System Clock Estimation based on Clock Wastage Minimization

Sanjiv Narayan
Daniel D. Gajski

Technical Report #91-49
December 5, 1991

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
(714) 856-8059

narayan@ics.uci.edu

Abstract

When synthesizing a hardware implementation from behavioral descriptions, an important decision is the selection of a clock cycle to schedule the datapath operations into control steps. Most existing behavioral synthesis systems either require the designer to specify the clock cycle explicitly or require that the delays of the operators used in the design be specified in multiples of a clock cycle. In the absence of any tool to guide the selection of a clock cycle, a bad choice of the clock period could adversely affect the performance of the synthesized design. We present an algorithm for estimating the system clock based on a clock wastage minimization criteria. Limitations of previous approaches to the problem are discussed. The results obtained prove that the clock cycle estimated by the Clock Wastage Minimization method produce faster designs than previous solutions to the problem.

Contents

List of Figures

1 Introduction

Behavioral Synthesis involves the transformation of a specification or design description into a set of interconnected micro-architectural components which satisfy the behavior and any specified constraints. Several tasks are performed during synthesis. *Component selection* chooses components from a design library to perform the desired functions. *Scheduling* determines the appropriate control step for each operation in the behavioral description. *Resource binding* assigns components to the specification's operations.

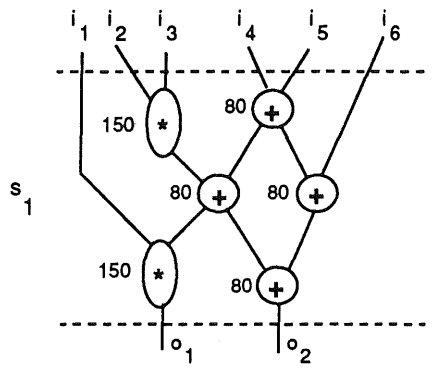
The scheduler tries to execute as many operations as possible in each control step to extract as much parallelism as possible. The overall performance (or execution time) of the design depends on the duration of each control step i.e. the *clock cycle.*

Most existing behavioral synthesis systems [Wa90, PaKnGi86, PaKn89, BaMa89] approach the problem of determining the clock cycle in one of following ways :

- The designer specifies a clock cycle explicitly.
- All operators are assumed to have identical delays. A unit clock cycle is assumed and each operator requires exactly one clock to execute.
- The designer specifies the operator delays in multiples of a clock. Once again, a unit clock cycle is assumed.

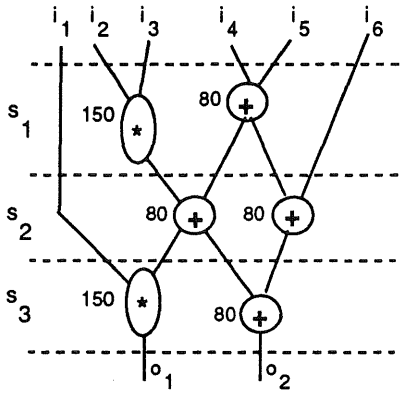
The quality, with respect to performance, of the schedule generated by the scheduler, depends on the clock cycle. In each of the approaches outlined above, the designer has nothing more than intuition to assist him while selecting a clock cycle. A bad choice of the clock cycle given to the synthesis tools could result in inefficient designs wherein the operators are greatly underutilized and the performance of the resulting design is unacceptably slow. Let us illustrate this with the example of Figure 1.

Figure 1(a) shows a dataflow graph (DFG) with 2 multiply operations and 4 add operations. The delays of the multiply and add functional units are 150 ns and 80 ns respectively. Assume that we are interested only in the output values o_1 and o_2. As shown in Figure 1(a), we use a clock cycle of 380 ns to give us the fastest possible DFG execution time of 380 ns. But we are using a large number of functional units - 2 multipliers and 4 adders. Since this is not desirable, we attempt to share functional units amongst the various operations. Figure 1(b) shows that a reduction of the clock cycle to 150 ns requires only 1 multiplier and 2 adders, giving us a higher completion time of 450 ns (3 clock cycles). We further reduce the functional units to 1 multiplier and 1 adder with the same clock cycle as shown in Figure 1(c). However, the completion time for the DFG is now 600 ns (4 clock cycles). Finally in Figure 1(d), we see that a clock cycle of 80 ns gives a completion time of 400 ns (5 clock cycles) *and* uses a minimal number of functional units - 1 multiplier and 1 adder. With respect to performance, this implementation is very close to the fastest implementation of Figure 1(a), while using significantly fewer number of functional untis.



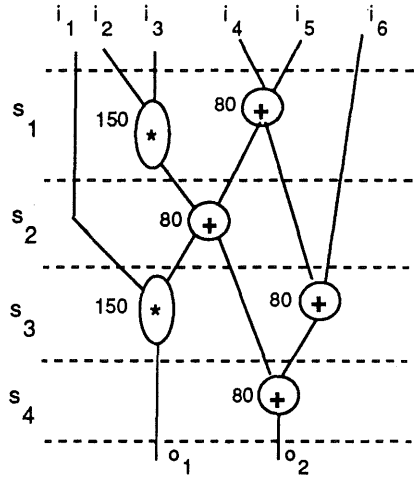
Clock Cycle : 380 ns
Completion Time : 380 ns
Resources used : 2 multipliers, 1 adder

(a)



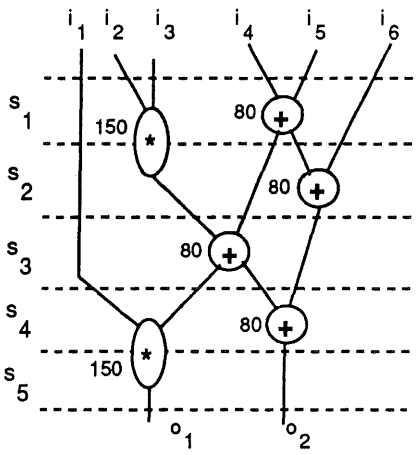
Clock Cycle : 150 ns
Completion Time : 450 ns
Resources used : 1 multiplier, 2 adders

(b)



Clock Cycle : 150 ns
Completion Time : 600 ns
Resources used : 1 multplier, 1 adder

(c)



Clock Cycle : 80 ns
Completion Time : 400 ns
Resources used : 1 multiplier, 1 adder

(d)

Figure 1: Effect of the clock cycle on DFG completion times (numbers adjacent to the operators represent respective operator delays, dotted lines represent state boundaries)

From Figure 1 it is evident that the choice of a particular clock cycle influences the sharing of functional units that will be used in the design as well the completion time or performance of the design.

Thus it is essential that clock estimation be an integral part of any synthesis system which should provide the designer with feedback as to how various clock cycle values could possibly affect the design performance.

In Section 3, we discuss previous approaches to estimating the clock. In Section 4, we explain the underlying design model which forms the basis of our clock cycle calculation. The Clock Wastage Minimization Algorithm is presented along with a detailed example in Section 5. Finally the results of estimations on well known examples are presented in Section 6.

2 Problem Definition

The goal of clock cycle calculation can be defined as follows :

> Given a description of a design and a list of components that will be used to implement the design, determine that value of the clock cycle which will minimize the execution time for the dataflow graph of the design.

The execution time of a dataflow graph is the length of the schedule determined for it. Since the scheduling of the operations in the dataflow graph depends on the operator delays and the clock cycle, the execution time is dependent on the clock cycle.

3 Previous Work

A few synthesis tools [PaPiMi86, PaPa85, JaMiPa88] have incorporated clock estimation techniques which are then used to either examine area-time tradeoffs in the design or to guide synthesis tasks such as scheduling.

In MAHA [PaPiMi86], first the critical path in the dataflow graph is determined. The maximum delay of any operator in the critical path is chosen as the clock cycle.

The clocking scheme proposed in [PaPa85] computes a lower bound for the clock cycle of a multistage system as being the longest stage time. Since the longest stage time is *at least* as large as the longest operator delay, this scheme computes a clock cycle greater than or equal to the largest operator delay.

A model for Area-Time Estimation is presented in [JaMiPa88]. The dataflow graph is divided into a number of *time steps*. The critical path delay and the number of time steps are used to compute the lower bound on the clock as given in the following equation :

$$clk = MAX[\ \frac{Critical\ Path\ Delay}{No.\ of\ Time\ Steps}\ ,\ MAX(Operator\ Delay)\ ] \quad (1)$$

Each of the above approaches assume that each operation must execute within one clock cycle. Multi-cycle operations, where an operation could be scheduled in two or more control steps, are not permitted. Consequently, they are similar to each other in one respect – the clock cycle calculated by each of the above methods is at least as long as the largest operator delay. We shall refer to these clock estimation methods as the *Maximum Operator Delay* methods. The advantages of the above methods is that they are simple to implement and their algorithm complexity is linear in the number of different operator types that will be used to implement the design.

Let the clock cycle computed by the Maximum Operator Delay (MOD) methods be denoted by CLK_{mod}. Let T denote the number of distinct operator types in the DFG and $OpDelay(t_i)$ denote the delay of an operator of type t_i. From the above analysis of Maximum Operator Delay methods we can clearly see that,

$$CLK_{mod} \geq MAX\ [OpDelay(t_i)\ ], \qquad for\ all\ operator\ types\ t_i \quad (2)$$

A shortcoming of these methods is evident in cases where the operators have widely differing delays. Consider a dataflow graph with a multiplier (delay : 200 ns) and an adder (delay : 50 ns) as the only components. The Maximum Operator Delay method would use a clock of 200 ns to schedule the dataflow graph. Thus whenever an add operation is scheduled in a control step, the adder is idle for 150 ns in that clock cycle. The utilization of the adder is only 25%. If instead, the graph was scheduled with a clock cycle of 50 ns, both the adder (delay : 1 clock) and the multiplier (delay : 4 clocks) can be utilized throughout the clock cycle.

Another approach to clock cycle calculation involves scheduling the dataflow graph for all possible clock cycle values, and selecting that clock which results in the fastest completion time. Scheduling algorithms typically have computational complexities of $O(n^2 logn)$, where n is the number of nodes in the dataflow graph. An *exhaustive* scheduling technique for all possible clock cycles will be computationally expensive even for small dataflow graphs. Also , to be able to schedule a dataflow graph, we require an allocation to have been already determined. In addition, it is difficult to define the fastest schedule for complex designs, such as a VHDL description with several concurrent processes.

However, the above exhaustive scheduling method could be feasible, if we could somehow reduce the search space involved with examining an entire range of clock values. The Clock Wastage Minimization algorithm presented in this paper achieves this by providing the designer with fast estimates of a feasible clock cycle. It calculates a clock cycle which would result in the maximum utilization of all the operators, **without performing any scheduling** and its computational complexity is linear with respect to the number of operations in the entire design.

4 Design Model

The underlying design model assumed for the purpose of clock calculation is shown in Fig 2. A two level bus structure is assumed for the interconnection across the registers and functional units. This model allows for easy analysis of performance issues since the delay of the tristate driver can be considered to be constant with respect to the number of the tristate drivers driving a bus.

Operations can execute over several clocks. Thus if a functional unit has a delay of 90 ns and the clock cycle is 50 ns, then the functional unit executes in two clock cycles. A typical register-to-register transfer involves operands being read from registers, an operation performed on the operands, and the results stored in another register. The delays associated with this type of register-to-register transfer are the following :

- Delay of the operation,
- Delay associated with two levels of tristate drivers.
- Register setup time and propagation delay.

Let $delay(t_i)$ represent the total delay in a register-to-register transfer involving an operator of type t_i. Then,

$$\begin{aligned} delay(t_i) \;=\; & OpDelay(t_i) \;+\; 2 \;*\; (tristate\ driver\ delay) \\ & +\; Register\ setup\ time \;+\; Register\ propagation\ delay \end{aligned} \tag{3}$$

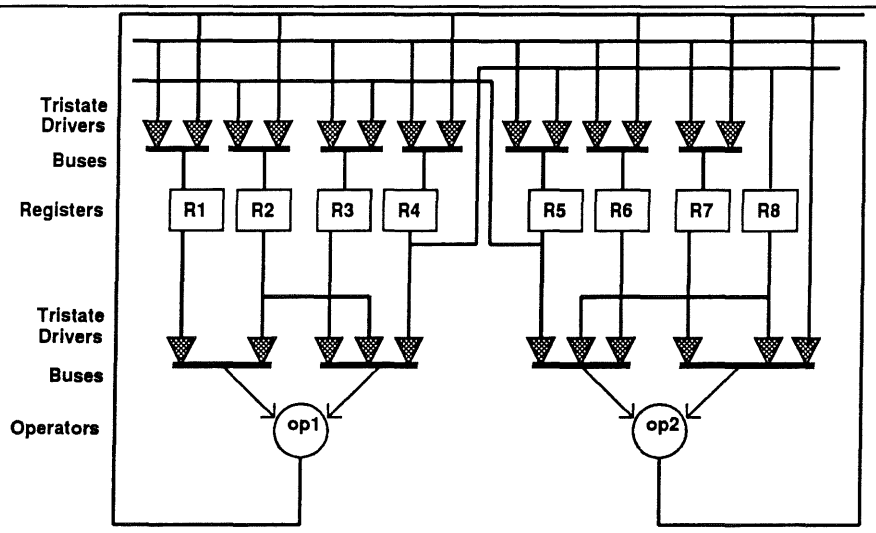


Figure 2: Design Model for Clock Calculation

Thus the operator delays used in our algorithm are actually the values, $delay(t_i)$, computed above. To support operator chaining, links are needed from the output ports of some functional units directly to the input ports of other functional units. This linking is accomplished by using the path from a functional unit's output port through one of the buses to some other functional unit's input port.

5 The Clock Wastage Minimization Algorithm

5.1 Definition of Terms

A few terms which will be referenced frequently are defined here.

DFG Completion Time : Represents the execution time of the DFG. If the DFG is scheduled into C control steps with a clock cycle clk, then the completion time of the DFG, d_{DFG}, is defined as :

$$d_{DFG} = C \times clk \tag{4}$$

Operator Occurrences : This represents the number of occurrences of an operator of type t_i in the behavioral statements (or the DFG) representing the design and is denoted by $occur(t_i)$.

Operator Wastage : The portion of the clock cycle for which the functional unit implementing an operation in the DFG is idle is defined as the operator wastage. Wastage for a particular operator can be defined as the difference between the operator delay and the *next higher multiple* of the clock cycle. For a given clock value, clk, the wastage for an operator type t_i is denoted by $Waste(clk, t_i)$. If an adder has delay of 80 ns and the clock cycle is 100 ns, then the wastage involved with every *use* of the adder is 20 ns, or $Waste(100, +) = 20\ ns$. However, a multiplier with a delay of 150 ns will take 2 clocks to execute and the wastage will be 50 ns. Note, that in this document, the term $Waste(100, \times)$ represents the wastage involved when a multiplier is used and the clock cycle is 100 ns.

Average_Wastage/Operator : This represents the average value of the time for which any operator will be idle in a clock cycle. For a given clock cycle, clk, the individual clock wastages for each operator are weighted by the number of occurrences of that operator in the dataflow graph and divided by the total number of operators in the dataflow graph. As an example, consider a design which has 2 add and 4 multiply operations in the DFG. Assume a clock cycle of 100 ns. If we use adders and multipliers with delays of 80 ns and 150 ns respectively, we have,

$clk = 100$ ns (given)
$T = number\ of\ operator\ types = |\{+, \times\}| = 2$
$occur(+) = 2$, $occur(\times) = 4$ (given)
$delay(+) = 80ns$, $delay(\times) = 150\ ns$ (given)
$Waste(clk, +) = 20$ ns, $Waste(clk, \times) = 50$ ns

$$\begin{aligned} Total\ Wastage &= [occur(+).Waste(clk, +)] + [occur(\times).Waste(clk, \times)] \\ &= [2 \times 20] + [4 \times 50] \\ &= 240ns \end{aligned}$$

$$\begin{aligned} Average\ Wastage/Operator &= Total\ Wastage \div (\textstyle\sum_{i=1}^{T} occur(t_i)) \\ &= 240 \div (2 + 4) \\ &= 40ns \end{aligned}$$

Clock Utilization : This is defined as the percentage of the clock cycle that is utilized for useful computation by all the functional units in the implementation. In the above example, the clock utilization for a clock cycle of 100 ns can be calculated as follows :

$$\begin{aligned} \textit{Clock Utilization} &= 1 - (\textit{Average\_Wastage/Operator} \div \textit{clk}) \\ &= 1 - (40 \div 100) \\ &= 0.6 \textit{ or } 60\% \end{aligned}$$

5.2 Notation

We use the following variables in our algorithm :

T	: Number of distinct operator types in the DFG.
$delay(t_i)$	: Delay of the operator of type t_i, as computed in Equation 3.
$occur(t_i)$	: The number of *occurrences* of operator of type t_i in the DFG.
N_{ops}	: Total number of operators in the DFG ($N_{ops} = \sum_{i=1}^{T} occur(t_i)$).
$Waste(clk, t_i)$	: Clock wastage for an operator of type t_i for a specific clock cycle, clk.
CLK_{lower}	: Lowest clock cycle examined by the Wastage Minimization method
CLK_{upper}	: Highest clock cycle examined by the Wastage Minimization method.
CLK_{mod}	: Clock cycle estimated by the Maximum Operator Delay method (Section 3).
CLK_{wm}	: Clock cycle estimated by the Wastage Minimization method.

5.3 Algorithm

The Clock Wastage Minimization method for calculating the clock examines the *occurrences* of operators in the DFG, and selects a clock cycle so as to minimize the wastage involved when an operator is idle for a portion of the clock cycle.

Intuitively, the algorithm examines a set of clock cycles within a specific range. For each clock cycle, the Wastage Minimization method computes the wastages associated with each operator and determines the clock utilization. The value of the clock cycle which yields the highest clock utilization is determined to be the best clock cycle. The complete algorithm is presented in Figure 3. An outline of the method is presented below.

Step 1 : Compute $delay(t_i)$, $occur(t_i)$

For each operator type t_i in the DFG, $delay(t_i)$ is computed as given in Equation 3 and $occur(t_i)$ is computed by examining all the nodes in the dataflow graph for occurrences of operator type t_i.

Step 2 : Determine CLK_{lower} and CLK_{upper}

The Wastage Minimization algorithm computes wastage for the operators over a range of possible clock values from CLK_{lower} to CLK_{upper}. CLK_{upper} is the largest value of $delay(t_i)$ (over all operator types t_i in the design), as computed in step 1 above.

Design libraries often specify the maximum clock frequency at which the clock input of a bistable circuit may be driven such that stable transitions of logic levels are maintained. For example, the VLSI Technology Inc. VDP100 Datapath Element Library [VTI88] specifies the maximum clock frequency for a D Flip-Flop as being 75 MHz. Thus the value of CLK_{lower} that will be used is 1/(75 MHz) or 14 ns. In case such a maximum clock frequency is not specified, then CLK_{lower} is approximated as the smallest value of $delay(t_i)$ computed in Step 1, over all operator types t_i in the design.

Step 3 : Clock Wastage Calculation Loop

Repeat the following steps for all values of the clock cycle, clk, where $CLK_{lower} < clk < CLK_{upper}$

- **Compute $Waste(clk, t_i)$, the Clock Wastage for each operator type t_i :** For each operator t_i, the difference between the operator delay and the next higher clock multiple can be calculated as follows:

$$Waste(clk, t_i) = ((\lceil delay(t_i) \div clk \rceil) \times clk) - delay(t_i) \tag{5}$$

- **Compute $Average\_Wastage/Operator(clk)$:** This is a measure of how much of the clock cycle is wasted on the average by each of the operators in the DFG, for a specific value of the clock, clk. First the total wastage is calculated by *weighing* the wastage $Waste(clk, t_i)$ involved with *each* operator type t_i by the number of its occurrences in the DFG, $occur(t_i)$. The total wastage is then divided by the total number of operators in the design to give the average value of the wastage per operator. If T represents the number of distinct operator types in the design, we can compute:

$$Average\_Wastage/Operator(clk) = \left\lceil \frac{\sum_{i=1}^{T} (occur(t_i) \times Waste(clk, t_i))}{\sum_{i=1}^{T} occur(t_i)} \right\rceil \tag{6}$$

- **Compute $Clock\_Utilization(clk)$:** The percentage of the clock cycle that is wasted is the ratio of the $Average\ Wastage/Operator(clk)$ calculated above and the clock cycle, clk. The utilization is then calculated as :

$$Clock\_Utilization(clk) = 1 - \left(\frac{Average\_Wastage/Operator(clk)}{clk} \right) \tag{7}$$

Step 4 : Calculating best clock, CLK_{wm}

The value of the clock cycle which maximizes the Clock Utilization is selected as the best clock, CLK_{wm}.

procedure MINIMUM_WASTAGE_CLOCK (DFG : data_flow_graph)

/* Given a dataflow graph, return a clock estimate which minimizes
the wastage of the clock cycle over all the operators in that graph. */

begin

for all operator types t_i **do**
Compute $occur(t_i)$ by examining the DFG
Compute $delay(t_i)$
end for

Determine CLK_{upper} and CLK_{lower}
$Max\_Utilization = 0$

for all values of clk, such that $CLK_{lower} < clk < CLK_{upper}$ **do**

for all operator types t_i **do**
$Waste(clk, t_i) = ((\lceil delay(t_i) \div clk \rceil) \times clk) - delay(t_i)$
end for

$$Average\_Wastage/Operator(clk) = \left\lceil \frac{\sum_{i=1}^{T} (occur(t_i) \times Waste(clk, t_i))}{\sum_{i=1}^{T} occur(t_i)} \right\rceil$$

$$Clock\_Utilization(clk) = 1 - \left(\frac{Average\_Wastage/Operator(clk)}{clk} \right)$$

If [$Clock\_Utilization(clk) > Max\_Utilization$] **then**
$Max\_Utilization = Clock\_Utilization(clk)$
$CLK_{wm} = clk$
end if
end for

return (CLK_{wm}) ;
end MINIMUM_WASTAGE_CLOCK ;

Figure 3: Minimal Wastage Clock Calculation Algorithm

5.4 The HAL Second Order Differential Equation Example

To illustrate how the Wastage Minimization Algorithm calculates CLK_{wm}, we apply it to the Second Order Differential Equation benchmark [PaKnGi86]. The components being used are from the VTI VDP100 datapath library given in Figure 6.

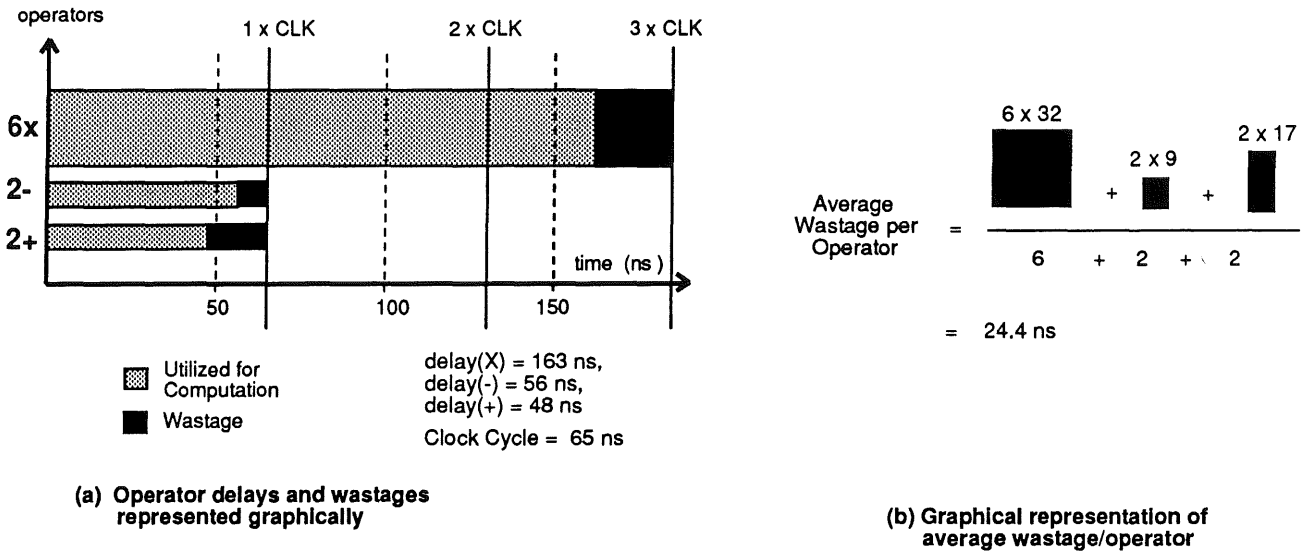


Figure 4: Graphical Representation of Clock Wastages for the HAL Differential Equation example

Step 1 : Compute $delay(t_i)$, $occur(t_i)$

The values of $delay(t_i)$ are calculated using Equation 3 for each operator type t_i. The values of $delay(t_i)$ are given in Figure 6(b). The dataflow graph for the differential equation benchmark has 2 add, 2 subtract and 2 multiply operations. Thus we have,

$$occur(\times) = 6 \qquad delay(\times) = 163\ ns$$
$$occur(-) = 2 \qquad delay(-) = 56\ ns$$
$$occur(+) = 2 \qquad delay(+) = 48\ ns$$

In Figure 4(a), the delays of the operators are represented graphically as the length of the lightly shaded regions along the X-axis. The *occurrences* of the operators is represented by the thickness of the shaded regions along the Y-axis.

Step 2 : Determine CLK_{lower} and CLK_{upper}

Since the VDP100 library specifies that the maximum frequency for clocking registers is 75 MHz, CLK_{lower} = 1 / 75 Mhz = 14 ns. CLK_{upper} is the largest operator delay, i.e. $CLK_{upper} = delay(\times) = 163$ ns.

Step 3 : Clock Wastage Calculation Loop

We shall illustrate the calculation of clock utilization for a clock cycle of 65 ns. Using Equation 5, we get the wastages for each operator as the difference between the delay and the next higher multiple of the clock cycle:

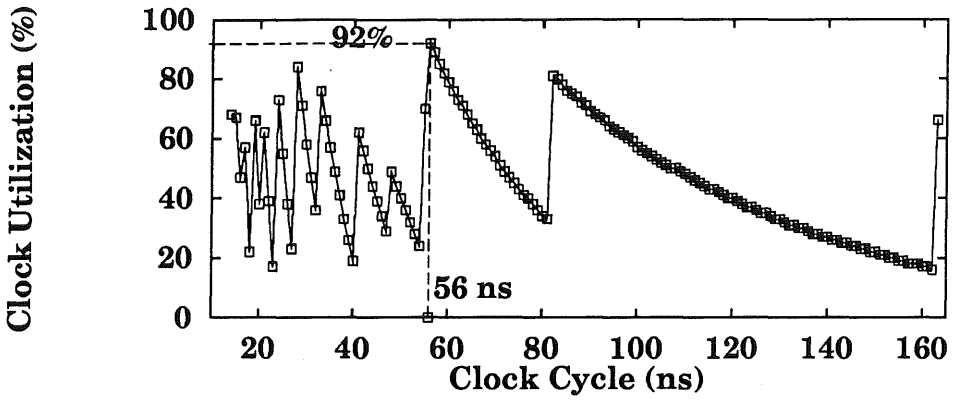


Figure 5: Clock Utilization as function of the Clock Cycle for the HAL Differential Equation example.

$$Waste(65, \times) = (3 \times 65) - 163 = 32\ ns$$
$$Waste(65, -) = (1 \times 65) - 56 = 9\ ns$$
$$Waste(65, +) = (1 \times 65) - 48 = 17\ ns$$

The wastages for each operator type for a clock cycle of 65 ns are shown as the dark shaded regions in Figure 4(a). The average wastage per operator for a clock cycle of 65 ns can be calculated by Equation 6. This is shown graphically in Figure 4(b).

$$Average\ Wastage/Operator(65) = \left\lceil \frac{6 \times 32\ +\ 2 \times 9\ +\ 2 \times 17}{6\ +\ 2\ +\ 2} \right\rceil = 24.4\ ns$$

Finally, Clock Utilization for a 65 ns clock cycle can be computed as given in Equation 7,

$$Clock\_Utilization(65) = 1 - (24.4/65) = 0.62\ or\ 62\%$$

Step 4 : Calculating best clock, CLK_{wm}

The Clock Wastage calculation loop of step 3 is repeated for all values of the clock cycle from 14 ns to 163 ns. We find that the maximum value of clock utilization of 92% is achieved at clock cycle value of 56 ns as shown in Figure 5. This is selected as the best value of the clock, CLK_{wm}.

The Maximum Operator Delay method estimated a clock value of 163 ns which resulted in a utilization of 73%. An analysis of the clock estimation results for this example is presented in Section 6.

5.5 Computational Complexity

Let T be the number of operator types in the design, N_{ops} be the total number of operators in the DFG and *range* be the number of clock cycles examined by the method, i.e.,

$$range = CLK_{upper} - CLK_{lower}$$

The *occurrences* of each operator in step 1 can computed in $O(N_{ops})$ time . For each of the *range* clock values considered (loop of step 3), $Waste(clk, t_i)$ and $Average\_Wastage/Operator$ can be computed in $O(T)$ time. The overall time complexity is thus $O(N_{ops} + T.range)$.

It can be shown that the clock cycle value which minimizes the clock wastage is either one of the operator delays or their divisors. This observation follows from Figure 5 where the peaks in the utilization curve occur at clock cycle values which are the divisors of the operator delays. By evaluating clock utilization at these points only, we can significantly reduce the number of clock cycle values (*range*) that are examined by the algorithm. In the current implementation the user can decide whether the entire range of clock cycles or only the divisors of the delays within the range of clock valu es CLK_{lower} and CLK_{upper} are to be examined.

Datapath Component	VTI Component Name	Delay
Adder	VDP3ALU001	38 ns
Multiplier	VDP3MLT001	153 ns
Subtractor	VDP5ALU001	46 ns
Tristate Buffer	VDP3TSB001	2.6 ns
Register	VDP3DFF001	setup time 3.8 ns
		hold time 1.0 ns
		max. Freq 75 Mhz

a) Components from the VTI VDP100 Datapath Element Library

Operator	Delay(i) (Equation 3)
Adder	48 ns
Multiplier	163 ns
Subtractor	56 ns

b) Functional Unit Delays used in the Wastage Minimization Algorithm. [derived from Equation 3 and (a)]

Figure 6: The VTI VDP100 Datapath Component Library

6 Experimental Results

6.1 Implementation

The Clock Wastage Minimization Algorithm presented in the previous section has been implemented. The input to the estimator is a SpecChart [NaVaGa91] description representing the behavior of the design for which clock has to be determined. In addition, the delays of the operators as given in Figure 6(b) are provided. The clock estimator outputs that value of the clock (CLK_{wm}) which minimizes wastage over all the operators. In addition, it calculates the utilization of the operators as a percentage of the clock cycle. The *manual* page for the clock cycle estimator, **scestclk**, is given in Appendix A.

We used the VLSI Technology Inc. VDP100 Datapath Element Library [VTI88] to obtain the delays of the functional units. The datapath elements used are given in Figure 6.

6.2 Experiments using the Clock Wastage Minimization method

The Clock Wastage Minimization Algorithm was tested on four well known examples : the HAL Second Order Differential Equation [PaKnGi86], a Fifth Order Elliptical Filter [KuWhKa85], the AR Lattice filter [JaMiPa88] and a Linear Phase B-Spline Interpolated Filter [PaFe90]. The VHDL for these example can be found in Appendix B.

To verify that the clock estimate produced by the Wastage Minimization algorithm did indeed produce faster completion times as compared to the Maximum Operator delay method, for *each* of the benchmarks above, we did the following :

1. We estimated the clock using both the Maximum Operator Delay method (CLK_{mod}) and the Clock Wastage Minimization method (CLK_{wm}). The values of the estimated clock cycles and clock utilization for both the methods are given in **columns A** and **B** of Figure 7. For example, in the case of the HAL Differential Equation example, the Maximum Operator Delay method estimates a clock of 163 ns (utilization : 73%), whereas the Clock Wastage Minimization method gives a clock estimate of 56 ns (utilization : 92%).

2. The completion time of a DFG scheduled with an estimated clock cycle is a good measure of how good that clock estimate is. We scheduled the DFG using a clock cycle equal to that estimated by both the methods, by employing the scheduling algorithm presented in [PaGa87]. For all the examples, we specified an allocation of two operators of each type as an input to the scheduler. For example, the Differential Equation Benchmark was scheduled with an allocation of 2 adders, 2 subtractors and 2 multipliers. The number of control steps produced as a result of scheduling *without chaining the functional units* is given in **column C** of Figure 7.

   It must be emphasized that **scheduling and allocation are not required by the Clock Wastage Minimization algorithm - they are simply being used to determine completion time (performance) on the basis of the clock cycle generated by the method. This ena es us to evaluate the quality of the estimate.**

3. We calculate the completion time for the DFGs as defined in Equation 4. The completion times for the DFG's can be found in **column D** of Figure 7. The improvement in the completion times produced by the clock estimated by the Clock Wastage Minimization method compared to the Maximum Operator Delay method is expressed as a percentage in **column E.**

$$Performance\ Improvement = 1 - \left(\frac{Completion\ Time(CLK_{wm})}{Completion\ Time(CLK_{mod})} \right) \qquad (8)$$

   Similar results for the case when *chaining of functional units* is permitted during scheduling is shown in **columns F, G, and H**

It is evident from the results of Figure 7 that the clock cycles estimated by the Clock Wastage Minimization method produce designs with better performance when compared to clock cycles

Example	Clock Estimation Method	Clock Estimated A (ns)	Clock Utilization B (%)	Scheduling without chained operators			Scheduling with chained operators		
				C Control Steps	D Completion Time (ns)	E Performance Improvement (%)	F Control Steps	G Completion Time (ns)	H Performance Improvement (%)
Differential Equation [PaKnGi86]	Max. Operator Delay	163	73%	4	652	-	4	652	-
	Clock Wastage Minimzation	56	92%	10	560	**14.1%**	10	560	**14.1%**
Fifth Order Digital Elliptic Filter [KuWhKa85]	Max. Operator Delay	163	46%	16	2608	-	11	1793	-
	Clock Wastage Minimzation	24	95%	49	1176	**54.9%**	47	1128	**37.1%**
AR Lattice Filter [JaMiPa88]	Max. Operator Delay	163	70%	11	1793	-	10	1630	-
	Clock Wastage Minimzation	55	92%	26	1430	**20.2%**	23	1265	**22.4%**
Linear Phase B-Spline Interpolated Filter [PaFe90]	Max. Operator Delay	163	57%	6	978	-	5	815	-
	Clock Wastage Minimzation	24	92%	24	576	**41.1%**	23	552	**32.2%**

Figure 7 : Comparison of Completion Times of DFG's when scheduled with the clock cycles estimated by the Maximum Operator Delay and Clock Wastage Minimization methods.

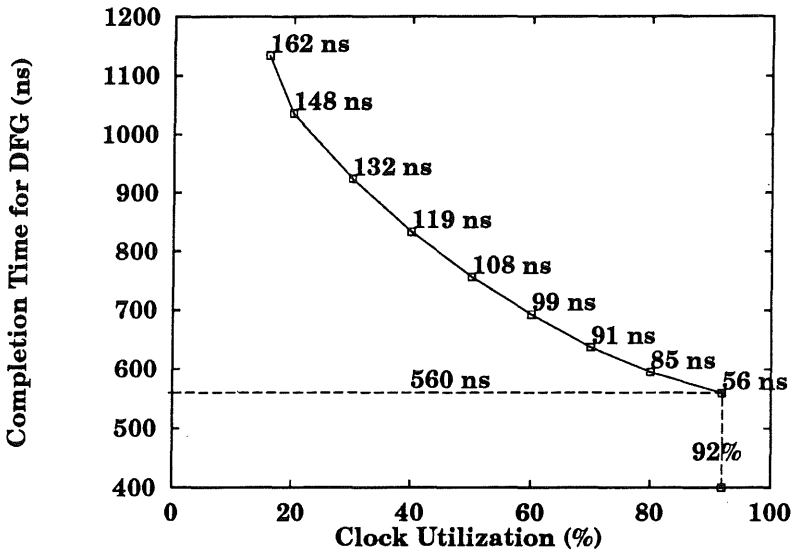


Figure 8: Effect of Clock Utilization on Completion times for the HAL Differential Equation example.

Clock Estimation Method	Allocation 1 (2 add, 2 mult)		Allocation 1 (4 add, 2 mult)		Allocation 1 (5 add, 1 mult)	
	Control Steps	Completion Time	Control Steps	Completion Time	Control Steps	Completion Time
Max. Operator Delay (CLK = 163 ns)	16	2608 ns	14	2282 ns	15	2445 ns
Clock Wastage Minimization (CLK = 24 ns)	49	1176 ns	47	1128 ns	65	1560 ns
Performance Improvement	—	**54.9%**	—	**50.5%**	—	**36.2%**

Faster completion times are obtained if the clock cycle computed by Clock Wastage Minimization method is used, regardless of the final allocation used to implement the design.

Figure 9: Comparing completion times for different functional unit allocations for the Elliptical Filter.

produced by Maximum Operator Delay methods. We must now verify our initial assertion that a clock cycle with a higher utilization will give us a better performance of the DFG.

Figure 8 examines whether the clock wastage minimization criteria that we adopted for selecting a clock cycle is justified. Completion times are plotted against clock utilization values for the HAL Differential Equation example. The points in the graph are labeled with the corresponding clock cycle. For example, a 56 ns clock cycle results in a clock utilization of 92% and the lowest DFG completion time of 560 ns. The figure clearly shows that better performance can be obtained if we use a clock cycle that maximizes the utilization of the functional units.

We now examine if the clock cycle estimated by the Wastage Minimization method will produce faster designs, regardless of the final allocation of functional units used to implement the design. To demonstrate this, we scheduled the DFG for the Fifth Order Digital Elliptical Filter with different allocations for both the clock cycles, (CLK_{mod} and CLK_{wm}) generated by the two estimation methods and compared the completion times of the DFG. This is shown in Figure 9. We observe that for each allocation, the completion time obtained on scheduling with CLK_{wm} is much faster than that obtained by scheduling with CLK_{mod}. Thus we can conclude that the clock estimated by the Clock Wastage Minimization algorithm produces faster completion times (as compared to the one estimated by the Maximum Operator Delay Method) *regardless of the allocation used for finally implementing the design.*

7 Conclusions and Future Work

Traditional high-level synthesis systems require the designer to specify the clock cycle explicitly or express operator delays in terms of multiple of a clock cycle. We have presented an algorithm for clock estimation from dataflow graphs, based on clock wastage minimization. This will provide both designers and synthesis tools with a realistic and useful estimate of the clock cycle that can be used to implement a design. By using real life components and examples, we have shown that that the clock estimates produced by our method yield faster execution times for the designs, as compared to the maximum operator delay methods. We also observe that the designs scheduled with our clock cycle estimates had faster execution times regardless of the components finally allocated for implementing the design during synthesis.

We plan to extend our model to incorporate wire delays in the register to register path. We are currently looking into ways by which it will be possible to derive the lower limit of the range of clock cycles that are examined by the algorithm (CLK_{lower}), based on the power constraints specified for the design.

We plan to further extend our algorithm to handle conditional branching in the VHDL description. In the current implementation, operators are counted exactly once when computing $occur(t_i)$, even though they may be mutually exclusive because they lie on different conditional branches. If the description is divided into statement blocks, and the branch probabilities are known, we can determine the frequency of execution of each statement block. While calculating $occur(t_i)$, the occurrences of operators of type t_i in each statement block is weighed by the frequency of execution of that block.

8 Acknowledgements

This work was supported by the Semiconductor Research Corporation (grant #91-DJ-146). We are grateful for their support. The authors would also like to thank Frank Vahid, Allen Wu and Lognath Ramachandran for their useful suggestions.

9 References

[BaMa89] M. Balakrishnan, and P. Marwedel, "Integrated Scheduling and Binding : A Synthesis approach for design space exploration", Proceeding of the Design Automation Conference, 1989, pages 68-74.

[JaMiPa88] R. Jain, M. Mlinar, and A. Parker, "Area-Time Model for Synthesis of Non-Pipelined Designs", ICCAD 88.

[KuWhKa85] S. Kung, H. Whitehouse, and T. Kailath, "VLSI and Modern Signal Processing", Prentice-Hall 1985, pp. 258-264.

[NaVaGa91] S. Narayan, F. Vahid and D.D. Gajski, "System Level Specification and Synthesis with the SpecCharts Language", International Conference on Computer Aided Design, Santa Clara, November 1991.

[PaFe90] D. Pang, and L. Ferrari, "Unified Approach to General IFIR Filter Design using the B-spline Function", Asilomar Conference on Signals, Systems and Computers, October 1989.

[PaGa87] B. Pangrle, and D. Gajski, "SLICER : A state Synthesizer for Intelligent Silicon Compilation", Proceedings of the International Conference on Computer Aided Design, 1987.

[PaKnGi86] P.G. Paulin, J.P. Knight, and E.F. Girzyc, "HAL : A Multi-Paradigm Approach to Datapath Synthesis", Proceedings of the Design Automation Conference, 1986.

[PaKn89] P.G. Paulin, and J.P. Knight, "Algorithms for High-Level Synthesis", IEEE Design & Test of Computers, December 1989.

[PaPa85] N. Park, and A. C. Parker, "Synthesis of Optimal Clocking Schemes", Proceedings of the Design Automation Conference, 1985

[PaPiMi86] A.C. Parker, T. Pizzaro, and M. Mlinar, "MAHA : A Program for Datapath Synthesis", Proceedings of the Design Automation Conference, 1986, pages 461-466.

[VTI88] VLSI Technologies Inc., "VDP100 1.5 Micron CMOS Datapath Cell Library"

[Wa90] R. Walker, "A Survey of High-Level Synthesis Systems", Report No. 90-30, Rensselaer Polytechnic Institute, October 1990

A SCESTCLK : Manual Page

SCESTCLK(L) LOCAL COMMANDS SCESTCLK(L)

NAME

scestclk - estimate clock cycle for given SpecChart

SYNOPSIS

scestclk [-wmb] [-vd] [-a allocfile] specchart

DESCRIPTION

scestclk estimates the clock cycle for a given SpecChart and an associated allocation file.

The allocation file contains the types of operators in the design and their respective delays. In case the allocation file is not specified using the '-a' option below, the clock cycle estimator will look in the file called ALLOC in the current directory.

The clock cycle estimator can estimate the clock using either the Clock Wastage Minimization method or the Maximum Operato Delay method. In the former, the clock cycle is computed so as to minimize the wastages involved when a particular operator is used in a control step. In the latter method, the largest delay of any operator is computed as the clock cycle.

If no options are specified, the estimator by default produces a non-verbose output using the Clock Wastage Minimization method and uses the file ALLOC.

OPTIONS

-w Estimate the clock cycle using the Wastage Minimization method

-m Estimate the clock cycle using the Maximum Operator Delay method

-b Estimate the clock cycle using both the Wastage Minimization and the Maximum Operator Delay methods

-v Verbose output which lists the number of occurrences of each operator and operator delays.

-d Output the results on three files : ",tmpclkest" which contains the clock cycle estimated, "tmpclkutil" which contains the utilization of the operators for that clock cycle, and ",tmpclkusage" which contains the number of operators of each type and their delays.

-a allocfile
Allocation file containing the operator delays. The default allocation file is set to ALLOC.

FILES

./*.sc	SpecChart files.
./*.scp	SpecChart packages.
,tmpclkutil	File containing the clock cycle utilization when the '-d' flag is used.
,tmpclkest	File containing the clock cycle estimated when the '-d' flag is used.
,tmpclkusage	File containing the number of operators of each type in the design and their delays.

SEE ALSO

xscestclk, scestarea

AUTHOR

Sanjiv Narayan (narayan@ics.uci.edu)

B Benchmarks/Examples used in the Document

B.1 HAL Differential Equation

```
--------------------------------------------------------------------------
-- Differential Equation Example
--
-- Source: Adapted from example in paper
--         "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis"
--          by P. Paulin, J. Knight and E. Girczyc
--          23rd DAC, June 1986, pp. 263-270
-- Benchmark author: Joe Lis
--    Copyright (c) 1989 by Joe Lis
--------------------------------------------------------------------------

entity HAL is
   port (dx: in BIT_VECTOR(0 to 7);
         a: in BIT_VECTOR(0 to 7);
         cntrl: out BIT);
end HAL;

--VSS: design_style BEHAVIORAL

architecture EX of HAL is
begin
   process
      variable x,y,u: BIT_VECTOR(0 to 7) ;
      variable u1,u2,u3,u4,u5,u6,y1: BIT_VECTOR(0 to 7) ;
   begin

      while (x < a) loop
         u1 := u * dx;
         u2 := 5 * x;
         u3 := 3 * y;
         y1 := u * dx;
         x := x + dx;
         u4 := u1 * u2;
         u5 := dx * u3;
         y := y + y1;
         u6 := u - u4;
         u := u6 - u5;
      end loop;

   end process;
end EX;
```

B.2 Fifth Order Digital Elliptic Filter

```
--------------------------------------------------------------------------
-- Fifth Order Digital Elliptic Filter Benchmark
--
-- Source: 1988 High Level Synthesis Workshop Benchmark
--         Example taken from "VLSI and Modern Signal Processing"
--         by S.Y. Kung, H.J. Whitehouse, T.Kailath (eds.)
--         Prentice-Hall 1985, pp. 258-264
-- Benchmark author: Joe Lis
--    Copyright (c) 1989 by Joe Lis
--------------------------------------------------------------------------
entity ELLIPTIC_FILTER is
       port (In_port: in BIT; Out_port: out BIT);
end ELLIPTIC_FILTER;

architecture EX of ELLIPTIC_FILTER is
begin
   process
       variable a,b,c,d,e,f,g,h,i,j,k,o: BIT ;
       variable t2,t13,t18,t33,t39,t26,t38 : BIT ;
       variable m21,m24,m9,m30,m40,m36,m16,m6: BIT ;
   begin
       i := In_port;
       a := i + t2;
       b := a + t13;
       g := t33 + t39;
       e := g + t26 + b;
       d := (m21 * e) + b;
       f := (m24 * e) + g;
       t26 := f + d + e;
       c := m9 * (b + d) + a;
       h := m30 * (f + g) + t39;
       j := t18 + c + d;
       k := t38 + f + h;
       o := m40 * (h + t39);
       t39 := o + h;
       t38 := t38 + (m36 * k);
       t33 := t38 + k;
       t18 := t18 + (m16 * j);
       t13 := t18 + j;
       t2 := c + i + m6 * (a + c);
       Out_port := o;
  end process;
end EX;
```

B.3 AR Lattice Filter

```
--------------------------------------------------------------------------
-- A. R Lattice Filter Example
--
-- Source: 1989  26th Design Automation Conference
--         " Experience with the ADAM Synthesis System"
--         by R.Jain, K. Kucukcar, M. Milnar, A. Parker
--
-- Benchmark author: Sanjiv Narayan
--    Copyright (c) 1991 by  Sanjiv Narayan
--------------------------------------------------------------------------

entity LATTICE_FILTER is
end  LATTICE_FILTER;

--VSS: design_style BEHAVIORAL

architecture EX of LATTICE_FILTER is
begin

process
   variable  i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13,i14:BIT;
   variable  a1, a2, a3, a4, a5, a6, a7, a8 : BIT ;
   variable  b1, b2, b3, b4 : BIT ;
   variable  c1, c2 : BIT ;
   variable  d1, d2, d3, d4 : BIT ;
   variable  e1, e2 : BIT ;
   variable  f1, f2, f3, f4 : BIT ;
   variable  g1, g2 : BIT ;
   variable  o1, o2, o3, o4 : BIT ;

  begin

     a1 := i1 * i2 ;
     a2 := i3 * i4 ;
     a3 := i1 * i4 ;
     a4 := i3 * i2 ;
     a5 := i5 * i6 ;
     a6 := i7 * i8 ;
     a7 := i5 * i8 ;
     a8 := i7 * i6 ;

     b1 := a1 + a2 ;
     b2 := a3 + a4 ;
```

```
      b3 := a5 + a6 ;
      b4 := a7 + a8 ;

      c1 := i9  + b3 ;
      c2 := i10 + b4 ;
      o1 := c1;
      o2 := c2;

      d1 := i11 * c2 ;
      d2 := i12 * c1 ;
      d3 := i11 * c1 ;
      d4 := i12 * c2 ;

      e1 := d1 + d2 ;
      e2 := d3 + d4 ;

      f1 := i13 * e2 ;
      f2 := i14 * e1 ;
      f3 := i13 * e1 ;
      f4 := i14 * e2 ;

      g1 := f1 + f2 ;
      g2 := f3 + f4 ;

      o3 := b1 + g1 ;
      o4 := b2 + g2 ;

   end process;

end EX;
```

B.4 Linear Phase B-Spline Interpolated Filter

```
--------------------------------------------------------------------------
--  Linear Phase B-Spline Interpolated Filter
--
-- Source: Adapted from example in paper  "Unified Approach to General
--         IFIR Filter Design using the B-spline Function"  by
--         D. Pang, and L. Ferrari, Asilomar Conference on Signals,
--         Systems and Computers, October 1989.
--
-- Benchmark author: Joe Lis
--    Copyright (c) 1989 by Joe Lis
--------------------------------------------------------------------------

entity LPBFIR_FILTER is
  port (In_port: in BIT);
end LPBFIR_FILTER;

-- VSS: design_style BEHAVIORAL

architecture EX of LPBFIR_FILTER is
begin

   process
      variable a0,a1,a2,a3,a4,a5,a6,a7,a8,x0,x1,x2,x3: BIT ;
      variable y0,y1,y2,y3,y4,z1,z2,z3,z4: BIT ;
      variable m0,m1,m2,m3,m4: BIT ;
   begin

      a8 := a7 and 1;
      a7 := a6 and 1;
      a6 := a5 and 1;
      a5 := a4 and 1;
      a4 := a3 and 1;
      a3 := a2 and 1;
      a2 := a1 and 1;
      a1 := a0 and 1;
      a0 := In_port and 1;

      x0 := a0 + a8;
      x1 := a1 + a7;
      x2 := a2 + a6;
      x3 := a3 + a5;

      y0 := m0 * x0;
```

```
        y1 := m1 * x1;
        y2 := m2 * x2;
        y3 := m3 * x3;
        y4 := m4 * a4;

        z1 := y0 + y1;
        z2 := z1 + y2;
        z3 := z2 + y3;
        z4 := z3 + y4;

    end process;

end EX;
```

DATE DUE

GAYLORD			PRINTED IN U.S.A.