4 Retiming

4.1 INTRODUCTION

Retiming [1] is a transformation technique used to change the locations of delay elements in a circuit without affecting the input/output characteristics of the circuit. For example, consider the IIR filter in Fig. 4.1(a). This filter is described by

$$\begin{aligned} w(n) &= ay(n-1) + by(n-2) \\ y(n) &= w(n-1) + x(n) \\ &= ay(n-2) + by(n-3) + x(n). \end{aligned}$$

The filter in Fig. 4.1(b) is described by

$$\begin{aligned} w_1(n) &= ay(n-1) \\ w_2(n) &= by(n-2) \\ y(n) &= w_1(n-1) + w_2(n-1) + x(n) \\ &= ay(n-2) + by(n-3) + x(n). \end{aligned}$$

Although the filters in Fig. 4.1(a) and Fig. 4.1(b) have delays at different locations, these filters have the same input/output characteristics. These 2 filters can be derived from one another using retiming.

Retiming has many applications in synchronous circuit design. These applications include reducing the clock period of the circuit, reducing the number of registers in the circuit, reducing the power consumption of the circuit, and

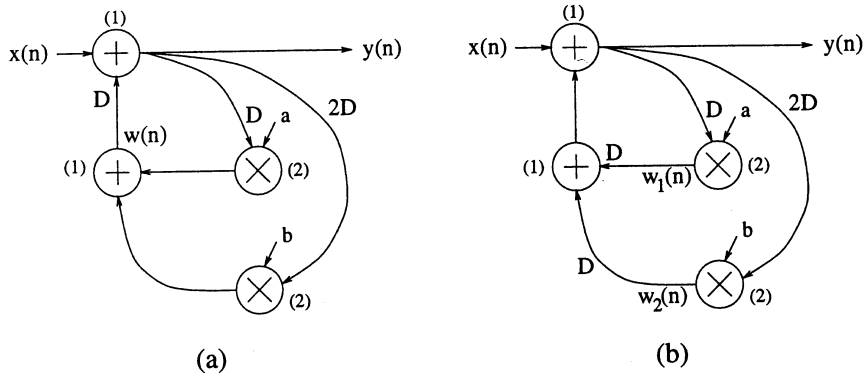

Fig. 4.1 Two versions of an IIR filter. The computation times of the nodes are shown in parentheses.

logic synthesis. The topics of retiming to reduce the clock period and to reduce the number of registers are discussed in detail in this chapter. Retiming for logic synthesis is beyond the scope of this book.

Retiming can be used to increase the clock rate of a circuit by reducing the computation time of the critical path. Recall that the critical path is defined to be the path with the longest computation time among all paths that contain zero delays, and the computation time of the critical path is the lower bound on the clock period of the circuit. The critical path of the filter in Fig. 4.1(a) passes through 1 multiplier and 1 adder and has a computation time of 3 u.t., so this filter cannot be clocked with a clock period of less than 3 u.t. The retimed filter in Fig. 4.1(b) has a critical path that passes through 2 adders and has a computation time of 2 u.t., so this filter can be clocked with a clock period of 2 u.t. By retiming the filter in Fig. 4.1(a) to obtain the filter in Fig. 4.1(b), the clock period has been reduced from 3 u.t. to 2 u.t., or by 33%. A polynomial-time algorithm for retiming for clock period minimization is described in Section 4.4.2.

Retiming can be used to decrease the number of registers in a circuit. The filter in Fig. 4.1(a) uses 4 registers while the filter in Fig. 4.1(b) uses 5 registers. Since retiming can affect the clock period *and* the number of registers, it is sometimes desirable to take both of these parameters into account. A polynomial time algorithm for retiming to minimize the number of registers for a given clock period is described in Section 4.4.3.

Retiming can be used to reduce the power consumption of a circuit by reducing switching, which can lead to dynamic power dissipation in static CMOS circuits [2]. Placing registers at the inputs of nodes with large capacitances can reduce the switching activities at these nodes, which can lead to low-power solutions (see Section 17.5.4).

In Section 4.2, a mathematical description of retiming is addressed and

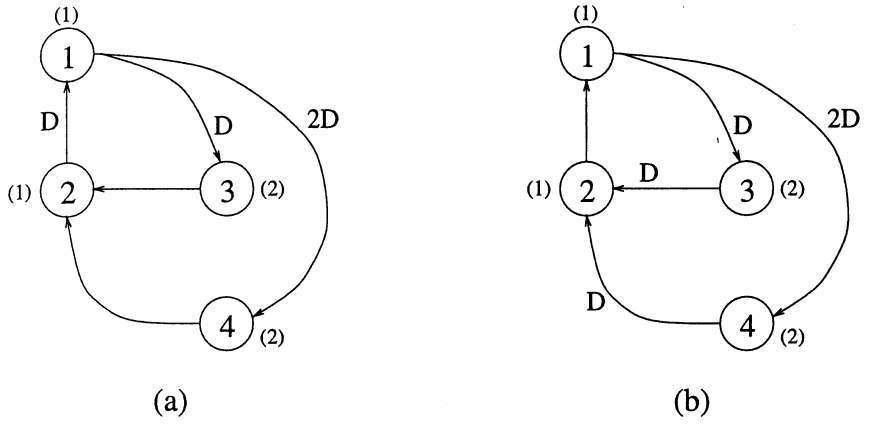

Fig. 4.2 (a) A DFG. (b) The retimed DFG obtained using $r(1) = 0$, $r(2) = 1$, $r(3) = 0$, and $r(4) = 0$.

some properties of retiming are discussed. Techniques for solving systems of inequalities are described in Section 4.3. These techniques are used in Section 4.4, where algorithms for retiming a circuit to achieve various objectives, such as clock period minimization, are described.

4.2 DEFINITIONS AND PROPERTIES

4.2.1 Quantitative Description of Retiming

Retiming maps a circuit G to a retimed circuit G_r. In the context of retiming, the terms circuit and graph and DFG are often used interchangeably, as they are in this chapter. A retiming solution is characterized by a value $r(V)$ for each node V in the graph. Let $w(e)$ denote the weight of the edge e in the original graph G, and let $w_r(e)$ denote the weight of the edge e in the retimed graph G_r. The weight of the edge $U \xrightarrow{e} V$ in the retimed graph is computed from the weight of the edge in the original graph using

$$w_r(e) = w(e) + r(V) - r(U). \tag{4.1}$$

To demonstrate some formal retiming concepts, the filter in Fig. 4.1(a) is redrawn in Fig. 4.2(a), and the retimed filter in Fig. 4.1(b) is redrawn in Fig. 4.2(b). The retiming values $r(1) = 0$, $r(2) = 1$, $r(3) = 0$, and $r(4) = 0$ can be used to obtain the retimed DFG in Fig. 4.2(b) from the DFG in Fig. 4.2(a). For example, the edge $3 \xrightarrow{e} 2$ in the retimed DFG contains

$$w_r(3 \xrightarrow{e} 2) \quad = \quad w(3 \xrightarrow{e} 2) + r(2) - r(3)$$

$$= 0+1-0=1$$

delay, and the edge $2 \stackrel{e}{\rightarrow} 1$ contains

$$\begin{aligned} w_r(2 \stackrel{e}{\rightarrow} 1) &= w(2 \stackrel{e}{\rightarrow} 1) + r(1) - r(2) \\ &= 1+0-1=0 \end{aligned}$$

delays.

A retiming solution is feasible if $w_r(e) \geq 0$ holds for all edges. While the solution that maps Fig. 4.2(a) to Fig. 4.2(b) is feasible because all of the edges in Fig. 4.2(b) have nonnegative weights, the solution $r(1) = 0$, $r(2) = -1$, $r(3) = 0$, and $r(4) = 0$ is infeasible because, for example, the edge $3 \stackrel{e}{\rightarrow} 2$ in the retimed system contains

$$\begin{aligned} w_r(3 \stackrel{e}{\rightarrow} 2) &= w(3 \stackrel{e}{\rightarrow} 2) + r(2) - r(3) \\ &= 0+(-1)-0=-1 \end{aligned}$$

delays.

4.2.2 Properties of Retiming

Several properties of retiming can be derived from the retiming equation (4.1). Before considering these, the concepts of paths and cycles are reviewed. A path is a sequence of edges and nodes $V_0 \stackrel{e_0}{\rightarrow} V_1 \stackrel{e_1}{\rightarrow} \cdots \stackrel{e_{k-2}}{\rightarrow} V_{k-1} \stackrel{e_{k-1}}{\rightarrow} V_k$. The weight of the path p is $w(p) = \sum_{i=0}^{k-1} w(e_i)$ and the computation time of the path is $t(p) = \sum_{i=0}^{k} t(V_i)$. A cycle is a closed path $V_0 \stackrel{e_0}{\rightarrow} V_1 \stackrel{e_1}{\rightarrow} \cdots \stackrel{e_{k-2}}{\rightarrow} V_{k-1} \stackrel{e_{k-1}}{\rightarrow} V_0$. The weight of the cycle c is $w(c) = \sum_{i=0}^{k-1} w(e_i)$ and the delay of the cycle is $t(c) = \sum_{i=0}^{k-1} t(V_i)$.

Property 4.2.1 *The weight of the retimed path* $p = V_0 \stackrel{e_0}{\rightarrow} V_1 \stackrel{e_1}{\rightarrow} \cdots \stackrel{e_{k-1}}{\rightarrow} V_k$ *is given by* $w_r(p) = w(p) + r(V_k) - r(V_0)$.

The retimed path weight is

$$\begin{aligned} w_r(p) &= \sum_{i=0}^{k-1} w_r(e_i) \\ &= \sum_{i=0}^{k-1} (w(e_i) + r(V_{i+1}) - r(V_i)) \\ &= \sum_{i=0}^{k-1} w(e_i) + \left(\sum_{i=0}^{k-1} r(V_{i+1}) - \sum_{i=0}^{k-1} r(V_i) \right) \\ &= w(p) + r(V_k) - r(V_0). \end{aligned}$$

For example, the path $2 \rightarrow 1 \rightarrow 3$ in Fig. 4.2(a) has 2 delays, and this path in the retimed DFG in Fig. 4.2(b) has $2 + r(3) - r(2) = 2 + 0 - 1 = 1$ delay.

Property 4.2.2 *Retiming does not change the number of delays in a cycle.*

This is a special case of Property 4.2.1 where $V_k = V_0$. The weight of the retimed cycle c is $w_r(c) = w(c) + r(V_0) - r(V_0) = w(c)$. In Fig. 4.2, the cycle $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$ contains 2 delays in the unretimed and retimed DFGs, and the cycle $1 \rightarrow 4 \rightarrow 2 \rightarrow 1$ contains 3 delays in the unretimed and retimed DFGs.

Property 4.2.3 *Retiming does not alter the iteration bound in a DFG.*

Property 4.2.4 *Adding the constant value j to the retiming value of each node does not change the mapping from G to G_r.*

After replacing $r(V)$ with $r(V) + j$ for each node, the weight of the retimed edge $U \xrightarrow{e} V$ in G_r is

$$w_r(e) = w(e) + (r(V) + j) - (r(U) + j) = w(e) + r(V) - r(U),$$

which is the same for any value of j (including $j = 0$). Recall that the retiming values $r(1) = 0$, $r(2) = 1$, $r(3) = 0$, and $r(4) = 0$ were used to obtain the retimed DFG in Fig. 4.2(b) from the unretimed DFG in Fig. 4.2(a). By adding, for example, the constant -38 to these retiming values, the retiming values $r(1) = -38$, $r(2) = -37$, $r(3) = -38$, and $r(4) = -38$ can be used to obtain the retimed DFG in Fig. 4.2(b) from the DFG in Fig. 4.2(a).

4.3 SOLVING SYSTEMS OF INEQUALITIES

Given a set of M inequalities in N variables, where each inequality has the form $r_i - r_j \leq k$ for integer values of k, one of the shortest path algorithms in Appendix A can be used to determine if a solution exists and to find a solution if one does indeed exist. This is done using the following procedure.

1. Draw a constraint graph.
   (a) Draw the node i for each of the N variables r_i, $i = 1, 2, \ldots, N$.
   (b) Draw the node $N + 1$.
   (c) For each inequality $r_i - r_j \leq k$, draw the edge $j \rightarrow i$ from node j to node i with length k.
   (d) For each node i, $i = 1, 2, \ldots, n$, draw the edge $N + 1 \rightarrow i$ from the node $N + 1$ to the node i with length 0.

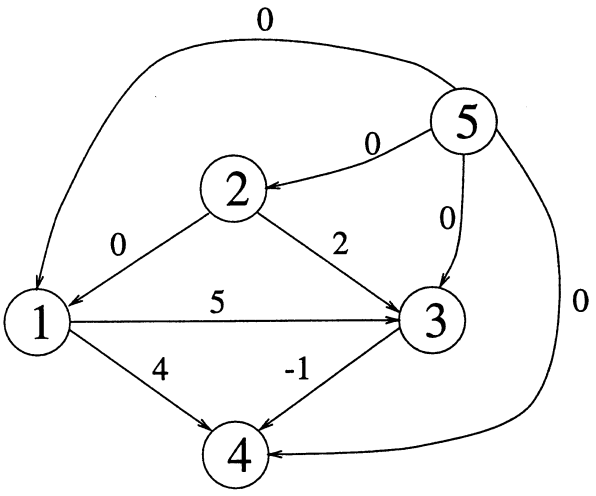

Fig. 4.3 The constraint graph for Example 4.3.1.

2. Solve using a shortest path algorithm.

   (a) The system of inequalities has a solution if and only if the constraint graph contains no negative cycles.

   (b) If a solution exists, one solution is where r_i is the minimum-length path from the node $N+1$ to the node i.

Example 4.3.1 *In this example we demonstrate how shortest path algorithms can be used to solve a system of* $M = 5$ *inequalities*

$$\begin{aligned} r_1 - r_2 &\leq 0 \\ r_3 - r_1 &\leq 5 \\ r_4 - r_1 &\leq 4 \\ r_4 - r_3 &\leq -1 \\ r_3 - r_2 &\leq 2. \end{aligned}$$

in $N = 4$ *variables. The 1st step is to draw the constraint graph, which is shown in Fig. 4.3.*

Using the Bellman-Ford algorithm (described in Section A.2 of Appendix A), where the origin U *is the node* 5*, we find that there are no negative cycles, so the solution can be found by examining* $r^{(4)}(V)$*. From* $r^{(4)}(1) = 0$*,* $r^{(4)}(2) = 0$*,* $r^{(4)}(3) = 0$*,* $r^{(4)}(4) = -1$*, and* $r^{(4)}(5) = 0$*, a solution to the system of inequalities is determined to be* $r_1 = 0$*,* $r_2 = 0$*,* $r_3 = 0$*, and* $r_4 = -1$*.*

Using the Floyd-Warshall algorithm (described in Section A.3 of Appendix A), we find that there are no cycles, so the solution can be found by examining

$$\mathbf{R}^{(6)} = \begin{bmatrix} \infty & \infty & 5 & 4 & \infty \\ 0 & \infty & 2 & 1 & \infty \\ \infty & \infty & \infty & -1 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 0 & -1 & \infty \end{bmatrix},$$

where the element U, V in the matrix $\mathbf{R}^{(6)}$ is $r^{(6)}(U,V)$. The bottom row of $\mathbf{R}^{(6)}$ gives $r^{(6)}(5,V)$ for $V = 1,2,3,4,5$. The solution to the system of inequalities, given by $r^{(6)}(5,V)$ for $V = 1,2,3,4$, is $r_1 = 0$, $r_2 = 0$, $r_3 = 0$, and $r_4 = -1$. ■

When solving systems of inequalities, there may be multiple inequalities with identical left-hand sides, which can lead to parallel edges in Step 1(c). For example, the 2 inequalities $r(1) - r(2) \leq 9$ and $r(1) - r(2) \leq 7$ would lead to 2 edges from node 2 to node 1 with weights 9 and 7. When this happens, the most restrictive of these inequalities should be selected to avoid drawing parallel edges in Step 1(c). For example, the 2 inequalities $r(1) - r(2) \leq 9$ and $r(1) - r(2) \leq 7$ can be represented by simply using $r(1) - r(2) \leq 7$ because this is the most restrictive of the two, and Step 1(c) results in only 1 edge from node 2 to node 1 with weight 7.

4.4 RETIMING TECHNIQUES

This section considers some techniques used for retiming. First, two special cases of retiming, namely, *cutset retiming* and *pipelining*, are considered. Two algorithms are then considered for retiming to minimize the clock period and retiming to minimize the number of registers that are required to implement the circuit.

4.4.1 Cutset Retiming and Pipelining

Cutset retiming is a useful technique that is a special case of retiming. A *cutset* is a set of edges that can be removed from the graph to create 2 disconnected subgraphs. Cutset retiming only affects the weights of the edges in the cutset. If the 2 disconnected subgraphs are labeled G_1 and G_2, then cutset retiming consists of adding k delays to each edge from G_1 to G_2 and removing k delays from each edge from G_2 to G_1. For example, a cutset is shown with a dashed line in Fig. 4.4(a). The 3 edges in the cutset are $2 \rightarrow 1$, $3 \rightarrow 2$, and $1 \rightarrow 4$. The 2 subgraphs G_1 and G_2 found by removing the 3 edges in the cutset are shown in Fig. 4.4(b). For $k = 1$, the result of cutset retiming is shown in Fig. 4.4(c). The edges from G_1 to G_2 are $3 \rightarrow 2$ and $1 \rightarrow 4$, and one delay

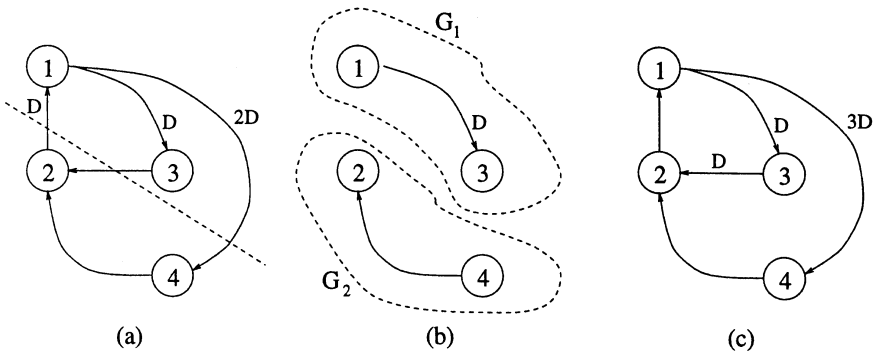

Fig. 4.4 (a) The unretimed DFG with a cutset shown as a dashed line. (b) The 2 graphs G_1 and G_2 formed by removing the edges in the cutset. (c) The retimed graph found using cutset retiming with $k = 1$.

is added to each of these edges. The edge from G_2 to G_1 is $2 \rightarrow 1$, and one delay is subtracted from this edge.

Cutset retiming is a special case of retiming where each node in the subgraph G_1 has the retiming value j and each node in the subgraph G_2 has the retiming value $j+k$. The value of j is unimportant due to Property 4.2.4. For the example in Fig. 4.4, using the values $j = 0$ and $k = 1$ results in $r(1) = 0$, $r(2) = 1$, $r(3) = 0$, and $r(4) = 1$, and this maps the DFG in Fig. 4.4(a) to the DFG in Fig. 4.4(c). Any value of j results in the same retimed graph.

For feasibility of the retimed graph, $w_r(e) \geq 0$ must hold for all edges e in G_r. Let $e_{1,2}$ represent an edge from G_1 to G_2, and let $e_{2,1}$ represent an edge from G_2 to G_1. Since cutset retiming adds k delays to each edge from G_1 to G_2, $w_r(e_{1,2}) \geq 0 \Rightarrow w(e_{1,2}) + k \geq 0$ must hold. Similarly, since k delays are subtracted from each edge $e_{2,1}$ from G_2 to G_1, $w_r(e_{2,1}) \geq 0 \Rightarrow w(e_{2,1}) - k \geq 0$ must hold. Combining these 2 inequalities and considering all of the edges in the cutset result in

$$-\min_{G_1 \overset{e}{\rightarrow} G_2} \{w(e)\} \leq k \leq \min_{G_2 \overset{e}{\rightarrow} G_1} \{w(e)\}$$

as the condition on k for cutset retiming to give a feasible solution. In Fig. 4.4, $\min_{G_1 \overset{e}{\rightarrow} G_2} \{w(e)\} = \min\{0, 2\} = 0$ and $\min_{G_2 \overset{e}{\rightarrow} G_1} \{w(e)\} = 1$, so $0 \leq k \leq 1$ must hold. Since $k = 0$ does nothing, $k = 1$ is the only value of k that results in a feasible retimed graph, and this graph is shown in Fig. 4.4(c).

It is interesting to see what happens when the cutset is chosen so the subgraph G_2 is a single node and the subgraph G_1 is the rest of the graph minus the edges going into and out of the chosen node. An example of such a cutset is shown in Fig. 4.5(a), and the 2 subgraphs G_1 and G_2 are shown in Fig. 4.5(b). The graph G_2 consists of the node 2, and the graph G_1 consists of the rest of the graph except the 3 edges that go into and out of the node 2.

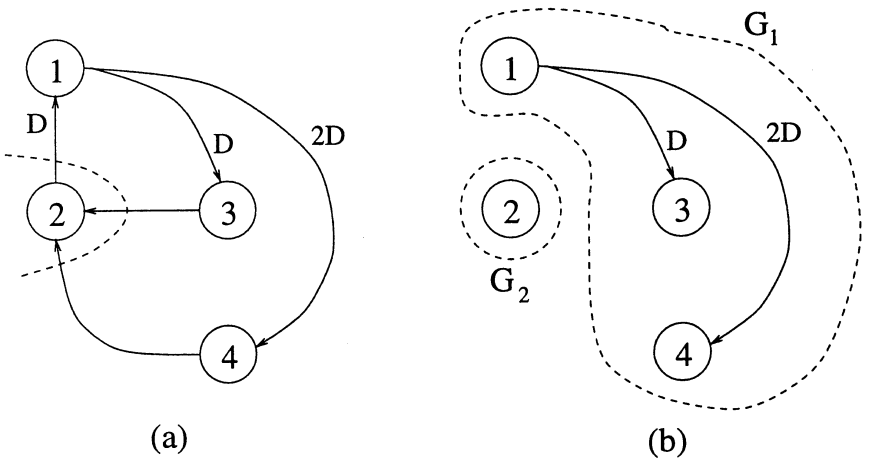

Fig. 4.5 (a) The unretimed DFG with a cutset shown as a dashed line. (b) The 2 graphs G_1 and G_2 formed by removing the edges in the cutset. The retimed graph is shown in Fig. 4.2(c).

The result of cutset retiming in this case is found by adding 1 delay to each edge incident into the node 2 and subtracting 1 delay from each edge outgoing from the node 2 as shown in Fig. 4.2(c). Therefore, this special case of cutset retiming consists of choosing a node as a cutset and subtracting one delay from each edge outgoing from the node and adding one delay to each edge incident into the node.

Pipelining is a special case of cutset retiming where there are no edges in the cutset from the subgraph G_2 to the subgraph G_1, i.e., pipelining applies to graphs without loops. These cutsets are referred to as feed-forward cutsets. The feed-forward cutset shown in Fig. 4.6(a) divides the graph into the 2 subgraphs shown in Fig. 4.6(b). All 3 of the edges in the cutset go from G_1 to G_2, and performing cutset retiming with $k = 2$ results in 2 additional delays on each edge in the cutset, resulting in the retimed (or pipelined, in this case) graph in Fig. 4.6(c). This demonstrates that pipelining can be viewed as a special case of retiming.

Cutset retiming is often used in combination with *slow-down*. The procedure is to first replace each delay in the DFG with N delays to create an *N-slow* version of the DFG and then to perform cutset retiming on the N-slow DFG. Note that in an N-slow system, $N - 1$ null operations (or 0 samples) must be interleaved after each useful signal sample to preserve the functionality of the algorithm. For example, consider the 100-stage lattice filter in Fig. 4.7(a), which has a critical path of 101 adders and 2 multipliers. Assuming that addition and multiplication take 1 and 2 u.t., respectively, the minimum sample period is $(101)(1) + (2)(2) = 105$ u.t. The 2-slow version of this circuit is shown in Fig. 4.7(b), and cutset retiming can be used to obtain

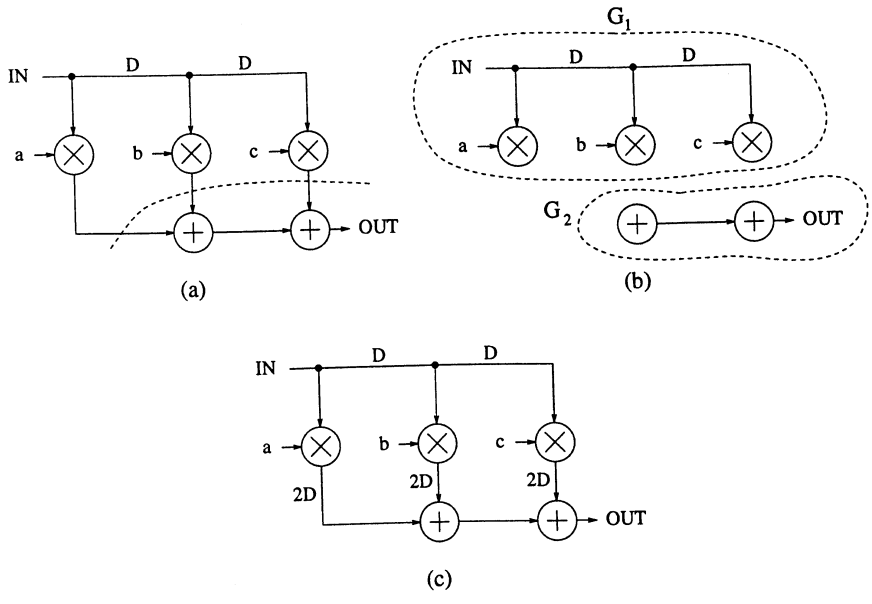

Fig. 4.6 (a) The unretimed DFG with a cutset shown as a dashed line. (b) The 2 graphs G_1 and G_2 formed by removing the edges in the cutset. (c) The graph obtained by cutset retiming with $k = 2$.

the circuit in Fig. 4.7(c). The critical path of the retimed circuit has 2 adders and 2 multipliers and has computation time (2)(1) + (2)(2) = 6 u.t. Since this circuit is 2-slow, the minimum sample period is (2)(6) = 12 u.t. In this example, slow-down and cutset retiming reduce the sample period from 105 u.t. to 12 u.t.

To summarize, cutset retiming is a special case of retiming, and pipelining is a special case of cutset retiming. Cutset retiming and pipelining are graphical techniques that can be used to perform complex retiming operations in a simple manner.

4.4.2 Retiming for Clock Period Minimization

This section presents a retiming algorithm for minimizing the clock period of a synchronous circuit. For a circuit G, the minimum feasible clock period is the computation time of the critical path, which is the path with the longest computation time among all paths with no delays. Mathematically, the minimum feasible clock period, $\Phi(G)$, is defined as

$$\Phi(G) = \max\{t(p) : w(p) = 0\}.$$

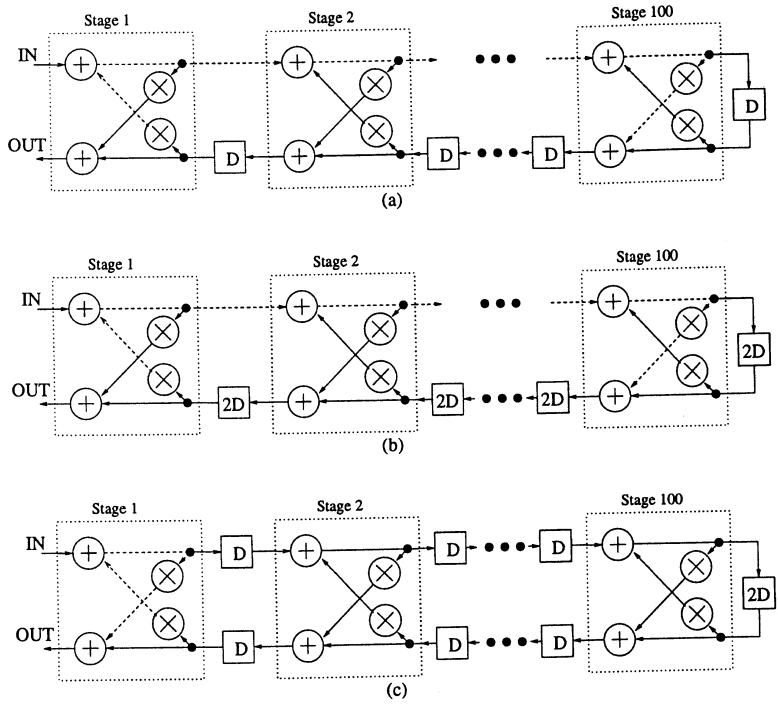

Fig. 4.7 In each of the 3 filters in this figure, the critical path is shown with dotted lines, and addition and multiplication are assumed to take 1 and 2 u.t., respectively. (a) A 100-stage lattice filter with minimum sample period of 105 u.t. (b) The 2-slow version of the circuit. (c) A retimed version of the 2-slow circuit with critical path of 6 u.t. and minimum sample period of 12 u.t.

The algorithm presented in this section finds a retiming solution r_0 such that $\Phi(G_{r_0}) \leq \Phi(G_r)$ for any other retiming solution r.

The 2 quantities $W(U,V)$ and $D(U,V)$ are used in this algorithm. The quantity $W(U,V)$ is the minimum number of registers on any path from node U to node V, and $D(U,V)$ is the maximum computation time among all paths from U to V with weight $W(U,V)$. Formally,

$$\begin{aligned} W(U,V) &= \min\{w(p) : U \overset{p}{\rightsquigarrow} V\} \\ D(U,V) &= \max\{t(p) : U \overset{p}{\rightsquigarrow} V \text{ and } w(p) = W(U,V)\}. \end{aligned}$$

The following algorithm can be used to compute $W(U,V)$ and $D(U,V)$.

1. Let $M = t_{max}n$, where t_{max} is the maximum computation time of the nodes in G and n is the number of nodes in G.

2. Form a new graph G' which is the same as G except the edge weights are replaced by $w'(e) = Mw(e) - t(U)$ for all edges $U \overset{e}{\rightarrow} V$.

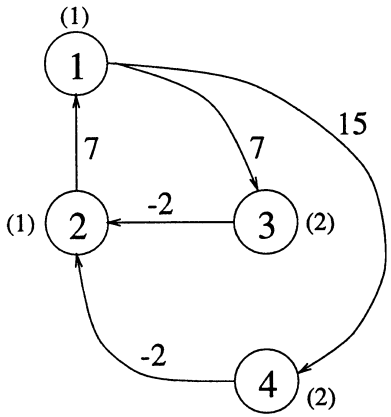

Fig. 4.8 The graph G' used to compute $W(U,V)$ and $D(U,V)$ for the DFG in Fig. 4.2(a).

3. Solve the all-pairs shortest path problem on G'. Let S'_{UV} be the shortest path from U to V.

4. If $U \neq V$, then $W(U,V) = \left\lceil \frac{S'_{UV}}{M} \right\rceil$ and $D(U,V) = MW(U,V) - S'_{UV} + t(V)$. If $U = V$, then $W(U,V) = 0$ and $D(U,V) = t(U)$.

The value of $\lceil x \rceil$ is the ceiling of x, which is the smallest integer greater than or equal to x.

To demonstrate this algorithm, the values of $W(U,V)$ and $D(U,V)$ are computed for the DFG in Fig. 4.2(a). In the first step, $t_{max} = 2$ and $n = 4$, so $M = 8$. The new graph G' (found in the second step) is shown in Fig. 4.8. The solution to the all-pairs shortest path problem for G' can be found using the Floyd-Warshall algorithm, which is described in Section A.3 of Appendix A. This solution (found in the third step), and the values of $W(U,V)$ and $D(U,V)$ (found in the final step) are given in Table 4.1.

The values of $W(U,V)$ and $D(U,V)$ are used to determine if there is a retiming solution that can achieve a desired clock period. Given a desired clock period c, there is a feasible retiming solution r such that $\Phi(G_r) \leq c$ if the following constraints hold

1. (feasibility constraint) $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$ of G, and

2. (critical path constraint) $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V in G such that $D(U,V) > c$.

The feasibility constraint forces the number of delays on each edge in the retimed graph to be nonnegative, and the critical path constraint enforces

Table 4.1 Values of S'_{UV}, $W(U,V)$, and $D(U,V)$ for the DFG in Fig. 4.2(a)

S'_{UV}	1	2	3	4
1	12	5	7	15
2	7	12	14	22
3	5	-2	12	20
4	5	-2	12	20

$W(U,V)$	1	2	3	4
1	0	1	1	2
2	1	0	2	3
3	1	0	0	3
4	1	0	2	0

$D(U,V)$	1	2	3	4
1	1	4	3	3
2	2	1	4	4
3	4	3	2	6
4	4	3	6	2

$\Phi(G) \leq c$. If $D(U,V) > c$, then $W(U,V) + r(V) - r(U) \geq 1$ must hold for the critical path to have computation time less than or equal to c. This leads to the critical path constraint. For the DFG in Fig. 4.2(a), if c is chosen to be 3, the inequalities $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$ are

$$\begin{aligned} r(1) - r(3) &\leq 1 \\ r(1) - r(4) &\leq 2 \\ r(2) - r(1) &\leq 1 \\ r(3) - r(2) &\leq 0 \\ r(4) - r(2) &\leq 0, \end{aligned} \tag{4.2}$$

and the inequalities $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V in G such that $D(U,V) > 3$, found using Table 4.1, are

$$\begin{aligned} r(1) - r(2) &\leq 0 \\ r(2) - r(3) &\leq 1 \\ r(2) - r(4) &\leq 2 \\ r(3) - r(1) &\leq 0 \\ r(3) - r(4) &\leq 2 \\ r(4) - r(1) &\leq 0 \\ r(4) - r(3) &\leq 1. \end{aligned} \tag{4.3}$$

If there is a solution to the 12 inequalities in (4.2) and (4.3), then the solution is a feasible retiming solution such that the circuit can be clocked with period $c = 3$. Checking for a solution can be performed by constructing a constraint graph and solving a single-source shortest path problem, as described in Section 4.3. The constraint graph for the 12 inequalities in (4.2) and (4.3) is shown in

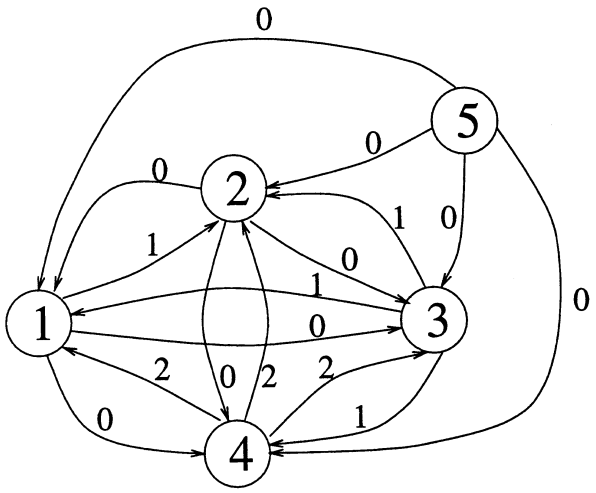

Fig. 4.9 The constraint graph for the inequalities in (4.2) and (4.3).

Fig. 4.9. Using either the Bellman-Ford algorithm or the Floyd-Warshall algorithm, it can be determined that the constraint graph in Fig. 4.9 contains no negative cycles, so the retiming solution for $c = 3$ is the solution to the single-source shortest path problem with the origin at the node 5, which is $r(1) = r(2) = r(3) = r(4) = 0$. This is intuitive since the DFG in Fig. 4.2(a) already has $\Phi(G) = 3$ so no retiming needs to be performed to achieve a clock period of 3 u.t.

The fact that the 12 inequalities in (4.2) and (4.3) have a solution indicates that the minimum clock period for the circuit in Fig. 4.2(a) is less than or equal to 3. It has been shown that the minimum clock period for the DFG is $D(U,V)$ for some pair U,V [1]. To find the minimum feasible clock period, we can attempt to solve the retiming equations for c equal to each unique value of $D(U,V)$ and find the minimum value of c with a solution. From the $D(U,V)$ values in Table 4.1, the unique values of $D(U,V)$ are 1, 2, 3, 4, and 6. A clock period of $c = 1$ is not feasible because the nodes 3 and 4 have computation time greater than 1 u.t. A clock period of $c = 3$ is feasible because the constraint graph in Fig. 4.9 has a solution, so in order to determine the minimum feasible clock period for the DFG in Fig. 4.2(a), only the value of $c = 2$ must be checked.

For $c = 2$, the inequalities $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$ are given in (4.2) (these inequalities are a function of the DFG and not c) and the inequalities $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U,V in G such that $D(U,V) > 2$ are

$$\begin{aligned} r(1) - r(2) &\leq 0 \\ r(1) - r(3) &\leq 0 \end{aligned}$$

$$
\begin{aligned}
r(1) - r(4) &\leq 1 \\
r(2) - r(3) &\leq 1 \\
r(2) - r(4) &\leq 2 \\
r(3) - r(1) &\leq 0 \\
r(3) - r(2) &\leq -1 \\
r(3) - r(4) &\leq 2 \\
r(4) - r(1) &\leq 0 \\
r(4) - r(2) &\leq -1 \\
r(4) - r(3) &\leq 1.
\end{aligned} \tag{4.4}
$$

The equations for a feasible retiming solution with clock period $c = 2$ are given in (4.2) and (4.4). Combining the equations in (4.2) and (4.4) that have identical left-hand sides results in the set of equations

$$
\begin{aligned}
r(1) - r(3) &\leq 0 \\
r(1) - r(4) &\leq 1 \\
r(2) - r(1) &\leq 1 \\
r(3) - r(2) &\leq -1 \\
r(4) - r(2) &\leq -1 \\
r(1) - r(2) &\leq 0 \\
r(2) - r(3) &\leq 1 \\
r(2) - r(4) &\leq 2 \\
r(3) - r(1) &\leq 0 \\
r(3) - r(4) &\leq 2 \\
r(4) - r(1) &\leq 0 \\
r(4) - r(3) &\leq 1.
\end{aligned} \tag{4.5}
$$

The constraint graph for the 12 inequalities in (4.5) is shown in Fig. 4.10. Using the Bellman-Ford algorithm or Floyd-Warshall algorithm, it can be determined that this graph contains no negative cycles, so the retiming solution for $c = 2$ is the solution to the single-source shortest path problem with the origin at the node 5, which is $r(1) = -1$, $r(2) = 0$, $r(3) = -1$, and $r(4) = -1$. This solution, which has clock period $\Phi(G_r) = 2$, is the retimed version of Fig. 4.2(a) and is shown in Fig. 4.2(b).

The general approach to finding a retiming solution with the minimum clock period is to 1st compute $W(U, V)$ and $D(U, V)$ and then sort the values of $D(U, V)$ and perform a *binary search* on these values to find the retiming solution with the minimum clock period. Computation of $W(U, V)$ and $D(U, V)$ can be done in $\mathcal{O}(n^3)$ time using the Floyd-Warshall algorithm. The maximum number of distinct values of $D(U, V)$ is n^2 (if all values of $D(U, V)$ are

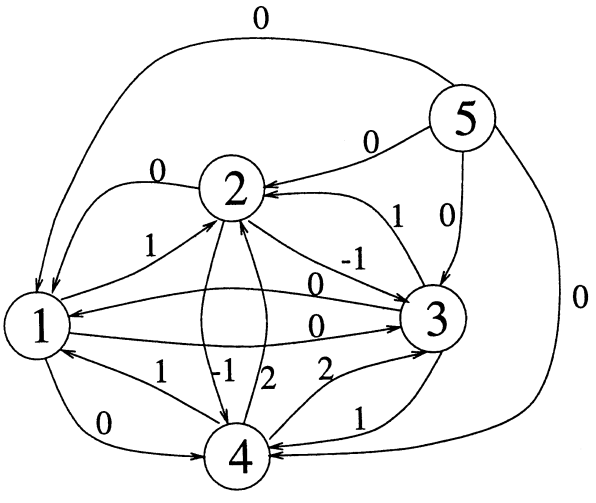

Fig. 4.10 The constraint graph for the inequalities in (4.5).

distinct), so the binary search takes $\mathcal{O}(\log n)$ steps. Checking if one of these values is greater than or equal to the minimum clock period takes $\mathcal{O}(n^3)$ time using either the Floyd-Warshall algorithm or the Bellman-Ford algorithm. Therefore, the time complexity required to find a retimed circuit with the minimum clock period is $\mathcal{O}(n^3 \log n)$. Note that if the iteration bound is known, then only those values in $D(U,V)$ that are greater than or equal to the iteration bound should be considered.

4.4.3 Retiming for Register Minimization

This section presents an algorithm for finding a retiming solution that uses the minimum number of registers while satisfying the clock period constraint.

If a node has several output edges carrying the same signal, the number of registers required to implement these edges is the maximum number of registers on any one of the edges. This is demonstrated in Fig. 4.11, where the naive implementation in Fig. 4.11(a) uses $1+3+7 = 11$ registers while the clever implementation in Fig. 4.11(b) uses $\max(1,3,7) = 7$ registers. Using this concept, the number of registers required to implement the output edges of the node V in the retimed graph is

$$R_V = \max_{V \xrightarrow{e} ?} \{w_r(e)\},$$

and the total register cost in the retimed circuit is $COST = \sum R_V$.

The formulation of retiming to minimize the number of registers under the constraint that the clock period is not greater than c is: Minimize $COST =$

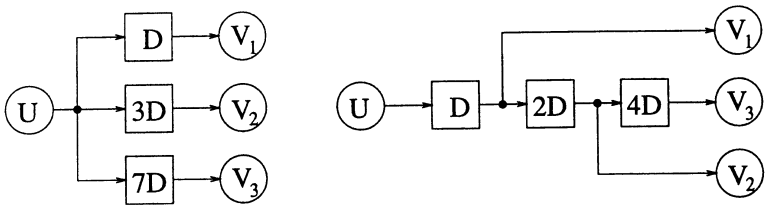

Fig. 4.11 (a) Fanout implementation using $1 + 3 + 7 = 11$ registers. (b) Fanout implementation using $\max(1, 3, 7) = 7$ registers.

$\sum R_V$ subject to

1. (fanout constraint) $R_V \geq w_r(e)$ for all V and all edges $V \xrightarrow{e} ?$.
2. (feasibility constraint) $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$.
3. (clock period constraint) $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V such that $D(U,V) > c$.

The fanout constraint simply makes sure that $R_V = \max_{V \xrightarrow{e} ?}\{w_r(e)\}$. An algorithm for computing $W(U,V)$ and $D(U,V)$ is presented in Section 4.4.2, along with descriptions of the feasibility constraint and clock period constraint.

While this formulation of retiming indeed minimizes the number of registers under a clock period constraint, it is not in a form that can be solved using linear programming (LP) techniques because some solutions may not be integers. To get the formulation in such a form, a "gadget" is used to represent nodes with multiple output edges [1]. This gadget is shown in Fig. 4.12. Fig. 4.12(a) shows a fanout node with k output edges. The gadget in Fig. 4.12(b) is used to model the fanout node. Each of the k edges e_i, $1 \leq i \leq k$, has an associated weight $w(e_i)$ which is known from the DFG. The node $\hat{U}$ is a dummy node with zero computation time ($t(\hat{U}) = 0$), and the edges $\hat{e}_i$, $1 \leq i \leq k$, are dummy edges introduced so the retiming for register minimization problem can be modeled as a linear programming problem. The weight of the edge $\hat{e}_i$ is defined to be $w(\hat{e}_i) = w_{max} - w(e_i)$, where $w_{max} = \max_{1 \leq i \leq k} w(e_i)$. In addition to its weight, each edge in this model also has a breadth β associated with it. This breadth of an edge is simply a number used so that the gadget in Fig. 4.12(b) properly models the memory required by the edges e_i, $1 \leq i \leq k$, in the retimed DFG (the details of this are explained a bit later). The breadth of each of the edges e_i and $\hat{e}_i$ for $1 \leq i \leq k$ is $\beta = 1/k$, as shown in Fig. 4.12(b).

Using the fanout model in Fig. 4.12, the retiming formulation is: Minimize $COST = \sum_e \beta(e) w_r(e)$ subject to

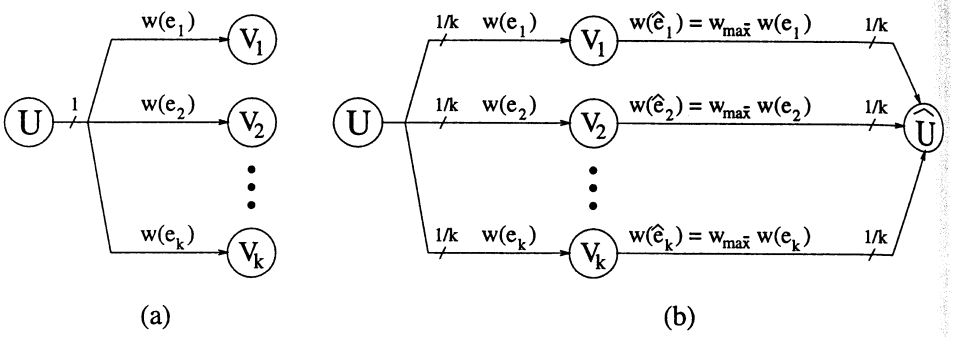

Fig. 4.12 (a) A node U with k output edges. (b) A gadget used to model the node U.

1. (feasibility constraint) $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$.
2. (clock period constraint) $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V such that $D(U,V) > c$.

The expression for $COST$ can be rewritten as

$$\begin{aligned} COST &= \sum_e \beta(e) w_r(e) \\ &= \sum_e \beta(e)(w(e) + r(V) - r(U)) \\ &= \sum_e \beta(e) w(e) + \sum_e \beta(e)(r(V) - r(U)) \\ &= K + \sum_e \beta(e)(r(V) - r(U)) \\ &= K + \sum_V r(V) \left(\sum_{?\xrightarrow{e}V} \beta(e) - \sum_{V\xrightarrow{e}?} \beta(e) \right). \end{aligned}$$

Since K is a constant, the formulation can be written as Minimize $COST' = \sum_V r(V) \left(\sum_{?\xrightarrow{e}V} \beta(e) - \sum_{V\xrightarrow{e}?} \beta(e) \right)$ subject to

1. (feasibility constraint) $r(U) - r(V) \leq w(e)$ for every edge $U \xrightarrow{e} V$.
2. (clock period constraint) $r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V such that $D(U,V) > c$.

To demonstrate this, the DFG in Fig. 4.2(a) is retimed so the number of registers is minimized while maintaining a clock period of $\Phi(G_r) \leq 2$. The 1st step is to redraw this DFG as shown in Fig. 4.13(a) using the fanout model for node 1 because this node has more than one output edge. The 2nd step

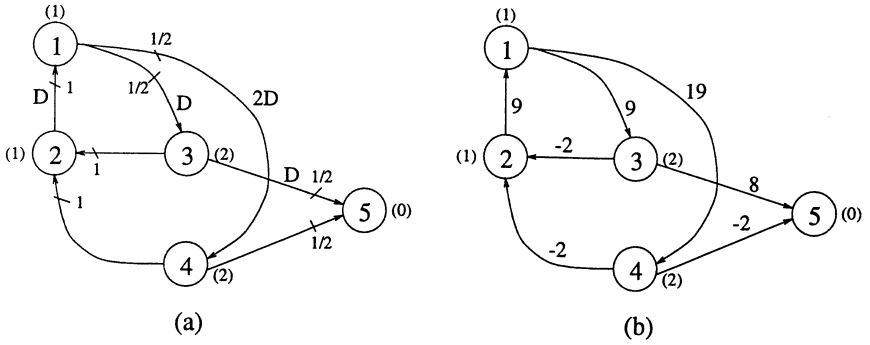

Fig. 4.13 (a) The DFG in Fig. 4.2(a) redrawn using the fanout gadget in Fig. 4.12. (b) The graph G' used to compute $W(U,V)$ and $D(U,V)$.

Table 4.2 Values of S'_{UV}, $W(U,V)$, and $D(U,V)$ for the DFG in Fig. 4.13(a)

S'_{UV}	1	2	3	4	5
1	16	7	9	19	17
2	9	16	18	28	26
3	7	-2	16	26	8
4	7	-2	16	26	-2
5	-	-	-	-	-

$W(U,V)$	1	2	3	4	5
1	0	1	1	2	2
2	1	0	2	3	3
3	1	0	0	3	1
4	1	0	2	0	0
5	-	-	-	-	0

$D(U,V)$	1	2	3	4	5
1	1	4	3	3	3
2	2	1	4	4	4
3	4	3	2	6	2
4	4	3	6	2	2
5	-	-	-	-	0

is to compute $W(U,V)$ and $D(U,V)$ for the DFG in Fig. 4.13(a). Using the procedure described in Section 4.4.2, $M = 10$, G' is shown in Fig. 4.13(b), and the values of S'_{UV}, $W(U,V)$, and $D(U,V)$ are shown in Table 4.2.

The LP formulation of the problem is: Minimize

$$\begin{aligned} COST' &= r(1)(1-1) + r(2)(2-1) + r(3)(1/2-3/2) + r(4)(1/2-3/2) \\ &\quad + r(5)(1-0) \\ &= r(2) - r(3) - r(4) + r(5) \end{aligned}$$

subject to the feasibility constraints

$$\begin{aligned} r(1) - r(3) &\leq 1 \\ r(1) - r(4) &\leq 2 \end{aligned}$$

$$\begin{aligned} r(2) - r(1) &\leq 1 \\ r(3) - r(2) &\leq 0 \\ r(3) - r(5) &\leq 1 \\ r(4) - r(2) &\leq 0 \\ r(4) - r(5) &\leq 0 \end{aligned}$$

and the clock period constraints

$$\begin{aligned} r(1) - r(2) &\leq 0 \\ r(1) - r(3) &\leq 0 \\ r(1) - r(4) &\leq 1 \\ r(1) - r(5) &\leq 1 \\ r(2) - r(3) &\leq 1 \\ r(2) - r(4) &\leq 2 \\ r(2) - r(5) &\leq 2 \\ r(3) - r(1) &\leq 0 \\ r(3) - r(2) &\leq -1 \\ r(3) - r(4) &\leq 2 \\ r(4) - r(1) &\leq 0 \\ r(4) - r(2) &\leq -1 \\ r(4) - r(3) &\leq 1. \end{aligned}$$

This linear programming problem is in the form of a minimum-cost flow problem, for which algorithms exist (e.g., see [3]); however, these algorithms can be quite involved. To solve this problem, a linear programming software package such as GAMS [4] can be used. Using GAMS, the solution to this problem is $r(1) = 1$, $r(2) = 2$, $r(3) = 1$, $r(4) = 0$, and $r(5) = 0$, and the retimed graph, which uses proper register sharing between the 2 output edges of the node 1, is shown in Fig. 4.14.

It remains to show that the fanout gadget in Fig. 4.12(b) properly models the amount of memory used by a node with multiple output edges. The goal is for the sum of the weights (scaled by the breadths β) of all of the edges e_i and $\hat{e}_i$ in the fanout gadget to equal the maximum of the weights on the edges e_i in the gadget. In other words, the goal is to show that

$$\sum_{i=1}^{k} (w_r(e_i)\beta(e_i) + w_r(\hat{e}_i)\beta(\hat{e}_i)) = w_{r,max},$$

where $w_{r,max} = \max_{1 \leq i \leq k} w_r(e_i)$. This is shown using the following 3 steps.

Step 1: Assume that $w_r(e_j) = w_{r,max}$. Show that $w_r(e_j) = w_{max} + r(\hat{U}) - r(U)$.

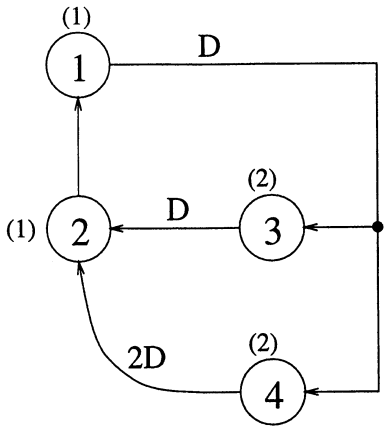

Fig. 4.14 The result of retiming the DFG in Fig. 4.2(a) to minimize the number of registers while maintaining a clock period of 2 u.t.

Step 2: Show that

$$\sum_{i=1}^{k} (w_r(e_i)\beta(e_i) + w_r(\hat{e}_i)\beta(\hat{e}_i)) = w_{max} + r(\hat{U}) - r(U).$$

Step 3: Combining steps 1 and 2 results in

$$\sum_{i=1}^{k} (w_r(e_i)\beta(e_i) + w_r(\hat{e}_i)\beta(\hat{e}_i)) = w_{max} + r(\hat{U}) - r(U) = w_r(e_j) = w_{r,max}.$$

While step 3 is trivial (assuming that steps 1 and 2 can be shown), steps 1 and 2 are not trivial. These 2 steps are described in detail.

Details of Step 1: Assume that $w_r(e_j) = w_{r,max}$. The weight of each path $p_i = U \xrightarrow{e_i} V_i \xrightarrow{\hat{e}_i} \hat{U}$ in the original DFG is

$$\begin{aligned} w(p_i) &= w(e_i) + (w_{max} - w(e_i)) \\ &= w_{max}. \end{aligned}$$

The weight of each path p_i in the retimed DFG is

$$\begin{aligned} w_r(p_i) &= w_r(e_i) + w_r(\hat{e}_i) \\ &= (w(e_i) + r(V_i) - r(U)) + \left(w_{max} - w(e_i) + r(\hat{U}) - r(V_i)\right) \\ &= w_{max} + r(\hat{U}) - r(U). \end{aligned}$$

Solving to minimize $COST$ causes $r(\hat{U})$ to be as small as possible while still

forcing $w_r(\hat{e}_i) \geq 0$ for $1 \leq i \leq k$. Therefore, $w_r(\hat{e}_i) = 0$ for at least 1 edge $\hat{e}_i$. Since the value of $w_r(p_i) = w_r(e_i) + w_r(\hat{e}_i) = w_{max} + r(\hat{U}) - r(U)$ is the same for $1 \leq i \leq k$, the edge e_j with $w_r(e_j) = w_{r,max}$ is in the path p_j which has $w_r(\hat{e}_j) = 0$. Thus, $w_r(e_j) = w_r(p_j) = w_{max} + r(\hat{U}) - r(U)$. ■

Details of Step 2: The cost of the edges in the fanout model is

$$\begin{aligned} COST &= \sum_{i=1}^{k} (w_r(e_i)\beta(e_i) + w_r(\hat{e}_i)\beta(\hat{e}_i)) \\ &= \frac{1}{k}\left(\sum_{i=1}^{k} (w_r(e_i) + w_r(\hat{e}_i))\right) \\ &= \frac{1}{k}(k)\left(w_{max} + r(\hat{U}) - r(U)\right). \blacksquare \end{aligned}$$

4.5 CONCLUSIONS

Synchronous systems can be retimed to reduce critical path or clock period, number of storage or delay elements, or power consumption. Shortest path algorithms can be used to obtain a retiming solution if one exists. All possible retiming solutions can also be obtained by *exhaustive retiming* using approaches in [5],[6]. Retiming can also be used as a preprocessing step for folding and for computation of roundoff noise as discussed in Chapters 6 and 11, respectively. Use of retiming for reduction of power consumption is covered in Section 17.5.4. Retiming of multirate DFGs is addressed in the context of multirate folding in Section 6.5. Retiming of 2D DFGs has been addressed in [7],[8].

4.6 PROBLEMS

1. Consider the wave digital filter shown in Fig. 4.15. Assume that each multiply operation requires 20 nsec and each add operation requires 8 nsec.

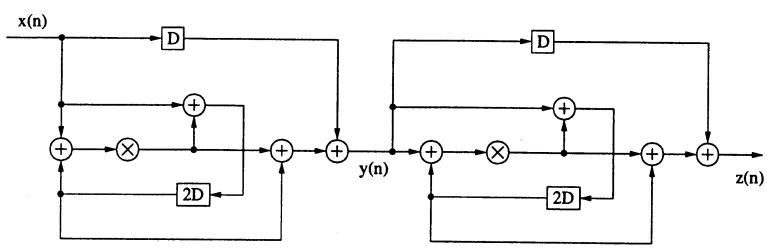

Fig. 4.15 The wave digital filter structure in Problem 1.

Appendix A: Shortest Path Algorithms

A.1 INTRODUCTION

In this appendix, the Bellman-Ford algorithm and the Floyd-Warshall algorithm shortest path algorithms are described. These algorithms can be used to solve several problems in VLSI signal processing such as computing the iteration bound of recursive DFGs (see Chapter 2), retiming (see Chapter 4), and retiming for folding (see Chapter 6).

The level of detail included here is intended to be enough so that the reader can use these algorithms to solve the problems mentioned above. The theory behind these algorithms has been omitted; however, for the interested reader, there is a reference that discusses the theory behind these algorithms (e.g., [1], [2]).

The concepts of shortest paths and negative cycles are demonstrated using the DFG in Figure A.1. The length of a path is the sum of the weights on the edges of the path. The path $2 \rightarrow 3 \rightarrow 4$ has length $1 + 2 = 3$. There are 2 paths from node 2 to node 4, namely the path $2 \rightarrow 4$ with length 1 and the path $2 \rightarrow 3 \rightarrow 4$ with length 3. The shortest path from node 2 to node 4 is $\min\{1, 3\} = 1$. A negative cycle in a directed graph is a directed cycle such that the sum of the lengths on the edges of the cycle is negative. Figure A.1

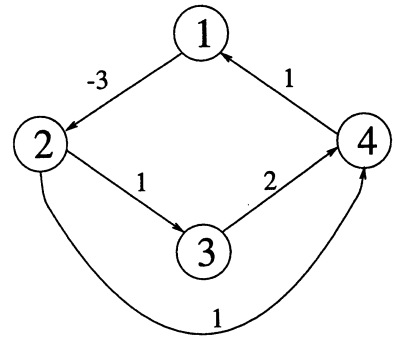

Fig. A.1 A graph containing 1 negative cycle.

contains the 2 cycles $2 \to 4 \to 1 \to 2$ and $2 \to 3 \to 4 \to 1 \to 2$. The cycle $2 \to 4 \to 1 \to 2$ is a negative cycle because the sum of the edge weights is $1 + 1 + (-3) = -1$. The other cycle, $2 \to 3 \to 4 \to 1 \to 2$, is not a negative cycle because the sum of the edge weights is $1 + 2 + 1 + (-3) = 1$. With this in mind, each of the 2 shortest path algorithms works as follows. If there are no negative cycles in the graph, the algorithm returns TRUE and provides some information about the lengths of the shortest path between the nodes (the amount of information varies between the two algorithms, as we will see in the following sections). If there exists at least 1 negative cycle in the graph, the algorithm returns FALSE. To summarize, each of the algorithms searches for a negative cycle and provides shortest path information if no negative cycle exists.

The following assumptions and notation are used. We assume that no parallel edges exist in the graph, i.e., for 2 nodes U and V, there is at most 1 edge from U to V ($U \xrightarrow{e} V$) and at most 1 edge from V to U ($V \xrightarrow{e} U$). Let

$$w(U \xrightarrow{e} V) = \begin{cases} \text{the length of the edge } U \xrightarrow{e} V, & \text{if } U \xrightarrow{e} V \text{ exists} \\ \infty, & \text{otherwise} \end{cases},$$

where the length of the edge is simply a number associated with the edge. Let n be the number of nodes in the graph, and assume that the n nodes are numbered $1, 2, \ldots, n$. The shortest path from the node U to the node V is denoted as S_{UV}.

A.2 THE BELLMAN-FORD ALGORITHM

The Bellman-Ford algorithm is a single-point shortest path algorithm. If no negative cycles exist in the graph, it finds the shortest path from an arbitrarily chosen node U (called the origin) to each node in the graph. For a graph with n nodes, the algorithm constructs $n-1$ vectors $\mathbf{r}^{(k)}$, $k = 1, 2, \ldots, n-1$, which

1. $r^{(1)}(U) = 0$
2. *For* $k = 1$ *to* n
3. *If* $k \neq U$
4. $r^{(1)}(k) = w(U \xrightarrow{e} k)$
5. *For* $k = 1$ *to* $n - 2$
6. *For* $V = 1$ *to* n
7. $r^{(k+1)}(V) = r^{(k)}(V)$
8. *For* $W = 1$ *to* n
9. *If* $r^{(k+1)}(V) > r^{(k)}(W) + w(W \xrightarrow{e} V)$
10. $r^{(k+1)}(V) = r^{(k)}(W) + w(W \xrightarrow{e} V)$
11. *For* $V = 1$ *to* n
12. *For* $W = 1$ *to* n
13. *If* $r^{(n-1)}(V) > r^{(n-1)}(W) + w(W \xrightarrow{e} V)$
14. *return FALSE and exit*
15. *return TRUE and exit*

Fig. A.2 The Bellman-Ford algorithm.

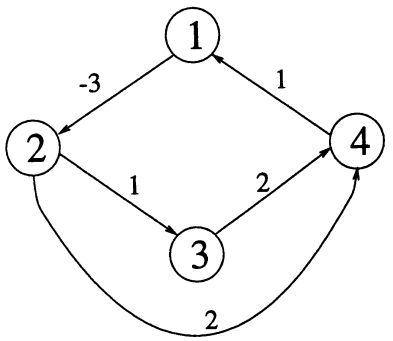

Fig. A.3 A graph containing no negative cycles.

are each of size $n \times 1$, as described in Figure A.2. Note that $\mathbf{r}^{(k)}(U)$ represents the U-th entry in the column vector $\mathbf{r}^{(k)}$.

If TRUE is returned, then $r^{(n-1)}(V)$ is S_{UV}, which is the shortest path from the node U to the node V. If FALSE is returned, then the graph contains at least one negative cycle.

Example A.2.1 *In this example, the Bellman-Ford algorithm in Figure A.2 is used to find the shortest path from the node* $U = 2$ *to the nodes in the graph in Figure A.3. The values of* $r^{(k)}(V)$ *for* $k = 1, 2, 3$ *and* $V = 1, 2, 3, 4$ *are shown in Table A.1. The Bellman-Ford algorithm returns TRUE in this example, so the value of* $r^{(3)}(V)$ *is the shortest path from node 2 to node V for* $V = 1, 2, 3, 4$. ■

Table A.1 Values of $r^{(k)}(V)$ for $k = 1, 2, 3$ and $V = 1, 2, 3, 4$ from Example A.2.1

$r^{(k)}(V)$	$k=1$	$k=2$	$k=3$
$V=1$	∞	3	3
$V=2$	0	0	0
$V=3$	1	1	1
$V=4$	2	2	2

Table A.2 Values of $r^{(k)}(V)$ for $k = 1, 2, 3$ and $V = 1, 2, 3, 4$ from Example A.2.2

$r^{(k)}(V)$	$k=1$	$k=2$	$k=3$
$V=1$	∞	2	2
$V=2$	0	0	-1
$V=3$	1	1	1
$V=4$	1	1	1

Example A.2.2 *In this example, the Bellman-Ford algorithm is used to find the shortest path from the origin $U = 2$ to the nodes in the graph in Figure A.1. The values of $r^{(k)}(V)$ for $k = 1, 2, 3$ and $V = 1, 2, 3, 4$ are shown in Table A.2. In this example, the Bellman-Ford algorithm returns FALSE because $r^{(3)}(3) > r^{(3)}(2) + w(2 \stackrel{e}{\rightarrow} 3)$, so a negative cycle is detected. This negative cycle can be seen in Figure A.1, where the cycle $2 \rightarrow 4 \rightarrow 1 \rightarrow 2$ has length $1 + 1 - 3 = -1$.* ■

A.3 THE FLOYD-WARSHALL ALGORITHM

The Floyd-Warshall algorithm is an all-points shortest path algorithm. If no negative cycles exist in the graph, the algorithm finds the shortest path between all possible pairs of nodes in the graph. The algorithm constructs $n + 1$ matrices $\mathbf{R}^{(k)}$, $k = 1, 2, \ldots, n + 1$, which are each of size $n \times n$, as described in Figure A.4.

If TRUE is returned, then $r^{(n+1)}(U, V)$ is S_{UV}, which is the shortest path from the node U to the node V. If FALSE is returned, then the graph contains a negative cycle.

Example A.3.1 *In this example, the Floyd-Warshall algorithm in Figure A.4 is used to solve the all-pairs shortest path problem for the graph in Figure A.3. The values of $r^{(k)}(U, V)$ for $U, V \in \{1, 2, 3, 4\}$ and $k = 1, 2, 3, 4, 5$ are given in Table A.3, where $r^{(k)}(U, V)$ is the element U, V in the matrix $\mathbf{R}^{(k)}$. The*

```
1.   For V = 1 to n
2.       For U = 1 to n
3.           r^(1)(U,V) = w(U -e-> V)
4.   For k = 1 to n
5.       For V = 1 to n
6.           For U = 1 to n
7.               r^(k+1)(U,V) = r^(k)(U,V)
8.               If r^(k+1)(U,V) > r^(k)(U,k) + r^(k)(k,V)
9.                   r^(k+1)(U,V) = r^(k)(U,k) + r^(k)(k,V)
10.  For k = 1 to n
11.      For U = 1 to n
12.          If r^(k)(U,U) < 0
13.              return FALSE and exit
14.  return TRUE and exit
```

Fig. A.4 The Floyd-Warshall algorithm.

Floyd-Warshall algorithm returns TRUE because all of the diagonal elements in the matrices $\mathbf{R}^{(k)}$, $k = 1, 2, 3, 4$, *are nonnegative, so* $r^{(5)}(U, V)$ *is the shortest path from the node* U *to the node* V. ■

Example A.3.2 *In this example, the Floyd-Warshall algorithm is used to solve the all-pairs shortest path problem for the graph in Figure A.1. The values of* $r^{(k)}(U, V)$ *for* $U, V \in \{1, 2, 3, 4\}$ *and* $k = 1, 2, 3, 4, 5$ *are given in Table A.4, where* $r^{(k)}(U, V)$ *is the element* U, V *in the matrix* $\mathbf{R}^{(k)}$. *The Floyd-Warshall algorithm returns FALSE because some of the diagonal elements in the matrices* $\mathbf{R}^{(k)}$, $k = 3$, 4 *and* 5, *are negative. Therefore, the graph contains a negative cycle.* ■

A.4 COMPUTATIONAL COMPLEXITIES

The Bellman-Ford algorithm detects negative cycles and solves the *single-source* shortest path problem if no negative cycles exist in $\mathcal{O}(n^3)$ time, while the Floyd-Warshall algorithm detects negative cycles and solves the *all-pairs* shortest path problem if no negative cycles exist in $\mathcal{O}(n^3)$ time. The Bellman-Ford algorithm requires $\mathcal{O}(n^4)$ time to solve the all-pairs shortest path problem since it needs to be run once for each of the n nodes. The Floyd-Warshall algorithm is preferred for applications that require all-pairs shortest path solution, and the Bellman-Ford algorithm and Floyd-Warshall algorithm can be used interchangeably for applications that require a single-source shortest path solution.

Modifications to the Bellman-Ford algorithm and early exit conditions for both algorithms can be utilized to improve computational efficiency. The

Table A.3 Values of $r^{(k)}(U, V)$ for Example A.3.1

$$\mathbf{R}^{(1)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & \infty & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(2)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(3)} = \begin{bmatrix} \infty & -3 & -2 & -1 \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix}$$

$$\mathbf{R}^{(4)} = \begin{bmatrix} \infty & -3 & -2 & -1 \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix} \quad \mathbf{R}^{(5)} = \begin{bmatrix} 0 & -3 & -2 & -1 \\ 3 & 0 & 1 & 2 \\ 3 & 0 & 1 & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix}$$

Table A.4 Values of $r^{(k)}(U, V)$ for Example A.3.2

$$\mathbf{R}^{(1)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & \infty & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(2)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(3)} = \begin{bmatrix} \infty & -3 & -2 & -2 \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & -1 \end{bmatrix}$$

$$\mathbf{R}^{(4)} = \begin{bmatrix} \infty & -3 & -2 & -2 \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & -1 \end{bmatrix} \quad \mathbf{R}^{(5)} = \begin{bmatrix} -1 & -4 & -3 & -3 \\ 2 & -1 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & -3 & -2 & -2 \end{bmatrix}$$

interested reader can find these details in [1].

REFERENCES

1. E. L. Lawler, *Combinatorial Optimization: Networks and Matriods.* Rinehart and Winston, 1976.

2. T. H. Cormen, C. E. Leiserson and R. C. Rivest, *Introduction to Algorithms*, MIT Press, 1990.