Clock Optimization for High-Performance Pipelined Design

Hsiao-Ping Juan, Daniel D. Gajski and Smita Bakshi
Department of Information and Computer Science
University of California, Irvine, CA 92717-3425, USA

Abstract

In order to reduce the design cost of pipelined systems, resources may be shared by operations within and across different pipe stages. In order to maximize resource sharing, a crucial decision is the selection of a clock period, since a bad choice can adversely affect the performance and cost of the design. In this paper, we present an algorithm to select a clock period that attempts to minimize design area while satisfying a given throughput constraint. Experimental results on several examples demonstrate the quality of our selection algorithm and the benefit of allowing resource sharing across pipe stages.

1 Introduction

In general, high-performance constraints are met by pipelining the design into several concurrently executing stages, such that the pipe stages operate one after the other on the same sample but at the same time on different samples. In designing such a pipelined system, the current practice is to partition the description under development into stages and then each stage, along with its performance and cost constraints, is given to a different design group. In this scheme, different stages are implemented separately and have their own datapaths and control units. Thus, different stages can use different clock signals, as long as their delays satisfy the throughput constraint.

Clearly, implementing each pipe stage separately would result in a large number of hardware resources, thereby increasing the cost of the design. However, the design cost can be reduced by sharing resources among these stages. For instance, the same functional unit can be utilized to perform several different operations from different pipe stages over different time-steps.

When performing resource sharing, an important decision is the selection of a clock period to schedule the operations into different states. A bad choice of the clock period could adversely affect the performance and cost of the final design. For instance, Figure 1(a) shows a two-stage pipeline with a pipe stage delay (the inverse of throughput) constraint of 120 ns. Figures 1(b), (c) and (d) show scheduling results of the given pipeline using 60, 30 and 20 ns as the clock period respectively. When the clock period is equal to either 60 or 30 ns, the pipe stage delay constraint is satisfied. The operations a and e can share the same multiplier since they are in different clock cycles. Note that the operations b and c are scheduled in the same clock cycle when the clock period is 60 ns, but in different

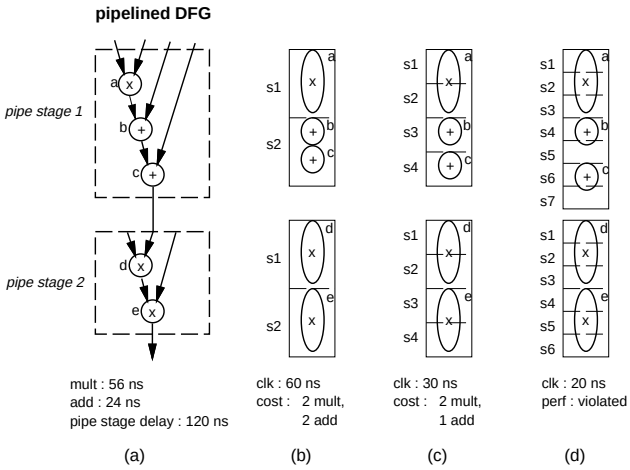

Figure 1: An example illustrating the effect of different clock periods on resource sharing of a pipelined design

clock cycles when the clock period is 30 ns. Consequently, the operation b and c can share one adder when the clock period is 30 ns, but not when the clock period is 60 ns. Thus, the implementation when the clock period is 30 ns requires one less adder than the implementation when the clock period is 60 ns. However, it does not imply that a shorter clock period is always preferable. For example, Figure 1(d) shows that, when the clock period is equal to 20 ns, pipe stage 1 requires seven clock cycles, which results in a delay of 140 ns and violates the pipe stage delay constraint. Using this example, we have shown that the selection of a clock period is a non-trivial problem. In this paper, we propose a clock estimation algorithm that determines the clock period which satisfies the throughput constraint and requires minimum number of resources.

The rest of the paper is organized as follows. In the next section, we discuss previous research done in this area and also explain how we differ from it. We give the problem definition and present the assumed design model in Section 3. The clock selection algorithm is explained in Section 4. Finally, we present experimental results and give conclusions.

2 Previous Work

Several previous papers addressed the issue of clock period estimation for a given data flow graph. For example, there are several clock estimation schemes [4] [7] [8] that use the delay of the slowest component as the estimated

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

clock period. However, using the slowest component delay as the clock period can lead to under-utilized functional units and consequently higher design cost in cases where the components have widely differing delays.

A clock estimation method based on slack minimization is proposed in [6]. This estimation method aims to select the clock period that optimizes the performance of the design. In our problem, performance optimization is not the goal; our goal is to minimize the design cost while satisfying the performance constraint.

In [1], a methodology is proposed to estimate the clock period for time-constrained scheduling as well as resource-constrained scheduling. However, this methodology does not consider pipelined designs, while our algorithm aims to select a clock period for pipelined designs.

Finally, the algorithm presented in this paper differs from all the algorithms mentioned above in that our algorithm also takes the control unit delay into account. When the number of states is very large, the control unit tends to become very complex and control unit delay contributes significantly to the clock period and cannot be neglected. By considering the control unit delay, our algorithm provides a more realistic estimation than the previously published work.

3 Problem Definition and Assumptions

Given (1) a pipeline of n pipe stages $PS_1 \cdots PS_n$, where each pipe stage PS_i is represented by a data flow graph DFG_i, (2) a component library, (3) the pipe stage delay constraint $PSDelay$ and (4) a range of allowable clock periods, represented by $[clkmin, clkmax]$, the goal of our algorithm is to find a clock period clk such that, for all i, DFG_i can be scheduled into $\lfloor PSDelay/clk \rfloor$ states of delay $\leq clk$ and the design area is minimized.

The maximum clock period allowed, $clkmax$, is equal to $PSDelay$. Design libraries often specify the maximum clock frequency at which the clock input of a bistate circuit may be driven such that stable transitions of logic levels are maintained. This frequency is used to determine the value of $clkmin$ if it is not already specified by the user. The cost of a pipeline is approximated by the total area of datapath components.

Our clock estimation algorithm assumes a design model, as shown in Figure 2, similar to the design model used in [6]. In this model, the datapath consists of registers, functional units and tri-state drivers. A two-level bus structure is assumed for the interconnection across the registers and functional units. Note that a register could be used to store a temporary value that is used in different states of the same pipe stage or could be used as a pipeline latch between pipe stages. Operation chaining is supported in this model by allowing connections from the output ports of some functional units directly to the input ports of other functional units. Moreover, operations can execute over several clock cycles; that is, multi-cycled operations are possible.

The control unit consists of the state register, a decoder, the control logic to drive the control lines for the datapath components, and the next-state logic to compute the next

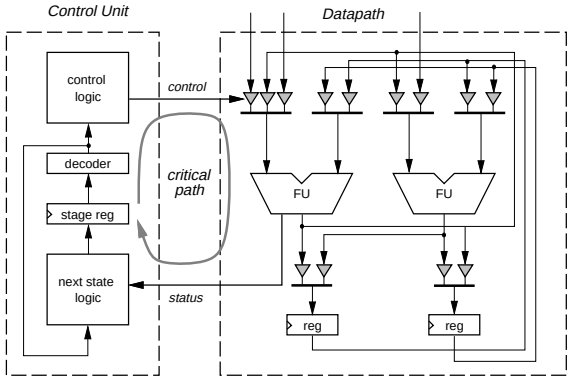

Figure 2: Design model for clock estimation

state to be stored in the state register. The control unit implements a state machine that sequences a design through a series of states, each of the states represents the set of datapath operations performed concurrently in the same or different pipe stages of the design.

The clock period is determined by the longest register-to-register delay. Typically, the path through the control logic, as shown in Figure 2, has the largest delay. Therefore, in our estimation, the minimal clock period is approximated using the sum of all the delays associated with the components in the path, including the datapath and the control unit.

4 Algorithm

Our algorithm selects the clock period in three basic steps.

1. **Pipe stage shape function generation:** The first step in our algorithm is to produce a shape function in terms of clock periods versus the pipe stage delay, individually, for each pipe stage of the description. This shape function can clearly indicate the clocks that can satisfy the pipe stage delay constraint.
2. **Clock candidates selection:** Next, given the pipe stage delay constraint, $PSDelay$, and the shape functions of each pipe stage, a set of clock periods that can satisfy the pipe stage delay constraint in all stages can be easily obtained. These clock periods are called *clock candidates*.
3. **Resource estimation:** Having obtained the set of clock candidates, the final step in our algorithm is to estimate the amount of resources required by each clock candidate. Then the algorithm would return the clock period that requires the least amount of resources.

Details of pipe stage shape function generation and resource estimation will be presented in the following sections.

4.1 Pipe Stage Shape Function Generation

The shape function generation algorithm basically consists of three steps. It first produces a shape function in terms of clock periods versus the minimum number of

```
Procedure: MinClkPeriod
  Inputs: a data flow graph DFG, the number of states N;
  Output: the minimum clock period;
begin Procedure
  Cstep = 1;
  ComputePathLength(DFG);
  MaxPathLength = delay of the longest path in DFG;
  MinClk = MaxPathLength/N;
  InsertReadyOps(DFG, PList);
  while (PList ≠ ∅) do
    if Cstep = N then
      schedule all the non-scheduled operations;
      MinClk = maximum state delay;
      PList = ∅;
    else
      op = First(PList);
      if op is a single-cycled operator then
        determine chaining or non-chaining;
        schedule op and update MinClk;
      else
        determine the number of cycles of op;
        schedule op and update MinClk;
      end if;
      InsertReadyOps(DFG, PList);
      Cstep = Cstep + 1;
    end if;
  end while;
  return MinClk;
end Procedure
```

Figure 3: The procedure to estimate the minimum clock period, given N states

states by considering only the datapath delay. Then the algorithm estimates the control unit delay and updates the shape function accordingly. Finally, the shape function of clock periods versus the pipe stage delay can be computed by multiplying the clock periods by the corresponding number of states.

Given a data flow graph DFG of a pipe stage and the range of allowable clock periods, $[clkmin, clkmax]$, the shape function is generated incrementally by fixing the number of states, and then computing the minimum clock period for the fixed number of states using the procedure $MinClkPeriod$ outlined in Figure 3. This process produces one (clock period, number of states) point in the shape function. To obtain the entire shape function, we iteratively increase the number of states, beginning with the smallest possible number, which is $\lceil PSDelay/clkmax \rceil$, and finishing with the largest possible number, which is $\lfloor PSDelay/clkmin \rfloor$.

Given a data flow graph DFG, the procedure $MinClkPeriod$ first computes the path length for each of the operations in DFG. The path length of an operation is defined as the longest path delay starting from this operation till the output node. Therefore, by definition, the maximum path length, $MaxPathLength$, of all operations in DFG is the critical path length. The variable $MinClk$ is initialized to the optimal clock period $MaxPathLength/N$, where N is the number of states that the DFG will be scheduled into. The next step of the procedure involves determining whether a ready operation can be scheduled and whether chaining or multi-cycling should be performed depending upon its effect on the clock period. The scheduling of an operation may increase the clock period and the variable $MinClk$ is updated if it does. Once an operation is scheduled, other non-ready operations become ready and these are then inserted into the ready list. This process continues and when it reaches the last state, all the non-scheduled operations are scheduled into the last state and the procedure returns the variable $MinClk$. Clearly, the result of this algorithm depends upon how chaining and multi-cycling is performed. We now illustrate how chaining and multi-cycling are determined on the example in Figure 4.

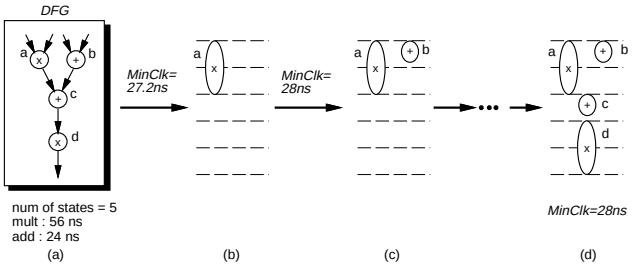

Figure 4: Determining the minimum clock period

Given that a multiplication operation takes 56 ns and an addition takes 24 ns, the procedure computes that the maximum path length is 136 ns. Since the data flow graph needs to be scheduled into five states, the optimal clock period, that is, the current $MinClk$, is 136/5=27.2 ns. In the first iteration, the procedure attempts to schedule operation a and needs to determine whether to schedule it across $\lfloor 56/27.2 \rfloor$=2 states or $\lceil 56/27.2 \rceil$=3 states. If a is scheduled across two states, the average delay of the first two states is 56/2=28 ns, and operations c and d could be scheduled across three states, which results in an estimated delay per state of (24+56)/3=26.7 ns. Thus, the clock period is 28 ns. If a is scheduled across three states, then the average state delay is 18.7 ns for the first three states. However, operations c and d now need to be finished within two states, which gives an estimated delay per state of (24+56)/2=40 ns. That is, the clock period in this case is 40 ns. Since scheduling the operation a into a two-cycled operation gives an estimation of shorter clock period, the procedure decides to schedule a across the first two states as shown in Figure 4(b), and the clock period $MinClk$ is updated to 28 ns.

Next, the operation b is scheduled. Since the delay of the operation b is less than 28 ns, it is a single-cycled operation and its scheduling does not change the current clock period. The result of this iteration is shown in Figure 4(c). The procedure continues this process for operations c and d, and the final result is shown in Figure 4(d). The algorithm obtains a minimum clock period of 28 ns.

Similarly, we can estimate that the minimum clock periods for scheduling the data flow graph in Figure 4(a) into one, two, three, or four states are 136 ns, 80 ns, 56 ns and 56 ns respectively. Therefore, we can conclude that for any clock period larger than 136 ns, the minimum number of

states the DFG requires is one; for any clock period between 136 and 80 ns, the minimum number of states is two, etc. Figure 5 shows the resulting shape function.

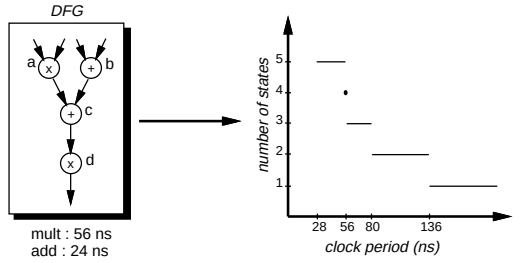

Figure 5: The shape function of clock periods versus number of states

Thus far, the shape function does not incorporate control unit delays. As illustrated in Figure 2, the control unit consists of a state register, a decoder, the control logic and the next-state logic. Therefore, given the number of states N, the delay of an N-state control unit, $T_{CU}(N)$, is estimated as the sum of the decoder delay (T_{DEC}), the control logic delay (T_{CL}), the next-state logic delay (T_{NS}), and the propagation delay and the setup time of the state register. The propagation delay and the setup time of the state register can be obtained from the component library. The following equations are used to estimate T_{DEC}, T_{CL}, and T_{NS}.

$$\begin{aligned} T_{DEC} &= T_{INV} + \lceil \log_M \log_2 N \rceil \times T_{AND} \\ T_{CL} &= \lceil \log_M N \rceil \times T_{OR} \\ T_{NS} &= \lceil \log_M (N/2) \rceil \times T_{OR} \end{aligned}$$

For lack of space, we will not explain how these equations have been derived; a detailed discussion is provided in [5].

Given a point (clk_i, N) in the shape function, the algorithm next updates the point (clk_i, N) to $(clk_i + T_{CU}(N), N)$. Note that given two points (clk_i, N) and $(clk_{i+1}, N+1)$, where $clk_i \leq clk_{i+1}$, it is possible that $clk_i + T_{CU}(N) \geq clk_{i+1} + T_{CU}(N+1)$. In this case, the algorithm drops the point $(clk_i + T_{CU}(N), N)$.

After the shape function of clock periods versus the number of states is updated, the shape function of clock periods versus the pipe stage delay can be obtained by multiplying the clock periods by the corresponding number of states.

4.2 Resource Estimation

From the shape functions for each pipe stage, the set of clock candidates can be easily obtained. The next step of our algorithm is to estimate the number and type of resources required for each clock candidate.

An example to illustrate the algorithm is shown in Figure 6(a). We know that all pipe stages are executed concurrently and in order to consider resource sharing across the stages, the algorithm needs to consider the operations in all stages at the same time. In order to demonstrate this, we put the $DFGs$ from two pipe stages, DFG_1 and DFG_2, side by side in Figure 6. Note that these pipe stages are executed in parallel but on different input samples.

Given the clock period and the number of states, the first step of the algorithm is to compute the time frame of each

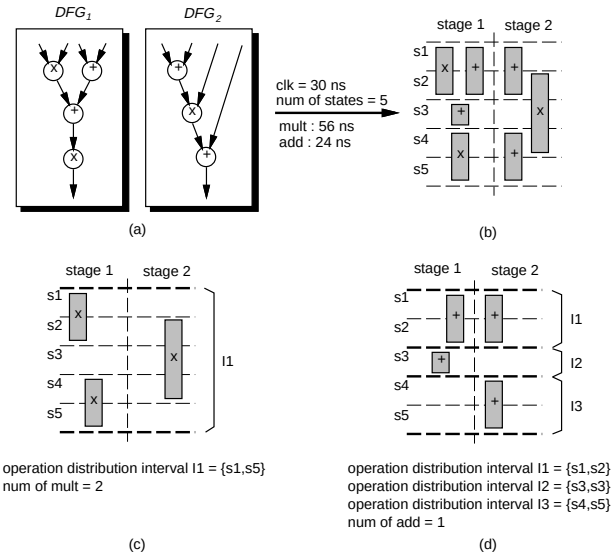

Figure 6: An example illustrating the resource estimation algorithm

operation. Let $ASAP_i$ and $ALAP_i$ denote the ASAP and ALAP value of operation o_i respectively, the time frame of o_i is defined as $(ALAP_i - ASAP_i + cycle(o_i))$, where $cycle(o_i)$ represents the number of clock cycles required to finish the operation o_i. Figure 6(b) shows the time frames of all the operations in Figure 6(a).

The next step is to partition states into a set of disjoint *operation distribution intervals* such that there are no overlapping time frames between two consecutive intervals. For example, in Figure 6(c), there is no way of partitioning the five states into intervals such that there are no overlapping time frames of the multiplication operations; therefore, there is only one operation distribution interval, {s1,s5}, for multiplication operations, where s1 is the starting state and s5 is the ending state of the interval. On the other hand, there are three operation distribution intervals for additions. After the operation intervals are obtained, the algorithm estimates the required number of components for each interval separately, and the maximum number of required components over all intervals is the minimum number of components needed to perform all the operations.

The underlying concept of determining the minimum number of components for one distribution interval is that, if there are n operations that need to be finished within s states, and a component used to perform an operation requires at least c clock cycles to finish the execution before it can be used again to execute another operation, then clearly, the minimum number of components required is equal to $\lceil (n \times c)/s \rceil$. For example, for the multiplication operations shown in Figure 6(c), $n = 3$, $c = 2$ and $s = 5$, hence, the minimum number of multipliers required is $\lceil (3 \times 2)/5 \rceil$, that is, at least two multipliers are needed. Similarly, from Figure 6(d), it can be estimated that at least one adder is required.

5 Experiments

In this section, we present results of three experiments with the clock estimation algorithm which we have implemented using C on a SUN SPARC 5 station. In the first experiment, we demonstrate the quality of our algorithm by comparing the selected clock against the "best" clock obtainable using force-directed scheduling. The second experiment studies the impact of resource sharing across different pipe stages on the cost of a design, and finally, the third experiment demonstrates the effect of considering control unit delay on the clock selection.

For all experiments we have used the VLSI Technology Inc. VDP370 1.0 micron Datapath Element Library [9] to obtain the area and delays of the functional units. The datapath elements used are shown in Figure 7.

component	delay(ns)	area(x1000 um*2)
adder	11.2	54
subtractor	15.5	60
multiplier	32.0	320

Figure 7: Datapath component library

5.1 Experiment 1: Quality of Results

As discussed in Section 2, there are no existing clock selection algorithms for pipelined designs; furthermore, the existing clock selection algorithms do not take control unit delay into account. Thus, in order to demonstrate the quality of our algorithm, we have been unable to compare our results with related research in clock selection; instead, we have utilized force-directed scheduling, which is a well known time-constrained scheduling algorithm.

This experiment is conducted on four examples: the AR lattice filter (AR) [4], the linear phase b-spline interpolated filter (BSpline) [6], the elliptical filter (EF) [2] and the HAL benchmark [2]. For each of the examples, we first generate a number of input descriptions by manually pipelining the specification into a different number of stages, where the delay of the pipe stages in each pipeline is as equal as possible. We then place different pipe stage delay constraints on each of the pipelined descriptions, and for a given pipe stage delay constraint we obtain the estimated and the "best" clock period. The estimated clock period is obtained by executing our clock-selection algorithm. The best clock period is obtained by executing the force-directed scheduling algorithm for a number of clock periods, each corresponding to a different number of states (from one state to fifteen states). The clock period that gives the minimal area design is then the best period.

The results of comparing the best and the estimated clock period for the four examples mentioned above are shown in Figure 8. The last column of the table shows the percentage difference in design area, which is approximated by the sum of the areas of all the components, obtained by the force-directed scheduling algorithm and by our clock-selection algorithm. As can be seen from the results, the estimated clock period was identical to the one obtained with FDS in most cases; however, in three cases our algorithm estimated a clock period that resulted in the use

Examples	# of stages	PSDelay (ns)	FDS		ours		res. diff.
			clk(ns)	resources	clk(ns)	resources	(%)
AR	2	150	16.6	4A,5M	16.6	4A,5M	0
	3	100	16.6	6A,8M	16.6	6A,8M	0
BSpline	2	150	21.4	2A,2M	18.75	2A,2M	0
	3	100	33.3	3A,2M	12.5	2A,3M	33.2
EF	2	300	27.2	3A,2M	33.3	4A,2M	6.7
	3	200	33.3	5A,2M	22.2	5A,3M	35.2
	4	150	16.6	5A,2M	16.6	5A,2M	0
HAL	2	150	50	1A,1S,2M	50	1A,1S,2M	0

A: adder, S: subtractor, M: multiplier

Figure 8: Comparing the best and the estimated clock period for four benchmarks: AR, BSpline, EF, and HAL

of either one more multiplier or one more adder than that obtained with FDS.

This discrepancy between the estimated and the best clock period may be explained by considering the fidelity of our resource estimation method, which essentially gives a lower bound on the number of resources. It is important to note that the correct selection of the clock depends more on the fidelity rather than on the accuracy of the resource estimation. In order to illustrate the role of fidelity of our resource estimates, we compared the results of our resource estimates with the resources obtained by the force-directed scheduling algorithm for all examples and $PSDelay$ constraints shown in Figure 8. Due to space limitations, we give only the results for the 3-stage AR and EF examples in Figure 9. From the results of comparison, we conclude that when fidelity is high, our clock selection algorithm selected the best clock - in spite of the fact that the accuracy of the estimation is low in some cases, such as in the 3-stage AR design. When fidelity is low, our clock selection algorithm selected the wrong clock, even though the accuracy may be high, such as in the 3-stage EF design.

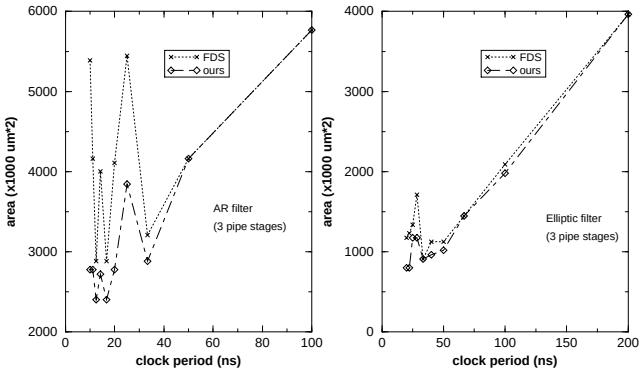

Figure 9: Comparing our resource estimates against the results of the FDS algorithm for the 3-stage AR and EF examples

From the results it may appear that the FDS approach is superior than our approach; however, we would like to point out that in the case of the elliptical filter example, whereas it took approximately 1 second to estimate and select the clock period for a given pipe stage delay constraint using our algorithm, it took more than 17 minutes to obtain the best clock period using the FDS algorithm since it had to be iterated over approximately fifteen different clock periods.

5.2 Experiment 2: Resource Sharing

This experiment is conducted on the same examples that were used in the previous section. For each description and constraint, we compare the minimum number of resources obtained by implementing all the pipe stages individually (that is, by dis-allowing resource-sharing across different pipe stages) to that obtained by implementing all the pipe stages together and thus allowing resource sharing across different pipe stages.

The minimum cost of a design without resource sharing is computed by first obtaining the best clock period and the minimum number of resources required for each pipe stage separately using force-directed scheduling, and then summing up the resources of all the pipe stages. To compute the minimum number of resources required with sharing, we first select a clock period by applying our algorithm to the pipelined descriptions and then generate the minimum number of resources required using force-directed scheduling.

Examples	# of stg.	PSDelay (ns)	without sharing		with sharing		imprv.
			clk(ns)	resources	clk(ns)	resources	(%)
AR	2	150	18.75	**6A,7M**	16.6	**4A,5M**	**29.2**
	3	100	12.5,50	**8A**,8M	16.6	**6A**,8M	**3.6**
BSpline	2	150	18.75,50	**3A**,2M	18.75	**2A**,2M	**6.7**
	3	100	20,50	**4A**,3M	12.5	**2A**,3M	**9.2**
EF	2	300	25,37.5	**5A**,2M	33.3	**4A**,2M	**5.9**
	3	200	20,25,50	**7A**,3M	22.2	**5A**,3M	**8.1**
	4	150	18.75,30,50	**6A**,4M	16.6	**5A**,2M	**43.3**
HAL	2	150	37.5,75	**2A**,1S,2M	50	**1A**,1S,2M	**6.7**

A: adder, S: subtractor, M: multiplier

Figure 10: The effects of resource sharing on four benchmarks AR, BSpline, EF, and HAL

The results on the four examples are shown in Figure 10. Note that when each stage is implemented separately, in some cases, more than one clock signal is used because different pipe stages can use different clock signals. In all the cases, the results indicate that resource sharing within and across different pipe stages reduces the design area from anywhere between 3.6 and 43.3 %. This shows a substantial reduction in area when resource-sharing across different pipe stages is allowed and it also indicates the effectiveness of our algorithm.

5.3 Experiment 3: Control Unit Delay

This experiment is conducted for the AR filter and the elliptical filter benchmarks. Figure 11 shows the result of the elliptical filter benchmark. There are two shape functions of clock periods versus total delay. The shape function in solid line is obtained by our shape function generation algorithm with the control unit delay estimation, while the shape function in dashed line is generated by our algorithm, but assuming the control unit delay is zero.

From the results, we observe that the difference between delays obtained with and without considering the control unit delay becomes larger when the clock period becomes smaller. Note that the difference can be as large as 720 ns. Therefore, we conclude that the control unit delay contributes significantly in the clock period and neglecting the control unit delay may result in a bad choice of the clock

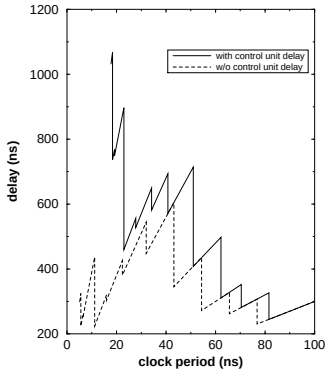

Figure 11: The clock period vs. delay shape functions of the EF example, generated with and without the control unit delay estimation

period. Same conclusion can be reached for the AR filter benchmark [5].

6 Conclusions and Future Work

In summary, we have presented a clock selection algorithm that, given a pipelined behavioral description and a throughput constraint, selects the clock period leading to the minimal-area design. We tested our clock-selection algorithm on several examples and the results show that, in most cases, our algorithm selects a clock period that uses minimal area resources within less than one second. We plan to extend our model to incorporate wire delays. Currently, we are working on a clock selection algorithm that allows multiple clock signals.

7 Acknowledgements

This work was supported by the Semiconducter Research Corporation (grant #94-DJ-146), and by the National Science Foundation (grant CDA-9422095). We gratefully acknowledge their support.

8 References

[1] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "An Exact Methodology for Scheduling in a 3D Design Space," in *Proc. 8th ISSS,* 1995.

[2] N. D. Dutt, and C. Ramachandran, "Benchmarks for the 1992 High-Level Synthesis workshop," *TR#92-107, Dept. of ICS, UCI,* 1992.

[3] D. D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design,* Kluwer Academic Publishers, 1992.

[4] R. Jain, A. C. Parker, and N. Park, "Module Selection for Pipelined Synthesis," in *Proc. 25th DAC,* 1988.

[5] H.-P. Juan, D. D. Gajski, and S. Bakshi, "Clock Optimization for High-Performance Pipelined Design," *TR#96-01, Dept. of ICS, UCI,* 1995.

[6] S. Narayan, and D. D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in *Proc. EuroDAC,* 1992.

[7] N. Park, and A. C. Parker, "Synthesis of Optimal Clocking Schemes," in *Proc. 22nd DAC,* 1985.

[8] A. C. Parker, T. Pizzaro, and M. Mlinar, "MAHA: A Program for Datapath Synthesis," in *Proc. 23th DAC,* 1986.

[9] VLSI Technology Inc., *VDP370 1.0 Micron CMOS Datapath Cell Library,* 1991.