AN ABSTRACT OF THE THESIS OF

Hubert J. Stier for the degree of Master of Science in Electrical and Computer Engineering presented on May 3, 1995. Title: Design of a 80/250-Msample/s FIR-Filter for a Pipelined ADC-FIR Interface.

Abstract approved: Redacted for Privacy

Shih-Lien Lu

With a growing demand for digital video signal processing applications, the demand for fast low-cost analog-to-digital converters (ADCs) is increasing rapidly. In order to meet this demand, recent research has provided a high variety of pipelined ADC devices. In contrast to traditional ADC methods, pipelined ADCs do not generate a parallel n-bit digital output, but n serial outputs. Often, analog-to-digital converters can be found in combination with digital filters that process the ADC output signals before they are further used.

The thesis project describes a pipelined FIR-filter that is designed to process the serial outputs of pipelined ADCs. It is shown that the serial architecture of the ADC can provide high-efficient FIR-filter architectures. Improvements over traditional FIR-filter applications are achieved through 2-dimensional pipelining and most significant bit first multiplication. Two pipelined FIR-filter architectures designed in CMOSN standard cells are shown. They can achieve a maximum sample rate of 80-Msamples/s and 250-Msamples/s and thereby meet the sample rate requirements set by the current serial ADC technology. By interfacing ADC and FIR-filter closely, further improvements in speed, area, and power consumption can be made. Thus, in addition to a stand-alone version of the FIR-filter, an interfaced version of ADC and FIR-filter is also considered.

Design of a 80/250-Msample/s FIR-Filter for a Pipelined ADC-FIR Interface

by

Hubert J. Stier

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed May 3, 1995
Commencement June 1995

Master of Science thesis of Hubert J. Stier presented on May 3, 1995

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Hubert J. Stier, Author

ACKNOWLEDGMENT

I would like to express my sincere appreciation and gratitude to my major professor, Dr. Shih-Lien Lu, for his continual support throughout my thesis project, Many thanks for his patience, understanding and encouragement along the way.

I would also like to thank friends, family and the Department of Electrical and Computer Engineering for their enduring support.

Special thanks as well to my sponsor, the Fulbright-Kommission, for their financial support. Without their generous help, this dream would have never come true.

TABLE OF CONTENTS

LIST OF FIGURES

Figure ***Page***

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

Table ***Page***

Design of a 80/250-Msample/s FIR-Filter for a Pipelined ADC-FIR Interface

1. INTRODUCTION

1.1 Motivation

The successive approximation (SA) analog to digital conversion method is an inexpensive and precise technique, extensively used in signal processing applications. Often, analog to digital converters (ADCs) are found in combination with digital filters. There are two major applications that suggest such an arrangement. First, would be applications in which analog filters cannot meet the requirements of a specific filtering process within the analog domain. So, continuous-time signals are converted to digital in order to apply digital filters. Second, would be applications in which digital processes require analog data as input. In order to remove noise from the signal, the ADC is succeeded by a digital filter.

The SA method generates a bit stream with the most significant bit (msb) first. In general, SA-ADCs store the output stream of each conversion in registers and thereby provide a parallel output. Thus, the filtering process cannot be started before all bits of the conversion process are available. Pipelined SA-ADCs, in combination with serial FIR-filters that start computation already when the first bit of the A/D conversion is ready, would be able to minimize the time required by this process. Such an architecture provides two advantages. First, the filtering process can finish sooner as it already starts with the first bit available. Second, the pipelined SA-ADC can read the next sample already after the first bit is converted.

However, this method requires a new approach in both the analog-to-digital conversion, and the digital filtering process. Furthermore, in order to reduce latency, area, and power consumption of the structure, the ADC and the FIR-filter can be interfaced in a single device.

1.2 Outline of the Thesis

The next chapter presents some general considerations that show the motivation for a pipelined ADC-FIR design in detail. Chapter 3 describes the techniques suggested for a serial ADC-FIR design. Methods like pipelined A/D conversion, msb-first multiplication and two-dimensional pipelining, as well as I/O formats, are explained in this section. Chapter 4 shows two different algorithms for a n-bit pipelined FIR-filter. It also describes the single modules used in the design and gives an overview of the logic applied. Chapter 5 evaluates the filter design. Information about hardware cost, speed, and scalability is given for the implemented 8-bit version as well as for other sizes of filters. Chapter 6 gives a summary and conclusion about the work. A short analysis shows the performance perspectives of the pipelined ADC-FIR designs in comparison to traditional methods. Appendices A and B show the implementation of 2 different designs for an 8-bit pipelined FIR-filter by means of Powerview tools [1].

2. MOTIVATION FOR A PIPELINED ADC-FIR DESIGN

2.1 FIR-Filters

Filter devices are used in a wide range of applications, such as removing noise from signals, removing signal distortion due to the transmission channel, and demodulating signals. Among all these applications two major families of filters can be identified: analog filters and digital filters. Analog filters involve signals in the analog domain (continuous-time signals), whereas digital filters relate to signals in the digital domain (discrete-time signals).

Most digital filters are based on the following relation between the filter input sequence X(n) and the filter output sequence Y(n):

$$Y(n) = \sum_{k=0}^{N} a_k * Y(n-k) + \sum_{k=0}^{M} b_k * X(n-k) \qquad (1)$$

The first term on the right side of the equation determines the part of the output that depends on the last N+1 foregoing outputs generated by the filter (infinite impulse response-IIR), whereas the second term determines the part that depends on the last M+1 foregoing inputs to the filter (finite impulse response-FIR). For FIR-filters all a_k are zero. Therefore, equation 1 reduces to

$$Y(n) = \sum_{k=0}^{M} b_k * X(n-k) \qquad (2)$$

where b_k is the weighted factor for input X(n-k). Thus, the output of the FIR-filter Y(n) is simply a finite length weighted sum of the current and the previous inputs to the filter.

2.2 Parallel SA-ADC Method in Combination with DSP

Most applications use digital signal processors (DSPs) in order to implement FIR-filters. DSPs are programmable multipurpose devices which handle a variety of tasks like filtering, fourier transformation, floating point arithmetic or digital sine wave generation. The traditional arrangement of a DSP with a preceding SA-ADC process is shown in Figure 1.

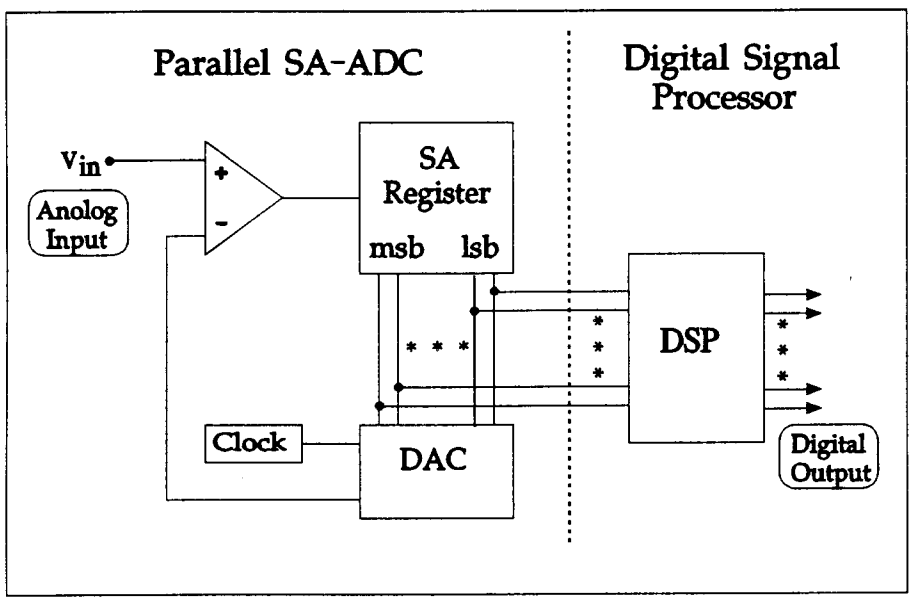

Figure 1. Parallel SA-ADC Method with DSP.

During the ADC stage, the parallel SA-ADC starts by enabling the digital inputs of the D/A converter one bit at a time, starting with the msb. As each bit is enabled, its amplitude is compared to the analog voltage, v_{in}, by the voltage comparator. If the output of the D/A converter is greater than the analog voltage, the msb is reset to zero, since it will not be required in the digital representation of the analog input. If the D/A converter output is less than the analog input, the msb is retained in the register.

After the ADC goes through this procedure once per bit, the DSP stage starts. First, the digital signal processor reads the unfiltered digital signals. Then, for each weighted factor a multiplication followed by the accumulation of the product has to be performed, a total of M+1 multiplications and additions. Finally, the accumulated result is sent to the output port.

This design is somewhat inefficient in two ways. First, the digital signal processor cannot start its operation until all bits of the ADC are available. This is due to the instruction set of the DSP which performs multiplication by means of two complete operands. The ALU of the DSP, however, performs multiplication bitwise through generation and accumulation of single partial products. Thus, the ALU may stay idle until all bits are available for operation even though it can process the bits only one by one. Second, as a single comparator has to generate all n output bits, the conversion of a single sample requires n consecutive conversion processes within the ADC. As a result, the total conversion time sums up to a n-fold of the time required by the inner process. The following paragraph will show an alternative design that takes a different approach.

2.3 Pipelined ADC-FIR Interface

The algorithm of the A/D converter in Figure 1 shows the characteristic of a loop. In Figure 2, the same algorithm is implemented as a chain of serial A/D units. Thus, we get a pipeline of A/D units, where each unit is made of a comparator plus some more logic. Each A/D unit generates a serial bit-stream with the unit on the left generating the msb and the unit on the right generating the lsb. In contrast to the looped version in Figure 1, the pipelined ADC in Figure 2 implements a full conversion each clock-cycle. The output bits of the serial A/D units are sent to the functional units of the FIR-filter. These units process the bit stream immediately, generating a partial product each cycle. The FIR units

succeed the pipeline of the ADC, accumulating the partial products according to the algorithm given in equation 2. Therefore, at the time when the DSP in Figure 1 starts its process, the pipelined FIR-filter in Figure 2 has only one bit to process left. The A/D converter and the FIR-filter in Figure 2 form a single pipeline. Thus, this circuit suggests an interface of ADC plus FIR-filter. A interfaced architecture reduces area, power consumption and latency of the device. The next paragraph describes the modules shown in Figure 2 in more detail.

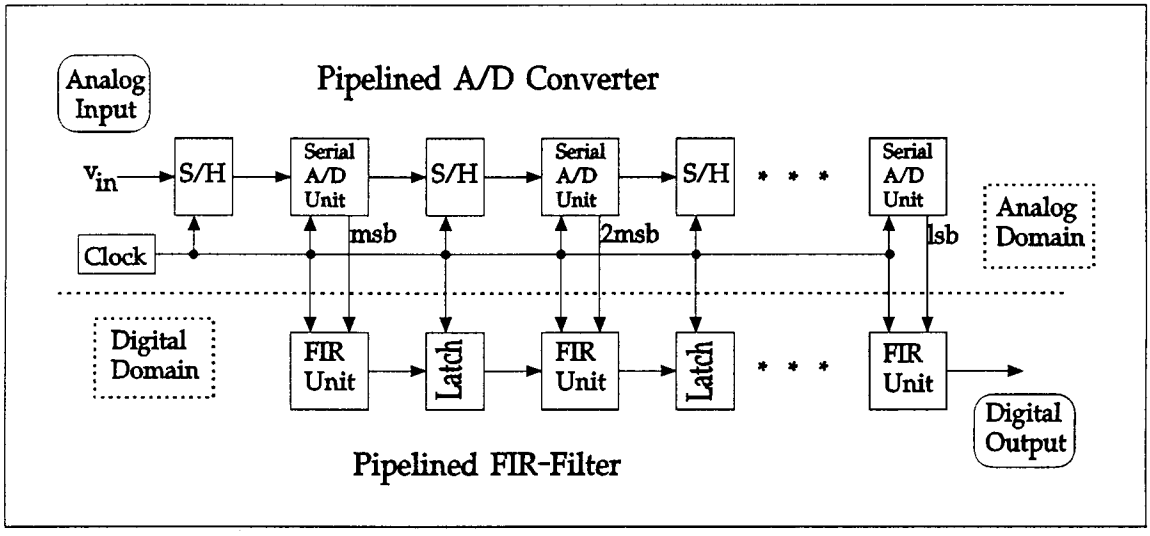

Figure 2. Pipelined ADC-FIR Interface.

3. FEATURES OF A PIPELINED ADC-FIR DESIGN

The architecture of a pipelined ADC-FIR design is characterized by two techniques, pipelined SA-A/D conversion and msb-first multiplication.

3.1 Pipelined SA-ADC

Figure 3 describes the architecture of a pipelined SA-ADC. It shows two serial successive A/D stages. There are two analog inputs, the signal input v_{in}, and the reference voltage input v_{ref} /2, which is one half of the maximum convertible signal.

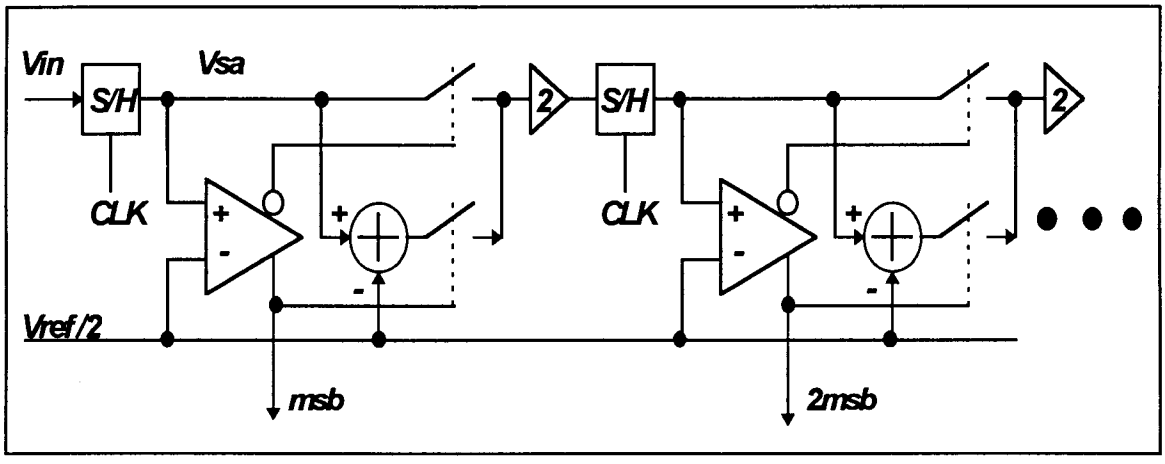

Figure 3. Pipelined SA-ADC.

The pipeline starts conversion with the msb in stage one, then the 2msb in stage two and so on. While stage two generates the second msb of the first input signal v_{in}, stage one already generates the msb of the second input signal

v_{in}. Each stage works as follows: First, the sampled signal v_{sa} is compared to $v_{ref}/2$. If v_{sa} is greater than v_{ref} /2, then the non-inverted output of the converter is 1. Thus, the digital output of this stage is set to 1 and the lower switch in the first stage feeds the signal v_{sa} - v_{ref} /2 into the succeeding multiplier. If v_{sa} is smaller than $v_{ref}/2$, then, as the inverted output of the comparator is 1, the digital output of the stage is set to zero. The upper switch feeds the sampled voltage directly into the succeeding multiplier which doubles the signal before it is sampled for the next stage. The following routine describes the function of each stage in short. The diagram in Figure 4 gives an example with six stages. v_{sa} of stage 1 is equal to the original input v_{in}.

```
IF Vsa > Vref/2 THEN
        out := 1
        Vsa := Vsa -Vref/2
ELSE
        out := 0
ENDIF
Vsa := Vsa * 2
```

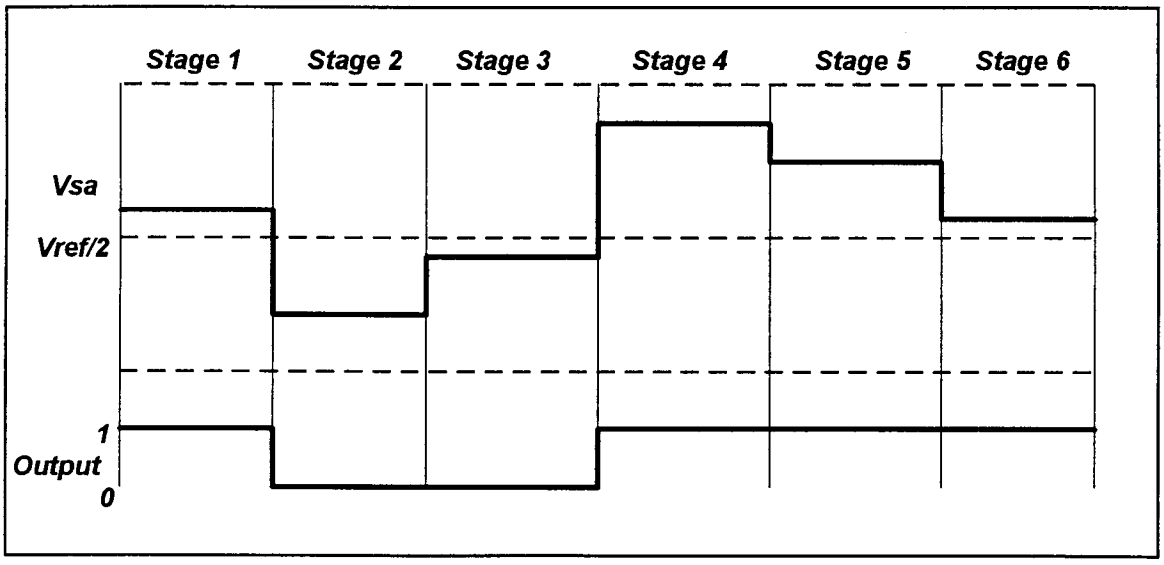

Figure 4. Example: Work-Sequence of a Pipelined SA-ADC with six Stages.

3.2 MSB-First Multiplication

A major feature of the alternative design is the serial multiplication within the FIR-filter, which is able to provide a higher speed than the parallel register-register multiplication performed in the DSP. There are two techniques that perform low cost serial multiplication: the add and shift method and the quasi serial method. These techniques perform multiplication with the least significant bit first. The pipelined SA-ADC described in 3.1, however, generates the most significant bit first and, therefore, requires another algorithm. This algorithm is illustrated in Figure 5.

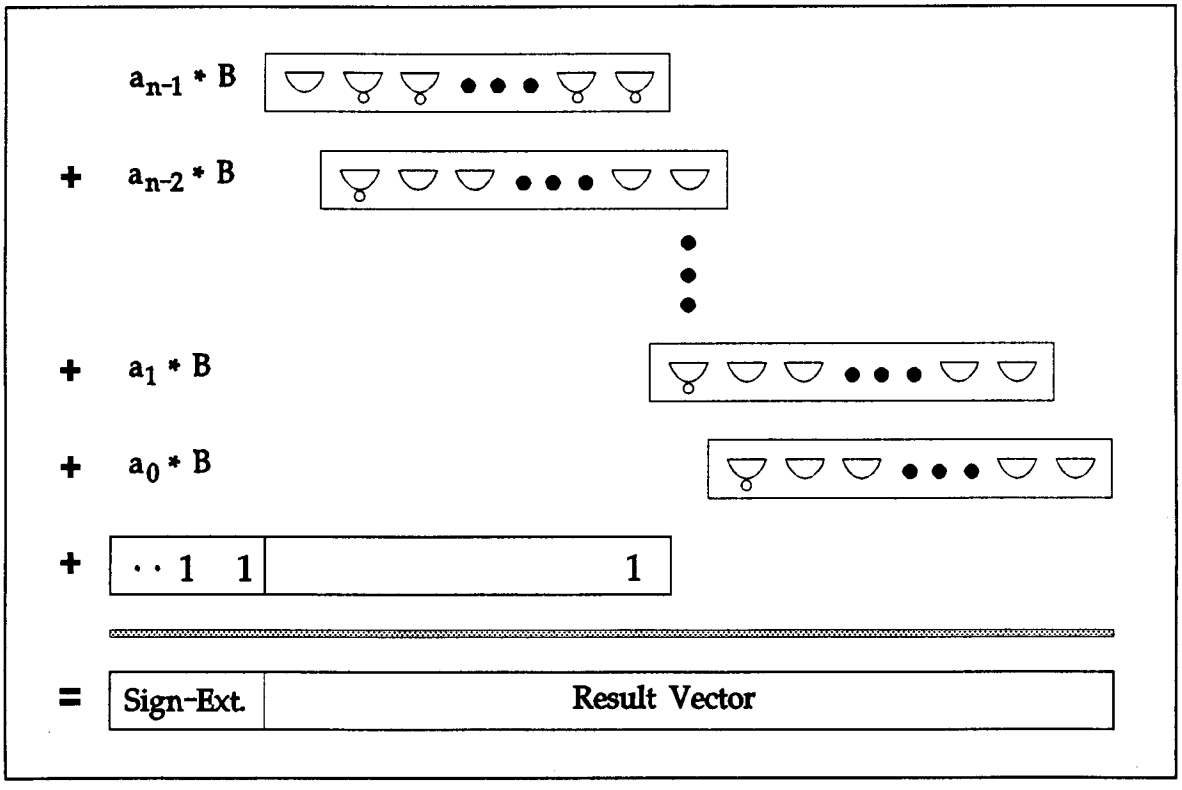

Figure 5. General Structure of the MSB-First Multiplier for a Multiplication A x B.

The msb-first multiplication algorithm applied [2] shows two special features that cannot be found in other multiplication methods. First, the partial products are not generated through straight AND operations, but through a combination of AND and NAND operations. The most significant partial product is generated in a AND-NAND array with all NANDs, but a single AND at the leading bit. All other partial products are generated in AND-NAND arrays with all ANDs, but a single NAND at the leading bit. Second, a logical '1' is added to the (n+1) least significant bit as well as to the bits on the left of the (2n-1) result vector. The '1's on the left are needed to generate a correct sign-extension.

3.3 Concepts for the Design

Equation (2) in 2.1 described the general FIR-filter function. The last product term in (2) can be also represented as shown in (3). In order to simplify the notation and deriviation without loss of generality, we will assume X(n-k) is an unsigned number.

$$X(n-k) = \sum_{m=0}^{u-1} x(n-k)_m * 2^m \tag{3}$$

With (3) in (2) we get a more detailed form of (2):

$$Y(n) = \sum_{k=0}^{M} \left[b_k * \sum_{m=0}^{u-1} x(n-k)_m * 2^m \right] \tag{4}$$

For practical use, (4) can be written in 2 forms:

$$Y(n) = \sum_{k=0}^{M} \left[\sum_{m=0}^{u-1} b_k * x(n-k)_m * 2^m \right] \tag{5}$$

$$Y(n) = \sum_{m=0}^{u-1} \left[\sum_{k=0}^{M} b_k * x(n-k)_m * 2^m \right] \tag{6}$$

Even though both forms of the equation are quite similar, they lead to different algorithms. According to (5), first, k products are generated by accumulating u-1 partial products which are shifted by m bits to the left. This first part is represented by the inner part of the brackets. In the second step, all k products are accumulated to a final result. In (6) all the partial products with the same exponent m are accumulated first. In the second step, the resulting terms are shifted by m bits to the left, before they are accumulated.

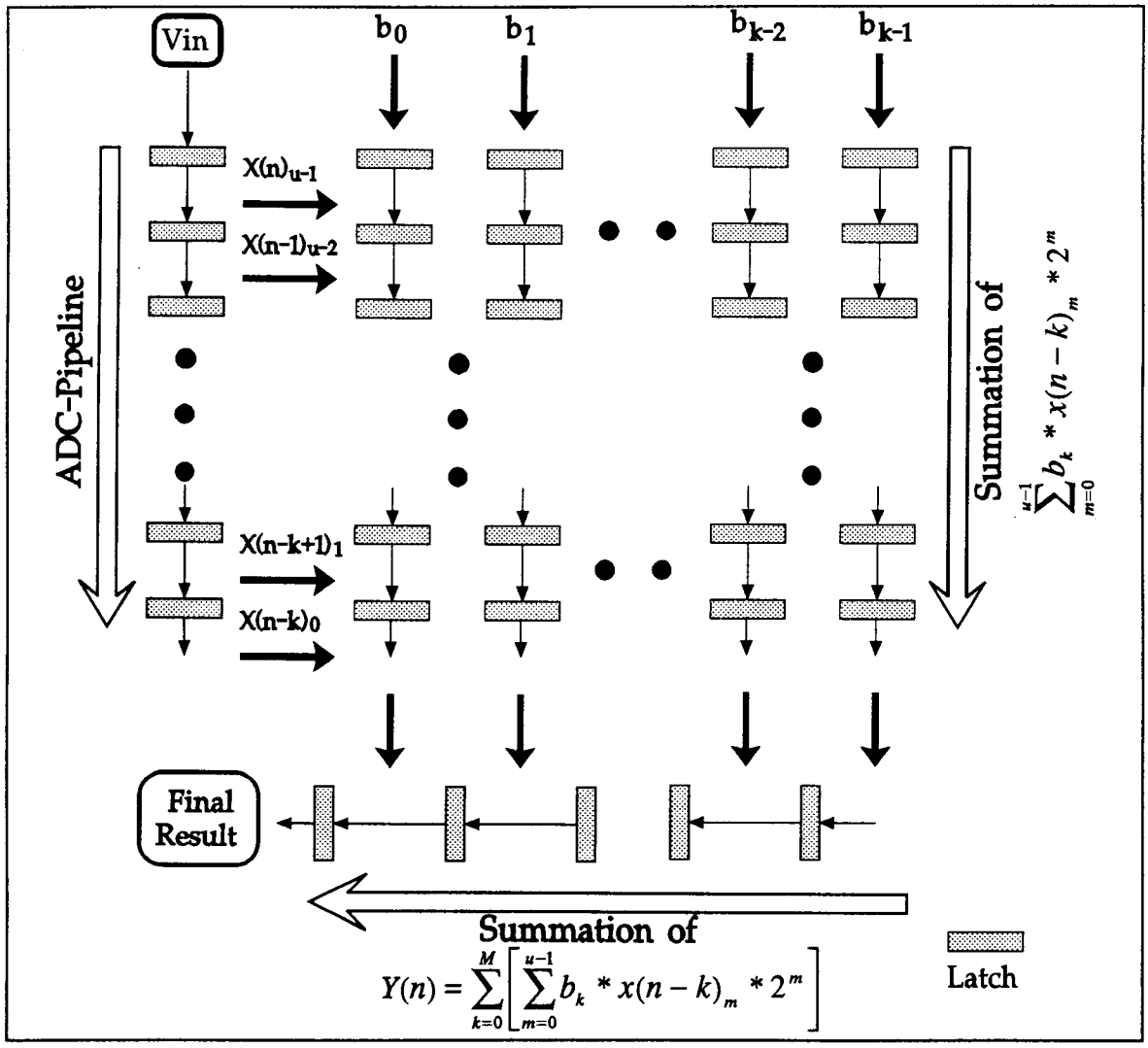

Figure 6. Algorithm According to Equation (5).

Both algorithms are made of two stages, where each stage performs a pipelined accumulation process. Thus, with the additional pipeline applied in the ADC we get a total of three pipelines in each algorithm. The three pipelines form a 2-dimensional matrix. Figures 6 and 7 show the matrixes according to equations (5) and (6). The large black arrows represent the inputs to the rows and columns of the matrix.

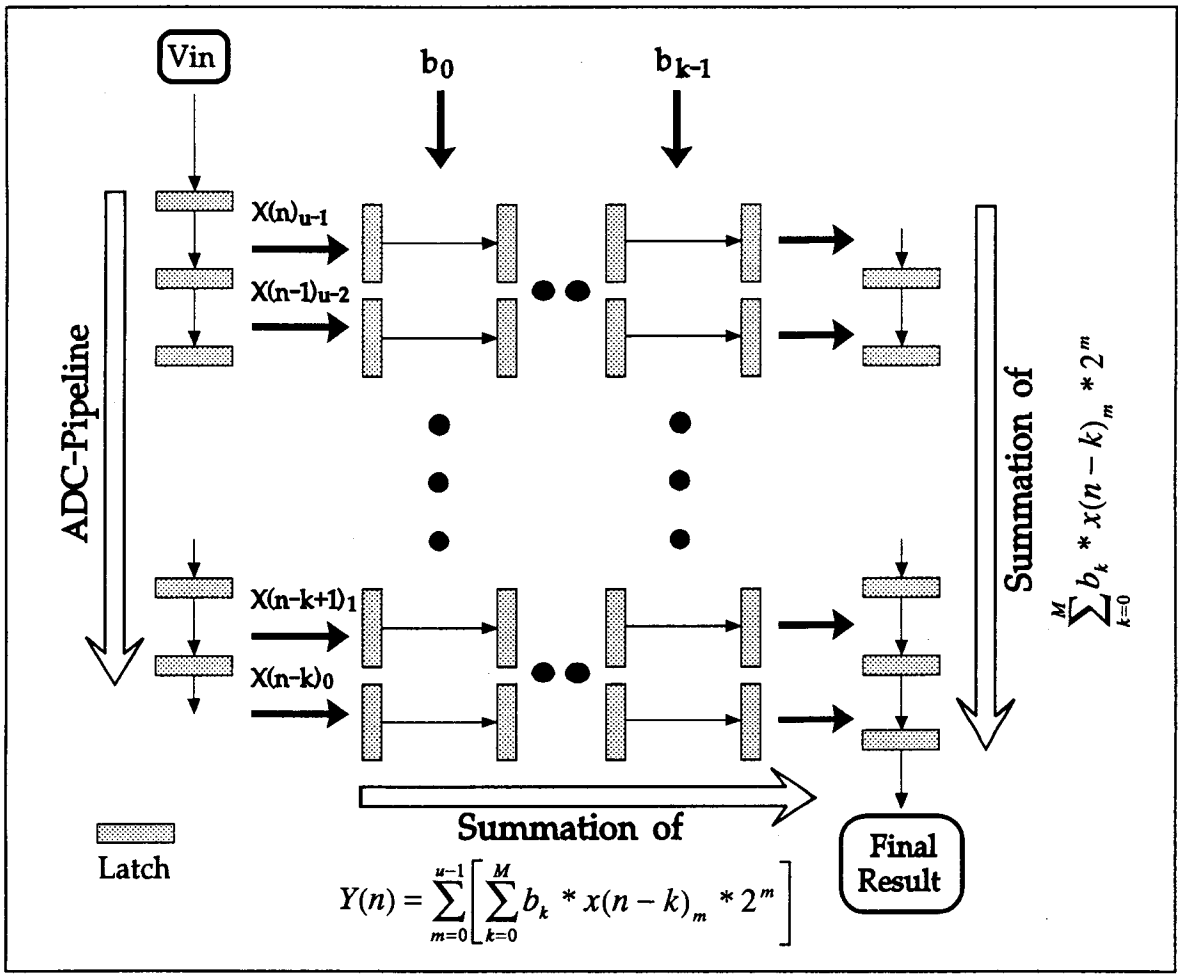

Figure 7. Algorithm According to Equation (6).

For both designs it is assumed that the number of ADC output bits is equal to the number of factors (u=M+1). This assumption provides a squared matrix that can be extended in size without structural changes. This assumption is held throughout the whole work.

3.4 I/O Requirements for the Filter

Input X from the ADC

Input X is an n-bit 2's complement number that can be in the whole range given by n.

Inputs for factors A

Input A has to be an n-bit number that can represent positive and negative numbers. There are n inputs A with $\sum A = 1$.

Output Y

Y has to be of the same format as X, with a stable input X, Y has to be equal to X. With this definition, no conversion back to the input format is needed. As both the inputs X and A can represent positive as well as negative numbers, a full representation of the result can require more than n bits. In the case when positive factors meet positive ADC outputs and negative factors meet negative ADC outputs, the result can sum up to a maximum of n*X. In order to avoid overflow, the output Y has to provide $\log_2 n$ additional bits. The design is implemented to meet this requirement. In applications where the problem of overflow does not exist, the output can be simply cut to n bits. Since the output is in 2's complement representation, no error will occur.

As we can see, the only format that is not completely defined yet is the format of the factors A. The task now is to define the format of the factors A in such a way that the given format of the input X produces an output of format Y.

The solution found is a 2's complement format with a virtual decimal point behind the most significant bit.

X.XXXXX.....

The most significant bit of the format is the 2's complement sign bit. Each factor can represent a decimal value in the range of -1 ... 0 ... $1-2^{-(n-1)}$. The sum of all factors has to be one. Thus, as we cannot represent a one in a single factor; there have to be at least two non-zero factors. Figure 8 shows an overview of all data inputs and outputs of the FIR-filter.

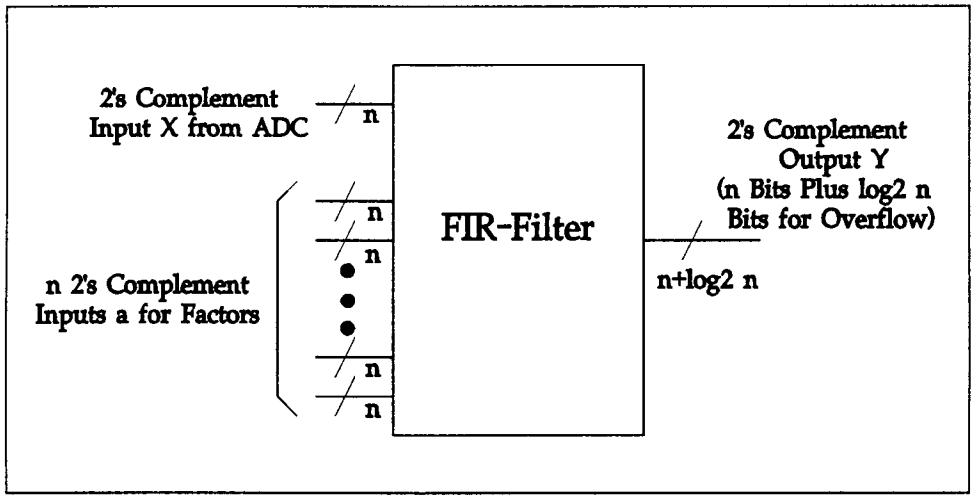

Figure 8. Inputs/Outputs for FIR-Filter.

4. FIR-FILTER DESIGN IN CMOSN

4.1 Goals and Properties

Before I explain the design in detail, I want to show the guidelines used in the design process. Thus, it will be easier for the reader to understand the specific solutions applied in the design. The most important goals are the following:

- Maximum speed
- Minimum hardware
- 100% functionality
- Straight forward design

Like in all pipelined circuits, there are two speed criteria that have to be considered. First, there is the clock cycle time which is the only input to the frequency of the circuit. Second, there is the number of clock cycles of the pipeline, which, together with the clock cycle time, determines the latency of the circuit (latency = clock cycle time * #cycles). A high frequency provides a high number of conversions per time, whereas a low latency provides a fast response to the input signals. Only a minimum of different modules are to be used. An extension of the ADC-output width should not require changes in the structure of the design.

Of course, there are some trade-offs between these goals. The trade-off between speed and hardware is a characteristic that can be found in most electrical designs. The analysis of both algorithms explained in 3.3 showed that the structure of algorithm 1 suggests a high degree of speed optimization, whereas the structure of algorithm 2 suggests a high degree of hardware optimization.

4.2 Design 1

Design 1 is shown in Figure 9. It is the practical implementation of the algorithm explained in Figure 5. n input-bits x_i are broadcasted to all pipelined array multipliers (PAM1) simultaneously. In general, a n-bit x n-bit multiplication results in a (2n-1) bit product. The pipelined multipliers, however, generate two result vectors of a different length. There are two aspects that require this alternative solution:

- As described in 3.4, in order to prevent overflow, $\log_2 n$ more bits have to be spent for the output. It is a general rule that sign-extension for 2's complement numbers has to be implemented before the numbers are accumulated. Since the accumulation process of the partial products starts in PAM1, the extension by $\log_2 n$ bits has to be implemented already within the pipelined array multipliers.
- Each stage of an array multiplier generates two output vectors, a carry vector CO and a sum vector SUM. The elimination of one vector would require a time consuming CLA-process. In order to avoid a loss in speed, both output vectors are sent directly to the pipelined adder. As the two least significant bits of the carry vector are eliminated during the accumulation process, the sum vector requires two bits more than the carry vector.

The pipelined adder (PAD) generates the final result. Design 1 only requires two different elements, PAM1, and PAM2.

Figure 10 shows the pipelined array multiplier in more detail. The inputs x are generated in the SA-ADC; they are updated every cycle. $x(n)_{u-1}$ is the most significant bit, $x(n-k)_0$ is the least significant bit. Three major modules are used to implement the circuit: AND-NAND arrays, latches, and accumulation arrays. The accumulation arrays are built of FAs which can handle up to 3 inputs. Thus, as the accumulation process does not need to start before 3 inputs are available, the two AND-NAND arrays at the top are not followed by an accumulation array. The AND-NAND arrays form a parallelogram, where the array for the msb is

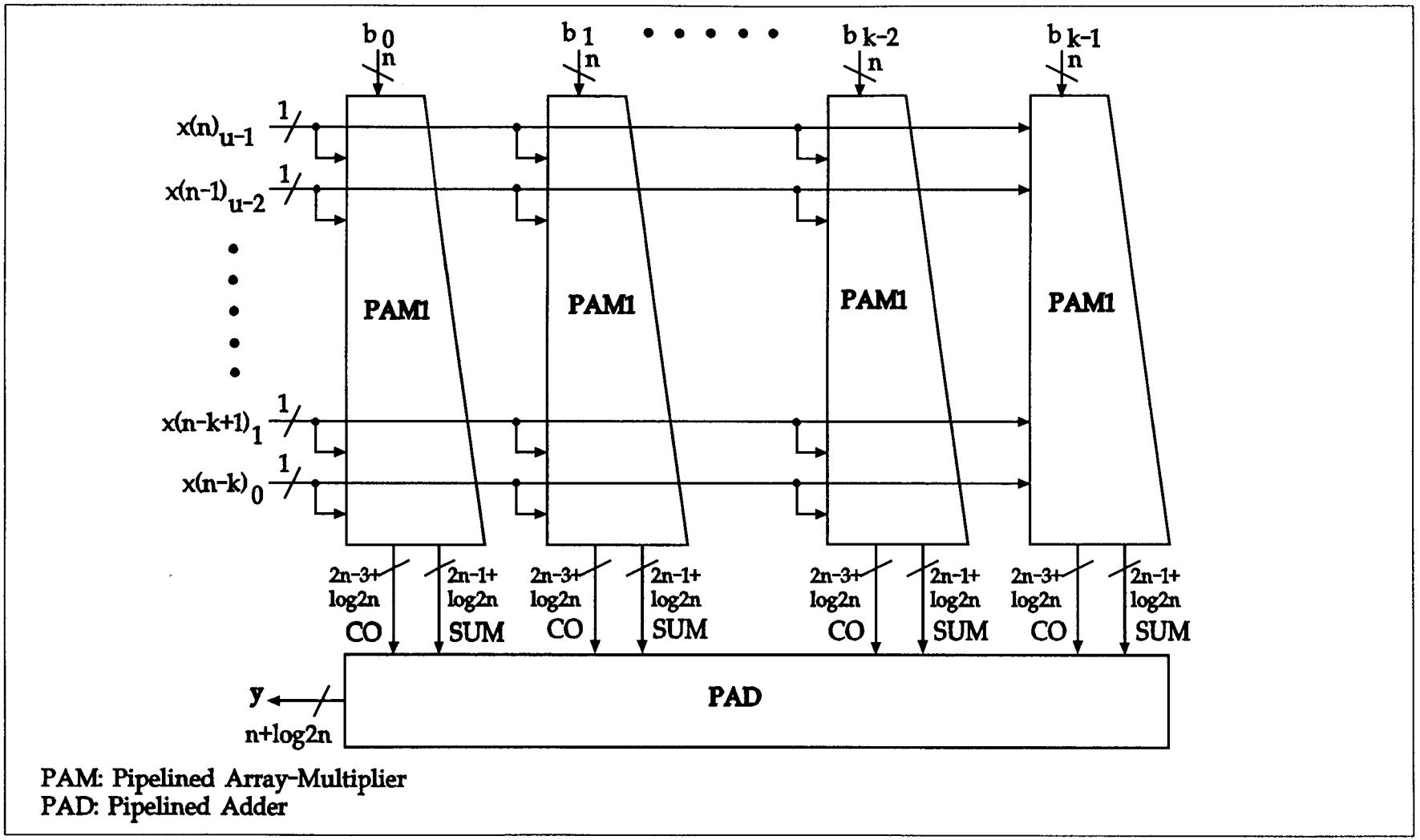

Figure 9. Design for FIR-Filter 1.

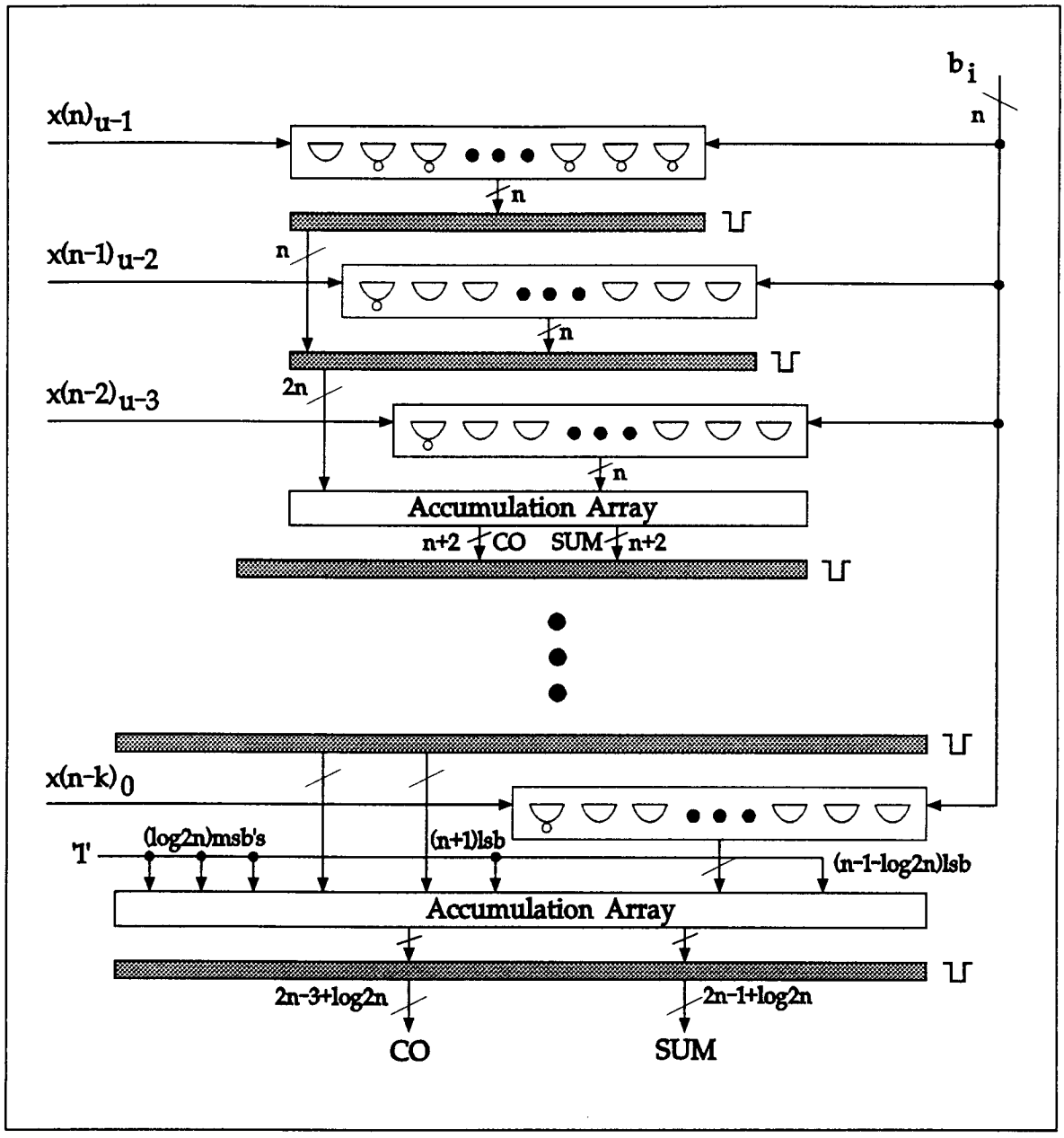

Figure 10. Pipelined Array Multiplier (PAM1).

placed on the left top, the array for the lsb is placed at the right bottom. As explained in 3.2, in order to implement msb-first multiplication, the circuit shows two special features. First, the partial products are not generated in n-bit AND arrays, but in AND-NAND arrays. Second, a logical '1' is added to the column

that represents the (n+1)lsb as well as to the columns that represent the (log_2n) most significant bits.

As we can see, there is another '1' added to the (n-1-log_2n)lsb. This is due to the rounding scheme. In Figure 9 we can see that the (2n-1+log_2n) bit input of the PAD produces a (n+log_2n) bit output, which means that the result has to be cut by (n-1) bits. In order to keep the maximum precision, the result has to be rounded. In the general rounding process, a '1' is added to the msb of the bits that are truncated afterwards. Such a final rounding process is very time consuming as it requires a succeeding CLA addition process. Thus, it is suggested that the '1' is already added before the final result is generated. The '1' can not be added within the PAD as the full adders in the PAD offer no unused inputs for this operation. Therefore, the '1' is added already during the multiplication process within the pipelined array multiplier. When n is a power of 2, a '1', added to the (n-1-log_2n)lsb in each PAM1 sums up to a '1' in the (n-1)lsb, which is exactly what is needed. When n is no power of 2, a '1' has to be added directly to the (n-1)lsb of a single PAM1. In both cases, no final rounding is required.

Figure 11 shows the pipelined adder (PAD). It consists of two accumulation trees, one for the carries and one for the sums. A solution with only one tree but two accumulations arrays between the latches would be also sufficient. However, since all other pipeline stages only apply one full adder in a row, two consecutive accumulation arrays would increase the clock cycle time of the circuit immensely. As a result, a more hardware-consuming 2-tree design is applied. For the final addition, a parallel prefix addition network according to Ladner/Fischer [3] is used. A general CLA network cannot be applied efficiently as the clock cycle time available is less than the time required by each CLA stage. The nodes of the parallel prefix circuit, however, meet the requirements of the circuit. Figure 12 illustrates the structure of the parallel prefix addition network. The three least significant bits of the carry vector are already eliminated during the accumulation process, thus, they do not have to be

considered. In the first step, the generation terms g_i and the propagation terms p_i are generated. Then, the parallel prefix matrix generates the carrys for all, except the (n-1) least significant bits that are cut off through rounding. The single nodes of the matrix are designed according to the Brent/Kung algorithm [4]. A XOR operation generates the final result.

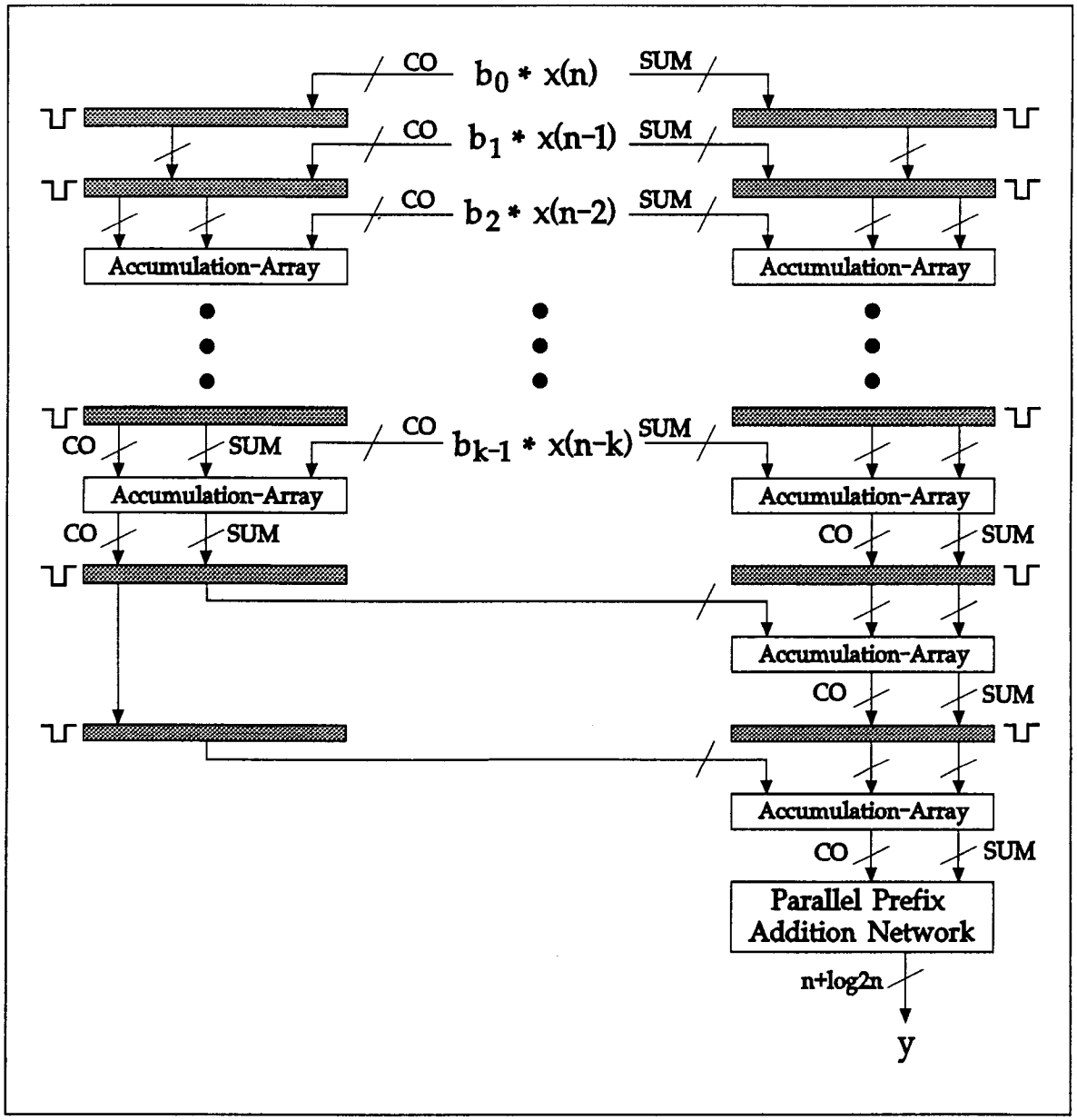

Figure 11. Pipelined Adder (PAD).

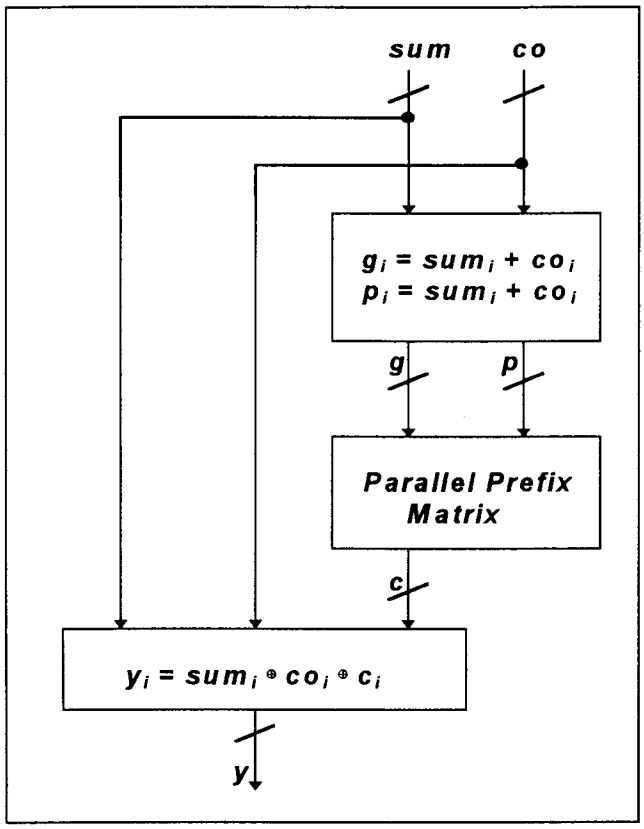

Figure 12. Parallel Prefix Addition Network.

4.3 Design 2

Design 2 (Figure 13) shows the implementation of the algorithm described in Figure 7. The inputs x_i are pipelined to a number of AND-NAND arrays, which generate the partial products. A carry save adder tree (CSA) [5] sums up all partial products of each row every clock cycle. As all the partial products of one row are generated with input bits of similar exponents, no shifting is required. Thus, the length of the result vectors we get in design 2 is less than the corresponding result vector in design 1. Again, there is no final CLA stage within the process; therefore, the CSA tree generates two outputs, a carry output and a sum output. All CSAs send their outputs to a pipelined array multiplier (PAM2), where the final result y is generated.

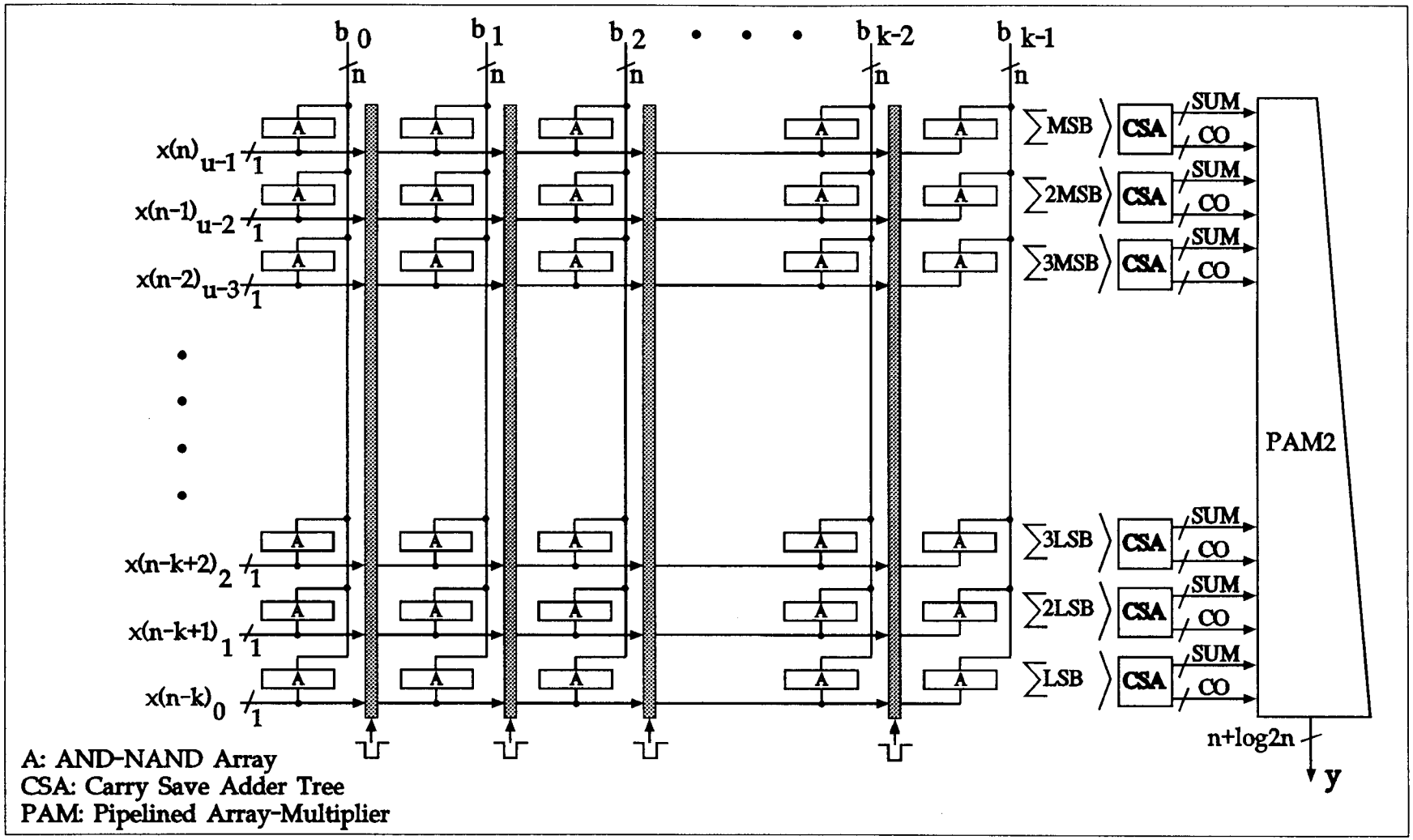

Figure 13. Design for FIR-Filter 2.

Figure 14 shows the architecture of PAM2 in more detail. As the partial products in design 2 are already generated earlier in the process, there are no AND-NAND arrays needed within PAM2. The inputs to design 2 also form a parallelogram, with the most significant pair of inputs placed at the left top, and the least significant pair of inputs placed at the right bottom. The clock cycle time of design 2 is determined by the time required for the CSA stage. As the CSAs use several consecutive full adders in one stage, the clock cycle time in design 2 adds up to a value larger than in design 1. Therefore, we can apply two consecutive accumulation stages between the latches without any increase in clock cycle time. PAM2 offers some free full adder input lines, thus, the rounding bit '1' can be added to the column that represents the (n-1)lsb directly.

The msb-first multiplication algorithm described in 3.2 does not meet the requirements found in PAM2. Unlike in PAM1, the inputs to PAM2 are not just single partial products, but a sum of M+1 partial products. Both pipelined multipliers, however, show the same kind of structure. The most significant input, generated by AND-NAND arrays with only one AND, is accumulated first, whereas the less significant inputs, generated by AND-NAND array with only one NAND, are accumulated later. PAM2 implements all the multiplications done by n PAM1s in design 1 simultaneously; it is an n-fold duplication of PAM1. Therefore, in order to get the correct result, we have to apply the corrective addition process shown in 3.2 once per factor. Since the number of factors M+1 is equal to n, we have to add n x 1 to the column that represents the (n+1)lsb as well as the columns that represent the ($\log_2 n$) most significant bits.

In design 2, no parallel prefix addition network is applied to generate the final result. As the clock cycle time is less critical as in design 1, a CLA-adder can be applied without increasing the clock cycle time.

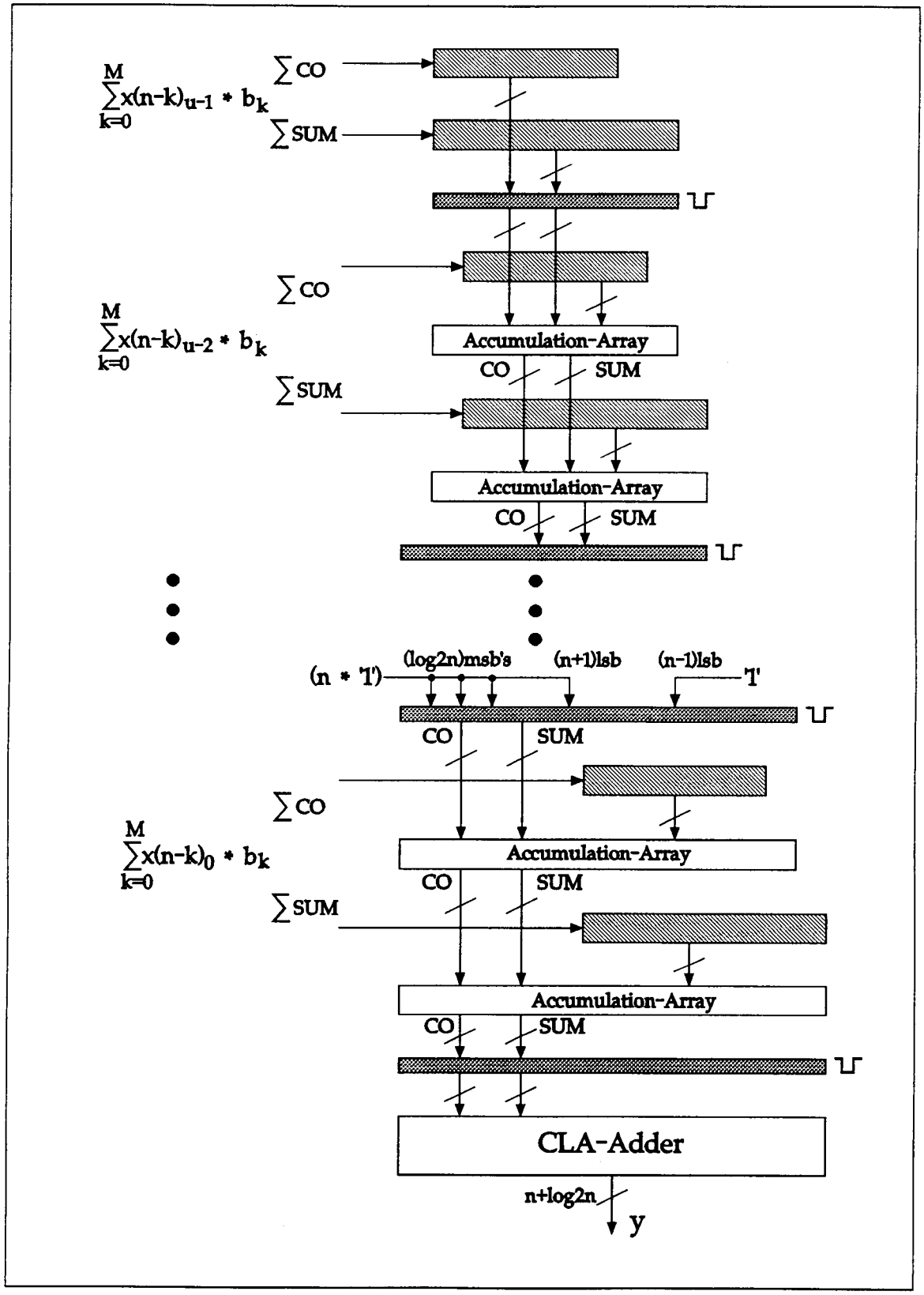

Figure 14. Pipelined Array Multiplier (PAM2).

4.4 Device Setup

Before the FIR-filter is able to work, it has to go through a setup stage. During the device setup, the FIR-filter reads k multiplication factors b, required for the process. The design of the setup logic is shown in Figure 15. It is applied for both circuits, design 1 and design 2.

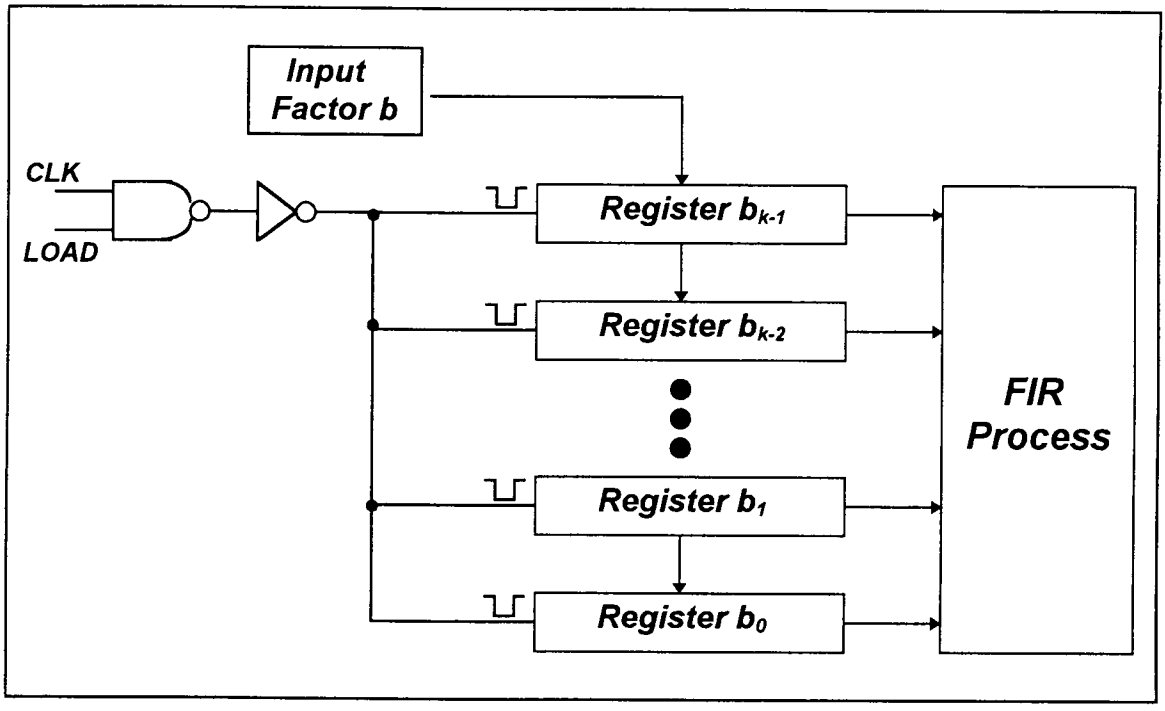

Figure 15. Setup Logic.

The setup logic requires k latched n-bit registers, each holding one multiplication factor. Whenever the load signal is set to '1', the clock reaches the registers. With the falling edge of the clock signal, all registers read the signals at their inputs. The register on the top directly reads the input at the pins, whereas all other registers read the signals held by the preceding registers. The setup requires k cycles. The serial arrangement of registers requires the

implementation of a new setup for every modification of factors. With a parallel structure or an additional address bus for the registers, modifications of single factors could also be implemented during the process. On the other hand, the serial structure only requires a minimum of input pins, Thus, in order to avoid additional control inputs, a serial structure is used.

4.5 I/O Pins

Figure 16 shows the data pins required for a serial ADC-FIR interface. Three major categories of pins are considered: pins for data input, pins for data output, and pins for control and setup. Additional pins (e.g. power supply pins) are not shown.

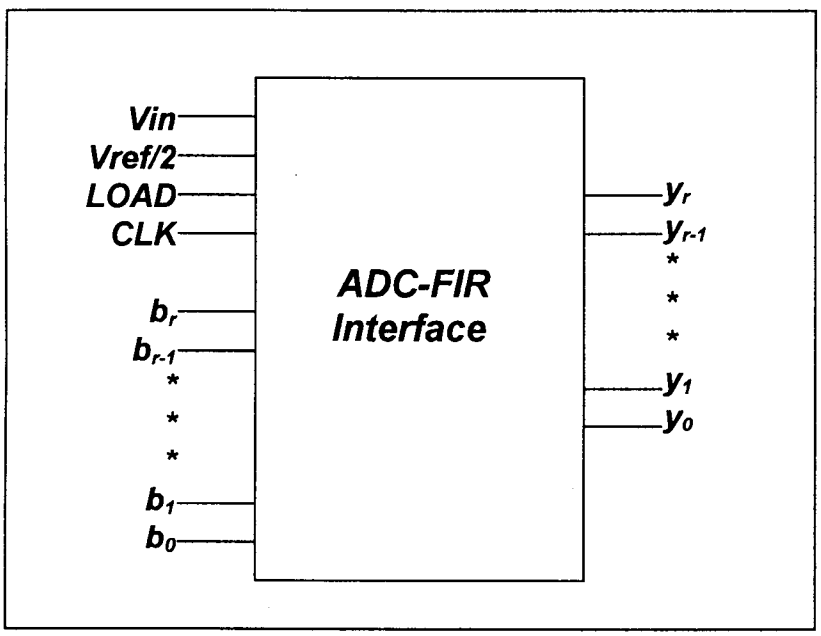

Figure 16. I/O Pins.

The ADC-FIR interface in Figure 16 shows a clock pin CLK. Even when provided with an internal clock, the device still requires an synchronization signal to the master process. However, this signal is only required during the setup, when the registers have to be latched. As a result, the use of an external clock is optional.

5. DESIGN EVALUATION

5.1 Hardware Requirements

Both designs only require a small variety of logic elements:

- Full adders (FAs) and half adders (HAs) for the accumulation process.
- D-flip-flops for the latches.
- AND/NAND gates for the generation of the partial products.
- CLA networks and XOR elements for the final addition stage.
- Pull-ups, pull-downs and inverters for special features.

An analysis of the circuits showed that the hardware cost for CLA networks, XOR elements, pull-ups, pull-downs and inverters adds up to less than 2% of the total hardware cost of the design. Thus, the following analysis only shows three groups of elements: AND/NAND gates, latches and FAs/HAs. Figures 17, 18, and 19 show a graphical comparison between the two designs. Tables 1 and 2 show specific numbers for n = 4, 8, 12 and 16 bits.

Hardware requirements for design 1 (in order of n):

- $n^3 + 10n$ AND/NAND gates
- $4n^3 + 8n^2 + 30n$ Latches
- $1.5n^3 + n^2$ FAs/HAs

Hardware requirements for design 2 (in order of n):

- n^3 AND/NAND gates
- $5n^2 + 2n$ Latches
- $n^3 + n^2$ FAs/HAs

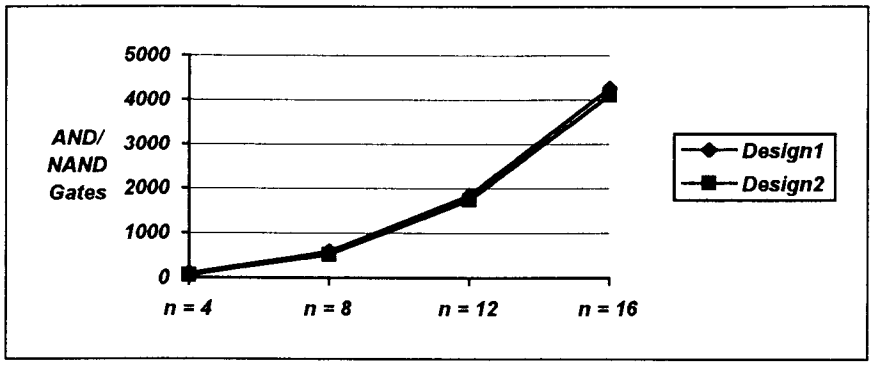

Figure 17. AND/NAND Gates Required for both Designs.

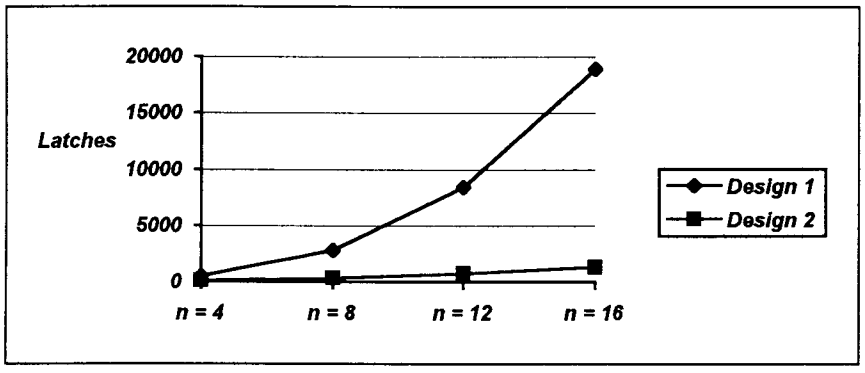

Figure 18. Latches Required for both Designs.

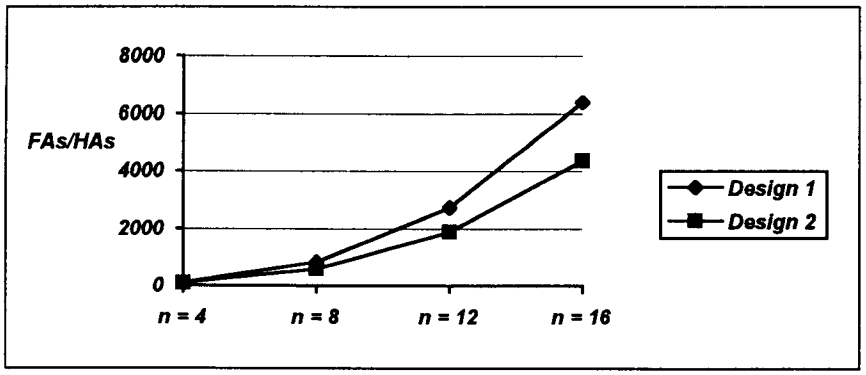

Figure 19. FAs/HAs Required for both Designs.

Size (# of bits n)	*4*	*8*	*12*	*16*
ANDs/NANDs	104	592	1848	4256
Latches	504	2800	8424	18912
FAs/HAs	112	832	2736	6400

Table 1. Hardware Required for Design 1.

Size (# of bits n)	*4*	*8*	*12*	*16*
ANDs/NANDs	64	5122	1727	4096
Latches	88	336	774	1312
FAs/HAs	80	576	1872	4352

Table 2. Hardware Required for Design 2.

The number of AND/NAND gates is about the same for both designs. Design 2 requires about 30% less FAs/HAs. The number of latches, however, differs by a factor of 6 to 15. Here, the gap between both designs is so large because the number of latches in design 1 grows in the order of n^3, whereas the number of the latches in design 2 grows in the order of n^2.

The FAs/HAs are the most costly elements. According to the CMOSN Cell Notebook [6], the size of one FA is 48µ x 150µ, one latch is 42µ x 150µ, and AND/NAND gates are an average of 21µ x 150µ. By weighing the number of elements with the area the elements require, we get some information about the total cost of the logic. As the cost for both logic and wiring grows in the same order, the data is also a good basis for the total hardware cost of both designs. Figure 20 shows the results of the analysis. All numbers are scaled to 1. As we can see, the hardware cost of design 1 exceeds the hardware cost of design 2 by factor 3.

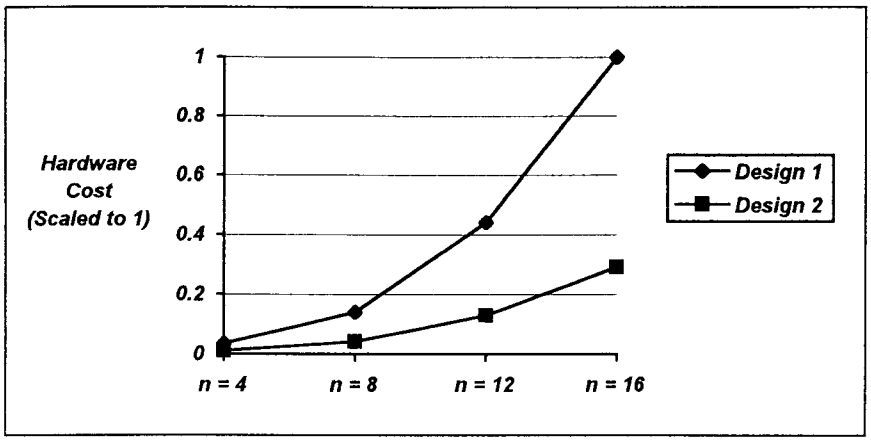

Figure 20. Overall Hardware Cost of both Designs.

5.2 Speed

The clock cycle time and the pipeline depth are the input parameters to the speed of the circuit. The clock cycle time t_c is the only parameter that determines the maximum frequency of the design. The product of the clock cycle time t_c and the depth of the circuit d determines the latency of the circuit. In most application, for instance video signal processing, a high performance in frequency is more important than a high performance in latency. In some applications, for instance control units with feedback loops, the performance in latency is more crucial. The simulation was implemented with Viewsim, the simulation package within the Powerview software [1]. Using the simulation parameters given in the CMOSN Cell Notebook [6], we obtained the data shown in Figures 21 and 22.

Figure 21 shows the clock cycle time of both circuits. In design 1, the final CLA stage of the PAD is pipelined. Thus, the most time consuming stages are found within the accumulation process of the partial products. The accumulation process only requires the sequential data flow through a latch and a full adder, which takes a maximum of 4ns. So, design 1 provides a 250-Msample/s FIR-filter. In design 2, there is another characteristic. The CSA stages and the final

CLA process are not pipelined. For n≤9, the final CLA in PAM2 consumes the most time of all stages. When n>9, however, the CSA tree takes the position of the critical path. This is due to the characteristic of the CSA method that grows in depth when the number of operands increases. Figure 21 illustrates this characteristic. When n exceeds 9, 13, 19 or 28 bits, the depth of the CSA tree increases by one and the clock cycle time increases by 4ns. Figure 22 shows the latency of both FIR-circuits. The figure does not include the latency of the ADC. The latency is measured from the point where all n ADC outputs of a conversion k are available to the point where the filtered signal y(n) is ready.

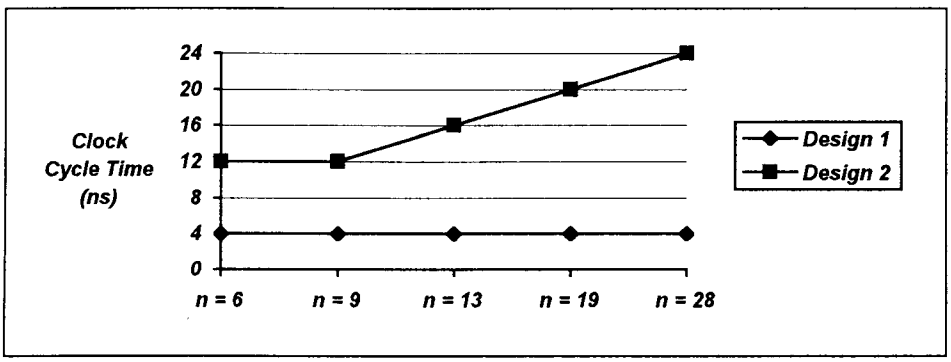

Figure 21. Clock Cycle Time of both Circuits.

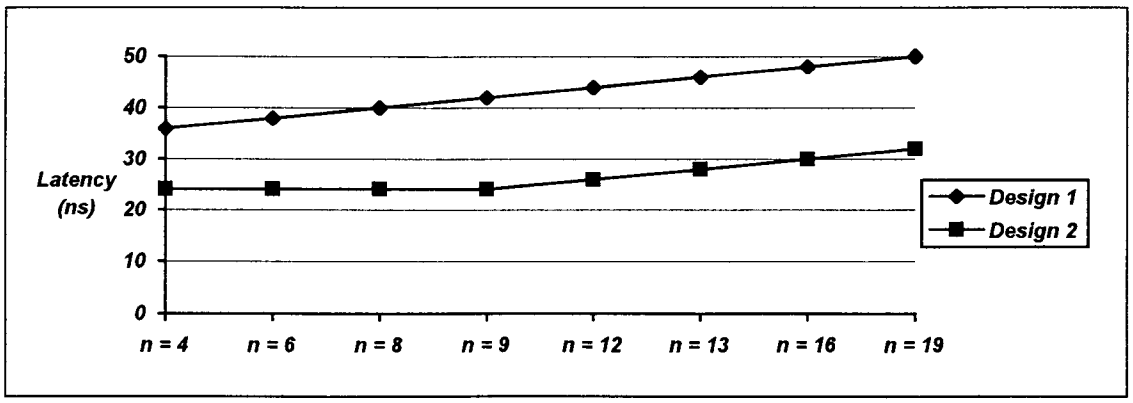

Figure 22. Latency of both Designs.

As shown in figure 21, design 1 provides a constant clock cycle time of 4ns. Thus, the latency of this design only varies with the pipeline depth d. The pipeline in design 1 consists of a basic number of stages plus some more variable stages:

- 1 stage to generate the partial products.
- 1 stage to accumulate the partial products.
- 3 stages to accumulate the carry and sum output of the 2 PAD trees.
- $(\log_2 n) + 1$ stages to generate the carrys for the final addition.
- 1 XOR stage to generate the final result.

This makes a total of $8+\log_2 n$ cycles. With a clock cycle time of 4ns the total latency is $32ns+4(\log_2 n)ns$.

Design 2, in contrast, provides a constant number of stages but a variable clock cycle time. The clock cycle time grows in order of $o(c + \log_{3/2} n)$, where c is the constant part, and $\log_{3/2} n$ is the variable part that describes the increasing depth of the CSA tree. The pipeline in design 2 only requires two stages:

- 1 stage to generate and accumulate the partial products.
- 1 stage to generate the final result.

As we can see in Figure 21, up to $n \leq 9$ bits, the total latency for design 2 is 24ns. For $n>9$ bits, the total latency is $4ns+4(\log_{3/2} n)ns$, where $\log_{3/2} n$ has to be rounded to the next integer.

6. SUMMARY AND CONCLUSION

The comparison mentioned in Chapter 5 showed the strengths and weaknesses of both designs. Design 1 provides a high sample rate of 250-Msamples/s, which has been achieved by a high degree of pipelining. This leads to a certain overhead in hardware. About one third of the hardware as well as the processing time in design 1 is consumed by latches. Design 2 limits this overhead by decreasing the degree of pipelining. By means of one third of the hardware required in design 1, design 2 processes about 80-Msamples/s which is about one third the sample rate of design 1. A look at the performance of pipelined ADC techniques, however, shows that the speed of design 2 is absolutely sufficient. Pipelined ADCs with 10 bits or more provide sample rates in the range up to about 20-Msamples/s [7].

Both FIR-designs exceed the speed of digital signal processors (DSPs) used for filtering by far. Even though DSP-FIRs provide a higher degree of flexibility, their speed is limited to less than 1/30 of the speed provided by both designs shown. Programmable DSPs are able to vary k, the number of samples considered in each process, whereas both designs shown limit the number of samples considered in the process to n. In other regards, both designs shown provide the same flexibility as DSPs. As the weighted factors in both designs can be set to zero, k can also be less than n. Inputs and outputs of both designs shown are in 2's complement format. Thus, it is possible to limit the width of the circuit to less than n. This can be simply implemented by extending the sign of the most significant input bit available to the input bits not considered.

This work showed two pipelined FIR-filter architectures, designed for a interfaced ADC-FIR device. Both design are featured by two techniques: msb-first multiplication and 2-dimensional pipelining. Both designs meet the speed requirements of the latest pipelined ADC circuits. It can be concluded that there is a high potential in ADC-FIR devices. A further increase in speed of pipelined ADCs will give the architectures proposed even greater potential.

BIBLIOGRAPHY

[1] Viewlogic, *Powerview Manual*, Viewlogic Systems Inc., 1992.

[2] Shih-Lien Lu, Jack Kenney, "*Design of a Most-Significant-Bit-First Serial Multiplier,*" Internal Research, Dept. of ECE, Oregon State University, Corvallis, OR 97330, June 1994.

[3] Richard E. Ladner, Michael J. Fischer, "Parallel Prefix Computation," *Journal of the Association for Computing Machinery*, vol. 27, no. 4, pp. 831-838, Oct. 1980.

[4] Richard P. Brent, H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. on Computers*, vol. C-31, no. 3, pp. 260-264, March 1982.

[5] C. S. Wallace, " A Suggestion for a Fast Multiplier," *IEEE Trans. on Computers*, vol. EC-13, pp. 14-17, Feb. 1964.

[6] The MOSIS Service, *CMOSN Cell Notebook*, Release 3.0A, 1991.

[7] Stephen H. Lewis, Scott Fetterman, George F. Gross, Jr., R. Ramachadran, T. R. Viswanathan, " A 10-b 20-Msample/s Analog-to-Digital Converter," *IEEE J. Solid-State Circuits*, vol. 27, no. 3, pp. 351-358, March 1992.

APPENDICES

APPENDIX A: SOFTWARE MODEL DESIGN 1

Appendix A illustrates the software model of design 1, implemented in Powerview [1]. Figure A1 shows an overview of the design. Figures A2-A13 show the schematics illustrated in Figure A1 in detail.

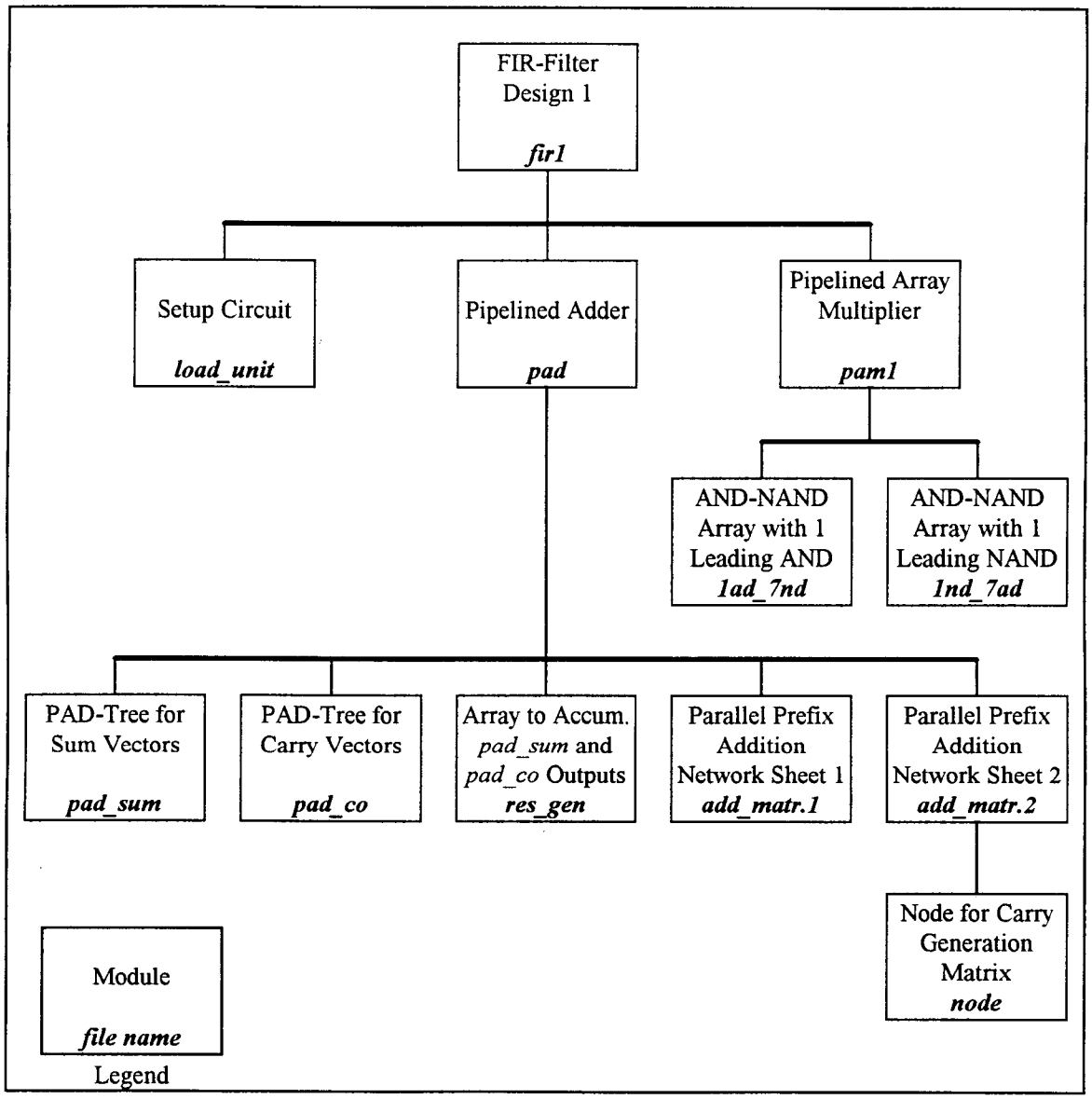

Figure A1. Hierarchy of Design 1.

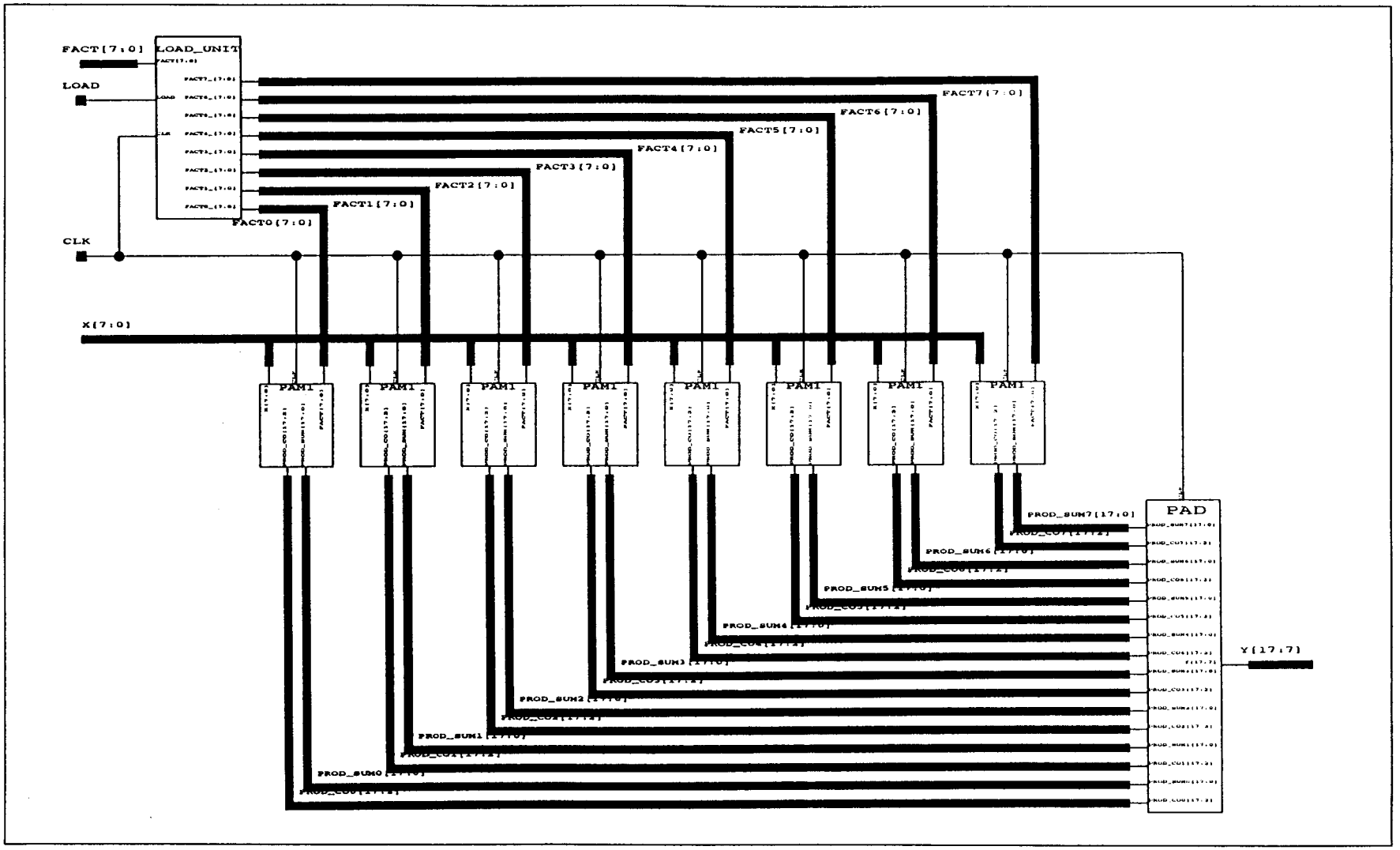

Figure A2. FIR-Filter Design 1.

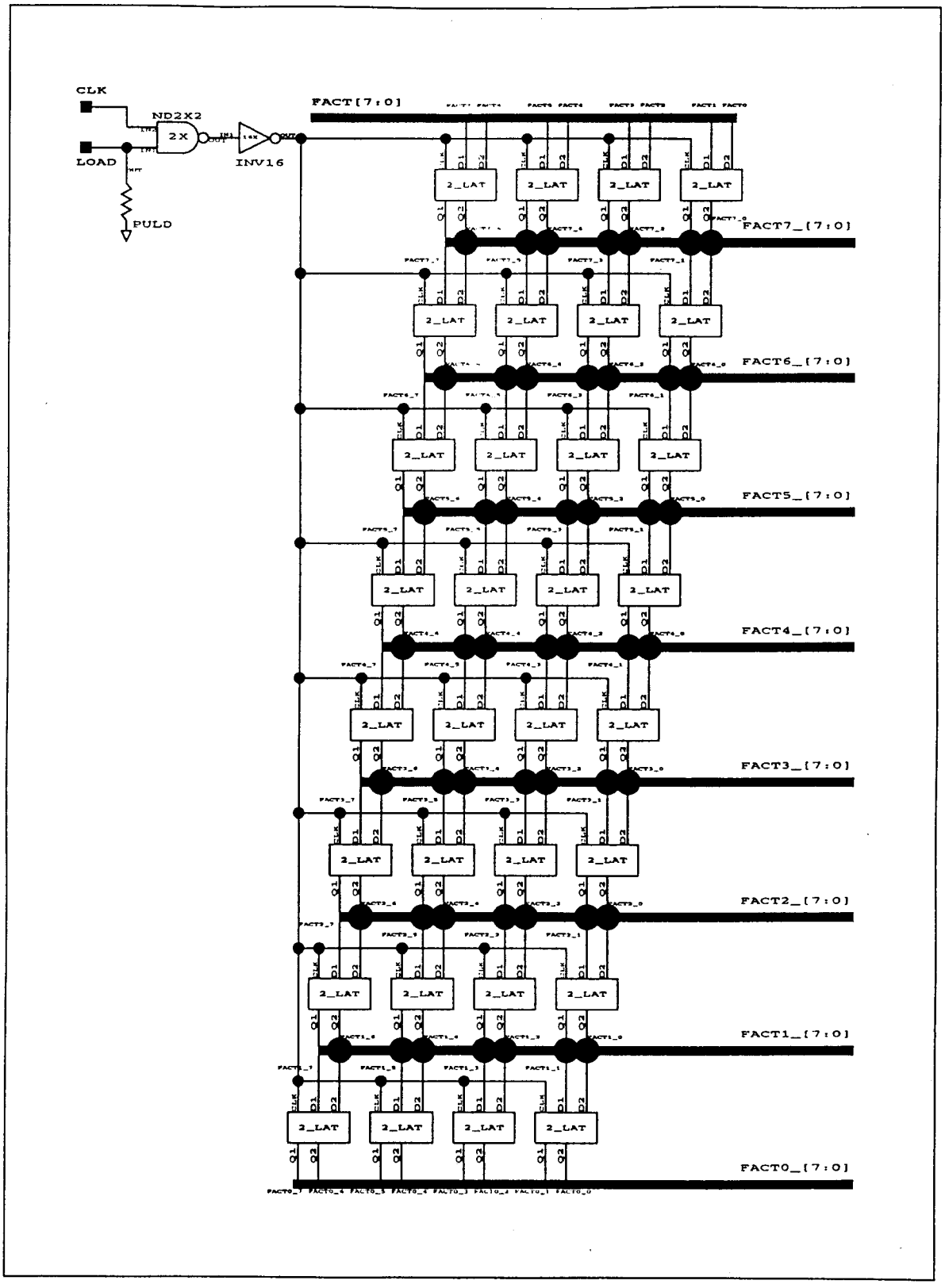

Figure A3. Setup Circuit.

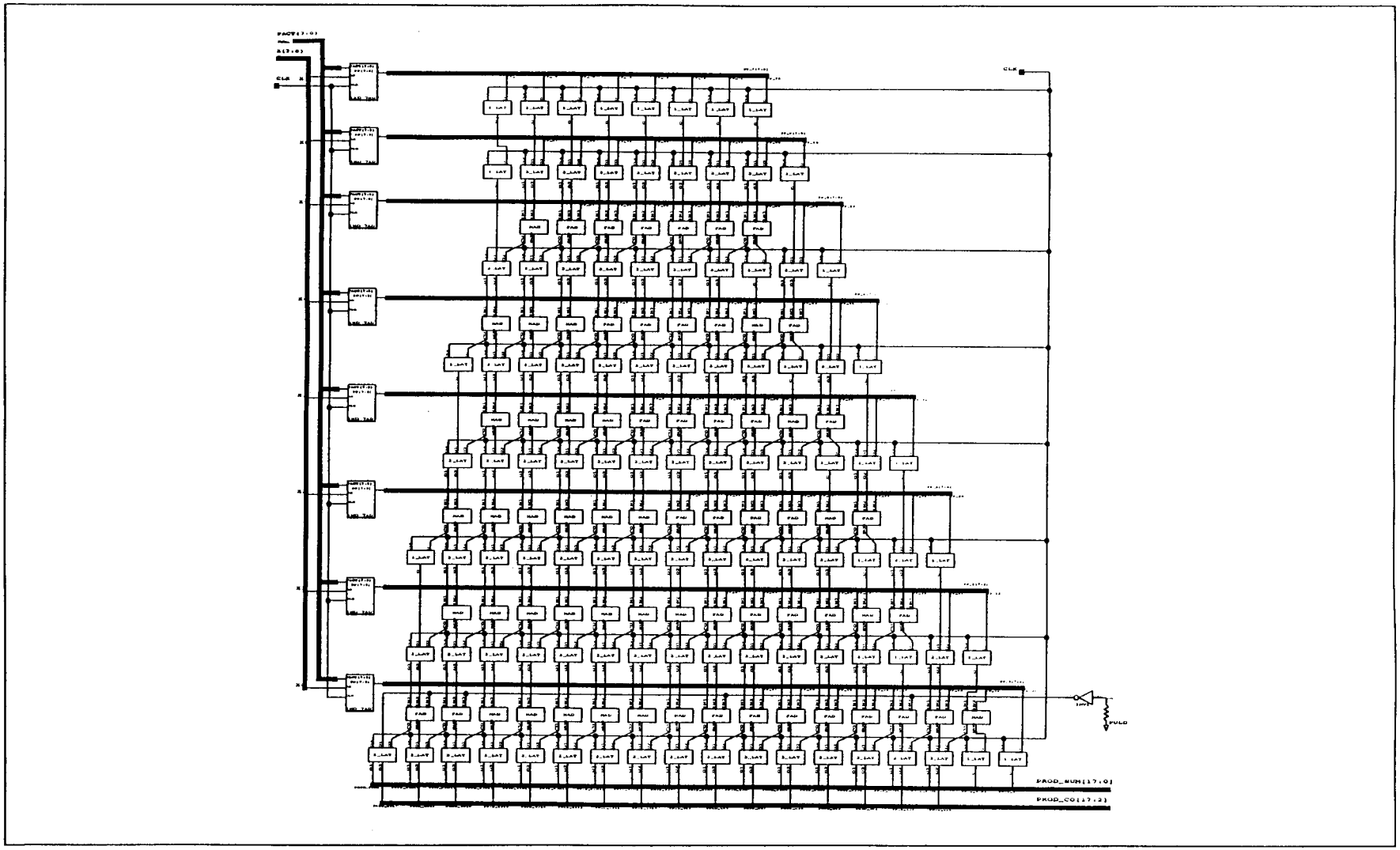

Figure A4. Pipelined Array Multiplier 1.

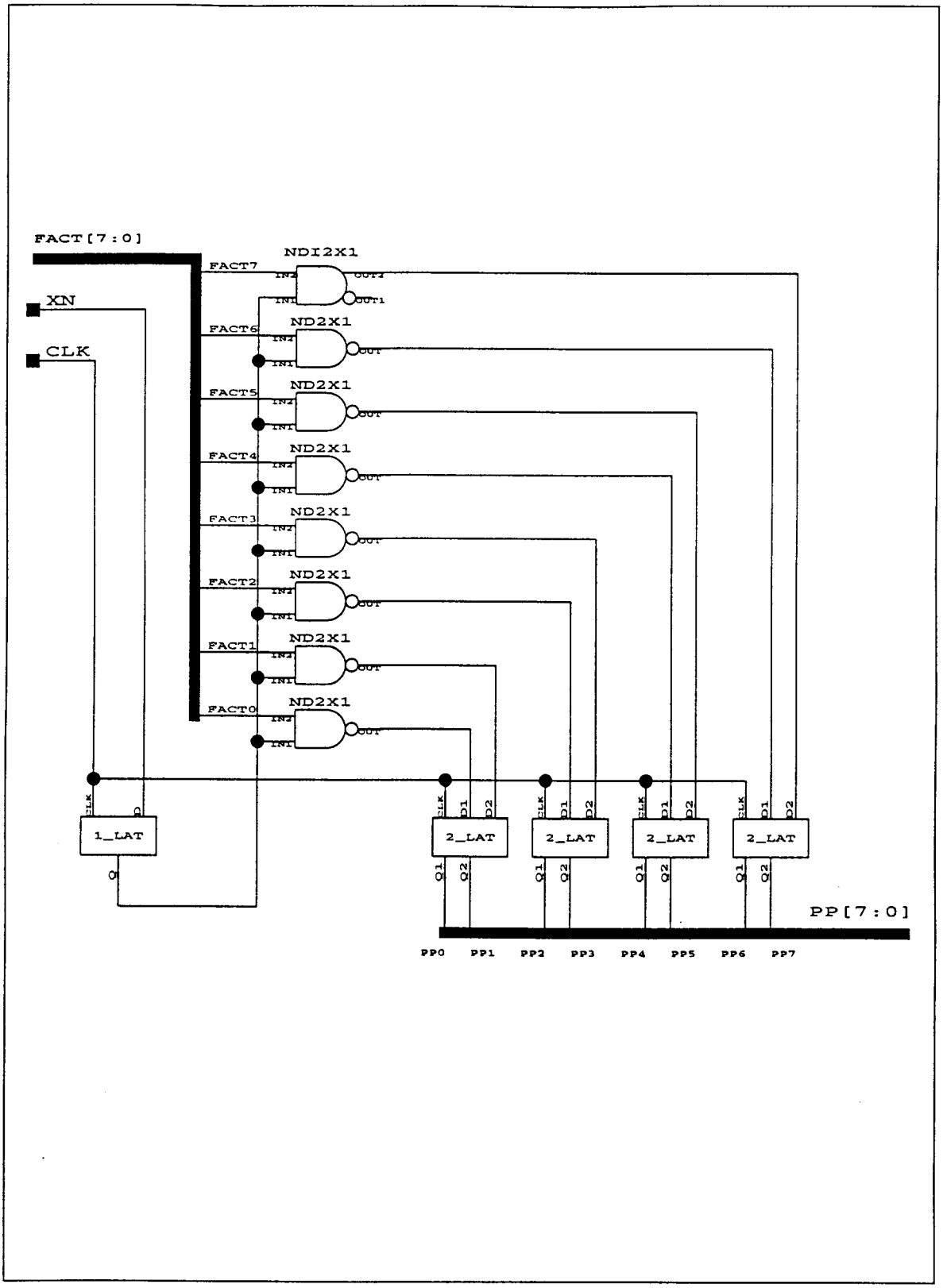

Figure A5. AND-NAND Array with 1 Leading AND.

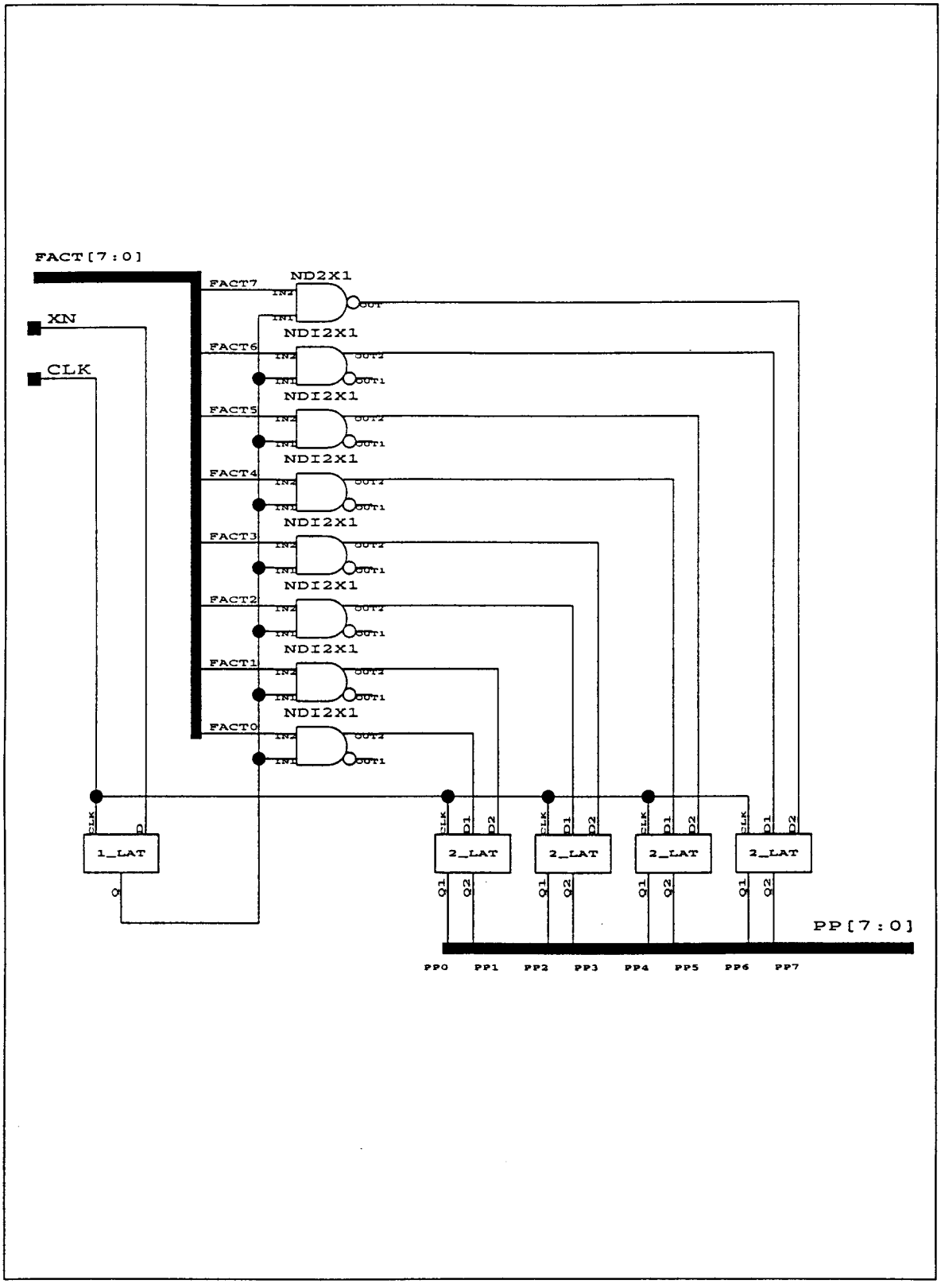

Figure A6. AND-NAND Array with 1 Leading NAND.

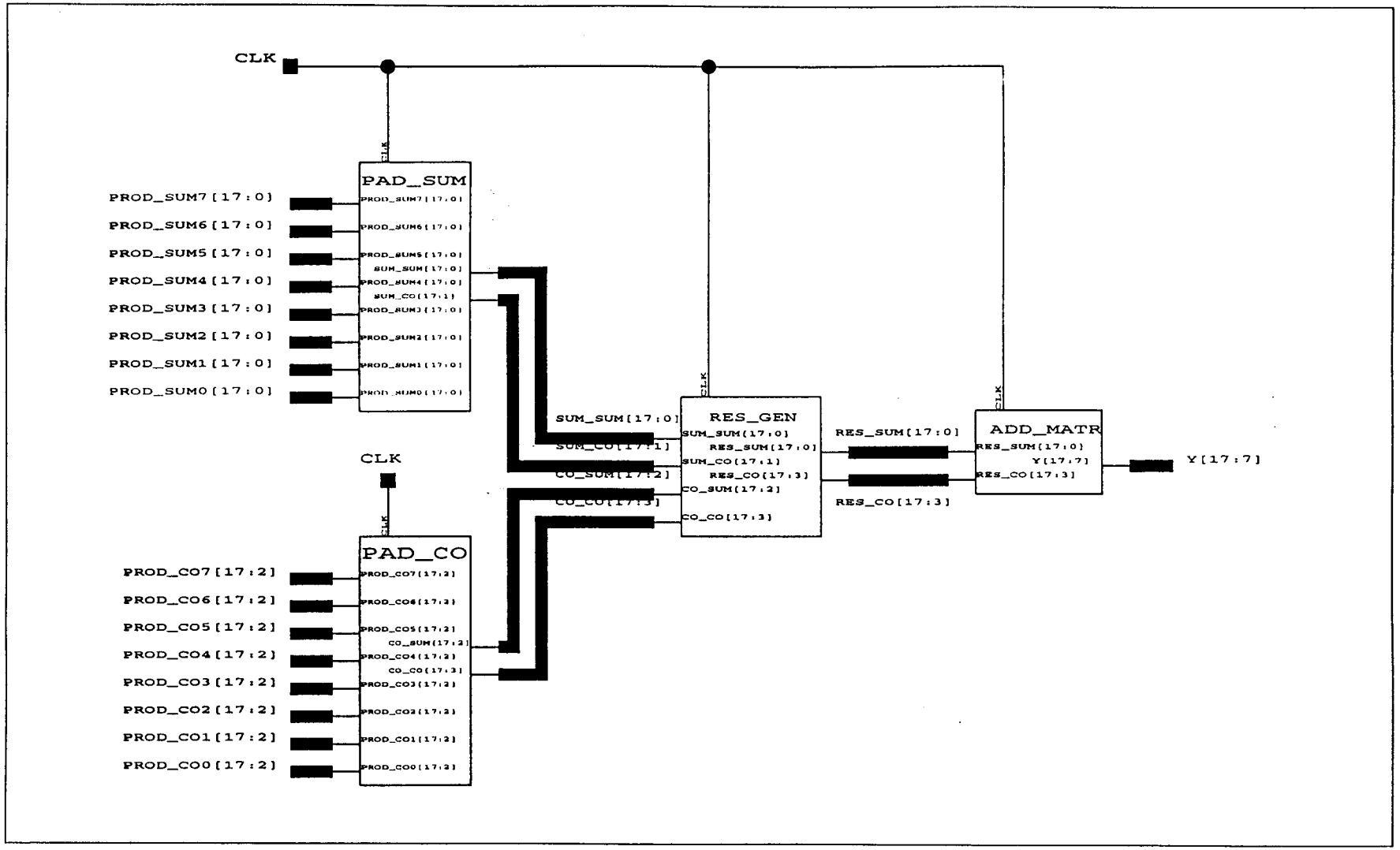

Figure A7. Pipelined Adder.

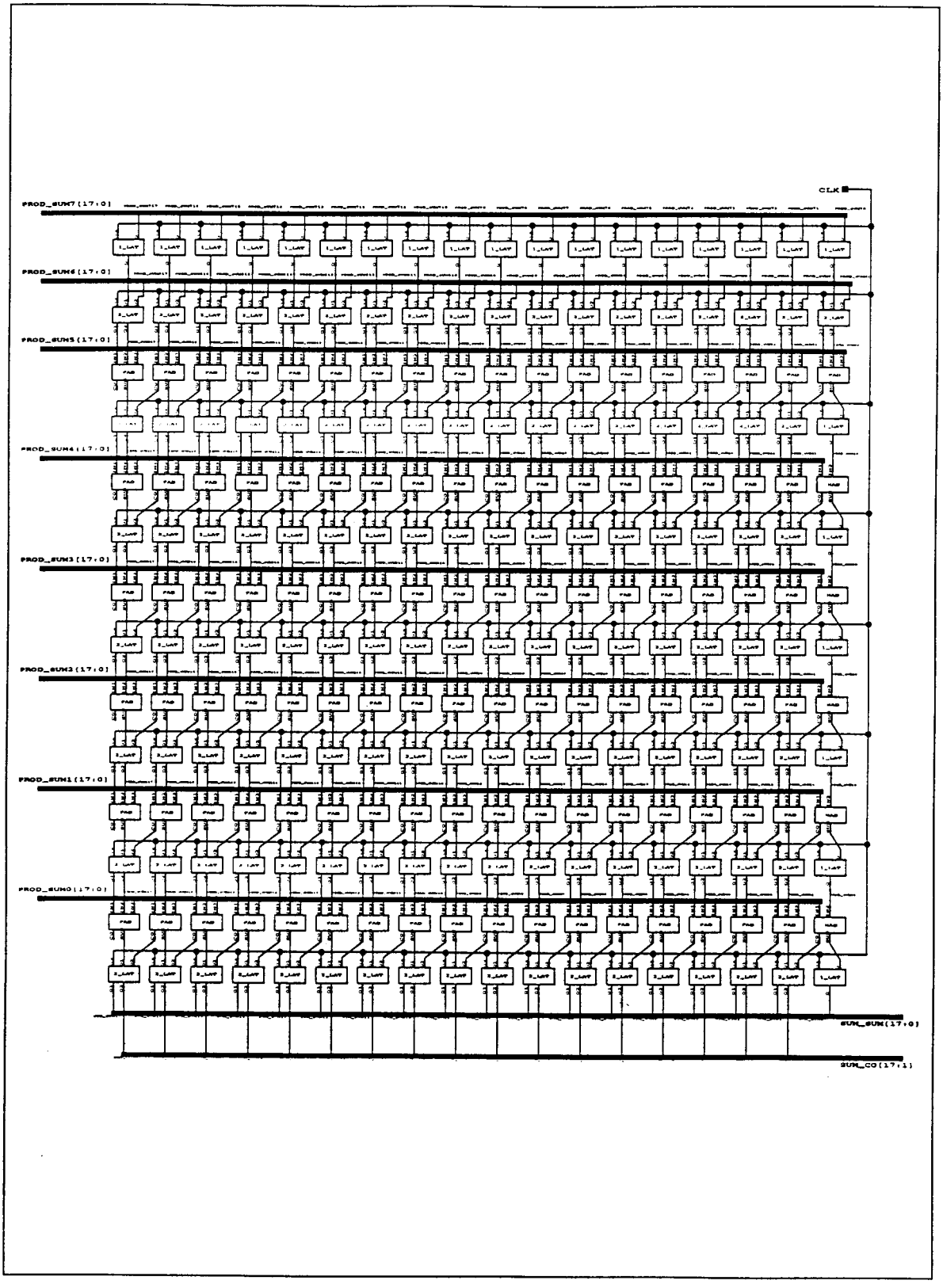

Figure A8. PAD-Tree for Sum Vectors.

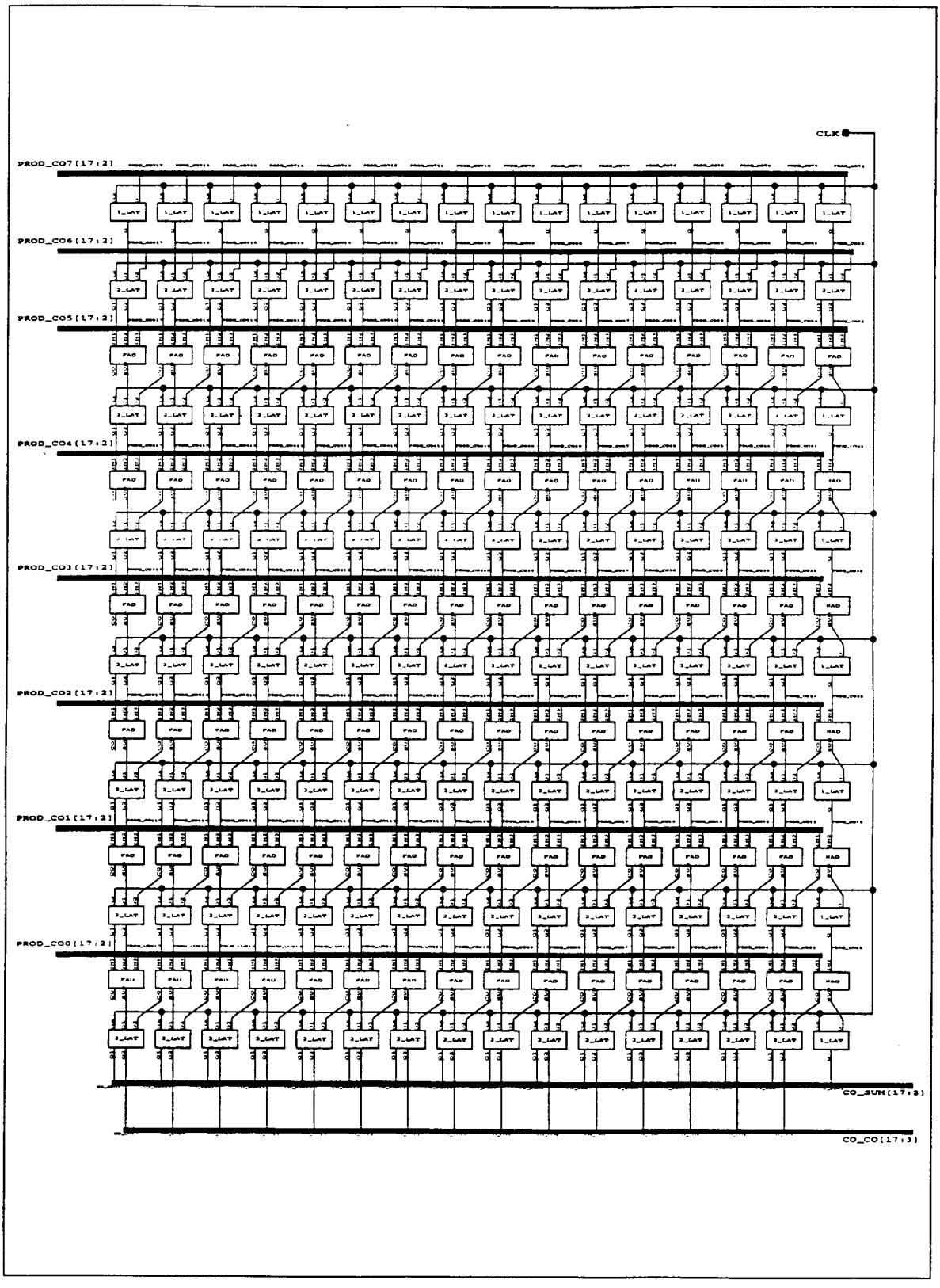

Figure A9. PAD-Tree for Carry Vectors.

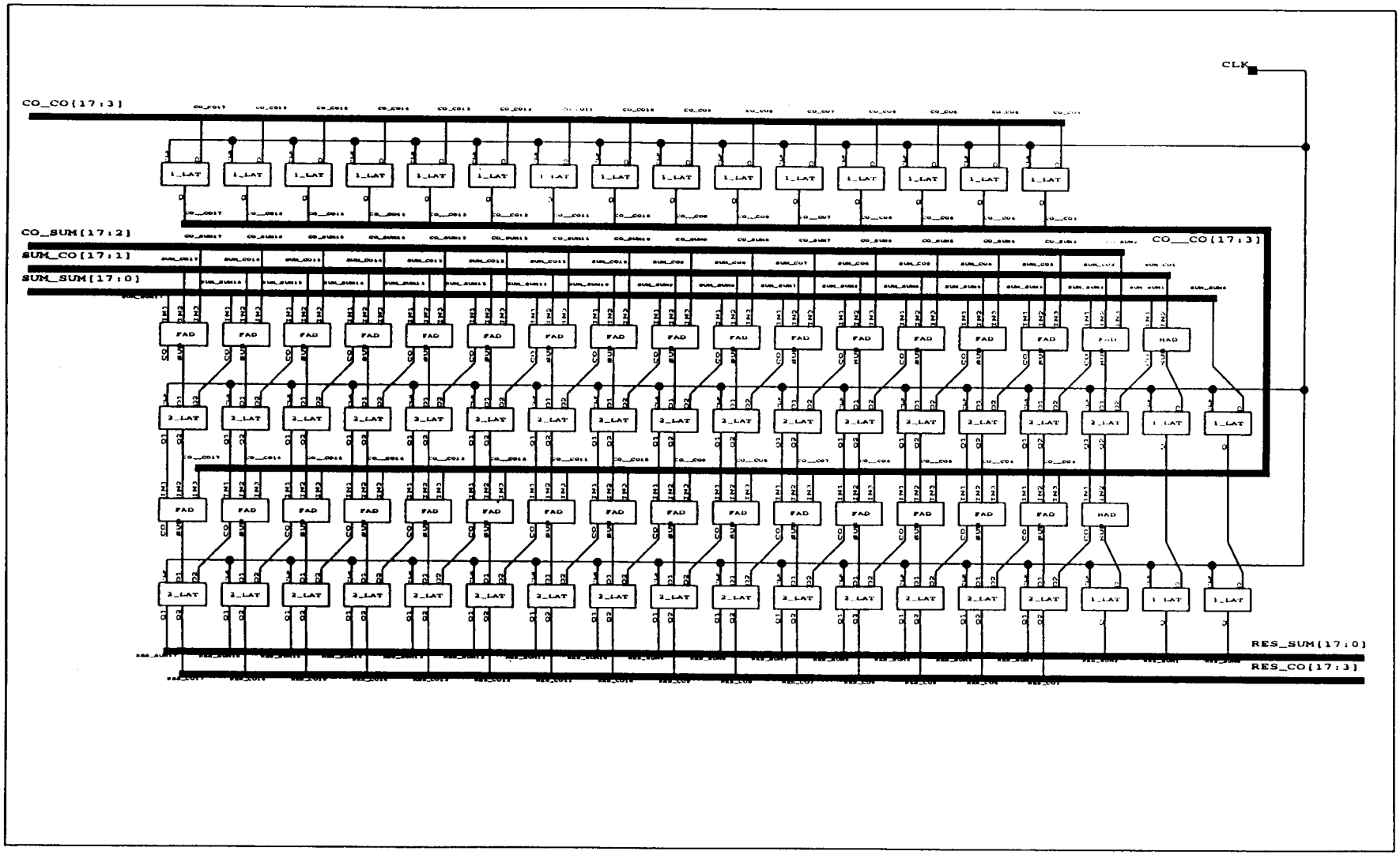

Figure A10. Array to Accumulate Carry and Sum Outputs of both PAD-Trees.

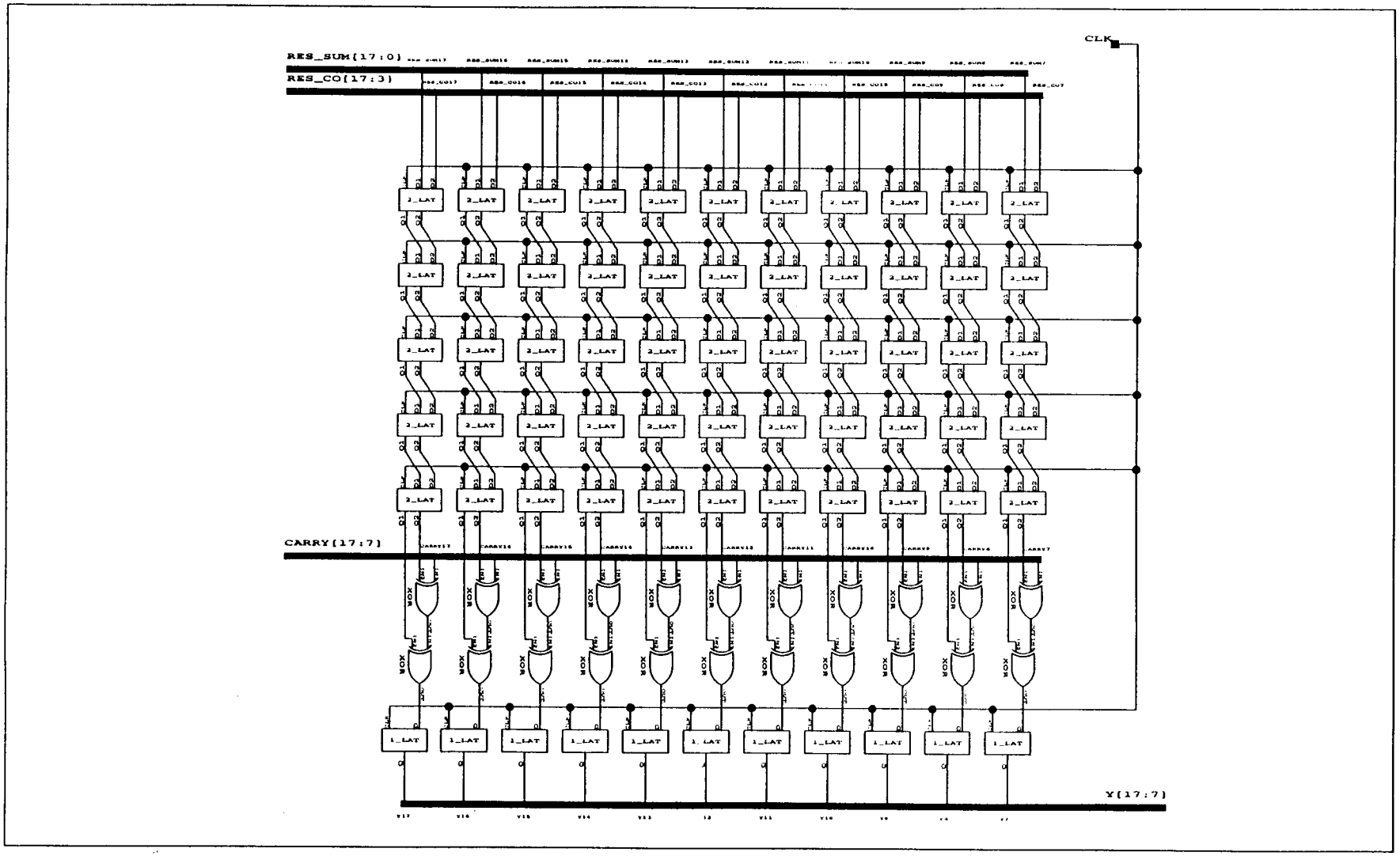

Figure A11. Parallel Prefix Addition Network Sheet 1.

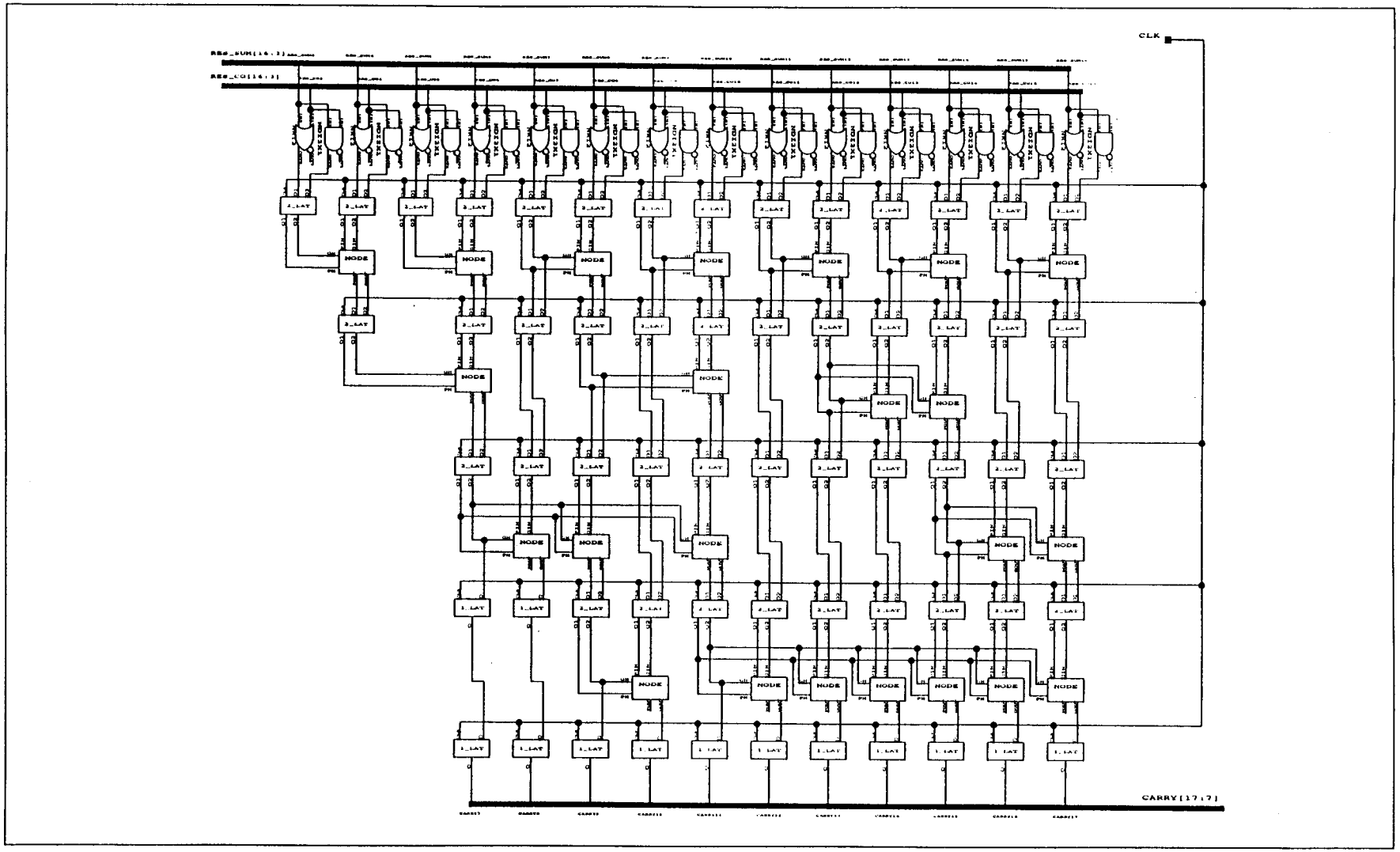

Figure A12. Parallel Prefix Addition Network Sheet 2.

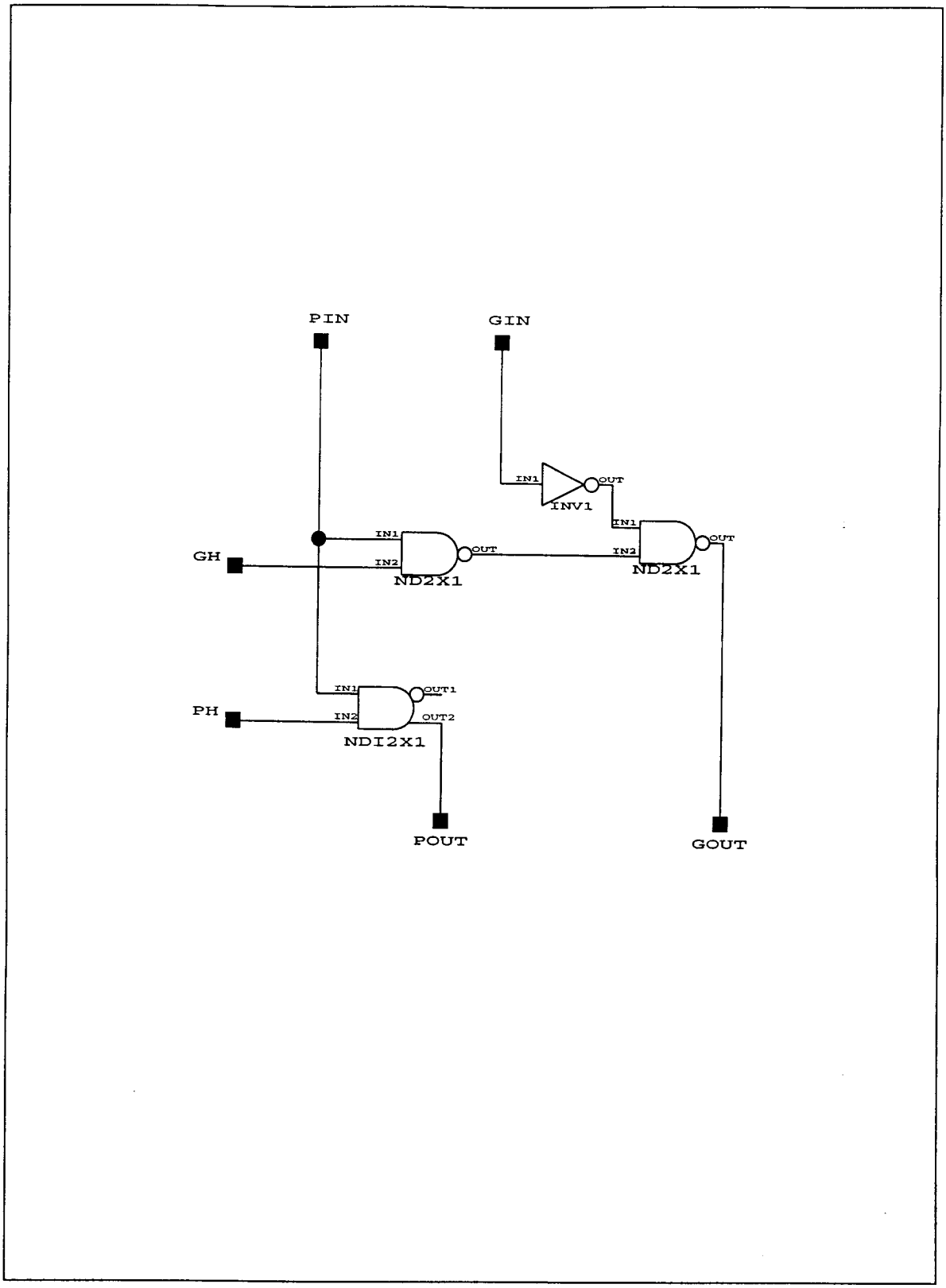

Figure A13. Node for Carry Generation Matrix.

APPENDIX B: SOFTWARE MODEL DESIGN 2

Appendix B illustrates the software model of design 2. Three of the modules used in design 1 can be also found in design 2: The setup circuit and both AND-NAND arrays. Figure B1 shows the hierarchy of design 2. Figures B2-B7 show the schematics illustrated in Figure B1 in detail.

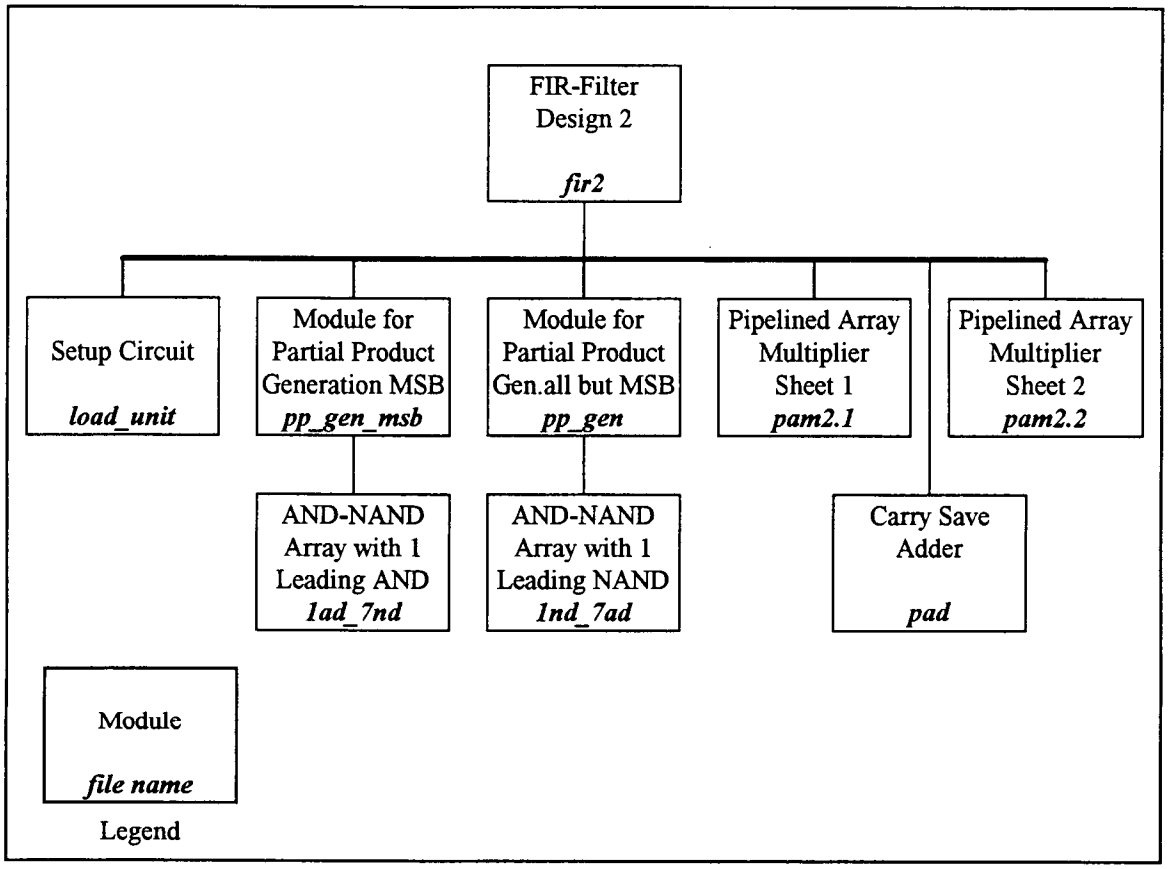

Figure B1. Hierarchy of Design 2.

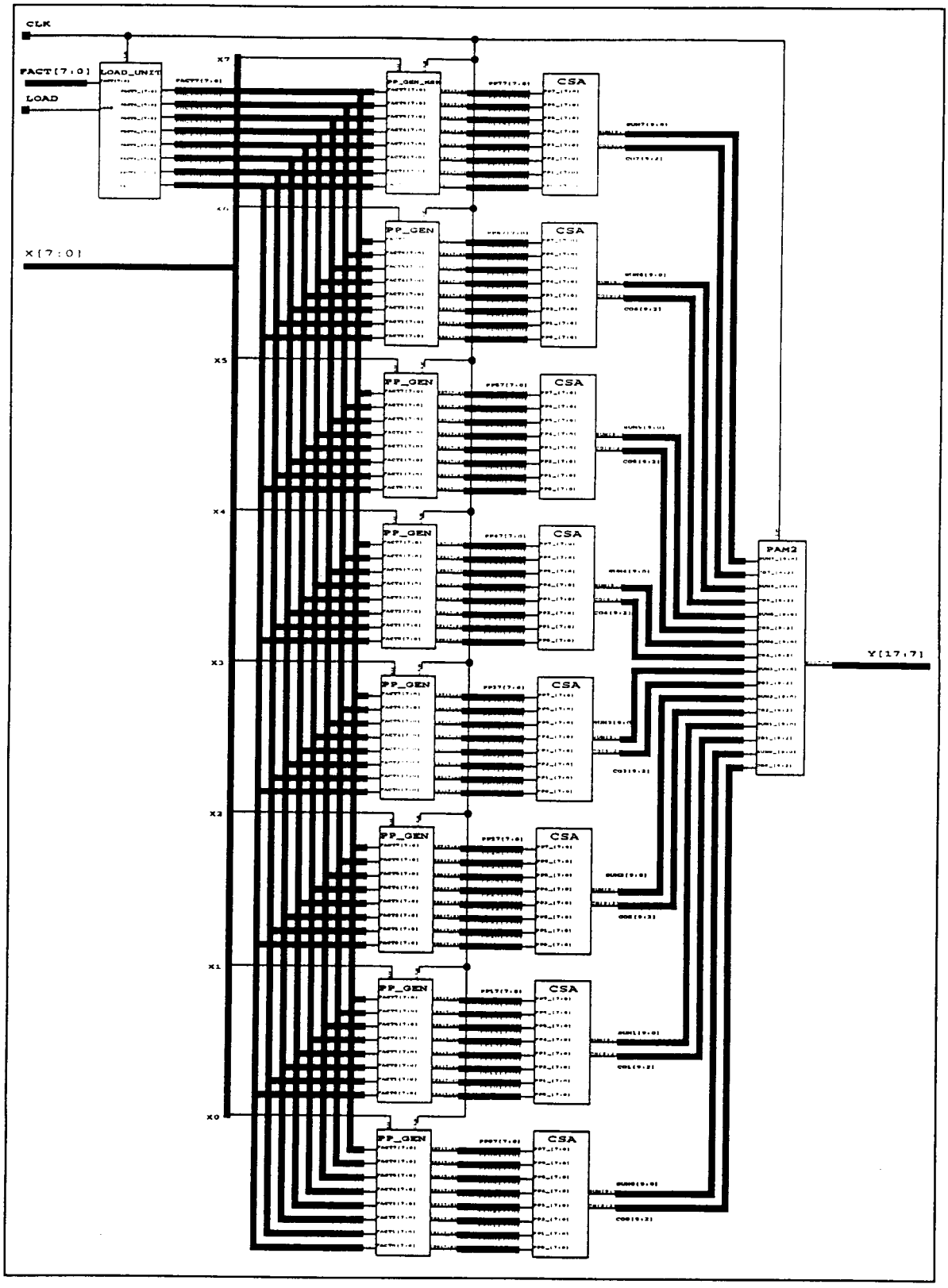

Figure B2. FIR-Filter Design 2.

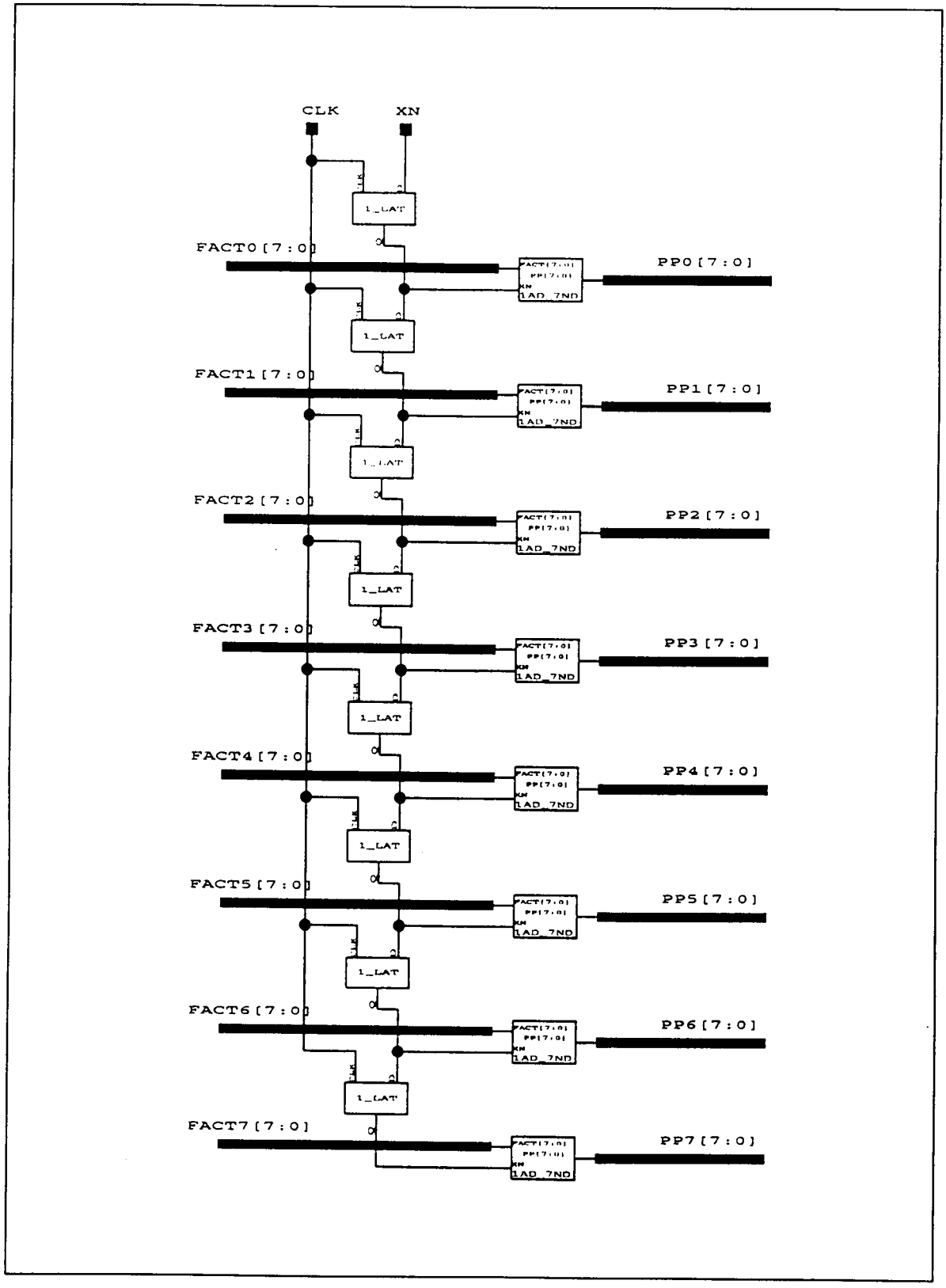

Figure B3. Module for Partial Product Generation MSB.

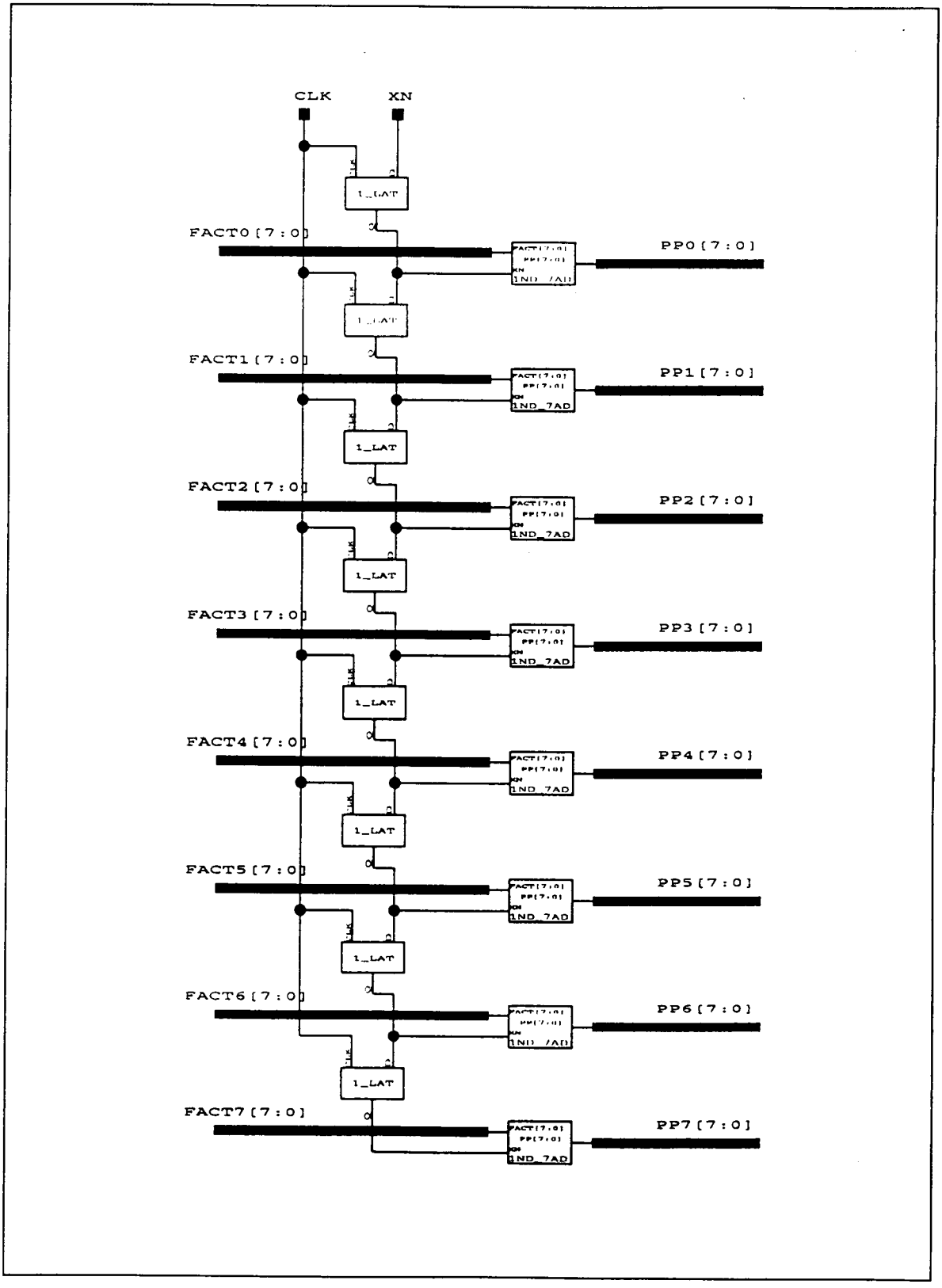

Figure B4. Module for Partial Product Generation all but MSB.

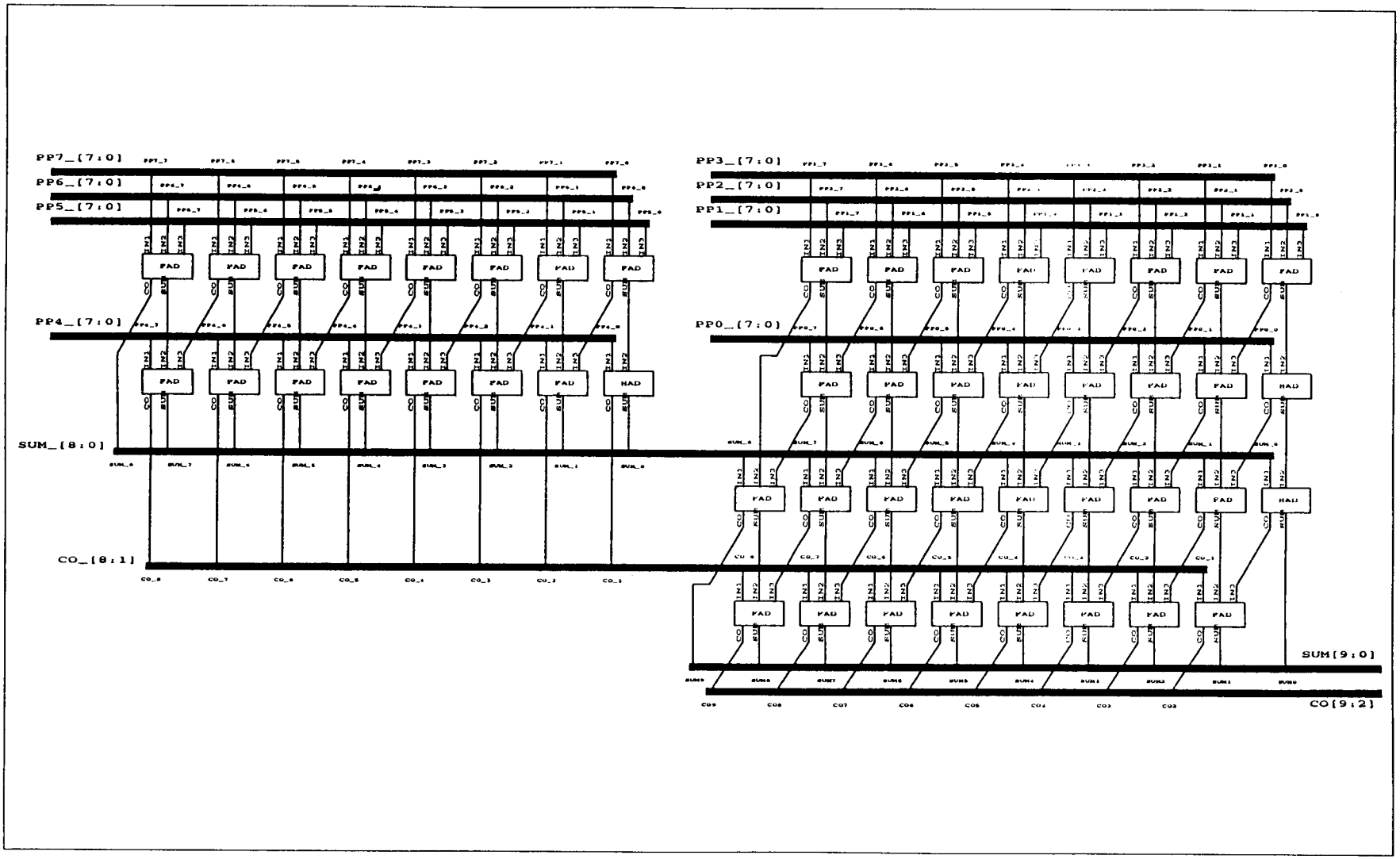

Figure B5. Carry Save Adder.

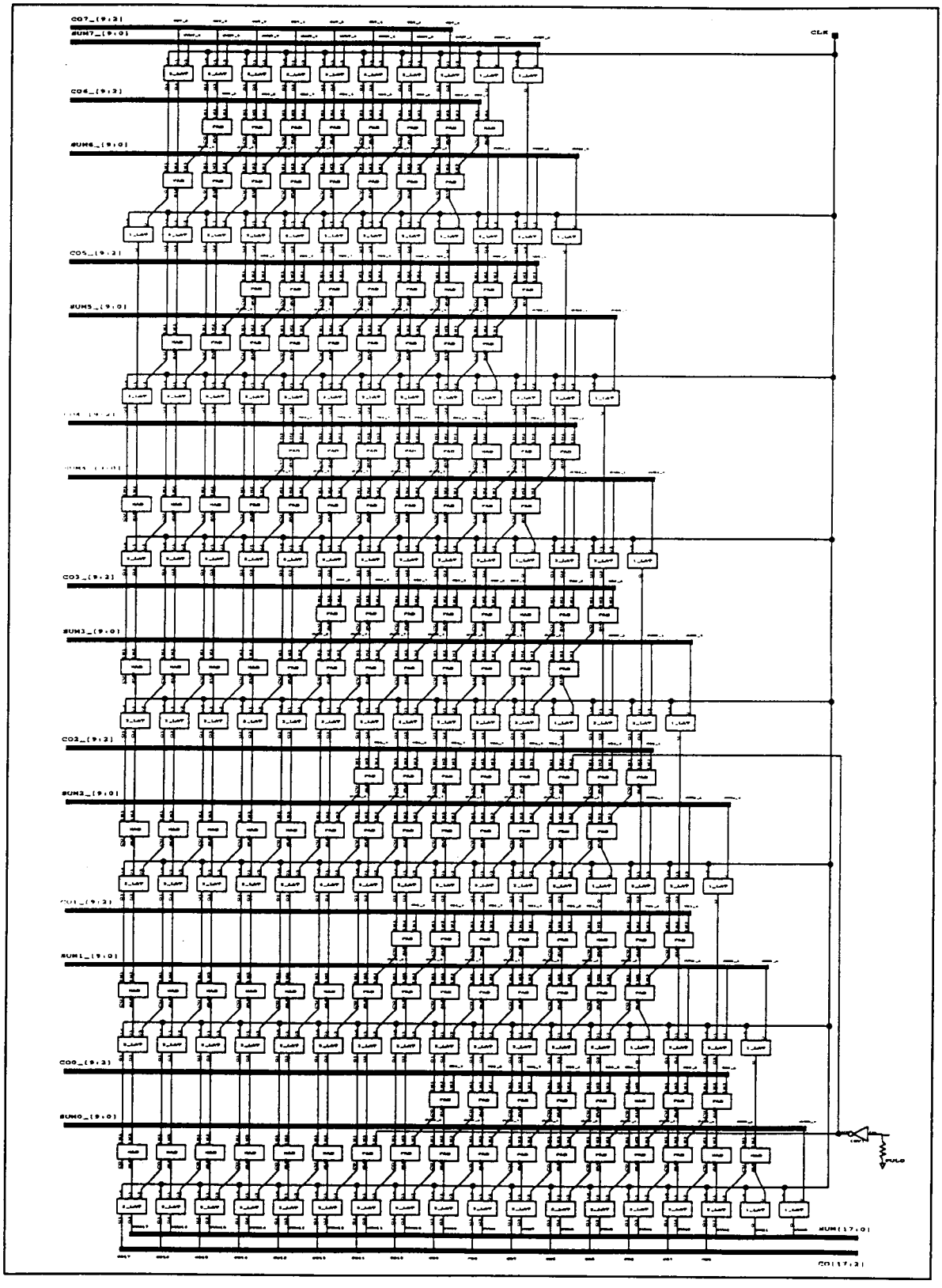

Figure B6. Pipelined Array Multiplier 2 Sheet 1.

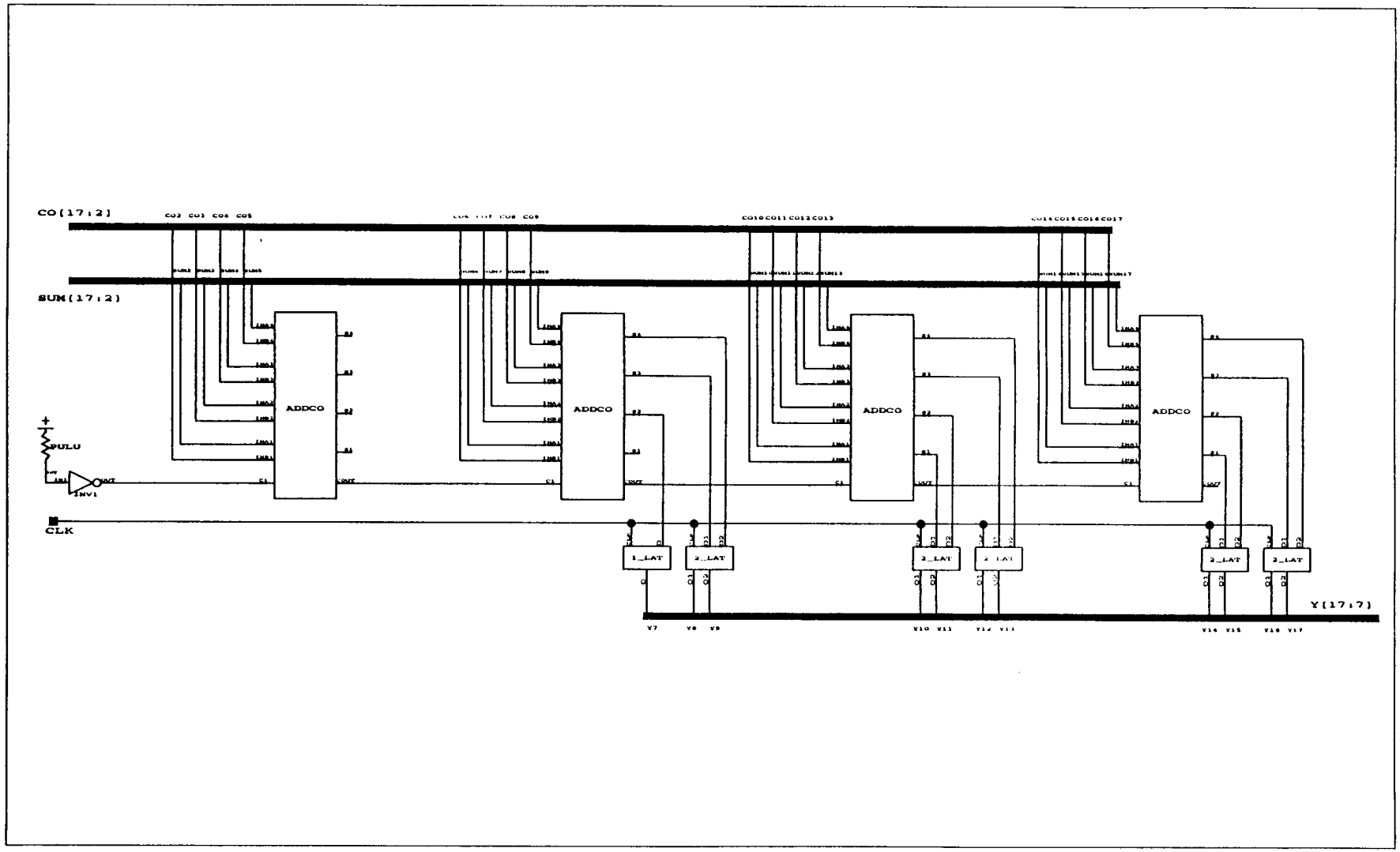

Figure B7. Pipelined Array Multiplier 2 Sheet 2.