S-38.220
Postgraduate Course on Signal Processing in Communications,
FALL - 99

Gibson Mwakipesile

JMT 1B 306
ESPOO 02150

gibson@cc.hut.fi

Date: 13.10.1999

ABSTRACT

This paper presents formulations of the retiming problem, and, based on these formulations, some techniques are introduced to determine the solutions to these problems. We also show that the problem of determining an equivalent circuit with minimum state (total number of registers) is the linear-programming dual of a minimum-cost flow problem, and hence can also be solved efficiently. The techniques are general in that many other constraints can be handled within the graph-theoretic framework

1. INTRODUCTION

Retiming is a technique used to move delay elements around in a circuit without changing its functionality i.e. the input/output characteristics of the circuit. One effect of changing the location of the delays is that combinatorial rippling can be reduced, allowing the circuit to be clocked at higher rate. Reducing combinatorial rippling also decreases the dynamic power dissipation in the circuit and allows the circuit to be operated at with the lower supply voltage, both of which lead to low-power implementations. Another effect of changing the locations of delays is that the number of number of delay elements required can be reduced, resulting in the area-efficient implementation. Retiming and scheduling are important tools used to map behavioral descriptions of algorithms to physical realizations. These tools are used during the design of software for programmable digital signal processors (DSP's). Retiming and time scheduling operate directly on a behavioral description of the algorithm, such as a data-flow graph (DFG). Scheduling consists of assigning execution times to the operations in a DFG such that the precedence constraints of the DFG are not violated. Unlike pipelining, retiming does not increase circuit latency.
Due to the recent demand for low-power digital circuits in portable devices, some recent work has focused on retiming for power minimization. An approach based on circuit theory can be used to generate all retiming solutions for a DFG.

2. PRELIMINARIES

Here we introduce the notations and terminology needed in the paper.
Critical path is defined as to be the path with the longest computational time among all paths that contain zero delays. Folding: Is the process of executing several algorithm operations on a single hardware module. Scheduling: Is the process of determining at which time units a given algorithm operation is to be executed in hardware. The computation time of the critical path, is the path with the longest computation time among all paths with no delays.

We can view circuit abstractly as a network of functional elements and globally clocked registers. Functional elements provide the computational power of the circuit. Our model is not concerned with the level of complexity of the functional elements. Each element has associated propagation delay.
We model a circuit as a *finite, rooted, vertex-weighted, edge-weighted, directed multigraph* $G = \langle V, E, v_h, d, w \rangle$ (thus, we shall simply say "graph" or, morefrequently, "circuit"). The vertices V of the graph model the functional elements of the circuit. Each vertex $v \in V$ is weighted with numerical its numerical *propagation delay* $d(v)$. A root vertex v_h , called the host, is included to represent the interface with the external world, and it is given zero propagation delay. The *directed edge* E of the graph model interconnection between functional element and is weighted with a *register count* $w(e)$. The register count is the number of registers along the connection. Between two vertices, there may be multiple edges with different register counts.

To avoid confusion between *vertex-weight functions* i.e. d and *edge-weight functions* i.e. w, we shall use the term weight for *edge-weight functions* only. We shall refer to the particular *edge-weights* $w(e)$ of a circuit as *register counts*. If e is an edge in a graph that goes from vertex u to vertex v, we shall denote as $u \xrightarrow{e} v$. In the event that the identity of either the head or the tail of an edge is unimportant, we shall use the symbol ?, as in $u \xrightarrow{e} ?$.

For a graph G, we shall view a path p in G as a sequence of vertices and edges. A simple path contains no vertex twice, and therefore the number of vertices exceeds the number of edges by exactly one. We extend the register count function w in a natural way from single edges to arbitrary paths. For any *path* $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} ... \xrightarrow{e_{k-1}} v_k$, we define the *path weight* as the sum of the weights of the edges of the path:

Equation 1

$$w(p) = \sum_{i=0}^{k-1} w(e_i)$$

Similarly, we extend the *propagation delay* d *function* to simple paths. For any *simple path* $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} ... \xrightarrow{e_{k-1}} v_k$, we define the *path delay* as the sum of the delays of the vertices of the path:

Equation 2

$$d(p) = \sum_{i=0}^{k} d(v_i)$$

In order that a graph $G = \langle V, E, d, w \rangle$ have well-defined physical meaning as a circuit, we place nonnegativity restrictions on the *propagation delays* $d(v)$ and the *register counts* $w(e)$:

D1. *The propagation delay* $d(v)$ *is nonnegative for each vertex* $v \in V$.

W1. *The register count* $w(e)$ *is a nonnegative integer for each edge* $e \in E$.

We also impose the restriction that there be no directed cycles of zero weight:

W2. *In any directed cycle of* G*, there is some edge with (strictly) positive register count*

We define *synchronous circuit* as a circuit that satisfies Conditions D1, W1, and W2. The reason for including Condition W2 is that whenever an edge e between two vertices u and v has zero weight, a signal entering vertex u can ripple unhindered through vertex u and subsequently through vertex v. If the rippling can feedback upon itself, problems of asynchronous latching, oscillation, and race condition can arise. By prohibiting zero-weight cycles, Condition W2 prevents these problems from occurring, provided that system clock runs slowly enough to allow the outputs of all the functional elements to settle between each two consecutive ticks.

For any synchronous circuit G, we define the (minimum feasible) clock period $\Phi(G)$ as the maximum amount of propagation delay through which any signal must ripple between clock ticks. Condition W2 guarantees that the *clock period* is well defined by the equation

Equation 3

$$\Phi(G) = \max\{d(p) \mid w(p) = 0\}$$

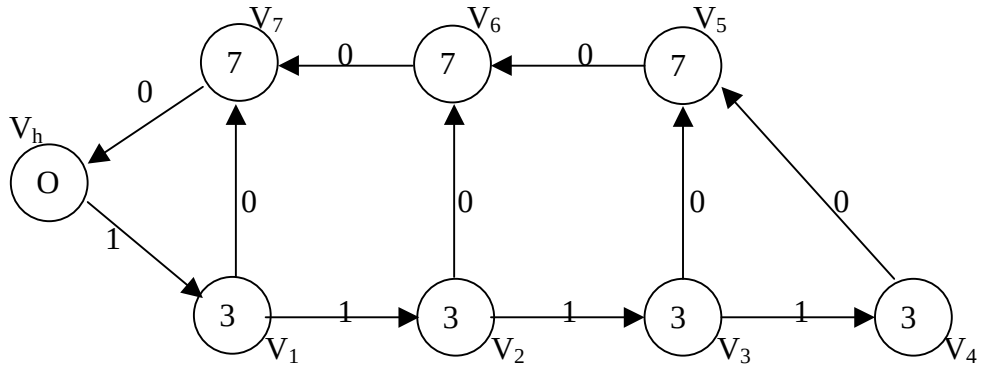


Figure1-1

for the circuit graph in Fig. 1 the clock period is 24, which corresponds to the sum of the *propagation delays along the path* $v_4 \to v_5 \to v_6 \to v_7$.

The *computation time of the path* is:

Equation 4

$$t(p) = \sum_{i=0}^{k-1} t(v_i)$$

A cycle is a closed path $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{k-2}} v_{k-1} \xrightarrow{e_{k-1}} v_0$.
The weight of the cycle c is:

Equation 5

$$w(c) = \sum_{i=0}^{k-1} w(e_i)$$

and the delay of the cycle is:

Equation 6

$$t(c) = \sum_{i=0}^{k-1} t(v_i)$$

2.1 Algorithm CP (Compute the clock period of a circuit)

This algorithm computes the *clock period* $\Phi(G)$ for asynchronous circuit $G = \langle V, E, v_h, d, w \rangle$.

1. Let G_0 be the subgraph of G that contains precisely those edges e with register count $w(e) = 0$.
2. by Condition W2, G_0 is acyclic. Perform a topological sort on G_0, totally ordering its vertices so that if there is an edge from vertex v in G_0, then u precedes v in the total order.
3. Go through the vertices in the order defined by the topological sort. On visiting each vertex v, compute the quantity $\Delta(v)$ as follows:
   a. If there is no incoming edge to v, set $\Delta(v) \leftarrow d(v)$.
   b. Otherwise, set $\Delta(x) \leftarrow d(v) + \max\{\Delta(u) \mid u \xrightarrow{e} v \ \ and \ \ w(e) = 0\}$
4. The clock period $\Phi(G)$ is $\max_{v \in V} \Delta(v)$.

The algorithm works because for each vertex v, the quantity $\Delta(v)$ equals the maximum sum $d(p)$ of vertex delays along any zero-weight directed path p in G such that $? \xrightarrow{p} v$. The running time is $O(|E|)$.

3. RETIMING

3.1 Basics

The basic retiming equation for the edge

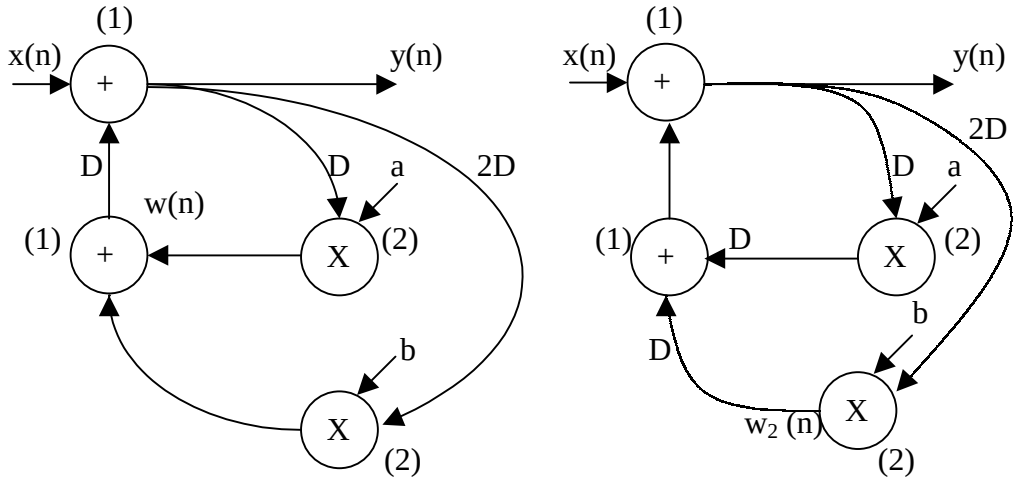


Figure3-1:Two versions of an IIR filter. The computation times are shown in parentheses.

Consider the IIR filter in Fig. 3-1(a). This filter is described by

Equation 7

$$w(n) = ay(n-1) + by(n-2)$$
$$y(n) = w(n-1) + x(n)$$
$$y(n) = ay(n-2) + by(n-3) + x(n)$$

The filter in Fig. 3-1(b) is described by

Equation 8

$$w_1(n) = ay(n-1)$$
$$w_2(n) = by(n-2)$$
$$y(n) = w_1(n-1) + w_2(n-1) + x(n)$$
$$y(n) = ay(n-2) + by(n-3) + x(n)$$

Although the filters in Fig. 3-1(a) and Fig. 3-1(b) have delays at different locations, these filters have the same input/output characteristics. These 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, increasing the clock rate of a circuit by reducing the computation time of the critical path, and logical synthesis.

Retiming can be used to increase the clock rate of a circuit by reducing the computation time of the critical path. 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. 3-1(a) passes through I multiplier and 1 adder and has a computational time of 3 u.t., so this filter cannot be clocked with a clock period less than 3 u.t. The retimed filter in Fig. 3-1(b) has a critical path that passes through 2 adders and has a computation time of 2 u.t. By retiming the filter in Fig. 3-1(a) to obtain the filter in Fig. 3-1(b), the clock period has been reduced from 3 u.t. to 2 u.t., or by 33%.

Retiming can be used to reduce the number of registers in a circuit. the filter in Fig. 3-1(a) uses 4 registers while the filter in Fig. 3-1(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.

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 circuit. 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 .

3.1.1 Properties

A retimed solution is characterized by a value $r(v)$ (this values are obtained by solving the linear programming equation which is formed by linear inequalities in eqn. 15) for each node v in the graph. Let $w(e)$ denote the weight if 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 edges $u \xrightarrow{e} v$ in the retimed graph is computed from the weight of the edge in the original graph using

Equation 9

$$w_r(e) = w(e) + r(v) - r(u)$$

For example, the edge $3 \xrightarrow{e} 2$ in the retimed DFG contains

Equation 10

$$w_r(3 \xrightarrow{e} 2) = w(3 \xrightarrow{e} 2) + r(2) - r(3)$$
$$w_r(3 \xrightarrow{e} 2) = 0 + 1 - 0 = 1$$

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

Equation 11

$$w_r(2 \xrightarrow{e} 1) = w(2 \xrightarrow{e} 1) + r(1) - r(2)$$
$$w_r(2 \xrightarrow{e} 1) = 1 + 0 - 1 = 0$$

delays.

The solution $r(1) = 0, r(2) = -1, r(3) = 0, and\ r(4) = 0$ is infeasible because, for example, the edge $3 \xrightarrow{e} 2$ in the retimed system contains

Equation 12

$$w_r(3 \xrightarrow{e} 2) = w(3 \xrightarrow{e} 2) + r(2) - r(3)$$
$$w_r(3 \xrightarrow{e} 2) = 0 + (-1) - 0 = -1$$

delays.

Several properties of retiming can be derived from the retiming equation (6).

Property 3.1.1 *The weight of the retimed path* $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \ldots \xrightarrow{e_{k-1}} v_k$ is *given by* $w_r(p) = w(p) + r(v_r) - r(v_0)$.

The *retimed path weight* is:

Equation 13

$$\begin{aligned} w_r(p) &= \sum_{i=0}^{k-1} w_r(e_i) \\ &= \sum_{i=0}^{k-1} (w(e_i) + r(v_{i+1}) - r(v_i)) \\ &= \sum_{i=0}^{k-1} w(e_i) + (\sum_{i=0}^{k-1} r(v_{i+1}) - \sum_{i=0}^{k-1} r(v_i)) \\ &= w(p) + r(v_k) - r(v_0). \end{aligned}$$

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

This is a special case of property 3.1.1 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 3.1.3 *Retiming does not alter the iteration bound in a DFG.*

Property 3.1.4 *Adding the constant value j to the retiming value of each node does not change the mapping the mapping from G to G_r.*

After replacing $r(v)$ with $r(v) + j$ for each node, the weight of the retimed edges $u \xrightarrow{e} v$ in G_r is:

Equation 14

$$w_r(e) = w(e) + (r(v) + j) - (r(u) + j) = w(e) + r(v) - r(u)$$

which is the same for any value j (including j = 0). By adding, for example, the constant -10 to the retiming values $r(1) = 0, r(2) = 1, r(3) = 0, \text{ and } r(4) = 0$, the retiming values $r(1) = -10, r(2) = -9, r(3) = -10, \text{ and } r(4) = -10$ can be used to obtain the retimed DFG in Fig. 4.2(b) from the DFG in Fig. 4.2(a).

4. DETERMINING AN OPTIONAL RETIMING

Here we presents a polynomial-time algorithm for relocating registers within a circuit so as to maximize the performance of the circuit. Specifically we'll solve the following problem: Given a circuit graph $G = \langle v, E, v_h, d, w \rangle$, find a legal retiming r of G such that the clock period $\Phi(G_r)$ of the retimed circuit G_r is as small as possible. The solution of this problem depends on some basic results from combinatorial optimization and graph theory. In particular, we rely on the fact that the following linear programming problem can be solved efficiently.

Problem LP. *Let S be a set of m linear inequalities of the form*

Equation 15

$$r_i - r_j \leq k_{ij}$$

on the unknowns $r_1, r_2, ..., r_n$, where the k_{ij} are given real constants. Determine feasible values for the unknowns r_i, or determine that no such values exist.

This is done using the following procedure:

1. Draw a constraint graph.
    a. Draw the node i for each of the N variables $r_i, i = 1,2,...,N.$
    b. Draw the node $N+1.$
    c. For each inequality $r_i - r_j \leq k$, draw the edge $j \to i$ from node j to node i with length k.
    d. For each node i, $i = 1,2,...,n,$ draw the edge $N+1 \to i$ from the node $N+1$ to the node i with length 0.
2. Solve using a shortest path algorithm.
    a. The system of inequality has a solution iff 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.

When solving systems of inequalities, there may be multiple inequalities with identical left-hand sides, which can lead to parallel edges in the Step 1(c). For example the two inequalities $r(1) - r(2) \leq 9$ *and* $r(1) - r(2) \leq 7$ would lead to two 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 Step1(c).

To demonstrate the just described algorithm, the values of $W(u,v)$ and $D(u,v)$ are computed for the DFG in Fig. 4-1(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 4-1(c). The solution to the all-pairs shortest path problem for G' can be found using the Floyd-Warshall algorithm. The 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 the Table 1-1.

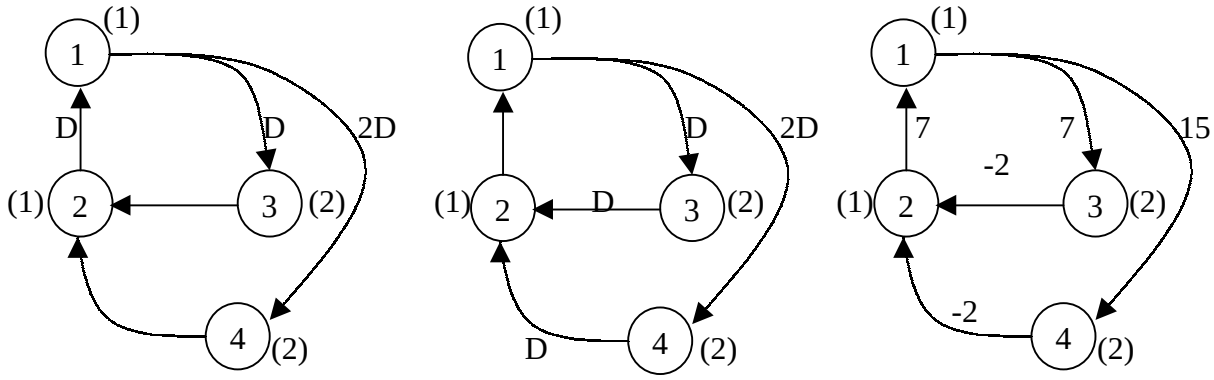


Figure 4-1: (a) A DFG. (b) The retimed DFG obtained using $r(1) = 0, r(2) = 1, r(3) = 0,$ *and* $r(4) = 0$. (c)The graph G' used to compute $W(u,v)$ and $D(u,v)$ for the DFG in (a).

Table 1-1:

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	3
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

The tables above shows the values of the functions $W(u,v)$ and $D(u,v)$ for the GDF in Fig. 4-The quantity $W(u,v)$ is the *number of registers on a minimum-weight path* from u to v, and $D(u,v)$ is the *maximum propagation delay* along any such *critical path*. The distinct entries in the table for D include all possible clock period less than c iff $W_r(u,v) > 0$ wherever $D(u,v) > c$.

Any such system of linear inequalities can be satisfied or determined to be inconsistent in $O(mn)$ time by the Bellman-Ford algorithm.

4.1 Retiming Techniques

4.1.1 Cutset Retiming and Pipelining

A cutset is a set of edges that can be removed from the graph to creat two disconnected subgraphs. Cutset retiming only affects the weights of the edges in the cutset. If the two 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. 4-1(a). The 3 edges in the cutset are $2 \to 1, 3 \to 2,$ *and* $1 \to 4.$ The two subgraphs G_1 *and* G_2 found by removing the 3 edges in the cutset are shown in Fig. 4-1(b). For $k = 1,$ the result of cutset retiming is shown in Fig.

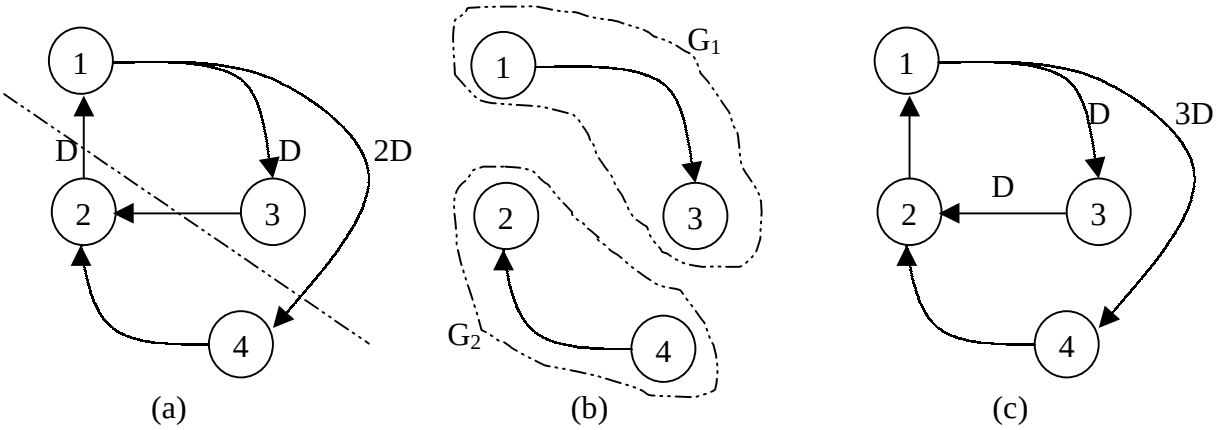


Figure 4-1: (a) unretimed DFG with a cutset shown as a dashed line. (b) The 2 graphs $G_1 and G_2$ formed by removing the edges in the cutset. (c) The retimed graph found using cutset retiming with $k = 1.$

4.4(c). The edges from G_1 *to* G_2 *are* $3 \to 2$ *and* $3 \to 2$ *and* $1 \to 4,$ and one delay is added to each of these edges. The edge from G_2 *to* G_1 *is* $2 \to 1,$ 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 example, in Fig. 4.4, using the values $j = 0$ *and* $k = 1$ results in $r(1) = 0, r(2) = 2, r(3) = 0,$ *and* $r(4) = 1,$ and this maps the DFG in Fig. 4-1(a) to the DFG in Fig. 4-1(c). Any value of j results in the 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* $G_2, w_r(e_{1,2}) \geq 0$ 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 two inequalities and considering all of the edges in the cutset result in

Equation 16

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

as the condition on k for cutset retiming to give a feasible solution. In Fig. 4.4, $\min_{G_1 \xrightarrow{e} G_2} \{w(e)\} = \min\{0,2\} = 0$ *and* $\min_{G_2 \xrightarrow{e} G_1} \{w(e)\} = 1$, *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 graph, and this graph is shown in Fig. 4-1(c).

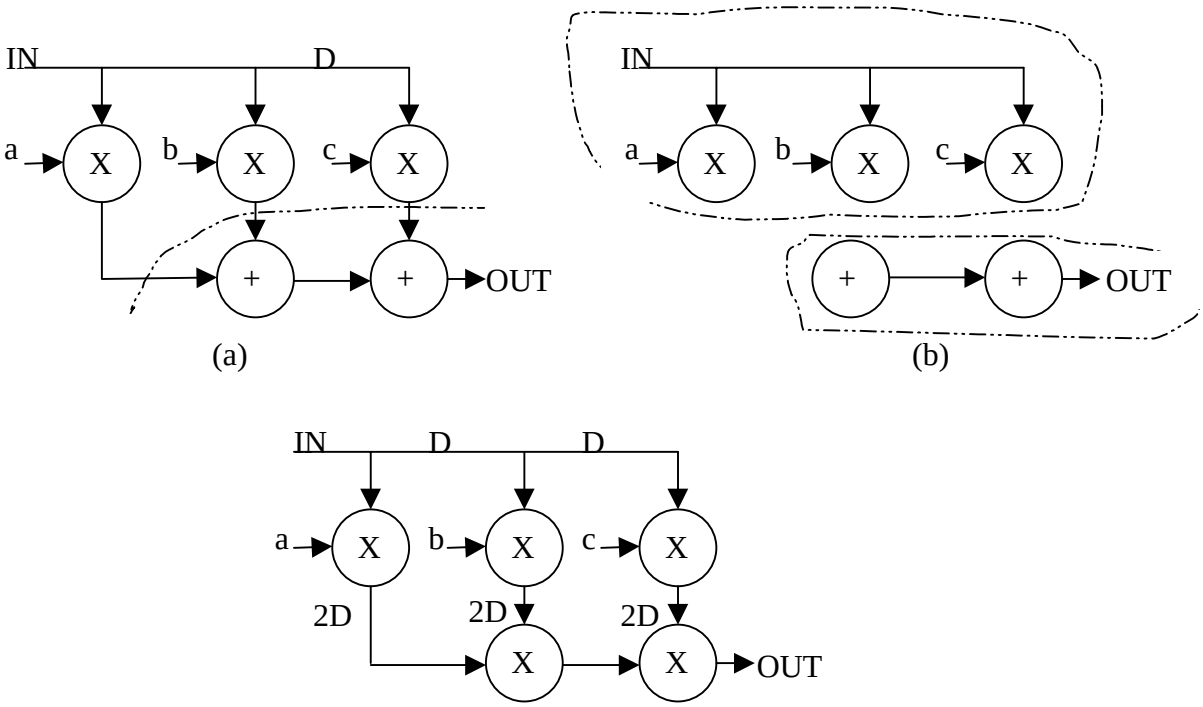


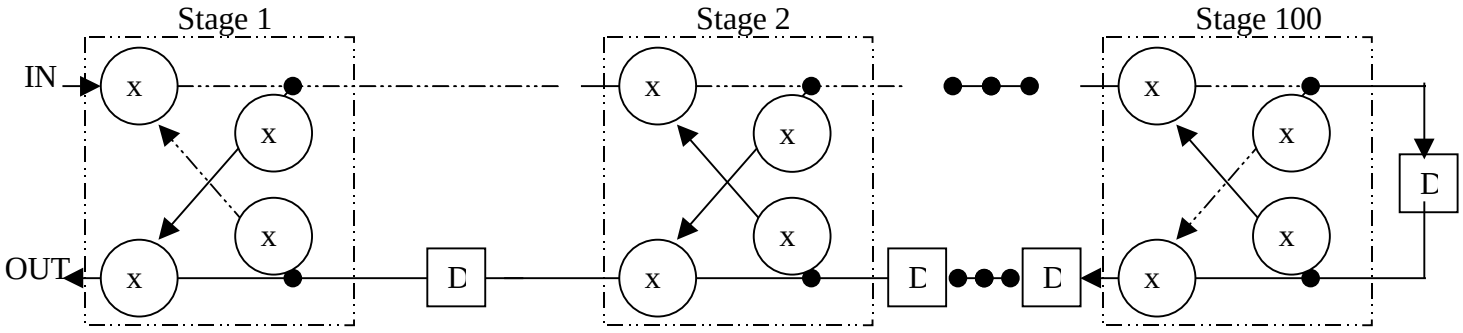
Figure4-2: (a) The unretimed DFG with a cutset shown as a dashed line. (b) The 2 graphs G_1 *and* G_2 formed by removing the edges in the cutset. (c) The graph obtained by cutset retiming with $k = 2$.

Pipelining is a special case of cutset retiming where there are no edges in thw 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. 4-2(a) divides the graph into the two subgraphs shown in Fig. 4-2(b). All three 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. 4-2(c). Tis demonstrates that the pipelining can be viewed as a special case of retiming.

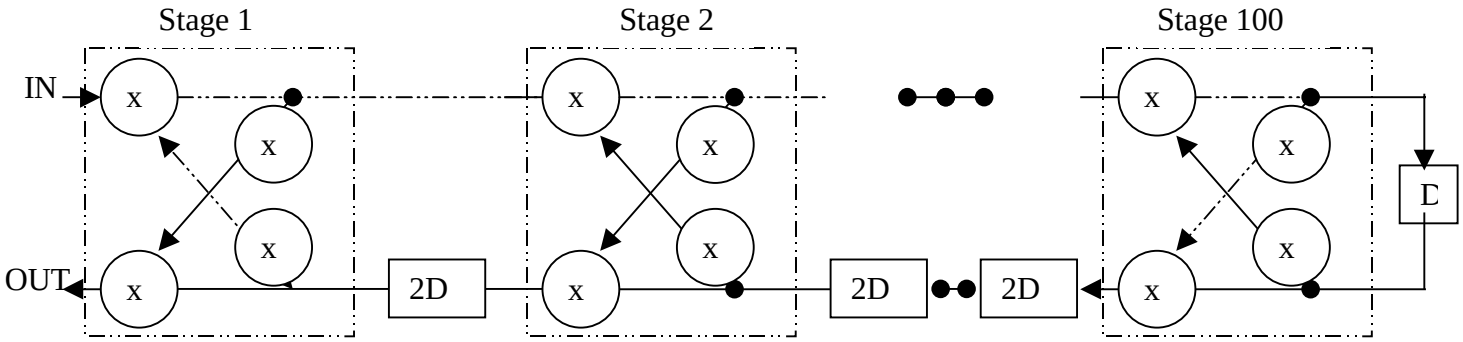
Cutset retiming is often used in combination with *Slow-down.* The procedure is to first replace each delay in the DFG with N delays to create an $N - Slow$ version of the DFG and the to perform cutset retiming on the $N - Slow$ DFG. Note that in an $N - Slow$ system, $N - 1$ null operations (or 0 samples) 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. 4-3(a), which has a critical path of 101 adders and 2 multipliers. Assuming that addition and multiplication takes 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. 4-3(b), and cutset retiming can be used to obtain the circuit in Fig. 4-3(c).

The critical path of the retimed circuit has 2 adders and 2 multipliers and has computation timw (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 102 u.t. to 12 u.t.

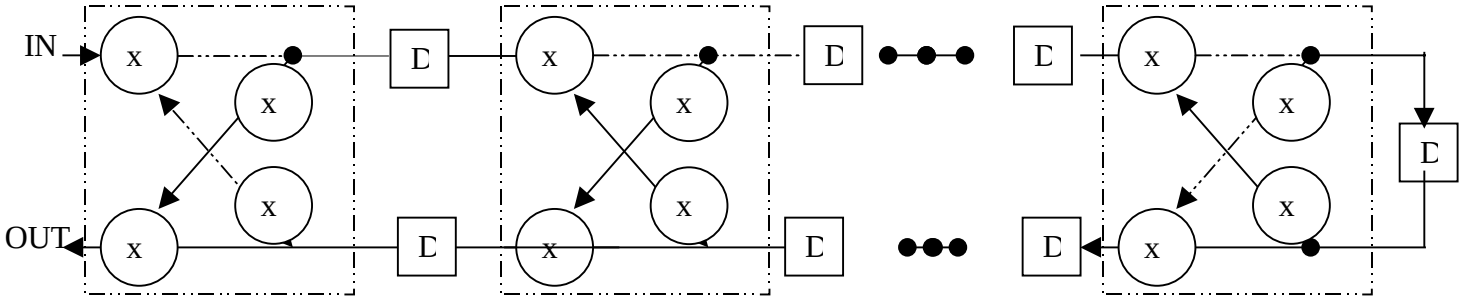


(a)



(b)

Stage 1 Stage 2 Stage 100



(c)

Figure 4-3: In each of the 3 filters, the critical path is shown woth the dotted lines. (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 vertion of the 2-slow circuit.

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

4.1.2 Retiming for Clock Period Minimization

Here we present 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:

Equation 17

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

The algorithm is concerned with finding a retiming solution r_0 such that $\Phi(G_{r_0}) \le \Phi(G_r)$ for any other retiming solution r.

The two quantities $W(u,v)$ *and* $D(u,v)$ are use 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:

Equation 18

$$W(u,v) = \min\{\, w(p) : u \xrightarrow{p} v \}$$
$$D(u,v) = \max\{\, t(p) : u \xrightarrow{p} v \text{ and } w(p) = W(u,v) \}$$

The algorithm for optimizing the clock period of a circuit is based on an alternate characterization of clock period in terms of the two quantities above.

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 node* in G and n is the *number of node* in G.

2. From 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 $u \xrightarrow{e} v$.

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 $D(u,v) = M(u,v) - S'_{uv} + t(v)$. If $u = v$, then $W(u,v) = 0$ *and* $W(u,v) = t(u)$.

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

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. (fanout constraint) $r(u) - r(v) \leq w(e)$ for every edges $u \xrightarrow{e} v$ of G, and

2. (critical path constraint) $r(u) - r(v) \leq W(u,v) - 1$ for all vertices u, v 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 $\Phi(G) \leq c$. If $W(u,v) + r(v) - r(u) \geq 1$ must hold for the critical path to have computation time less than o equal to c. This leads to the critical path constraint.

4.1.3 Retiming Register Minimization

Here we present the 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. 4-4(a) uses 1+3+7=11 registers with the clever implementation in Fig. 4-4(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:

Equation 19

$$R_v = \max_{v \xrightarrow{e} ?} \{w_r(e)\}$$

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

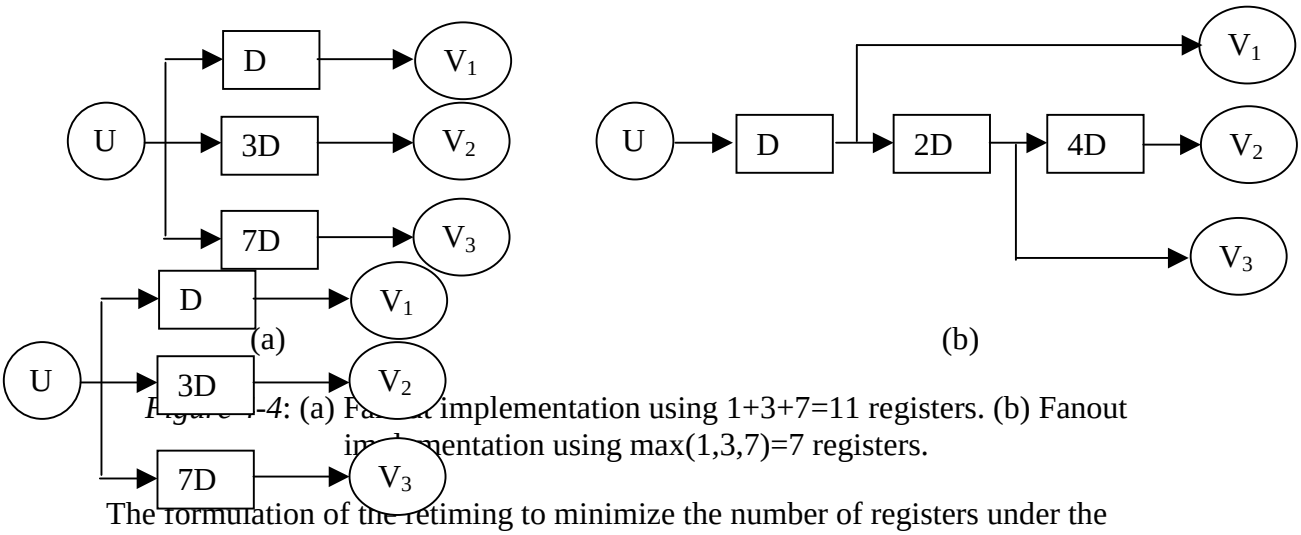


Figure 7-4: (a) Fanout implementation using 1+3+7=11 registers. (b) Fanout implementation using max(1,3,7)=7 registers.

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

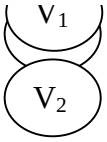
Equation 20

$$COST = \sum R_v$$

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 makes sure that $R_v = \max_{v \xrightarrow{e} ?} \{w_r(e)\}$.



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 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. This gadget is shown in Fig. 4.12. Fig. 4-5(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 $e_i,\ 1 \le i \le k,$ 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 $\hat{e}_i,\ 1 \le i \le k,$ 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 $w_{\max} = \max_{1 \le i \le k} w(e_i).$

In addition to its weight, each edge in this model also has a breadth β associated with it. This comes from the fact that the cost of adding a register is not the same on all edges. For example, it may be cheaper to add a register along a one-bit wide control path than along a 32-bit wide data path; thus we model this phenomenon by assigning to each edge e a breadth β. The breadth of an edge is a number used so that the gadget in Fig. 4-5(b) properly models the memory required by the edges $e_i,\ 1 \le i \le k,$ in the retimed DFG. The breadth of each edge the e_i and $\hat{e}_i$ for $1 \le i \le k$ is $\beta = 1/k,$ as shown in Fig. 4-5(b).

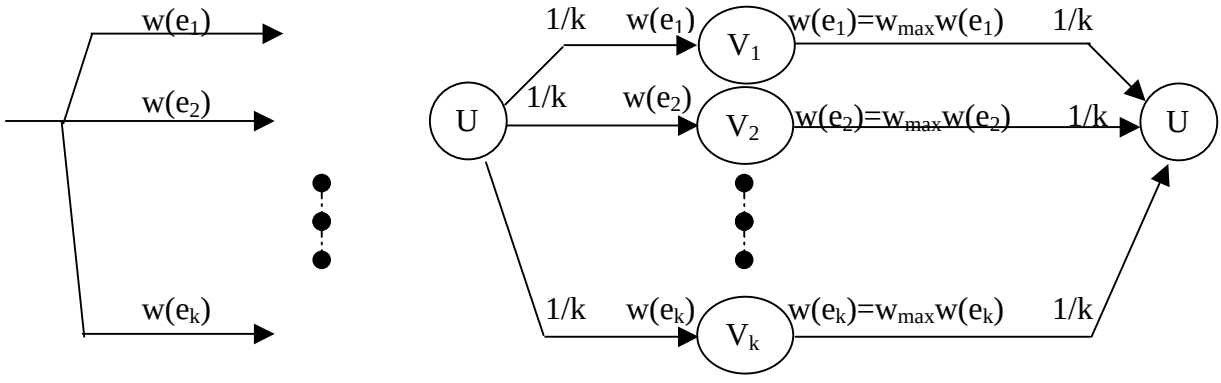


Figure 4-5: (a) A node u with k output edges. (b) A gadget used to model the node u.

Using the fanout model in Fig. 4.12, the retiming formulation is: Minimize

Equation 21

$$COST = \sum_e \beta(e) w_r(e)$$

subject to:-

1. (fanout constraint) $r(u) - r(v) \leq w(e)$ for every edges $u \xrightarrow{e} v$.

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

Equation 22

$$\begin{aligned} COST &= \sum_r \beta(e) w_r(e) \\ &= \sum_e \beta(e)(w(e) + r(v) - r(u)) \\ &= \sum_e \beta(e) w(e) + \sum_e \beta(e)(r(v) - r(u)) \\ &= K + \sum_e \beta(e)(r(v) - r(u)) \\ &= K + \sum_v r(v)\Big(\sum_{v \xrightarrow{e} v} \beta(e) - \sum_{v \xrightarrow{e} ?} \beta(e) \Big) \end{aligned}$$

since K is aconstant, the formulation can be written as Minimize

Equation 23

$$COST = \sum_v r(v)\Big(\sum_{? \xrightarrow{e} v} \beta(e) - \sum_{v \xrightarrow{e} ?} \beta(e)\Big)$$

subject to the constraints immediately above.

5. CONCLUSIONS

Systems can be retimed to reduce critical path or clock period, number of storage or delay elements. Shortest path algorithm can be used to obtain a retiming solution if one exists. General framework for the understanding of circuit timing has been presented through the use of graphical model with which we have been able to deal with an otherwise complex problem of circuit retiming.

REFERENCES

[1] K.K. Parhi. VLSI Digital Signal Processing Systems Design and Implementation. chapter 4. J. Wiley & Sons, 1999.

[2] R. Bryant. Third Caltech Conference on VLSI. Optimizing Synchronous Circutry by Retiming. C.E. Leiserson, F.M. Rose, J.B. Saxe. Computer Science Press,1988.

[3] T.C. Denk and K.K. Parhi, “Two-dimentional retiming”, IEEE Trans. On VLSI Systems, vol. 7, 1999.

[4] T.C. Denk and K.K. Parhi, “Exhaustive scheduling and retiming of digital signal processing systems”, IEEE Trans. On Circuits and Systems-II:Analog and Digital Signal Processing, vol. 45, no.7, pp. 821-838, July 1998.