to the output of H_u to make the effective folding order of u zero. The $(l + i)$th iteration of V, V_{l+i}, is executed in cycle $[N_v \lfloor (l + i)/d_v \rfloor + v]$. To find the number of registers needed for proper execution, we express all time units in terms of the clock of H_u. When referenced to H_u's clock, H_v starts its execution in cycle

$$r_u \left\lceil \frac{N_v \left\lfloor \frac{(l+i)}{d_v} \right\rfloor + v}{r_v} \right\rceil.$$

To simplify the above expression, we assume a) i is a multiple of d_v, and b) N_v is a multiple of r_v. Furthermore, it is not necessary to convert the folding order v to the reference clock of H_u (this can be taken care of by adding v offset delays at rate H_v to the input of H_v). Incorporating these conditions and the delay of the d_u/d_v converter delay, we obtain the number of folded arc registers connected to H_u to be

$$\frac{r_u N_v}{r_v} \left\lfloor \frac{l}{d_v} \right\rfloor + \frac{r_u N_v i}{d_v r_v} - N_u \left\lfloor \frac{l}{d_u} \right\rfloor - P_u - d_v \left\lfloor \frac{l}{d_v} \right\rfloor + d_u \left\lfloor \frac{l}{d_u} \right\rfloor - d_v + 1 + d_u - u.$$

The above is iteration independent if

- i is a multiple of d_v,
- $N_u = d_u$, and N_v is a multiple of r_v,
- $d_v = r_u N_v / r_v$.

The number of folded arc delays is then given by

$$D_F(U \rightarrow V) = i - P_u - u + d_u - d_v + 1.$$

Note that in these designs, the data for H_u are valid for one cycle only, whereas the data for H_v are valid for r_v/r_u number of cycles. The total path and synchronization delays must now incorporate the converter delays and the clock rate conversions. Other procedures for folding remain identical.

Example 17: Consider the fourth-order FIR filter in Fig. 18(a). Let the folding sets be $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$, where S_1 has a data size of three and S_2 has a data size of one. The clock rate of the serial operator is twice that of the three-unfolded operator. The complete folded architecture is shown in Fig. 19. □

VIII. CAD Tool for Folding

We have written a design tool for the automated pipelining and retiming preprocessing of DFG's and for folding of the retimed/pipelined DFG's. In this section, we present the input and output files of one example DFG. The program takes two files, one file containing a description of the topology of the DFG and a second file containing a description of the folding set. Fig. 20(a) shows the input topology file for the second-order retimed biquad DFG in Fig. 12(c). Each line represents an edge with syntax: (input node num), (input terminal num), (output node num), (output terminal num), (number of delays). Fig. 20(b) shows the folding set file. This file describes the number of partitions and the folding order of nodes to processors. Fig. 20(c) contains the output file.

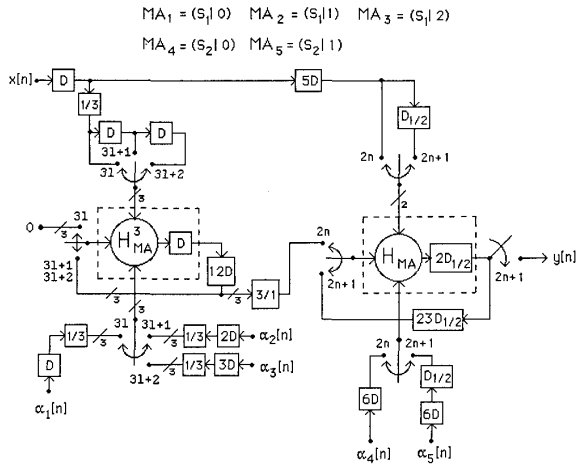

Fig. 19. A complete folded architecture of the fourth-order FIR filter of Fig. 18(a) using multiple design styles and multiple clocks. The folding sets are $S_1 = [MA1, MA2, MA3]$ and $S_2 = [MA4, MA5]$ with the parameters: $d_1 = 3$, $P_1 = 1$, $N_1 = 3$, $d_2 = 1$, $P_2 = 2$, $N_2 = 2$, $r_1 : r_2 = 1:2$. The latch $D_{1/2}$ is 2-fast.

```
x 0 a1 0 0
a1 0 a2 0 0
a1 0 m1 0 1
a1 0 m2 0 1
a1 0 m3 0 1
a1 0 m4 0 1
m1 0 a3 0 0
m2 0 a4 1 0
m3 0 a3 1 1
m4 0 a4 0 1
a3 0 a1 1 0
a4 0 a2 1 0
a2 0 y 0 0
```

(a)

```
4
1 a4 a2 a3 a1
2 m1 m3 m2 m4
```

(b)

```
Retimed algorithm DFG:
x 0 a1 0 0
a1 0 a2 0 1
a1 0 m1 0 1
a1 0 m2 0 1
a1 0 m3 0 1
a1 0 m4 0 1
m1 0 a3 0 0
m2 0 a4 1 1
m3 0 a3 1 1
m4 0 a4 0 2
a3 0 a1 1 0
a4 0 a2 1 0
a2 0 y 0 0


Hardware DFG:
x.0      --0D->  H[0].0 at 4l+3
H[0].0 at 4l+0  --1D->  H[0].0 at 4l+1
H[0].0 at 4l+0  --0D->  H[1].0 at 4l+0
H[0].0 at 4l+0  --2D->  H[1].0 at 4l+2
H[0].0 at 4l+0  --1D->  H[1].0 at 4l+1
H[0].0 at 4l+0  --3D->  H[1].0 at 4l+3
H[1].0 at 4l+2  --0D->  H[0].0 at 4l+2
H[1].0 at 4l+0  --0D->  H[0].1 at 4l+0
H[1].0 at 4l+3  --3D->  H[0].1 at 4l+2
H[1].0 at 4l+1  --3D->  H[0].0 at 4l+0
H[0].0 at 4l+3  --0D->  H[0].1 at 4l+3
H[0].0 at 4l+1  --0D->  H[0].1 at 4l+1
H[0].0 at 4l+2  --0D->  y.0
```

(c)

Fig. 20. (a) The topology file for the DFG in Fig. 12(c). (b) The folding set file for Fig. 12(c). (c) Output of the folding program describes the folded hardware architecture of Fig. 12(d).

This file describes the retimed/pipelined DFG for folding, and the syntax of the hardware DFG. One can verify that the hardware description of Fig. 20(c) indeed describes the hardware DFG in Fig. 12(d).

IX. Conclusion

By using graph-based approaches, we have presented folding algorithms for synthesis of control circuits in pipelined digital signal processing architectures. A sufficient theory of pipelining has been developed to ensure iteration independence of the registers used in control circuits of the dedicated architectures. We have presented some refinement steps that can reduce the complexity of the design by collapsing repeated arcs and minimizing the number of latches. Automatic pipelining/retiming for folding has been proposed to preprocess data-flow graphs prior to folding. The relationship between folding and resource allocation and scheduling for synthesis needs to be explored better.

References

[1] P. R. Cappello and K. Steiglitz, "Unifying VLSI array design with linear transformation of space time," *Advances Comput. Res.*, vol. 2, pp. 23–65, 1984.

[2] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," in *Proc. 11th Annual Symp. Comput. Architecture*, June 1984, pp. 208–214.

[3] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping of algorithms into fixed size systolic arrays," *IEEE Trans. Comput.*, vol. C-35, pp. 1–12, Jan. 1986.

[4] G. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 67–77, Jan. 1985.

[5] H. V. Jagadish, S. K. Rao, and T. Kailath, "Array architectures for iterative algorithms," *Proc. IEEE*, vol. 75, no. 9, pp. 1304–1321, Sept. 1987.

[6] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[7] E. F. Girczyc, "Loop winding—A data flow approach to functional pipelining," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1987, pp. 382–385.

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

[9] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to implementation of digital filters," *IEEE Trans. Audio Electroacoust.*, vol. AE-16, no. 3, pp. 413–421, 1968.

[10] R. Jain *et al.*, "Custom design of a VLSI PCM-FDM transmultiplexor from system specification to circuit layout using a computer aided design system," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 1, pp. 73–85, Feb. 1986.

[11] R. F. Lyon, "Two's complement pipelined multipliers," *IEEE Trans. Commun.*, vol. COM-24, pp. 418–424, 1976.

[12] S. G. Smith and P. B. Denyer, *Serial Data Computation*. Boston: Kluwer Academic, 1988.

[13] R. I. Hartley and P. F. Corbett, "Digital serial processing techniques," *IEEE Trans. Circuits Syst.*, vol. 37, no. 6, pp. 707–719, June 1990.

[14] K. K. Parhi, "A systematic approach for design of digit-serial signal processing architectures," *IEEE Trans. Circuits Syst.*, vol. 38, no. 4, pp. 358–375, Apr. 1991.

[15] K. K. Parhi and C.-Y. Wang, "Digit-serial DSP architectures," in *Proc. Int. Conf. Applications Specific Array Processors*. Princeton, NJ: IEEE Computer Society, Sept. 1990, pp. 341–351.

[16] J. Rabaey *et al.*, "Resource driven synthesis in hyper system," in *Proc. 1990 IEEE ISCAS* (New Orleans), May 1990, pp. 2592–2595.

[17] 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. 661–675, June 1989.

[18] C. T. Hwang, Y. C. Hsu, and Y. L. Lin, "Optimum and heuristic

data path scheduling under resource constraints," in *Proc. 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 65–70.
[19] K. S. Hwang *et al.*, "Scheduling and hardware sharing in pipelined data paths," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 24–27.
[20] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, pp. 301–318, Feb. 1990.
[21] N. Park and A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 356–370, Mar. 1988.
[22] H. De Man *et al.*, "Cathedral-II: A silicon compiler for digital signal processing," *IEEE Design Test Comput.*, pp. 13–25, Dec. 1986.
[23] J-H. Lee, Y-C. Hsu, and Y-L. Lin, "A new integer linear programming formulation for the scheduling problem in data path synthesis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 20–23.
[24] C. A. Papachristou and H. Konuk, "A linear program driven scheduling and allocation method followed by interconnect optimization algorithm," in *Proc. 27th ACM/IEEE DA Conf.*, 1990, pp. 77–83.
[25] C. Y. Huang *et al.*, "Data path allocation based on bipartite weighted matching," in *Proc. 27th ACM/IEEE DA Conf.*, 1990, pp. 499–504.
[26] R. Potasman *et al.*, "Percolation based synthesis," in *Proc. 27th ACM/IEEE DA Conf.*, 1990, pp. 444–449.
[27] R. J. Cloutier and D. E. Thomas, "Combination of scheduling, allocation, and mapping in a single algorithm," in *Proc. 27th ACM/IEEE DA Conf.*, 1990, pp. 71–76.
[28] C.-Y. Wang and K. K. Parhi, "Dedicated DSP architecture synthesis using the MARS design system," in *Proc. 1991 IEEE Int. Conf. Acoust., Speech, Signal Processing* (Toronto), May 1991, pp. 1253–1256.
[29] C.-Y. Wang and K. K. Parhi, "Automatic generation of control circuits in pipelined DSP architectures," in *Proc. IEEE Int. Conf. Computer Design* (Cambridge, MA), Sept. 1990, pp. 324–327.
[30] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. 3rd Caltech Conf VLSI*, Mar. 1983, pp. 87–116.
[31] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 178–195, Feb. 1991.
[32] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE (Special Issue on Supercomputer Technology)*, pp. 1879–1895, Dec. 1989.
[33] K. K. Parhi and J. S. Lee, "Register allocation for design of data format converters," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, May 1991, pp. 1133–1136.
[34] K. K. Parhi, "Register minimization in DSP data format converters," in *Proc. IEEE Int. Symp. Circuits Syst.* (Singapore), June 1991, pp. 2367–2370.

Keshab K. Parhi (S'85–M'88–SM'91) received the B.Tech. (Honors) degree from the Indian Institute of Technology, Kharagpur, India, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988.

He is currently an Assistant Professor of Electrical Engineering at the University of Minnesota, Minneapolis. He has held short-term positions at AT&T Bell Laboratories, Holmdel, NJ, the IBM T.J. Watson Research Center, Yorktown Heights, NY, and the Tata Engineering and Locomotive Company, Jamshedpur, India, and has been a consultant to U.S. West Advanced Technologies. His research interests include concurrent algorithm and architecture designs for communications, signal and image processing systems, neural network signal processing, digital integrated circuits, VLSI digital filters, computer arithmetic, high-level DSP synthesis, and multiprocessor prototyping and task scheduling for programmable software systems. He has published over 60 papers in these areas.

Dr. Parhi received the 1991 Browder Thompson prize paper award of the IEEE, the 1989 research initiation award of the National Science Foundation, and the 1987 Eliahu Jury award and the 1987 Demetri Angelakos award of the University of California (U.C.), Berkeley, where he was a U.C. Regents Fellow and an IBM Graduate Fellow. He is a former Associate Editor for Image Processing and VLSI Applications of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and is a member of Eta Kappa Nu and the Association for Computing Machinery.

Ching-Yi Wang (S'83) received the B.S. degree in electrical engineering in 1985 from the University of Idaho, Moscow. He obtained the M.S. degree in electrical engineering in 1989 and is currently a Ph.D. candidate at the University of Minnesota, Minneapolis.

He has held short-term positions at the Hewlett-Packard Disc Memory Division, Boise, ID, Sandia National Laboratory, Livermore, CA, Los Alamos National Laboratory, Los Alamos, NM, and General Electric Corporate Research and Development, Schenectady, NY. His research interests include synthesis of high-speed architectures, VLSI design, signal and image processing, and computer-aided design.

Mr. Wang is a member of Tau Beta Pi.

Andrew P. Brown received the B.S.E.E. degree from Iowa State University, Ames, in 1985 and the M.S.E.E. degree from the University of Minnesota, Minneapolis, in 1991.

He has worked at Digital Equipment Corporation, and currently is a Senior Engineer in 3M's Applied Technology Laboratory, St. Paul, MN.