An optimal slack minimization method

En-Shou Chang

September 11, 1995

Technical Report #95–34

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
(714) 856-7063

echang@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 operation into control steps. Prior to scheduling, 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. A bad choice of the clock cycle could adversely affect the performance of the synthesized design. We present mathematical proofs of setting clock cycle length with zero clock slack and an algorithm for estimating the system clock based on a clock slack minimization criteria. This algorithm guarantee the minimum average clock slack.

1 Introduction

In these 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**[1, 2] way. The advantage of this new methodology is that it allow us to describe a design in a purely behavioral form, void of any implementation details. e.g. we can describe the design via Boolean equations, finite-state machines...etc. Then, the design structure is generated by automatic synthesis using CAD tools, instead of manually designing detail, since manual design is usually tedious.

The describe-and-synthesize methodology can be applied on several levels[2]: (1) On the circuit level, we describe the system in transfer functions and timing diagrams, then transform them into transistor circuit via circuit synthesis techniques. (2) On the logic level, we use Boolean expressions and state diagrams, then transform them into interconnected logic gates and flip-flop's via logic synthesis techniques. (3) On the Register-Transfer(RT) level, register-transfer descriptions which specify the data transfer from some registers to other registers, usually through some functional units, are used to specify the system, then microchips are synthesized using **behavioral synthesis** techniques.(4) On the system level, we use variables and language operators to express functionality of system components, then synthesis them into printed circuit(PC) boards or multi-chip modules(MCM's) via system synthesis techniques.

On the RT level, the microchips, which represent processors, memories and ASIC's, can be synthesized using behavioral synthesis techniques. The structure of these microchips consist functional units, storage units, and control units that are pre-designed and stored in RT-level libraries. The behavior of these microchips are described by means of programs, algorithms, flowcharts, dataflow graphs, instruction sets or generalized finite-state machines in which each state can perform arbitrarily complex computations.

Behavioral synthesis involves the transformation of a specification or design description into a set of interconnected micro-architectural component which satisfy the behavior and any specified constraints. Among the several tasks performed during synthesis, **scheduling** determines the appropriate control step for each operation in the behavioral description.

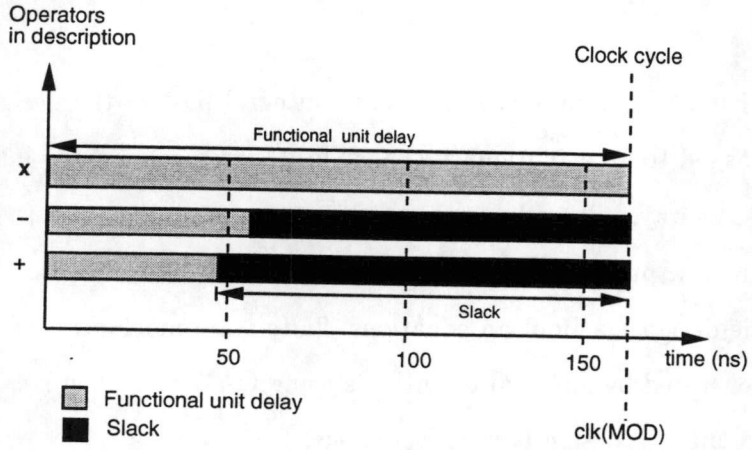


Figure 1: Functional-unit slack associated with clock cycle of 163 ns

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**.

In most synthesis tools[3, 4, 5, 6, 7], the clock cycle must be specified by the designer before synthesis. Either the clock cycle is specified explicitly or the delays of components are expressed in multiples of a clock cycle. Designer-specified clock cycles are applicable when the design which is developed is part of a larger system. In this case, the clock cycle used for some of the standard components in the system is known and can be used for the remainder of the design. In the other case, the clock is not specified by the designer, we need to estimate a good one.

Some synthesis tools[8, 9, 10] equate the clock cycle with the delay of the slowest operation in the design. However, the maximum operator delay clock leads to under-utilization of the faster functional units. In order to improve the performance of the design, we need to minimize the idle time of functional units. The **clock slack**[1] associated with a functional unit represents the portion of the clock cycle for which this functional unit is idle. Smaller slack associated with a functional unit will result in a higher utilization of the functional unit, and in shorter execution time for the same number of resources. Definition of **slack** will be given in Section 2

In Section 2, we will quote definitions about the slack minimization method from[1].

Previous works and their flaws will also be discussed in Section 2. In Section 3, we will show how to achieve 100% clock cycle utilization with setting clock cycle as long as possible and prove it mathematically. In Section 4, an optimal slack minimization method which can find the clock cycle length within any given range with minimum average clock slack is provided. Experimental results is shown in Section 5. Then, we will summarize our achievements in Section 6.

2 Problem definition and previous work

In order to improve the performance of the design, we need to minimize the idle time of the functional units. In the following sections, we will quantify the idle, then minimize it via mathematical analysis or algorithm approaches.

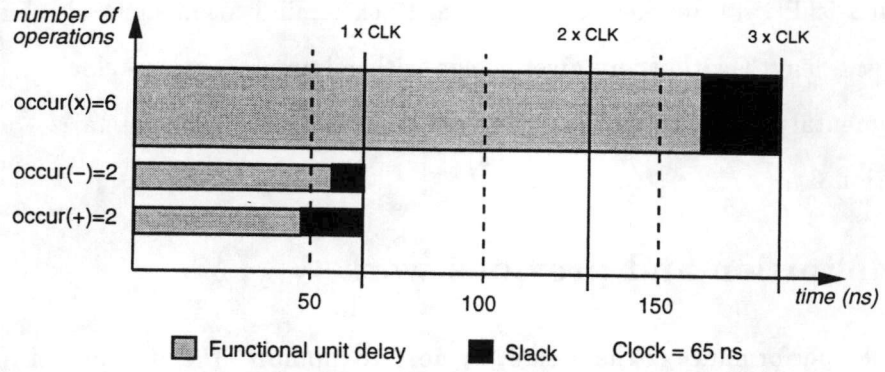
The idle time **slack**[1], associated with a functional unit, is defined as the difference between the functional unit delay and the next higher multiple of clock cycle. The $delay(oper_i)$ represents the delay associated with a functional unit implementing an operation of type $oper_i$. For a given clock cycle clk and operation type $oper_i$, the slack $slack(clk, oper_i)$ associated with corresponding functional unit is computed by the following equation:

$$slack(clk, oper_i) = (\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i) \tag{1}$$

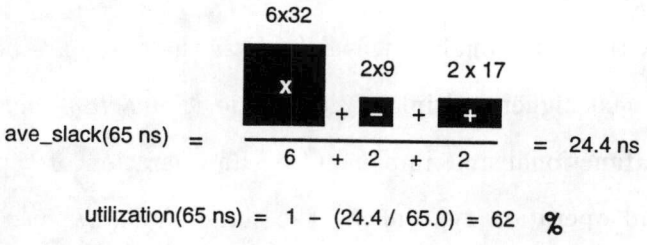
For a given clock cycle clk, the **average slack**, denoted by $ave\_slack(clk)$, is defined as the portion of the clock cycle during which each unit in the design is idle in average. Let $occur(oper_i)$ represents the number of occurrences of operation type i in a behavior, then the average slack would be:

$$ave\_slack(clk) = \frac{\sum_i occur(oper_i) \times slack(clk, oper_i)}{\sum_i occur(oper_i)} \tag{2}$$

Figure 2 graphically depicts the slack associated with the different operations for a second-order differential equation example [4]. The occurrences of each operation and delays of the functional units that will be used to implement them are shown in Table 1. Following the same design model in[11], we put the summation of the delay of the operator associated with two levels of tristate drivers and register setup time and propagation delay as the functional unit delay, since a typical register-to-register transfer involves operands being



(a)



(b)

Figure 2: **(a)** Computing slack, **(b)** Average slack for a clock cycle of 65 ns

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

Table 1: Occurrences and delays of operations in the differential equation example.

read from registers, an operation performed on the operands, and the results stored in another register. The components used are from the VDP100 datapath library[12].

In Figure 2(a), the delays of the functional units are represented graphically as the length of the lightly shaded regions along the X-axis. The number of occurrences of the operations in the behavior is represented by height of the shaded region along the Y-axis. The dark shaded regions represent the slack for each operation type. The average slack for a clock cycle of 65 ns is 24.4 ns, graphically shown in Figure 2(b). It is calculated via Eq. (2).

The main motivation behind the **slack minimization method**[13] is to minimize the slack in each clock cycle under the assumption that a smaller slack will increase the functional unit utilization, and in turn decrease the execution time for the behavior. This slack minimization method[13] examines the clock cycle of grid points from *clkmin* to *clkmax*, computing clock utilization which is defined in Eq. (3) on each grid points. The clock cycle that maximizes the utilization is selected as the best slack-minimal clock. Design libraries usually 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*. The longest delay among all of the functional unit is chosen as *clkmax*.

$$utilization(clk) = 1 - \frac{ave\_slack(clk)}{clk} \tag{3}$$

There are two flaws in the slack minimization method mentioned above. First, the best length of clock cycle is not necessary on a grid point. e.g. it might be 26.2422 ns or $26\frac{1}{3}$ ns. We can't guarantee the highest utilization which is found via examining all of the grid points is the best utilization.

Second, the definition of clock utilization in[13] is incorrect. Since $ave\_slack(clk)$ means the average slack in each operation, not in each clock, the utilization can't be derived from dividing $ave\_slack(clk)$ with clock length, as in Eq.3. It might happen performance associated with clock cycles that have the same utilization value calculated via Eq.(3) is different and performance associated with clock cycles that have the different utilization value calculated via Eq.(3) is the same. Thus, it is not suitable to search the best performance via

clock cycle length	calculated utilization value	average slack	execution time under ASAP scheduling scheduling	execution time under resource-constrained scheduling
56 ns	91.8%	4.6 ns	448 ns	560 ns
28 ns	83.7%	4.6 ns	448 ns	560 ns
14 ns	67.2%	4.6 ns	448 ns	560 ns

Table 2: different calculated utilization values but totally the same performance

Eq.3

As an example, we take the same benchmark in[1], HAL Second Order Differential Equation[4] associated with VDP100 datapath library shown in Table 1. Two schedulers, as-soon-as-possible(ASAP) and resource-constrained, are involved in Table 2. The resource-constrained scheduler limits two functional units for each operation type, as in[13]. We can see different calculated utilization values but totally the same scheduling and performance.

3 Longest clock cycle with no slack

One nature lower bound of slack minimization method is obviously "zero slack". When will the nature lower bound be reached? Before we go further reasoning, let's introduce two extended definitions of **Common Divisor** and **Greatest Common Divisor (GCD)**:

Definition 1: An extended definition of **Common Divisor** over real number space:

A real number r is a Common Divisor of a set of real numbers $\{t_1, t_2, \ldots, t_n\} \equiv \exists$ positive integer $k_i \ni rk_i = t_i, \forall i$

Definition 2: An extended definition of **GCD** over real number space:

GCD : set of $R \rightarrow R$

$GCD\{t_1, t_2, \ldots, t_n\} \equiv$ the greatest $CommonDivisor$ associated with $\{t_1, t_2, \ldots, t_n\}$

Whenever we select a Common Divisor of delay time of those functional units being used as the length of the system clock cycle, it is directly induced from Definition 1 that the delay time of those functional units can be exactly fitted into one or several clock cycles without any slack. We prove this property formally below:

Theorem 1: A given clock cycle length causes no clock slack associated a number of given functional units if and only if the clock cycle is a common divisor of the delay time of those functional units.

Proof: $\Longleftarrow$) Let the delay of the functional units be $\{t_1, t_2, \ldots, t_n\}$ and the given clock cycle be a Common Divisor r. According to Definition 1, there are corresponded positive integers $\{k_1, k_2, \ldots, k_i\}$ such that $rk_i = t_i$ for all i. i.e. the delay time t_i of functional unit i can be exactly fitted into k_i clock cycles with no slack.

$\Longrightarrow$) Assume the delay of the functional units is $\{t_1, t_2, \ldots, t_n\}$ and there are corresponded positive integers $\{k_1, k_2, \ldots, k_i\}$ such that the delay time t_i of functional unit i can be exactly fitted into k_i clock cycles with no slack, then the clock cycle r satisfies Definition 1: $\exists$ positive integer $k_i \ni rk_i = t_i, \forall i$. $\|$

Since a clock cycle length which causes no clock slack should be a common divisor of the delay time of the functional units used, it is trivial to infer the next theorem:

Theorem 2: *The longest clock cycle which causes no clock slack associated a number of given functional units is GCD of the delay of these functional units.*

4 Optimal slack minimization method

Although putting GCD of delay of the functional units as the system clock cycle can induce 100% utilization of each clock cycle, it is too small to be practically implemented in most cases. Thus, we need a method to find a feasible clock cycle length with highest utilization

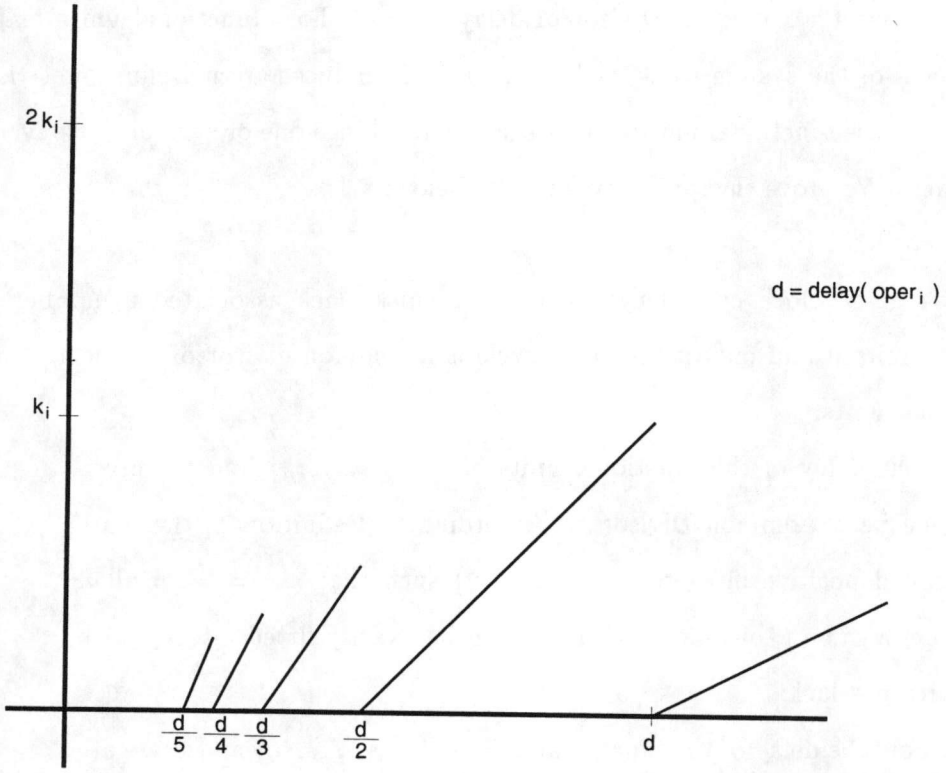


Figure 3: $f(clk) = k_i \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i))$

of each clock cycle, i.e. with lowest average slack.

Following definitions in[13], a slack minimization method, which works on finding the minimum average slack, can be defined in mathematical style below:

$$\textbf{Minimize} \quad \frac{\sum_i occur(oper_i) \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i))}{\sum_i occur(oper_i)}, \forall\ clk \qquad (4)$$

Since we are minimizing Eq. (4) along various clk, those terms that are unchanged associated with various clk can be viewed as constants. Thus, the problem is simplified into

$$\textbf{Minimize} \quad \sum_i k_i \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i)), \forall\ clk \qquad (5)$$

What does the function above (Eq. (5)) look like? Let's see the function $f(clk)$, shown in Eq. (6), of one single term of Eq. (5) first.

$$f(clk) = k_i \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i)) \qquad (6)$$

When $delay(oper_i) \leq clk$, we have $f(clk) = k_i \times clk - delay(oper_i)$, and when $\frac{1}{m} delay(oper_i) \leq clk < \frac{1}{m-1} delay(oper_i)$, we have $f(clk) = k_i \times m \times clk - delay(oper_i)$

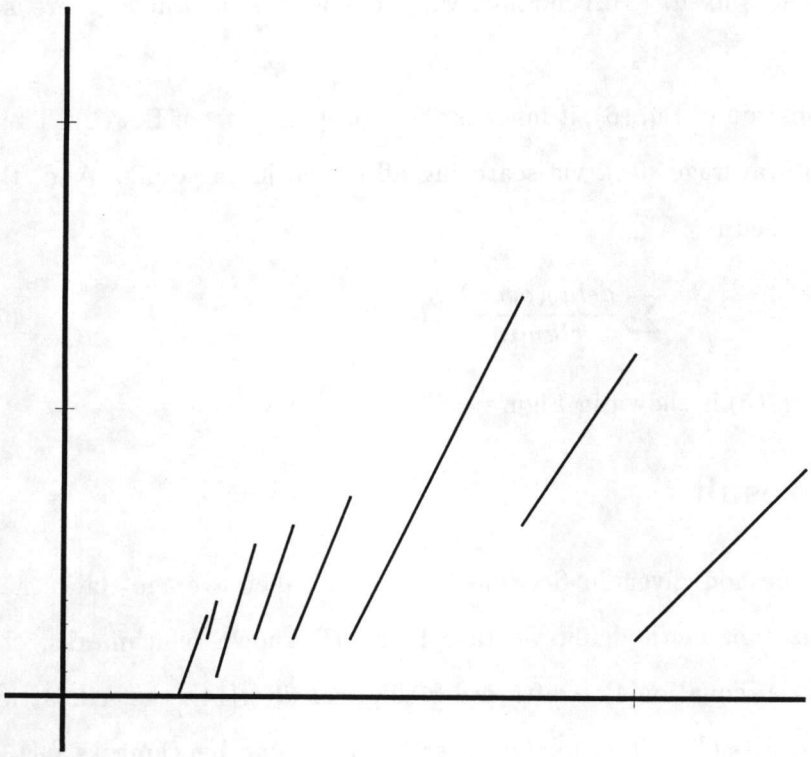


Figure 4: $f(clk) = \sum_i k_i \times ((\lceil delay(oper_i) \div clk \rceil \times clk) - delay(oper_i))$

for all positive integer greater than 1. i.e.

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

when $\frac{1}{3}delay(oper_i) \leq clk < \frac{1}{2}delay(oper_i)$, $f(clk) = k_i \times 3 \times clk - delay(oper_i)$;

when $\frac{1}{4}delay(oper_i) \leq clk < \frac{1}{3}delay(oper_i)$, $f(clk) = k_i \times 4 \times clk - delay(oper_i)$;

...etc. The function diagram of $f(clk)$ in Eq. (6) is shown in Figure 3.

There are three useful properties on Eq. (6). (1) A jump-point of this function happens if and only it is on $\frac{1}{m \times delay(oper_i)}$ where m is a positive integer. (2) The gradient between any two adjacent jump-point is fixed. (3) A minimal value happens if and only if there is a jump-point.

Since Eq. (5) is a summation of Eq. (6), it inherits these properties from Eq. (6). Thus, we can derive the minimum average slack via searching all of the jump-points. And, the number of points been searched is

$$\sum_i \frac{delay(oper_i)}{clkmin} \tag{7}$$

The function diagram of Eq. (5) is shown in Figure 4.

5 Experimental result

To demonstrate that the method given in Section 4 results smaller average slack than the previous slack minimization method[13], we take four well known benchmarks, the HAL second order differential equation[4], a fifth order elliptical filter[14], the AR lattice filter[10], and a linear phase B-spline interpolated filter[15], the same benchmarks in[13], with VDP100 datapath library[12], shown in Table 1, as examples.

Following[13], the lower bound *clkmin* of searched space is determined by the maximum operating frequency for clocking registers of VDP100 library. Result of maximum operator delay method, the previous slack minimization method[13], and the method we proposed are shown in Table 3. It can be seen the method we proposed in Section 4 achieves much less average clock slack than the previous[13] one.

example	clock estimation method	clock length estimated	average slack
differential equation[4]	max. operator delay	163.0 ns	44.40 ns
	slack minimization[13]	56.0 ns	4.60 ns
	opt. slack minimization	16.3 ns	2.02 ns
fifth order digital elliptic filter[14]	max. operator delay	163.0 ns	87.94 ns
	slack minimization[13]	24.0 ns	1.18 ns
	opt. slack minimization	16.3 ns	0.69 ns
AR lattice filter[10]	max. operator delay	163.0 ns	49.29 ns
	slack minimization[13]	55.0 ns	4.40 ns
	opt. slack minimization	16.3 ns	0.39 ns
linear phase B-spline inter. filter[15]	max. operator delay	163.0 ns	70.77 ns
	slack minimization[13]	24.0 ns	1.92 ns
	opt. slack minimization	16.3 ns	0.55 ns

Table 3: Result of two slack minimization methods

6 Conclusion

In this paper, an optimal method for clock estimation, based on clock slack minimization, is presented. It can provide both designers and synthesis tools with a useful estimating of the clock cycle that can be used to implement a design.

We proved that longest system clock cycle with 100% clock cycle utilization is extended GCD, which is defined in Section 3. In some cases, the extended GCD might be too small to be practically implemented. When it is not applicable, a method to find the clock cycle length with smallest average slack in a given range is also provided.

References

[1] D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and design of embedded systems.* New Jersey: Prentice Hall, 1994.

[2] D. Gajski, N. Dutt, C. Wu, and Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design.* Boston, Massachusetts: Kluwer Academic Publishers, 1991.

[3] M. Balakrishnan and P. Marwedel, "A synthesis approach for design space exploration," in *Proceedings of the Design Automation Conference*, pp. 68–74, 1989.

[4] P. Paulin, J. Knight, and E. Girzyc, "HAL: A multi-paradigm approach to datapath synthesis," in *Proceedings of the Design Automation Conference*, 1986.

[5] P. Paulin and J. Knight, "Algorithms for high-level synthesis," in *IEEE Design & Test of Computers*, Dec. 1989.

[6] R. Walker and R. Camposano, *A Survey of High-Level Synthesis Systems.* Kluwer Academic Publishers, 1991.

[7] M. McFarland and T. Kowalski, "Incorporating bottom-up design into hardware synthesis," *IEEE Transactions on Computer-Aided Design*, September 1990.

[8] A. Parker, T. Pizzaro, and M. Mlinar, "MAHA: A program for datapath synthesis," in *Proceedings of the Design Automation Conference*, 1986.

[9] N. Park and A. Parker, "Synthesis of optimal clocking schemes," in *Proceedings of the Design Automation Conference*, 1985.

[10] R. Jain, M. Mlinar, and A. Parker, "Area-time model for synthesis of non-pipelined designs," in *Proceedings of the International Conference on Computer-Aided Design*, 1988.

[11] S. Narayan and D. Gajski, "System clock estimation based on clock wastage minimization." UC Irvine, Dept. of ICS, Technical Report 91-49,1991.

[12] *VDP100 1.5 Micron CMOS Datapath Cell Library*, 1988.

[13] S. Narayan and D. Gajski, "System clock estimation based on clock slack minimization," in *Proceedings of the European Design Automation Conference (EuroDAC)*, 1992.

[14] S. Kung, H. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing.* Prentice-Hall, 1985.

[15] D. Pang and L. Ferrari, "Unified approach to general IFIR filter design using the B-spline function," in *Proceedings of Asilomar Conference on Signals, Systems & Computers*, 1989.