An Optimal Clock Period Selection Method Based on Slack Minimization Criteria

EN-SHOU CHANG and DANIEL D. GAJSKI
University of California at Irvine
SANJIV NARAYAN
Viewlogic Systems, Inc.

An important decision in synthesizing a hardware implementation from a behavioral description is selecting the clock period to schedule the datapath operations into control steps. Prior to scheduling, most existing behavioral synthesis systems either require the designer to specify the clock period explicitly or require that the delays of the operators used in the design be specified in multiples of the clock period. An unfavorable choice of clock period could result in operations being idle for a large portion of the clock period and, consequently, affect the performance of the synthesized design. In this article, we demonstrate the effect of clock slack on the performance of designs and present an algorithm to find a slack-minimal clock period. We prove the optimality of our method and apply it to several examples to demonstrate its effectiveness in maximizing design performance.

Categories and Subject Descriptors: B.5.2 [**Hardware**]: Register-Transfer-Level Implementation

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Clock period, clock slack, performance estimation, scheduling

1. INTRODUCTION

In recent years, logic synthesis has come to be recognized as an integral part of the design process, and this recognition has led to an evolutionary change in design methodology into a *describe-and-synthesize* [Gajski et al. 1994, 1991] approach. The advantage of this new methodology is that it allows us to specify a design in a purely behavioral form, devoid of any implementation details. For example, we can describe a design using

Authors' addresses: E. S. Chang and D. J. Gajski, University of California at Irvine, Department of Information and Computer Science, Irvine, CA 92717; S. Narayan, Viewlogic Systems, Inc., 293 Boston Post Rd. West, Marlboro, MA 01752.

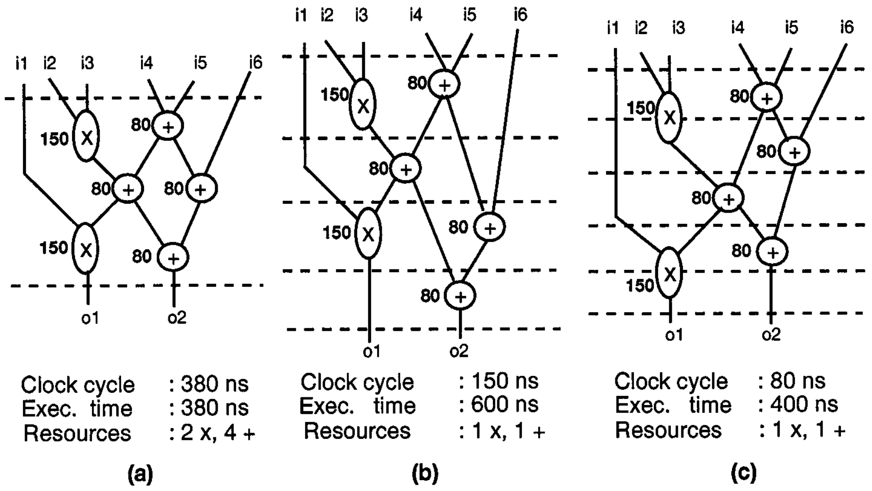


Fig. 1. Effect of clock period on execution time and resources required.

Boolean equations, finite-state, and the like. An implementation for the design can be generated by automatic synthesis tools, instead of manual design, which is usually tedious.

Behavioral synthesis involves the transformation of a design specification into a set of interconnected *RT-components* [Gajski et al. 1991] that satisfy the behavior and some specified constraints, such as the number of functional units, performance, and so on. Three major synthesis tasks are applied during the transformation [Gajski et al. 1991]: *allocation, scheduling,* and *binding.* The purpose of allocation is to determine the number of resources, such as registers, buses, and functional units, that will be used in the implementation. The task of scheduling is intended to partition the behavioral description into time intervals, called *control steps.* During each control step, which is usually one clock-cycle long, data will be fetched from a register, transformed by a functional unit, and written back to a register. All register transfers in any given control step will be executed concurrently. The binding task assigns variables to storage units and operations to functional units, as well as insures that there is a distinct communication path or bus assigned for each transfer of data between the storage and functional units.

Another major task in behavioral synthesis is the selection of the clock period that will be used for implementing the design. Selecting the clock period before performing synthesis tasks is important because the choice of clock period can affect the execution time and the resources required to implement the design. For example, consider using three different clock periods (380 ns, 150 ns, and 80 ns) for implementing the dataflow graph in Figure 1. In Figure 1(a), clock period 380 ns allows the fastest possible execution time but requires two multipliers and four adders to implement the design (the multiplier and adder have a delay of 150 ns and 80 ns, respectively). On the other hand, a clock period of 150 ns in Figure 1(b) requires only one adder and one multiplier, but results in a total execution time of 600 ns. The most efficient implementation, in terms of performance per resource, is obtained with an 80 ns clock period, as shown in Figure 1(c). Its execution time is comparable to that of the first implementation, and it requires the same minimal number of resources required by the second implementation.

In most synthesis tools [Balakrishnan and Marwedel 1989; Paulin et al. 1986; Paulin and Knight 1989; Walker and Camposano 1991; McFarland and Kowalski 1990], the clock period must be specified by the designer before synthesis—either the clock period is specified explicitly or the delays of components are expressed in multiples of the clock period. A designer-specified clock period is applicable when the design is developed as part of a larger system. In this case, the clock period used for some of the standard components in the system is known and can be used for the remainder of the design. In some other cases, where the clock period is not specified, clock period selection assumes great importance. It is evident from Figure 1 that the choice of a particular clock period has a strong impact on the quality of the design both in terms of hardware size and its performance. Thus it is essential that clock period selection become an integral part of synthesis tools, which should provide the designer with feedback as to how various clock periods could possibly affect the design quality.

Some synthesis tools [Parker et al. 1986; Park and Parker 1985; Jain et al. 1988] equate the clock period with the delay of the slowest functional unit in the design. However, this scheme leads to underutilization of the faster functional units. The reason for low utilization of faster units is that in the presence of a slower functional unit such as a multiplier, which has a large delay, the clock period will be at least as long as the multiplier delay, and faster functional units implementing other operations (such as an addition) will be idle for a significant portion of the clock cycle. Hence longer execution times can be expected for a design using the maximum-operator-delay clock.

In order to improve the performance of the design, we need to minimize the time that operations are idle (i.e., slack time) in any given control step. In this article we present a method to compute the clock period for implementing a given behavioral description with a view to eliminating or minimizing the idle time associated with datapath operations.

An overview of existing approaches for estimating the clock period is presented in Section 2. We explain our design model in Section 3. Section 4 presents a formulation of the clock estimation problem. We formally prove some properties of the zero-slack clock period in relation to the delays of the functional units that are used in the design in Section 5. In cases where it is not possible to entirely eliminate the slack associated with design operations, we present an algorithm to determine a slack-minimal clock period. Idle time of the functional units is discussed in Section 6. Experimental results on several benchmarks are shown in Section 7, and we present our conclusions in Section 8.

2. PREVIOUS WORK

A few synthesis tools [Parker et al. 1986; Park and Parker 1985; Jain et al. 1988] have incorporated clock estimation techniques that are used to either examine area-time tradeoff in the design or to guide synthesis tasks such as scheduling.

In MAHA [Parker et al. 1986], the critical path in the dataflow graph is determined first. The maximum delay of any operator in the critical path is chosen as the clock period.

The clocking scheme proposed in Park and Parker [1985] computes a lower bound for the clock period of a multistage system to be the longest stage time. Because the longest stage time is at least as long as the longest operator delay, this scheme computes a clock period greater than or equal to the longest operator delay.

A model for area-time estimation is presented in Jain et al. [1988]. 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.

$$CLK = MAX\begin{bmatrix} Critical\ Path\ Delay / Time\ Steps, \\ MAX[Operator\ Delay] \end{bmatrix}. \tag{1}$$

Each of the preceding approaches assumes that each operation must be executed within one clock cycle. Multicycle 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 period calculated by each of the preceding methods is at least as long as the largest operator delay. We refer to these clock estimation methods as the *maximum-operator-delay* methods. The advantage of these methods is that they are simple to implement and their algorithmic complexity is linear with respect to the number of different operation types that are used to implement the design.

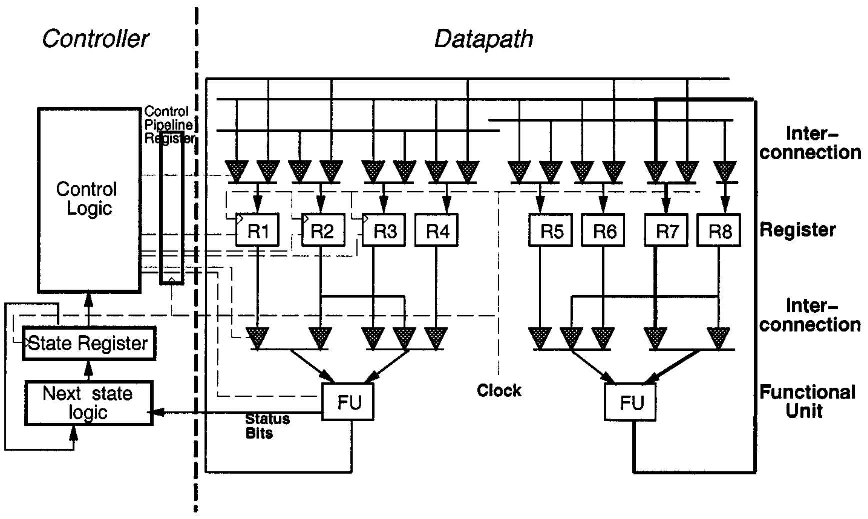
Let the clock period computed by these methods be denoted by CLK_{MOD}. Let $Delay(oper_i)$ denote the delay of operation type $oper_i$. From the preceding analysis of maximum-operator-delay methods,

$$CLK_{MOD} \geq MAX[Delay(oper_i)], \quad for\ all\ operator\ types\ t_i. \tag{2}$$

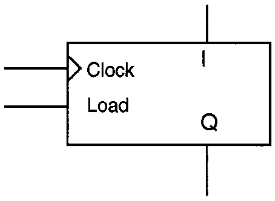
However, using the maximum-operator-delay clock period will lead to underutilization of the functional units in cases where the delays differ widely. Consequently, the performance of the design (start to finish execution time) is slower than cases where the idle time is somehow minimized. Incorporating the effect of clock period on the idle times of operations, during clock selection, forms the main motivation of the slack minimization method presented in this article.

3. DESIGN MODEL

The design model on which the clock calculation is based is the *Finite-State Machine with Datapath* (*FSMD*) [Gajski et al. 1991]. Figure 2(a) shows the

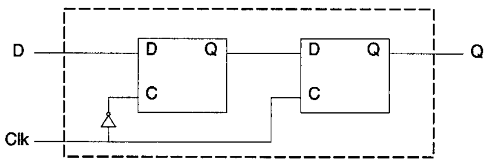


(a)



inputs			output
Clock	Load	I	Q
↗	H	L	L
↗	H	H	H
↗	L	X	Q
L	L	X	Q
H	L	X	Q

(b)



(c)

Fig. 2. (a) Design model for clock calculation; (b) graphic symbol and function table for a register; (c) master-slave flip-flop.

design model. Thin dash lines in the figure indicate part of the control lines. Some of them are omitted for simplicity.

A FSMD is comprised of a controller and a datapath. The controller is a finite-state machine that contains a State Register, a Control Pipeline Register, and two combinatorial logic blocks, namely, the Next-State Logic and the Control (Output) Logic. The Control Logic drives the control lines for the datapath components, for example, functional units, registers, MUXes, and the like. The Next-State Logic computes the next state that will be stored in the State Register. The State Register and the Control Pipeline Register are master-slave type flip-flops [Gajski 1996; VLSI Technologies Inc. 1992], as illustrated in Figure 2(c). Thus the State Register can retain current state for the combinatorial logic blocks and the Control Pipeline Register can retain control signals for the datapath components through the entire clock.

The datapath is comprised of registers; functional units (FU), such as adders, subtracters, and multipliers; and interconnections, such as selectors or buses. A register in the datapath has a load-enable control, as shown in Figure 2(b). In Figure 2(a), the load-enable control for each register is individually controlled by the controller. Only a one-phase clock is required in the design model. Each register is not only controlled by the clock signal but also controlled by the load-enable signal. Thus the register is controlled by the controller to store the data present on a demanded clock and will retain the data until it is controlled to store the data present on another clock. It is not necessary to store data present on every clock.

For example, if the left FU is going to output the result to register R1 and the right FU is going to output the result to register R3, in addition to properly switching those MUXes in front of the registers, the controller will switch on load-enable controls for register R1 and register R3 and keep off load-enable controls for other registers. Thus register R1 and register R3 will input their new data, and the data stored in other registers will not be changed.

The bold solid lines in Figure 2(a) show the dataflow for a typical operation. A typical operation reads the operands from the registers, computes the result in the functional units, and finally writes the result into the destination register. Therefore the delay $delay(oper_i)$ associated with operation type $oper_i$ is the following,

$$delay(oper_i) = T_{reg} + OpDelay(oper_i) + T_{interconnection}, \tag{3}$$

where T_{reg} is the delay for reading data from a register and writing data to a register, $OpDelay(oper_i)$ is the delay for the functional unit of type $oper_i$, and $T_{interconnection}$ is the delay for the interconnection. For a MUX-based design, the $T_{interconnection}$ includes delay for two selectors; for a bus-based design, the $T_{interconnection}$ includes the total delay for two tristate drivers and two selectors.

Operations can be executed over several clock cycles. For example, when an operation with delay of 90 ns is executed and the clock period is 50 ns, the correspondent source registers will continue to output the data stored for two clock cycles to meet the time requirement for the operation, namely, to allow sufficient time for computing the result and storing the result in the destination register. Thus it would take two clock cycles to execute the operation.

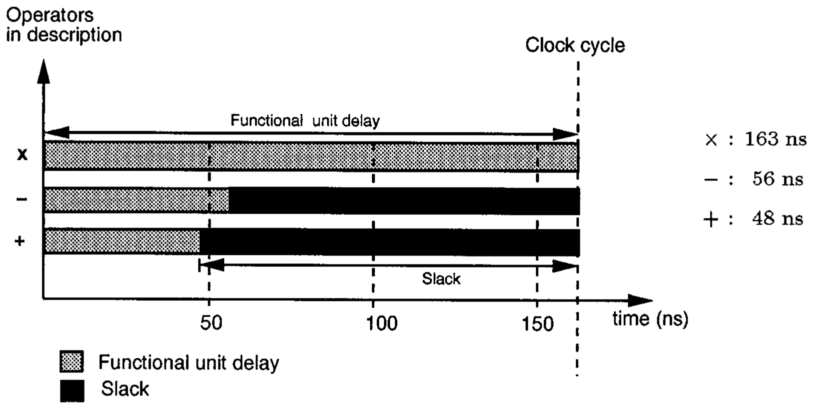


Fig. 3. Functional unit clock slack with clock period 163 ns.

4. THE SLACK MINIMIZATION CRITERIA

In this section, we present a new approach to guiding clock period selection based on *slack minimization criteria*. We first define a few terms that are used frequently throughout this article.

DFG completion time. It represents the execution time of a DFG (Data Flow Graph). If the DFG is scheduled into C control steps with a clock period clk, then the completion time of the DFG, T_{DFG}, is defined as:

$$T_{DFG} = C \times clk. \tag{4}$$

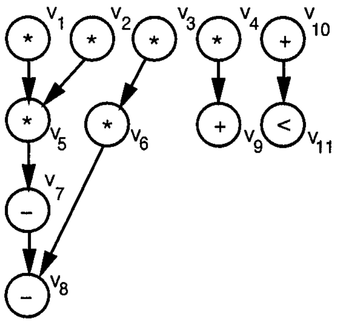
Operator occurrences. This represents the number of occurrences of an operation type $oper_i$ in a given behavioral description or its corresponding DFG, and is denoted by $occur(oper_i)$.

Clock slack [Gajski et al. 1994]. For a given clock period, the clock slack associated with an operation (or its corresponding functional unit) is defined as the difference between the operation delay and the next higher multiple of the clock cycle. In other words, the clock slack is equivalent to the time that the functional unit would be idle if it were scheduled into a control step. For a given clock period clk and operation type $oper_i$, the clock slack, denoted by $slack(clk, oper_i)$, is computed using the following equation,

$$slack(clk, oper_i) = (\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i). \tag{5}$$

Figure 3 shows the clock slack associated with three types of operations. In this example, the clock period is determined by the maximum-operator-delay method. The lightly shaded regions represent the delays of three operation types. The clock slacks are represented by the dark regions.

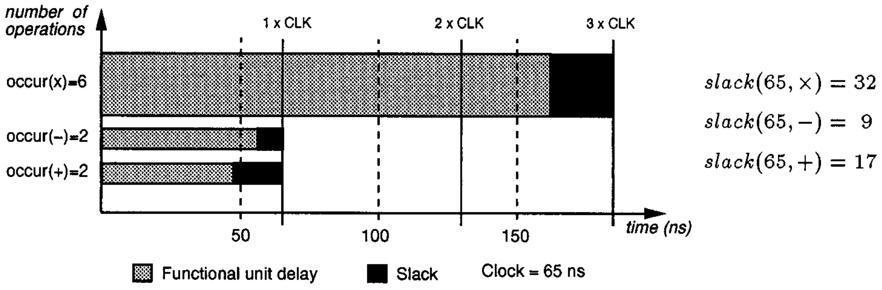
Average slack. For a given clock period clk, the average slack, denoted by *ave_slack(clk)*, is defined as the average clock slack of each operation in



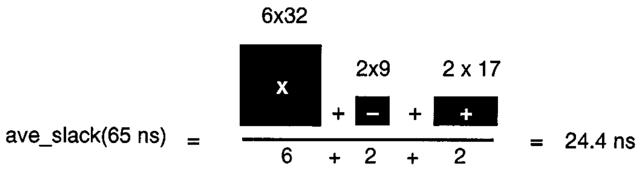
(a)

operation type	occurrences	delay
add	2	48 ns
subtract	2	56 ns
multiply	6	163 ns

(b)



(c)



(d)

Fig. 4. Clock slack and average slack for HAL differential equation with clock period 65 ns: (a) dataflow graph; (b) occurrences and delays of each operation type; (c) clock slack; (d) average slack.

the design. Let *occur*($oper_i$) represent the number of occurrences of operation type $oper_i$ in the design, then the average slack is defined as:

$$ave\_slack(clk) = \frac{\sum_i \{occur(oper_i) \times slack(clk,\ oper_i)\}}{\sum_i occur(oper_i)}. \tag{6}$$

Let us consider the example shown in Figure 4, which graphically depicts the clock slack associated with the different operations in HAL [Paulin et al. 1986], a second-order differential equation example. The components used are from the VDP100 datapath library [VLSI Technologies Inc. 1988]. Figure 4(a) shows the dataflow graph for this design. Figure 4(b) shows the occurrences and the delays of each operation type. In Figure 4(c), the delay of each operation type is shown graphically as the length of the lightly shaded regions along the X-axis. The number of occurrences of the operations in the behavior is the height of the shaded region along the Y-axis. The dark shaded regions represent the clock slack for each operation type. The average slack for a design of clock period 65 ns is 24.4 ns, graphically shown in Figure 4(d).

We now formulate the *Slack Minimization* problem. The main objective of the slack minimization problem is to minimize the clock slack in each clock cycle with the assumption that a smaller clock slack on the average will decrease the execution time of the given behavior. The clock period that produces minimum average slack, within a certain clock range, is selected as the slack-minimal clock period. Thus the problem is defined as follows.

$$\begin{array}{c}\textbf{Find } clk \geq clkmin \text{ that} \\ \textbf{Minimizes } ave\_slack(clk),\end{array} \tag{7}$$

where *clkmin* is the lower bound of the clock period. The designer or physical restrictions determine *clkmin*. For example, a component library usually specifies the shortest clock period at which the clock input of a bistate circuit may be driven with stable transitions of logic levels.

5. COMPUTING A ZERO-SLACK CLOCK

Obviously, "zero slack" is a lower bound of the clock slack. In this section, we introduce two extended definitions of *common divisor* and *greatest common divisor* (*GCD*) and then demonstrate how a clock period is selected to obtain zero slack.

Definition 5.1. A *common divisor* in the domain of real numbers is defined as:

A real number r is a common divisor of a set of real numbers $\{t_1, t_2, \ldots, t_n\}$
$\equiv$ positive integer $k_i \ni rk_i = t_i, \quad i.$

Definition 5.2. A *greatest common divisor* (GCD) in the domain of real numbers is defined as:

GCD: set of $R \rightarrow R$
$GCD\{t_1, t_2, \ldots, t_n\} \equiv$
the largest common divisor associated with $\{t_1, t_2, \ldots, t_n\}$.

Whenever we select a common divisor for the delays of all the operation types used in the design to be the system clock period, it is directly induced from Definition 5.1 that all the operations can be executed completely in one or more clock cycles without any clock slack. We prove this property formally in the following.

Theorem 5.1. *A given clock period causes no clock slacks for a given set of operations $\{oper_1, oper_2, \ldots, oper_n\}$ if and only if the clock period is a common divisor of the delay time $\{t_1, t_2, \ldots, t_n\}$ of those operations.*

Proof. First, we prove the necessity. Let the delays of the operation types be $\{t_1, t_2, \ldots, t_n\}$ and the given clock period be a common divisor r of them. According to Definition 5.1, there are corresponding positive integers $\{k_1, k_2, \ldots, k_i\}$ such that $rk_i = t_i$ for all i; that is, an operation of type $oper_i$ can be executed exactly in k_i clock cycles with no slack.

Second, we prove the sufficiency. Assume that the delays of the operation types are $\{t_1, t_2, \ldots, t_n\}$ and that there are corresponding positive integers $\{k_1, k_2, \ldots, k_i\}$ such that an operation of type $oper_i$ can be executed exactly in k_i clock cycles with no slack, then the clock period r satisfies Definition 5.1: positive integer $k_i \ni rk_i = t_i$, i. □

Because a clock period that causes zero-slack should be a common divisor of the delays of those operation types, it is trivial to infer the next theorem.

Theorem 5.2. *The longest clock period that causes no clock slack for a given set of operations is the GCD of the delays of these operations.*

From Theorem 5.2, we can see that if we select the GCD of the delays of all the operation types used in the design to be the clock period, there will be no idle portion in all of the clock cycles, no matter what operations are executed in each clock cycle; that is, the longest clock period for which we will have a zero clock slack for *all* operation types is the GCD of their delays.

6. SLACK MINIMIZATION METHOD

Although using the GCD of all the operation delays as the clock period will result in zero clock slack, it is sometimes too small to be practically implemented. Thus we need a method to select a good clock period with the smallest average slack within the feasible range when the GCD is not applicable.

Consider the definition of the slack minimization problem [Eq. (7)]. To find out the properties of Eq. (7), we expand the equation into a formula of primary terms, shown in the following.

$$\textbf{Find } clk \geq clkmin \text{ that} \tag{8}$$

Minimizes

$$\frac{\sum_i \{occur(oper_i) \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i))\}}{\sum_i occur(oper_i)}.$$

Because we are minimizing Eq. (8) over a range of clock period values, terms in the equation that are invariant regardless of the specific clock period value being considered (clk) can be treated as constants. Thus the problem can be further simplified into Eq. (9) shown in the following.

$$\textbf{Find } clk \geq clkmin \text{ that } \textbf{Minimizes} \sum_i \{k_i \times ((\lceil d_i \div clk \rceil \times clk) - d_i)\}, \tag{9}$$

where both $k_i = (occur(oper_i))/\Sigma_i\, occur(oper_i))$ and $d_i = delay(oper_i)$ are constants under various clk.

Let $f_i(clk)$ denote a single term of Eq. (9); we can obtain the following equation.

$$f_i(clk) = k_i \times ((\lceil d_i \div clk \rceil \times clk) - d_i) \tag{10}$$

$$= \begin{cases} k_i \times (clk - d_i) \text{ where } d_i \leq clk \\ k_i \times (m \times clk - d_i) \quad \text{where} \quad \frac{1}{m} d_i \leq clk < \frac{1}{m-1} d_i, \\ \qquad \forall \text{ integer } m > 1. \end{cases}$$

For example,

$$\text{when } \frac{1}{2} d_i \leq clk < d_i, \quad f(clk) = k_i \times 2 \times clk - d_i;$$

$$\text{when } \frac{1}{3} d_i \leq clk < \frac{1}{2} d_i, \quad f_i(clk) = k_i \times 3 \times clk - d_i;$$

$$\text{when } \frac{1}{4} d_i \leq clk < \frac{1}{3} d_i, \quad f_i(clk) = k_i \times 4 \times clk - d_i.$$

Figure 5 graphically depicts the function Eq. (10). Therefore we observe the following useful properties on Eq. (10).

(1) A discontinuous point (break point) of this function is created if and only if it is on $clk = d_i/m$, where m is a positive integer.
(2) The gradient between any two adjacent discontinuous points is fixed.
(3) A minimal value is generated if and only if it is on a discontinuous point.

Because Eq. (9) is a summation of Eq. (10), it inherits these properties from Eq. (10). These properties of Eq. (9) can be seen clearly in Figure 6, an example of the function diagram of Eq. (9) which is computed from the HAL second-order differential equation example [Paulin et al. 1986]. With these

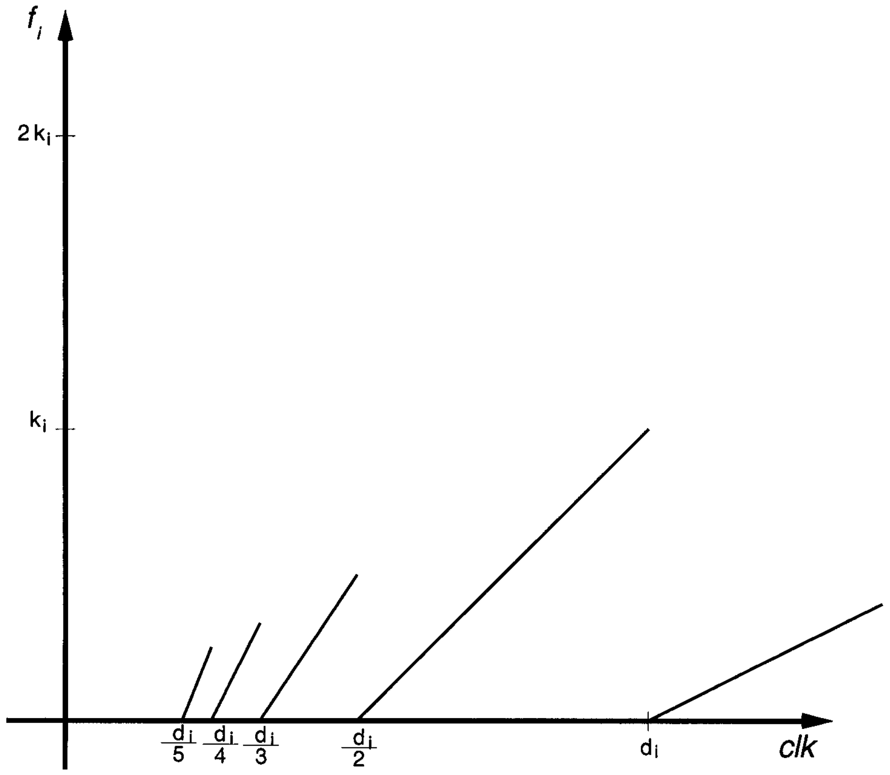


Fig. 5. $f_i(clk) = k_i \times ((\lceil d_i \div clk \rceil \times clk) - d_i)$.

properties we can derive the minimal average slack by examining all the discontinuous points and the boundary, *clkmin*.

The slack minimization algorithm, which computes the clock period with the minimum average slack, is outlined in Figure 7. First we follow the definition of operator occurrences to compute $occur(oper_i)$ of each operation type $oper_i$. Then we search all the discontinuous points of the function *ave_slack*(*clk*) defined in Eq. (6) to find the clock period min_slack_clk that will cause minimum average slack within the clock range specified. The value *clkmin* is the lower bound of the clock range.

The time complexity of computing operator occurrences is $O(n)$, where n is the number of nodes in the DFG provided. The time complexities of computing function *ave_slack*(*clk*) is $O(m)$, where m is the number of the operation types used. The number of points searched is

$$\sum_i \left\lfloor \frac{delay(oper_i)}{clkmin} \right\rfloor + 1, \tag{11}$$

which is $O(m)$. Thus the overall time complexity is $O(n + m^2)$.

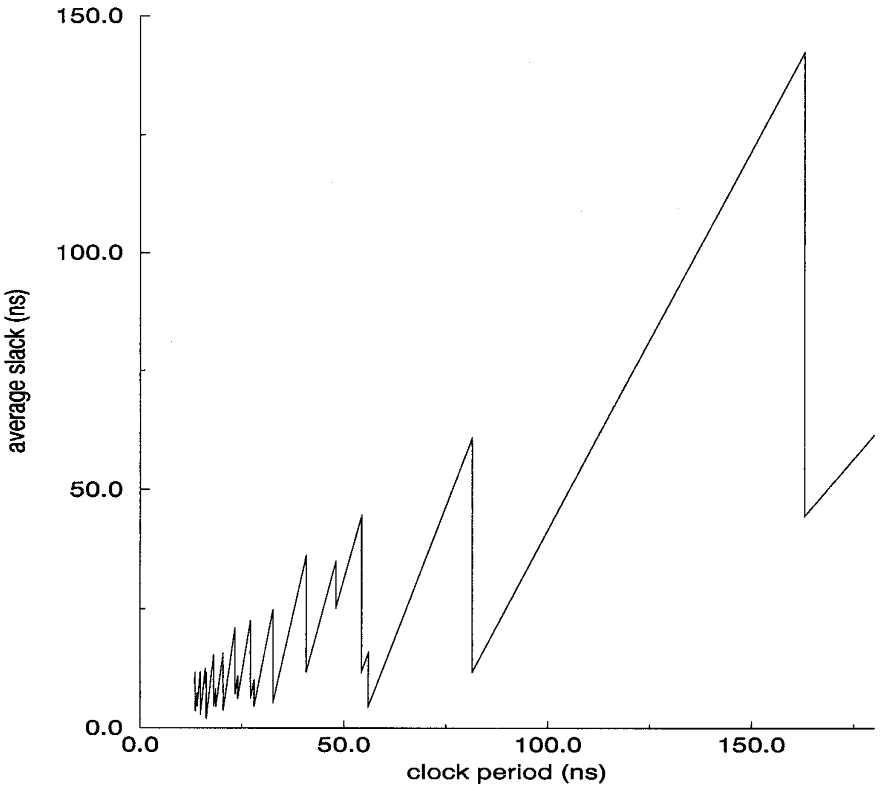


Fig. 6. *ave_slack* $(clk) = \Sigma_i f_i(clk)$ of HAL example.

7. EXPERIMENTAL RESULT

To verify the accuracy of the slack minimization criteria and to prove that the clock period selected by the slack minimization algorithm can produce significantly improved design performance, the Slack Minimization method was applied to several well-known benchmarks, the HAL second order differential equation [Paulin et al. 1986], a fifth order elliptical filter [Kung et al. 1985], a AR lattice filter [Jain et al. 1988], a linear phase B-spline interpolated filter [Pang and Ferrari 1989], and a design with conditional branches [Kim et al. 1991]. We use the resource-constrained scheduler of BDA [Ramachandran and Gajski 1994] to perform the scheduling.

The datapath elements we used, shown in Table I(a), are taken from VLSI Technology Inc. [1992] VCC4DP3 Datapath Library. Computed by Eq. (3), the $delay(oper_i)$ we used are shown in Table I(b). The minimum clock period *clkmin* is 2.54 ns, which is determined by the speed of the global control input noninverting buffer of the VCC4DP3 Library.

7.1 Relation Between Average Slack and DFG Completion Time

Figure 8 illustrates the correlation between the average clock slack and the execution time for the entire behavior. Using an allocation of two functional

```
procedure minimum_slack_clock( $DFG$: data flow graph,
                               $delay(oper_1, oper_2 \ldots oper_m)$ : delays of all operation types,
                               $clkmin$: real number )
begin
    compute $occur(oper_i)$ , $\forall$ operation type $oper_i$
    min_slack := $+\infty$;
    for clk $\in$ $\{ clkmin \} \cup \{ \frac{1}{m \times delay(oper_i)} > clkmin \mid \forall i \; \forall m \}$ do
        begin
            if $ave\_slack$(clk) < min_slack then
                begin
                    min_slack := $ave\_slack$(clk);
                    min_slack_clk := clk;
                end;
        end;
    return( min_slack_clk );
end minimum_slack_clock
```

Fig. 7. Slack minimization algorithm.

units for each operation type, the DFG completion time and the average slack are plotted over a clock range for the fifth order digital elliptical filter [Kung et al. 1985].

From this figure, we can find clearly the relationship between DFG completion time and the average slack: *smaller average slack is associated with shorter DFG completion time in a range wherever the functions are continuous.* This property is true regardless of the behavior being synthesized, the resources allocated, or the scheduler involved. When we shrink the clock period, as long as it does not change the number of clock cycles needed to execute each operation type, it does not change the number of total clock cycles needed to execute the behavior. Under such conditions, not only is the average slack reduced, but the DFG completion time is shortened as well. However, when the clock period is shrunk such that it changes the number of clock cycles needed to execute one of the operation types, the clock slack for the operation type will jump up one clock period and consequently the average slack will increase. This accounts for the discontinuous point of the average slack function. At the same time, owing to the change in the number of clock cycles required to perform an operation of a specific type, it may change the schedule and consequently change the number of total clock cycles required to execute the behavior. Thus a discontinuous point of the DFG completion time function is also created at the same clock period.

Because there is a strong relationship between the average slack and DFG completion time, the clock period with the smallest average slack is an ideal metric to be used while estimating the clock period that will result

Table I. Operation delays derived from VCC4DP3 Datapath Library: (a) datapath elements used in the designs; (b) *delay*($oper_i$) we used.

datapath component	VCC4DP3 cell name	delay time
adder	DPADD001H	26.90 ns
multiplier	DPMLT020M	84.10 ns
subtractor	DPSUB001H	27.40 ns
interconnection	DPBUF0011	1.56 ns
register	DPDFF0201	5.24 ns
control buffer	DPCLKGLO1-4	2.54 ns

(a)

operation type	delay
add	33.70 ns
subtract	34.20 ns
multiply	90.90 ns

(b)

in the shortest DFG completion time. Thus we can obtain the best performance of the design if we select the clock period with the smallest average slack.

Moreover, we can find the clock period that results in the fastest DFG completion time by scheduling the DFG with every clock period at which the average slack function is discontinuous, and then selecting the one with the shortest DFG completion time. According to the properties of Eq. (6) discussed in Section 6, this approach guarantees the optimal solution. However, this approach is impractical, because repetitive scheduling is computationally expensive.

The clock periods shown in the following subsections are the exact values obtained by our algorithm. In actual design, some clock margins might be considered. The user could utilize a ceiling function to round off the theoretical result into suitable precision. The user also could choose a suitable *clkmin*, which is the lower limit for the clock period.

7.2 Benchmark Results

Table II shows the experimental results of four benchmarks using three approaches: the maximum-operator-delay method, the slack minimization method, and the optimal clock period. The optimal clock period is found by using exhausted search declared in Section 7.1.

In Table II, the third column is the clock period selected by each method. The fourth column is the average slack computed by Eq. (6). The fifth

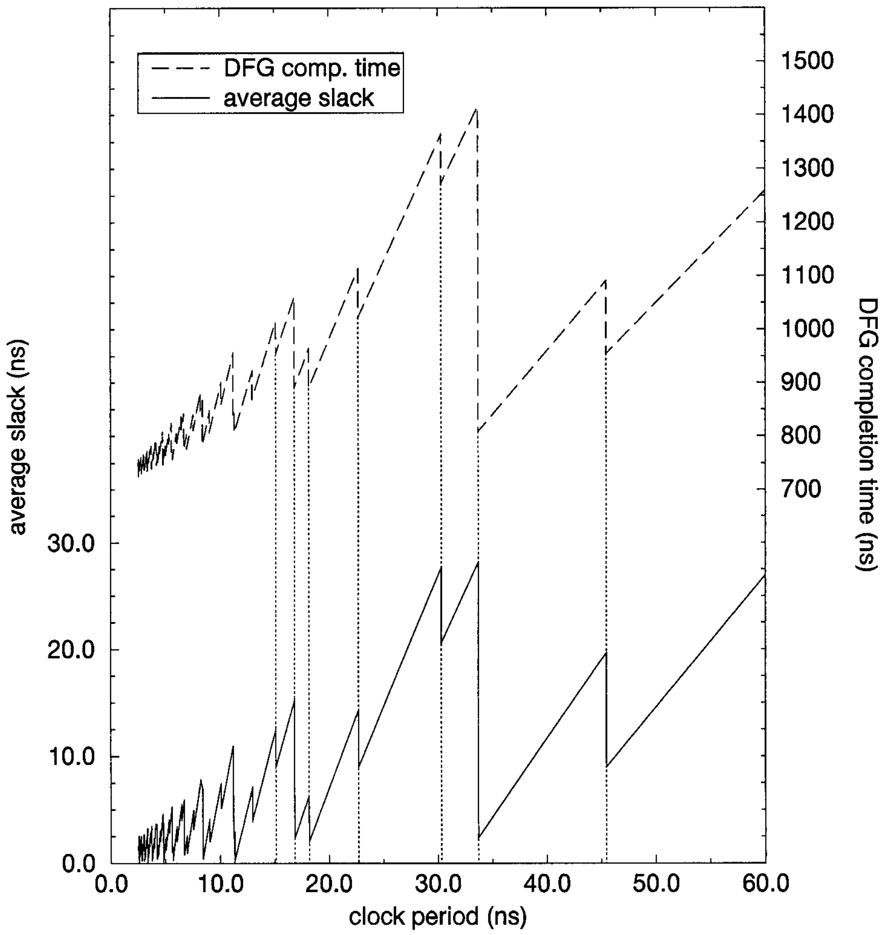


Fig. 8. Relation between average slack and DFG completion time using the fifth order elliptical filter as example.

column is the DFG completion time after we scheduled each benchmark with the clock period selected by each method. We schedule them with the allocation of two functional units of each operation type. Finally, the sixth column shows the slow-down factors of two clock estimation methods: the maximum-operator-delay method and the slack minimization method. The slow-down factor is calculated by Eq. (12) shown in the following.

$$slow\_down = \frac{DFG\ completion\ time\ for\ the\ clock\ estimation\ method}{DFG\ completion\ time\ for\ the\ optimal\ clock\ period} - 1 \tag{12}$$

From this table, we can see that the DFG completion time for the clock period selected by the slack minimization method is very close to the DFG completion time for the optimal clock period.

Table II. Result of four benchmarks scheduled with allocating two functional units of each operation type.

example	clock period selection method	clock period (ns)	average slack (ns)	DFG comp. time(ns)	slow down
HAL differential equation	max. operator delay	90.900	22.780	363.600	15.3%
	slack minimization	3.134	0.212	316.582	0.4%
	optimal clock period	2.674	0.322	315.475	–
digital elliptic filter	max. operator delay	90.900	43.741	1454.400	100.7%
	slack minimization	3.370	0.021	731.288	0.9%
	optimal clock period	2.597	0.048	724.602	–
AR lattice filter	max. operator delay	90.900	24.514	909.000	5.0%
	slack minimization	2.597	0.027	916.790	5.9%
	optimal clock period	4.784	1.960	865.942	–
B-spline interpolated filter	max. operator delay	90.900	35.200	636.300	81.5%
	slack minimization	3.370	0.035	353.849	0.9%
	optimal clock period	2.597	0.039	350.614	–

7.3 Effects of Varying Allocation

In Table III, we examine the fact of whether the clock period selected by the slack minimization method can achieve the DFG completion time that is closed to the DFG completion time for the optimal clock period, regardless of the final allocation of functional units used to implement the design. We scheduled the DFG of the fifth order digital elliptical filter [Kung et al. 1985] with different allocations for the clock period selected by the three approaches and compared their DFG completion time.

Similar to the results in Section 7.2, the DFG completion time for the clock period selected by the slack minimization method is very close to the DFG completion time for the optimal clock period regardless of the allocation used for finally implementing the design.

7.4 Designs with Conditional Branch

In the case including conditional branches, the *Operator Occurrences*, $occur(oper_i)$ may be defined as following,

$$occur(oper_i) = \sum_{branch\ k} \{probability(k) \times block\_occur(oper_i, k)\}, \qquad (13)$$

where $probability(k)$ is the expectation for executing the branch k, and $block\_occur(oper_i, k)$ is the number of occurrences of the operation type $oper_i$ in the branch k.

Table III. Result of fifth order elliptical filter scheduled with allocating different number of functional units.

allocated resources	clock period selection method	clock period (ns)	DFG comp. time(ns)	slow down
2 adder 2 multiplier	max. operator delay	90.900	1454.400	100.7%
	slack minimization	3.370	731.288	0.9%
	optimal clock period	2.597	724.602	–
4 adder 2 multiplier	max. operator delay	90.900	1272.600	84.2%
	slack minimization	3.370	697.588	1.0%
	optimal clock period	2.597	690.839	–
2 adder 3 multiplier	max. operator delay	90.900	1454.400	107.4%
	slack minimization	3.370	707.698	0.9%
	optimal clock period	2.597	701.228	–

Table IV. Result of benchmark including conditional branches.

example	clock period selection method	clock period (ns)	average slack (ns)	average DFG comp. time (ns)	slow down
Kim	max. operator delay	90.900	55.773	749.925	153.0%
	slack minimization	11.400	0.340	296.400	0.0%
	optimal clock period	11.400	0.340	296.400	–

Table IV shows the experimental results for the Kim's example [Kim et al. 1991], which includes conditional branches, using three approaches. We presume equal probability on each branch. From this table, we can see the slack minimization method also can be used for designs including conditional branches.

Moreover, the GCD method developed in Section 5 can be used for the case including conditional branches, because the $occur(oper_i)$ does not affect the analysis for the GCD method.

8. CONCLUSION

In this article we present a new approach for clock period selection, based on clock slack minimization criteria, which provide both designers and synthesis tools with useful methods for the clock period selection.

We prove that the longest clock period that produces no slack for the functional units used in each clock cycle is the extended GCD defined in Section 5.

In some cases the extended GCD may be too small to be practically implementable. When the extended GCD is not applicable, a method of finding the clock period with the smallest average slack in any given range is presented. Experimental results show that the DFG completion time for the clock period selected by the method we propose is very close to the optimal solution.

REFERENCES

BALAKRISHNAN, M. AND MARWEDEL, P. 1989. A synthesis approach for design space exploration. In *Proceedings of the Design Automation Conference* (Las Vegas, NV), 68–74.

GAJSKI, D. 1996. *Principles of Digital Design.* Prentice Hall, NJ.

GAJSKI, D., DUTT, N., WU, C., AND LIN, Y. 1991. *High-Level Synthesis: Introduction to Chip and System Design.* Kluwer Academic, Boston, MA.

GAJSKI, D., VAHID, F., NARAYAN, S., AND GONG, J. 1994. *Specification and Design of Embedded Systems.* Prentice Hall, Englewood Cliffs, NJ.

JAIN, R., MLINAR, M., AND PARKER, A. 1988. Area-time model for synthesis of nonpipelined designs. In *Proceedings of the International Conference on Computer-Aided Design* (Santa Clara, CA).

KIM, T., LIU, J., AND LIU, C. 1991. A scheduling algorithm for conditional resource sharing. In *Proceedings of the International Conference on Computer-Aided Design* (Santa, Clara, CA).

KUNG, S., WHITEHOUSE, H., AND KAILATH, T. 1985. *VLSI and Modern Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ.

MCFARLAND, M. AND KOWALSKI, T. 1990. Incorporating bottom-up design into hardware synthesis. *IEEE Trans. Comput.-Aided Des. 9*, 9 (Sept.).

PANG, D. AND FERRARI, L. 1989. Unified approach to general IFIR filter design using the B-spline function. In *Proceedings of the Asilomar Conference on Signals, Systems & Computers.*

PARK, N. AND PARKER, A. 1985. Synthesis of optimal clocking schemes. In *Proceedings of the Design Automation Conference* (Las Vegas, NV).

PARKER, A., PIZZARO, T., AND MLINAR, M. 1986. MAHA: A program for datapath synthesis. In *Proceedings of the Design Automation Conference* (Las Vegas, NV).

PAULIN, P. AND KNIGHT, J. 1989. Algorithms for high-level synthesis. In *IEEE Des. Test Comput.* (Dec.).

PAULIN, P., KNIGHT, J., AND GIRZYC, E. 1986. HAL: A multi-paradigm approach to datapath synthesis. In *Proceedings of the Design Automation Conference.*

RAMACHANDRAN, L. AND GAJSKI, D. 1994. Behavioral design assistant (bda) user's manual. UC Irvine, Dept. of ICS, Tech. Rep. 94-36.

VLSI TECHNOLOGIES INC. 1988. VDP*100 1.5 Micron* CMOS *Datapath Cell Library.*

VLSI TECHNOLOGIES INC. 1992. *0.8-Micron Datapath Library (*VCC4DP3*).*

WALKER, R. AND CAMPOSANO, R. 1991. *A Survey of High-Level Synthesis Systems.* Kluwer Academic, Boston, MA.

Received January 1996; revised April 1996; accepted April 1996