Retiming Arithmetic Datapaths using Timed Taylor Expansion Diagrams

Daniel Gomez-Prado[1] Dusung Kim[1] Maciej Ciesielski[1] Emmanuel Boutillon[2]
[1]University of Massachusetts Amherst, USA. *{dgomezpr,ciesiel,dukim}@ecs.umass.edu*
[2]Lab-STICC, Université de Bretagne Sud, France. *emmanuel.boutillon@univ-ubs.fr*

Abstract - This paper describes an extension to the Taylor Expansion Diagrams (TED), called Timed TEDs, which makes it possible to represent sequential arithmetic datapaths. Timed TEDs enable register and clock period minimization while performing factorizations and common sub expression eliminations in the data flow graph (DFG). Specifically, timed TEDs allow a wider range of retiming options as the computations in the DFG can be modified while performing retiming. In this paper we discuss the formalism of timed TEDs and the restrictions it imposes on the TED variable ordering.

1 Introduction

It is known that certain data flow operations in a design can be optimized by performing retiming [1]. Retiming is an important step of scheduling in high level synthesis; it can decrease the clock period of the design [2], its switching activity [3], power consumptions and its area [4]. A classic example of retiming is an n-tap FIR filter, which in its original version as shown in Figure 1(a) has a combinational delay of one constant multiplier and n adders, with $n + 1$ being the number of filter taps. After retiming, the critical path delay of the filter is reduced to 1 multiplier and 1 adder, as shown in Figure 1(b). This reduction in the critical path obtained through retiming allows the circuit to be clocked at higher speeds.

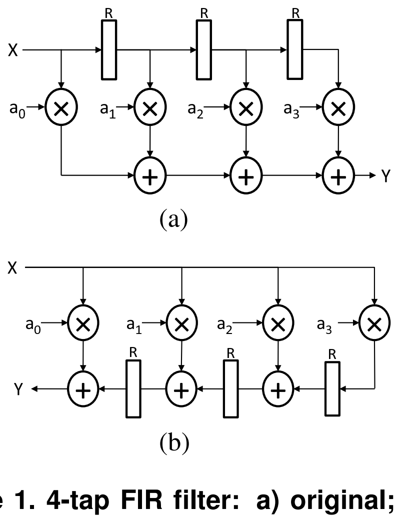


Figure 1. 4-tap FIR filter: a) original; b) retimed.

Although retiming has been extensively studied, it has been applied to fixed Data Flow Graphs (DFGs). This work focuses on retiming of Taylor Expansion Diagrams (TEDs). TED captures the functional description of a DFG and optimizes it, in order to minimize the hardware implementation [5]. TED enables structural changes such as factorization and common sub expression elimination to occur in the DFG. In this work, the TED is enhanced by inserting registers into the basic data structure to enable retiming. We refer to such a representation as *Timed TED*. We will show that such a retiming and the resulting delay optimization can be efficiently captured using the TED model.

2 Previous work

Taylor Decomposition System (TDS) is a framework to optimize data flow graphs (DFGs) using a canonical data structure, called Taylor Expansion Diagram (TED). TDS offers a systematic way to transform the initial specification, written in C or C++, into a DFG that addresses the constraints of hardware implementation and explores a larger space of architectural solutions. Its applications to high-level synthesis can be found in [6, 7, 5]. TED is a compact, graph-based data structure that provides an efficient way to represent word-level computation in a canonical, factored form. It is particularly suitable for computation-intensive applications, such as signal and image processing, with computations modeled as polynomial expressions. The basic formalism of TED is described in detail in [6].

Briefly, a multi-variate polynomial expression, $f(x, y, ...)$, is represented using Taylor series expansion w.r.t. variable x around the origin $x = 0$ as follows:

$$f(x, y, \cdots) = \sum_{k=0}^{m} \frac{x^k}{k!} f^k(0, y, \cdots) \qquad (1)$$

where $f^k(0, y, \cdots)$ are the successive derivatives of f w.r.t. x, evaluated at $x = 0$. The individual terms of the expression are then decomposed iteratively with respect to the remaining variables on which they depend ($y, \cdots$, etc.), one variable at a time.

The resulting decomposition is stored as a directed acyclic graph, called TED. Each node of the TED is labeled with the name of the variable at the current decomposition level and represents the expression rooted at this

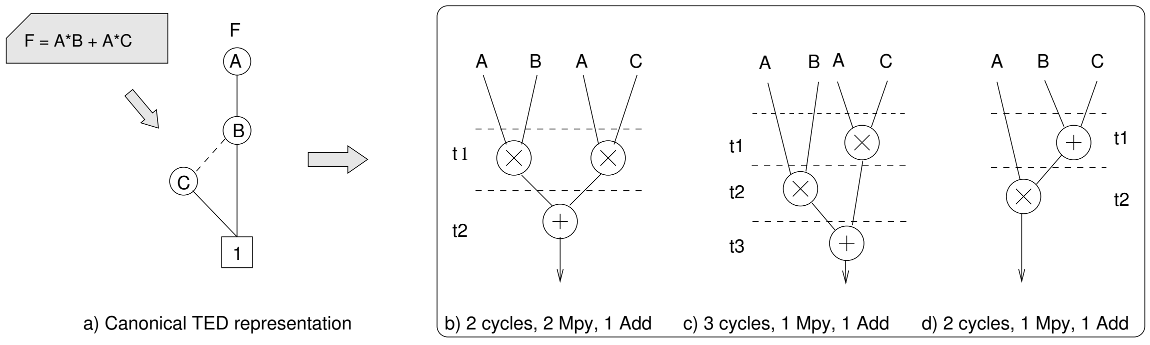

F = A*B + A*C
F
A
B
C
1
a) Canonical TED representation
A B A C
t1
t2
b) 2 cycles, 2 Mpy, 1 Add
A B A C
t1
t2
t3
c) 3 cycles, 1 Mpy, 1 Add
A B C
t1
t2
d) 2 cycles, 1 Mpy, 1 Add

Figure 2. Behavioral transformations: (a) Canonical TED representation. (b),(c),(d) Functionally-equivalent DFGs.

node. The top node of the TED represents the main function $f(x, y, \dots)$ and is associated with the first variable, x. Each term of the expansion at a given decomposition level is represented as a directed edge from the current decomposition node to its respective derivative term. Each edge is labeled with a pair $(^\wedge p, w)$, where $^\wedge p$ represents the power of the corresponding variable and w represents the edge weight (multiplicative constant) associated with this term. An edge with a pair $(^\wedge 0, w)$ corresponds to an additive edge, and it is drawn with a dotted line. An edge with a pair $(^\wedge 1, w)$ is a linear (multiplicative) edge and it is drawn with a solid line. Explicit labels on the edges are dropped for additive and linear edges; similarly, edges with weight 0 are not shown as they correspond to empty terms. The final reduced, normalized and ordered TED is canonical for a given variable order [6].

The construction of a TED is illustrated in Figure 2(a) for expression $F = AB + AC$, where variables A, B, C are word-level signals of arbitrary bit-width. The expression encoded in the graph is computed as a sum of two paths from TED root to terminal node 1: $A \cdot B$ and $A \cdot B^0 \cdot C = A \cdot C$, i.e., $AB + AC$. In fact, TED encodes such an expression in *factored form*, $F = A(B+C)$, since variable A is common to both paths. This is manifested in the graph by the presence of the subexpression $(B + C)$, rooted at node B, which can be factored out. It is this feature of the TED structure that makes it particularly applicable to factorization and common subexpression extraction of algebraic expressions.

To illustrate the concept of data flow transformations supported by TED consider the simple computation $F = AB + AC$, and the two possible schedules of a DFG derived directly from this expression, Figures 2(b) and (c). The two DFGs have the same structure and differ only in the scheduling of arithmetic operations. The DFG in Fig. 2(b) has minimum latency and requires two multipliers and one adder. The one in (c) needs only one multiplier and one adder, at a cost of the increased latency. In contrast, Figure 2(d) shows a solution that can be obtained by transforming the original specification $F = AB + AC$ into $F = A(B + C)$, which results in a *different DFG*. This DFG requires only one adder and one multiplier and can be scheduled in two control steps. Such an implementation cannot be obtained from the initial DFG by simple structural transformation and requires *functional* transformation (in this case factorization) of the original expression which preserves its original behavior. The solution with best hardware cost can then be chosen for the final hardware implementation.

3 Timed TEDs

We will now show how the TED structure can be enhanced to handle sequential designs by introducing registers. The resulting TED is called *Timed TED*. A timed TED is a TED with register information annotated on its nodes and edges using a register label nR, where n is the number of registers.

3.1 Registers annotation in the nodes

Recall that a node in a TED represents a primary input (symbolic word-level variable). The label nR associated with a node, denots that the given input has n registers, or that it is delayed by n clock cycles. For instance, function $F = (a \times 1)^{1R}$ shown in Figure 3(a) represents the multiplication of constant node (ONE) with a primary input a delayed one clock cycle. This is equivalent to function $F = a^{1R}$ as shown in Figure 3(b).

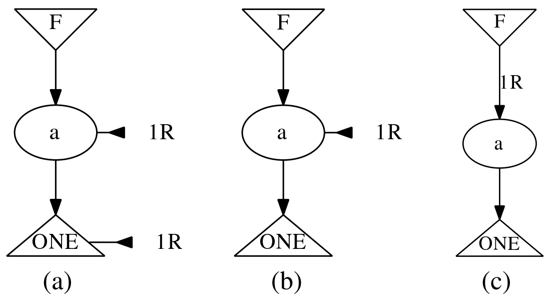


Figure 3. Timed TED Representation. (a) Annotated node. (b) After Normalization. (c) Annotated edge.

Variables that represent constant values, called *constant nodes*, can take an arbitrary number of registers and still represent the same functionality (delaying the constant does not change its value). Therefore we normalize the TED representation by removing all registers at constant nodes. That is, after any operation in the TED, the registers left at constant nodes can be safely removed.

3.2 Registers annotation in the edges

Labels in the TED edges are extended to carry register information; that is, each edge is labeled with a triple $(\hat{}p, w, n\text{R})$. The label nR associated with an edge means that the entire function rooted at the node below that edge is delayed by n clock cycles. That is, the labeled edge denotes n registers positioned at the output of the corresponding function. This is shown for function $F = a^{1R}$ in Figure 3(c), where the output of the multiplication of node a with the constant node 1 has been retimed once.

Similarly to dropping additive and multiplicative edge labels mentioned earlier, weight labels equal to 1 and registers labels equal to 0R are also dropped for simplicity. For instance, the additive edge with weight of 1 and 1 register shown in Figure 4(b), is labeled as 1R instead of $(\hat{}0, 1, 1\text{R})$; it is implicit from the dotted line that it is an additive edge with $\hat{}0$ and because there is no weight information it is implicitly set to one.

It should be observed that the TEDs shown in Figures 3(b) and 3(c) produce the same DFG: the primary input a connected to the output F through a register. This is true for all nodes connecting to node ONE through a multiplicative edge with weight 1, as the function represented by the node and the node itself are the same.

3.3 Forward and Backward register propagation

We will refer to a node with non-zero register label as a *retimed node*, and to an edge with non-zero register label as a *retimed edge*.

Theorem 3.1 *A register at the root of a TED node can be retimed backward by propagating the register to the node itself and to all its children edges.*

Proof Let's denote by F an arbitrary TED function rooted at node a with a register on top of it; and let's denote by G the TED function rooted at node a. Let's define by $()^{nR}$ the lineal operator *retime* . Then F can be composed in terms of G by:

$$F_{(a,\cdots)} = G^{1R}_{(a,\cdots)} \tag{2}$$

Consider an abitrary decomposition of G, as the one shown in Figure 4(a); then G can be described in terms of its Taylor expansion as:

$$G = G_{(a=0)} + a\frac{\partial G}{\partial a}_{|a=0} + a^2\frac{\partial^2 G}{\partial a^2}_{|a=0} + a^3\frac{\partial^3 G}{\partial a^3}_{|a=0} \tag{3}$$

replacing equation (3) in (2) and applying the properties of a linear operator

$$F = G^{1R}_{(a=0)} + a^{1R}(\frac{\partial G}{\partial a})^{1R}_{|a=0} + (a^{1R})^2(\frac{\partial^2 G}{\partial a^2})^{1R}_{|a=0} + (a^{1R})^3(\frac{\partial^3 G}{\partial a^3})^{1R}_{|a=0} \tag{4}$$

equation (4) shows that register in equation (2) has been propagated to node a and to its children edges, denoted as partial derivates of G. Therefore, equation (4) corresponds to a backward retiming as shown in Figure 4(b). ∎

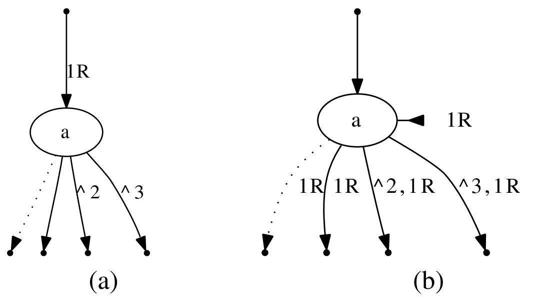


Figure 4. Function G rooted at node a. (a) Register at the root of node a. (b) Register moved inside function G.

It can be proved that the retime operator $()^{nR}$ is the convolution with the time delayed delta Dirac δ, which is by definition a lineal operator. Therefore the retime operator is guaranteed to be linear. It can also be shown that the retime operator commutes with the TED function being implemented. That is, applying the retime operator to the primary inputs of a TED is equivalent to applying the retime operator to the output of a TED. Therefore, the guarantee that a register in a TED can be propagated backward or forward is a direct consequence of the time invariant function being implemented in the TED.

Corollary 3.2 *A node with a register in all its children edges and in itself can be retimed forward by moving all those registers to the output of the node as a single register.*

Proof Starting from Figure 4(b) we define the function F at root node a as:

$$F = G_0^{1R} + a^{1R}G_1^{1R} + (a^{1R})^2G_2^{1R} + (a^{1R})^3G_3^{1R} \tag{5}$$

Since $()^{nR}$ is a lineal operator we can rewrite the above equation as:

$$F = (G_0 + aG_1 + a^2G_2 + a^3G_3)^{1R} \tag{6}$$

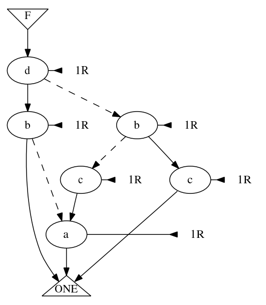
Therefore, it is easy to prove that there exist a function G, such that $G_{(a=0)} = G_0$ and $\frac{\partial^i G}{\partial a^i}_{|a=0} = G_i$. ∎

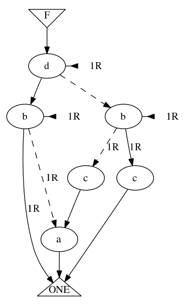
Both backward and forward register propagation (retiming) can be clearly observed in the TED data structure by the displacement of the register label. Backward retiming is shown by the movement of the register located at the top of node a in Figure 4(a), to the registers located in the node a and its edges, as shown in Figure 4(b). Forward retiming is shown by the movement of the registers in Figure 4(b) to the register in Figure 4(a).

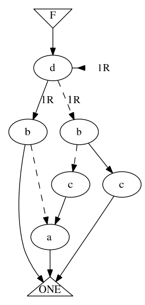
To better understand retiming in TEDs, let's take as an example a synthetic polynomial given by:

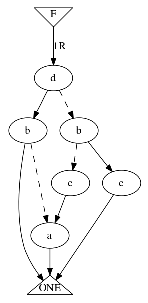
$$F = d^{1R}(b^{1R} + a^{1R}) + c^{1R}a^{1R} + b^{1R}c^{1R} \qquad (7)$$

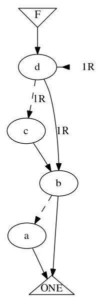
The TED and DFG corresponding to equation (7) for the variable ordering d, b, c, a are shown in Figures 5(a) and 6(a) respectively. At this point all registers are located at the primary inputs. Recall from Section 3.2 that a constant node is normalized, and it can be assumed to have as many registers as needed; therefore node ONE can be retimed forward as many times as needed. Forward retiming of node ONE enables nodes a and then node c to be forward retimed as shown in Figure 5(b). Its associated DFG is shown in Figure 6(b). In this TED, it can be seen that node b satisfies Corollary 3.2. Therefore node b can be forward retimed as well, thus obtaining the TED shown in Figure 5(c). Its corresponding DFG is shown in Figure 6(c).


(a)


(b)


(c)


(d)


(e)

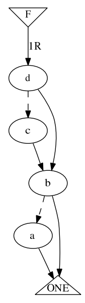
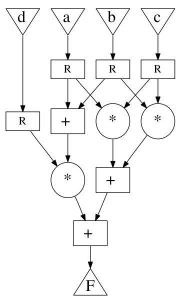
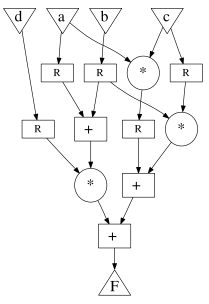
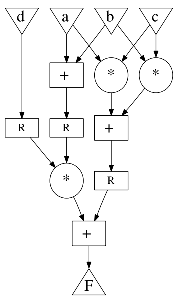

(f)

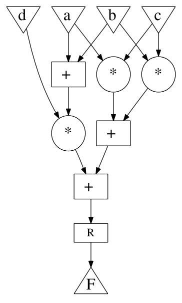
Figure 5. Step by step forward retiming in a TED. (a) Initial TED. (b) Retiming nodes a and c. (c) Retiming node b. (d) retiming node d. (e) Swapping variables b and c from (c). (f) Final retimed TED.

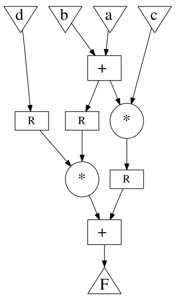
The effects of forward retiming can be observed by looking at the changes in the TED from Figures 5(b) to 5(c). Similarly, backward retiming can be observed by looking at the changes in the TED from Figures 5(c) to 5(b).


(a)


(b)


(c)


(d)


(e)

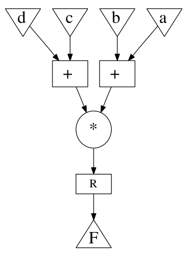

(f)

Figure 6. DFGs associated with the corresponding TEDs shown in Fig. 5

From the TED in Figure 5(c) two different approaches can be taken to reach the TED show in Figure 5(f) which generates the best DFG architecture as shown in Figure 6(f). The first approach could be to forward retime node d as shown in Figure 5(d), and afterwards swap the variables b and c. The second approach could be to swap the variables b and c as shown in Figure 5(e), and then forward retime node d.

Therefore, the TED that produces the best DFG architecture for a given cost function can be obtained through variable ordering and retiming, regardless of the initial order of the TED or its initial register grouping. It is important to note that existing retiming approaches can modify the DFG shown in Figure 6(a) to the DFGs shown in Figures 6(b)(c) and (d), but to the best of our knowledge none can get to the DFG shown in Figure 6(f), which is optimal. Timed TED can discover this optimal solution because it can explore solution space not explored by other tools. This is timed TED can perform retiming jointly with factorization and common sub expression elimination.

3.4 Canonicity

Timed TED is canonical for a given variable order, but in addition to the graph structure the labeling of registers must be taken into consideration. Specifically, the canonical form of TTED is defined as one in which all registers are retimed to its fully backward (or forward) position in the graph. Then, two TTEDs are equivalent if the underlying TEDs are equivalent and the labeling of registers is the same.

3.5 Implications of retiming on variable ordering

Theorem 3.3 *A forward register propagation in the TED data structure produces a set of boundaries in the variable ordering space. That is, a node with variable x having a set of children connecting through retimed edges to a set of nodes y_i cannot be reordered below the highest order of the set of nodes y_j. Similarly none of the nodes with variables y_i can be moved above node x.*

Proof Given the multivariate polynomial $F(\mathbf{X}, \mathbf{Y^{1R}}) \in C^m$, with $\mathbf{X}$ corresponding to a vector of none delayed primary inputs, $\mathbf{Y^{1R}}$ corresponding to a vector of delayed primary inputs, and assuming the initial variable order of the TED equal to $(x_1, \cdots, x_n, y_1, \cdots, y_n)$. Then the Taylor expansion of F, can be written as:

$$\begin{aligned}
F &= F(X, Y^{1R}) \\
&= \sum_{k_1=0}^{m} \frac{x_1^{k_1}}{k_1!} \cdots \sum_{k_n=0}^{m} \frac{x_n^{k_n}}{k_n!} \sum_{j_1=0}^{m} \frac{(y_1^{1R})^{j_1}}{j_1!} \cdots \\
&\quad \sum_{j_n=0}^{m} \frac{(y_n^{1R})^{j_n}}{j_n!} \frac{\partial^{k_1+\ldots+k_n+j_1+\ldots+j_n}}{\partial x_1^{k_1} \ldots \partial x_n^{k_n} \partial y_1^{j_1} \ldots \partial y_n^{j_n}} F_{(\mathbf{0})} \\
&= \sum_{k_1=0}^{m} \frac{x_1^{k_1}}{k_1!} \cdots \sum_{k_n=0}^{m} \frac{x_n^{k_n}}{k_n!} \frac{\partial^{k_1+\ldots+k_n}}{\partial x_1^{k_1} \ldots \partial x_n^{k_n}} \\
&\quad \left(\sum_{j_1=0}^{m} \frac{(y_1^{1R})^{j_1}}{j_1!} \cdots \sum_{j_n=0}^{m} \frac{(y_n^{1R})^{j_n}}{j_n!} \frac{\partial^{j_1+\ldots+j_n}}{\partial y_1^{j_1} \ldots \partial y_n^{j_n}} F_{(Y=\mathbf{0})} \right)
\end{aligned} \tag{8}$$

Because the decomposition of F through Taylor expansion is done recursively, the most inner partial derivatives are all of class C^1 (once differentiable), that is, the inner derivatives in equation (8) can be seen as a constant value. Therefore, applying Theorem 3.1, we can propagate forward the registers on primary inputs $\mathbf{Y}$ and rewrite equation (8) as:

$$\begin{aligned}
F = &\sum_{k_1=0}^{m} \frac{x_1^{k_1}}{k_1!} \cdots \sum_{k_n=0}^{m} \frac{x_n^{k_n}}{k_n!} \frac{\partial^{k_1+\ldots+k_n}}{\partial x_1^{k_1} \ldots \partial x_n^{k_n}} \\
&\left(\sum_{j_1=0}^{m} \frac{y_1^{j_1}}{j_1!} \cdots \sum_{j_n=0}^{m} \frac{y_n^{j_n}}{j_n!} \frac{\partial^{j_1+\ldots+j_n}}{\partial y_1^{j_1} \ldots \partial y_n^{j_n}} F_{(Y=\mathbf{0})} \right)^{1R}_{(X=\mathbf{0})}
\end{aligned} \tag{9}$$

From equation (9), it can be observed that variable ordering as discussed in [8] cannot be done across the register boundary. That is, two adjecent variables x_n and y_1 connected through a retimed edge cannot be swapped, because moving the partial derivatives inside or outside the operator $()^{nR}$ requires a retiming operation first. Therefore, while the register is located in the edge (and no further forward or backward retiming is done) the two variables x_n and y_1 cannot be swapped. This proves that if a node with variable x_i connects to a set of nodes with variables y_j, y_k through a retimed edge, then x_i cannot be moved below the highest of the variables y_j, y_k; and that variables y_j, y_k cannot be moved above variable x_i. ∎

As an example, let's consider the TED shown in Figure 5(b). In this TED, node b connects through retimed edges to nodes c and a; therefore node b cannot be moved below those nodes. Similarly nodes a and c cannot be moved above node b. It is important to note in this TED, that nodes a and c can be swapped because they are not connected through retimed edges.

Corollary 3.4 *To remove all restrictions imposed on the variable order by forward retiming, all registers have to be backward retimed to the primary inputs.*

Proof Forward retiming produces restrictions in the variable order as shown in Theorem 3.3, therefore to remove these restrictions we have to apply the backward register propagation until all registers are located at the primary inputs only. ∎

Let's consider the TED shown in Figure 5(b) again. This TED has some restrictions in its variable order, namely node b has to be above nodes a and c. Although we could remove this restriction by forward retiming node b as shown in 5(c), new restrictions arise. In the TED given by Figure 5(c) node d has a fixed possition as it cannot be moved below any other node in the TED. In certain cases it is possible to remove all restrictions by fully forward retiming the TED; this is true for the TED shown in Figure 5(d), but in general not all restrictions are removed as forward retiming a TED does not guarantee to move all register to the output of the TED. An example depicting this, is shown in Figure 7(d), where the fully forward retimed TED has left registers along the edges of the TED. In this case the approach of fully forward retiming has lead to constraint even futher the variable order. In general, only through backward retiming one can guarantee to remove all restrictions on the variable order.

Corollary 3.5 *A set of nodes z_i that are connected with each other through retimed edges, have a relative lock in their variable ordering $z_1, \cdots, z_n$.*

Proof Let's consider three adjacent variables z_i, z_j and z_k with ordering $z_i > z_j > z_k$. By applying Theorem 3.3 to the pairs (z_i, z_j) and (z_j, z_k), it can be observed that variable z_j cannot be moved neither above z_i nor below z_k; therefore the position of variable z_j has a relative lock with respect to variables z_i and z_k. Applying the same theorem to all connecting pairs in z_i we prove that the position of variables in z_i are locked relative to each other. ∎

Let's consider the effects retimed edges have on the variable ordering space of the TEDs shown in Figure 7. The TED shown in Figure 7(a) has 120 possible variable orders; the TEDs shown in Figures 7(a) and 7(b) have 24 possible orders; and the TED shown in Figure 7(d) has only 2 possible variable orders. In this last TED, the nodes

(a_0, a_1), (a_1, a_2) and (a_2, a_3) are connected with each other through retimed edges, therefore these nodes have a relative lock in their variable order. In fact, the varible order of nodes a_0, a_1 and a_2 is fixed. The variable order of node a_3 is not fixed as it can be swapped with node x. But swapping node x with a_3, causes x to be in a relative lock with respect to variables a_0, a_1 and a_2. Therefore, the only two variable orders allowed for the TED in Figure 7(d) are: $a_0 > a_1 > a_2 > a_3 > x$ and $a_0 > a_1 > a_2 > x > a_3$;

3.6 Complexity

The complexity of constructing a Timed TED is equal to the complexity of constructing a TED and then fully forward retiming it; which is in practice polynomial in time. The complexity of reordering a TTED, as it is for TEDs, is an exponential problem (factorial in the number of variables) [9]. It can be shown that finding the ordering that minimizes some metric of the TTED is in worst case $\mathbf{O}(n \times n!)$, where n is the number of variables in the TED representation.

4 Examples

4.1 Retiming for clock period minimization

The equation of the 4-tap FIR filter shown in Figure 1(a) is given by:

$$y = a_0x + a_1x^{1R} + a_2x^{2R} + a_3x^{3R} \qquad (10)$$

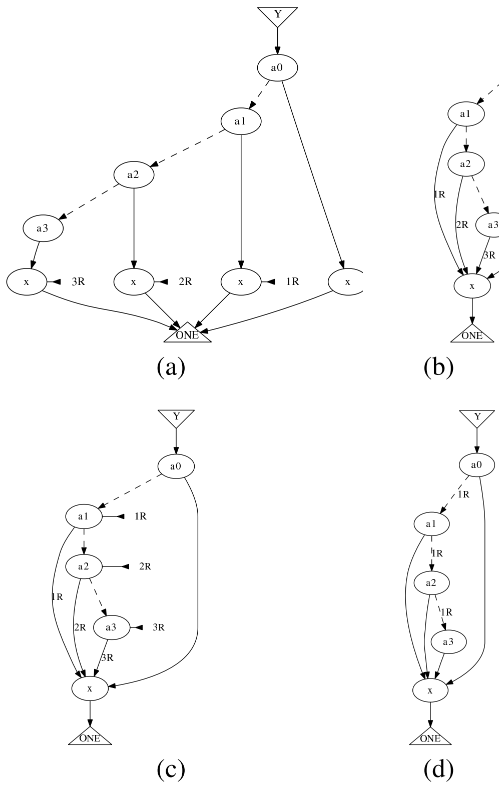


Figure 7. TEDs for the FIR filter.

Its TED and DFG are shown in Figures 7(a) and 8(a) respectively. Given that there are no resource contraints, the architecture generated by this DFG has a minimum clock period of 1MPY and 3ADD operations. But the node x of the TED in Figure 7(a) can be forward retimed as shown in Figure 7(b).

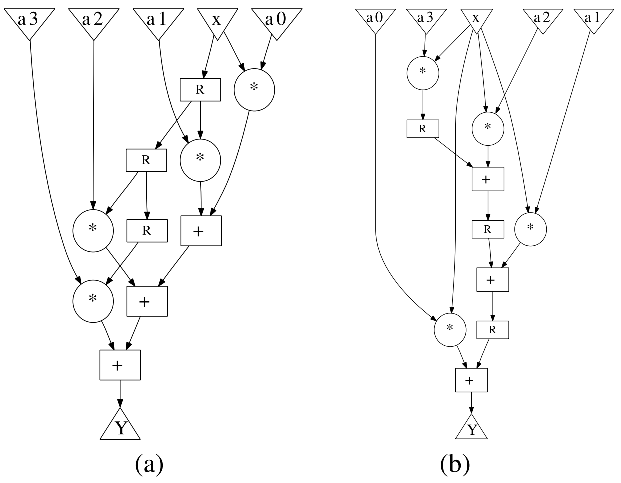


Figure 8. FIR transformation to reduce clock period: (a) 1MPY and 3ADD operations. (b) 1MPY and 1ADD operations.

At this point, nodes a_0, a_1, a_2 and a_3 can take as many registers as needed, because these nodes represent constant values. Thus by assigning i registers to a_i as shown in Figure 7(c), the TED can be forward retimed through nodes a_3, a_2 and a_1 as shown in Figure 7(d).

From the TED in Figure 7(d) the DFG in Figure 8(b) is obtained. The architecture generated by this DFG (given no resource constraints) has 1MPY and 1ADD operations. This shows that clock period minimization can be achieved through timed TED.

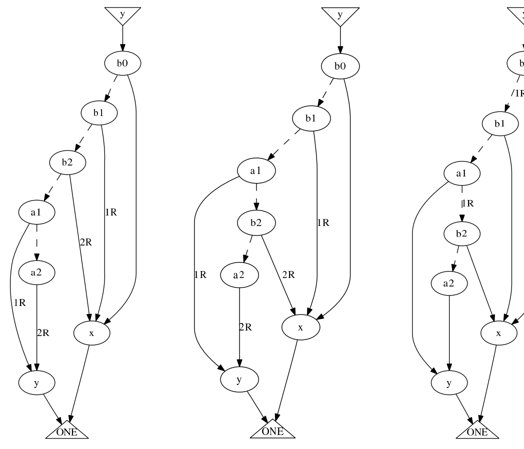
4.2 Retiming for register minimization

The TED and DFG, for the Bi-quadratic filter described by equation (11), after forward retiming nodes x and y are shown in Figures 9(a) and 10(a).

$$y = b_0x + b_1x^{1R} + b_2x^{1R} + a_1y^{1R} + a_2y^{2R} \qquad (11)$$

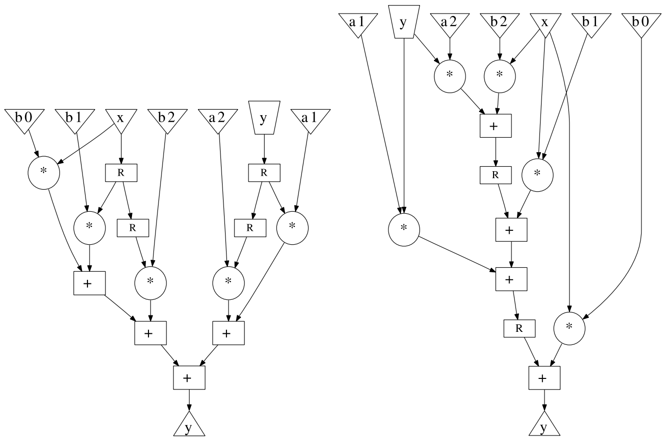
Assuming no resource constraints, this DFG produces an architecture which requires 4 registers and a clock period of 1MPY and 3ADD operations. The TED in Figure 9(a) has a relative variable order lock between a_1, a_2 and x; and b_1, b_2 and y; but there is no lock between variables a_i and b_j, therefore we can swap varaibles a_2 and b_1 as shown in Figure 9(b). From the bi-quadratic filter, nodes a_i and b_j are constants, therefore we can forward retime the TED in Figure 9(b) as to obtain the TED shown in 9(c).

The DFG associated with the TED in Figure 9(c) is shown in Figure 10(b). If there are no resource constraints, this DFG produces an architecture with 2 registers and a clock period of 2MPY and 2ADD. It can be observed that the register count has decreased, but the clock period has increased (assuming the delay of a multiplier is greater than the delay of an adder). This shows that register minimization can be achieved through timed TED; and that register and clock period minimization are different, but interrelated objectives.



(a) (b) (c)

Figure 9. (a) Bi-quadratic TED. (b) After variable ordering. (c) After retiming.



(a) (b)

Figure 10. Bi-quadratic transformation to reduce the number of registers: (a) 4 registers. (b) 2 registers.

Although the implicit assumption that $b_0 \neq 1$ was made in equation (11) to show there is a tradeoff between register and clock period minimization, normaly constants a_i and b_j are normaly scaled to have $b_0 = 1$. This careful selection of contants reduces the clock period of the DFG in Figure 10(b) by 1MPY, so that its generated architecture requires fewer registers and has a lower clock period than the one obtained through the DFG in Figure 10(a).

5 Conclusions

We have extended the TED data structure to cope with registers and handle sequential designs. This extension of the data structure, called Timed TED, allows the developper to express the design specification at a higher level of abstraction. One can specify as constraint that certain computation is needed at n clock cycles and Timed TED can be used to find an efficient DFG optimized for it. We have proved that timed TEDs support backward and forward retiming and that retiming can be done to perform register and clock period minimization. Furthermore, we have proved that retiming can be done jointly with variable ordering under certain restrictions; therefore different factorizations and decompositions can be attained. This allows to explore a larger space and obtain other solutions that are not reachable otherwise.

As future work we plan to deal with recursive equations, such as FII, and explore the potential of TTEDs for doing loop unrolling to exhibit higher level of parallelism and increase throughput.

6 Acknowledgements

This work was supported by the US National Science Foundation, award no. CCF-0702506.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry", in *Algorithmica*, 1991, vol. 6, pp. 5–35.

[2] Giovanni DeMichelli, *Synthesis and Optimization of Digital Circuits*, 1994.

[3] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power", in *in Proc. IEEE Int. Conf. Computer Aided Design*, 1993, pp. 398–402.

[4] Keshab K. Parhi, *VLSI Digital Signal Processing Systems*, 1999.

[5] D. Gomez-Prado, Q. Ren, M. Ciesielski, J. Guillot, and E. Boutillon, "Optimizing Data Flow Graphs to Minimize Hardware Implementations", in *Design Automation and Test in Europe, DATE 09*, April 2009, pp. 117–122.

[6] M. Ciesielski, D. Gomez-Prado, Q. Ren, J. Guillot, and E. Boutillon, "Optimization of data-flow computation using canonical ted representation", *IEEE Trans. on Computers*, vol. 28, no. 9, pp. 1321–1333, September 2009.

[7] M. Ciesielski, J. Guillot, D. Gomez-Prado, and E. Boutillon, "High-level dataflow transformations using taylor expansion diagrams", *IEEE Design & Test*, vol. 26, no. 4, pp. 46–57, July/August 2009.

[8] D. Gomez-Prado, Q. Ren, S. Askar, M. Ciesielski, and E. Boutillon, "Variable Ordering for Taylor Expansions Diagrams", in *IEEE Intl. High Level Design Validation and Test Workshop*, November 2004, pp. 55–59.

[9] Daniel. Gomez-Prado, "Variable Ordering for Taylor Expansion Diagrams", Master's thesis, University of Massacusetts Amherst, 2006.