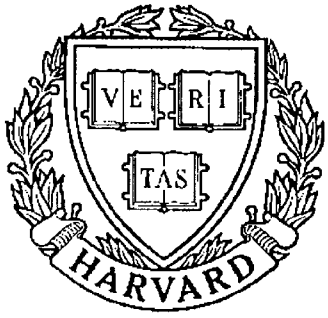
THESIS REPORT
Master's Degree

S Y S T E M S
R E S E A R C H
C E N T E R



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Design, Implementation and Testing of an 8 x 8 DCT Chip

by: S. Sachidanandan
Advisor: J. JáJá

M.S. 89-5
Formerly TR 89-54

Design, Implementation and Testing of an 8×8 DCT Chip [1]

Sambandan Sachidanandan
Department of Electrical Engineering
and
Systems Research Center
University of Maryland
College Park, MD. 20742
(301)454-0522
E-mail: parveen@vlsi5.src.umd.edu

and

Joseph Já Já
Department of Electrical Engineering
Institute For Advanced Computer Studies
and
Systems Research Center
University of Maryland
College Park, MD. 20742
(301)454-1987
E-mail: joseph@wafer.src.umd.edu

[1]Supported in part by NSF grant number DCR86-00375, NSF Engineering Research Centers Program: NSF CDR 88003012, and by Ford Aerospace Corporation under the MIPS program

ABSTRACT

Title of Thesis:	Design, Implementation, and Testing of an 8×8 DCT chip
Name of degree candidate:	Sambandan Sachidanandan
Degree and Year:	Master of Science, 1989
Thesis directed by:	Dr. Joseph F. JaJa Professor Department of Electrical Engineering

An implementation of a fully pipelined bit serial architecture to compute the 2-D Discrete Cosine Transform of an 8×8 element matrix is presented. The algorithm used requires the minimal number of multipliers to perform the computation. The basic hardware components required by our implementation are the one bit serial adder, one bit serial subtractor, one bit pipeline multiplier and the dynamic shift register. We use two's complement arithmetic. An internal precision of 18 bits is maintained throughout the chip. We have used the two phase non-overlapping clocking scheme. The basic architecture consists of an 1-D ROW DCT followed by a TRANSPOSE operation and another 1-D COLUMN DCT. All the components were custom designed and simulated at various levels. The chips were laid out using the layout editor MAGIC and were fabricated by MOSIS. The chips were

tested using the IMS Logic Master. The results of the simulation and testing are also presented here. The throughput is very high and inputs can be processed successively with no delay and just two controls. The chip is designed to operate at clock speeds of 10Mhz or more.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER
ONE

INTRODUCTION

In dealing with imagery a rapidly growing need for higher bandwidth and/or larger storage capacity along with the attendant increases in the cost of such bandwidth and/or storage expansion, is of great concern in the fields of Communications and Computers. A solution to this problem is image compression. An efficient image compression scheme can result in considerable reduction in transmission bandwidth and storage requirements.

In a typical transform coding system, the image is segmented into blocks of equal size and each block is operated upon by a linear transformation in an effort to reduce (or eliminate) correlation between coefficients. The transform coefficients are then quantized by a scalar quantizer.

The Discrete Cosine Transform chip presented here will form a part of one such image compression system. The system is based on a 2-D block cosine transform coding scheme, where the image is of size 256×256 and each block is of size 8×8. Each entry of the image is assumed to be represented by an 8-bit binary integer. There are two main computational tasks involved. The first consists of computing the 2-D Discrete Cosine Transform (DCT) on blocks of size 8×8, while the second task consists of quantizing the transform coefficients using scalar quantizers. We

present an implementation of a chip that computes the DCT of an 8 X 8 element block.

The DCT was presented by Ahmed et al [1] in 1974. They showed that the performance of the transform compares more closely to that of the Karhunen-Loeve transform relative to the performances of the Discrete Fourier Transform, the Walsh-Hadamard Transform and the Hartley Transform. The Karhunen-Loeve transform is known to be optimal with respect to variance distribution, estimation using the mean- square error criterion and the rate-distortion function.

Let $X_{i,j}, i, j = 0, 1, ..., L-1$, be a block, denoted by X, such that each entry is an 8-bit integer. A 2-D DCT operating on blocks of size $L \times L$ is described by:

$$Y_{k,l} = \frac{2}{L} C_k C_l \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} X_{i,j} cos \frac{(2i+1)k\pi}{2L} cos \frac{(2j+1)l\pi}{2L}, k, l = 0, 1, ..., L-1,$$

where $C_0 = 1/\sqrt{2}$ and $C_k = 1, k = 1, 2, ..., L-1$. In our case $L = 8$. Here, $Y_{k,l}$ is called the (k,l)th transform coefficient. The above formula shows that the 2-D DCT can be computed by applying 1-D DCT to each of the columns of the matrix separately and then applying 1-D DCT to each of the rows separately.

There are several popular methods of computing the one dimensional DCT. Most of these methods use either the one dimensional Discrete Fourier Transform (DFT) or the one dimensional Discrete Hartley Transform (DHT) on permuted data. After a careful analysis of these methods, we found that none of these methods is suitable for our purposes. The method that we used was due to Chen et al [4] . It is similar to the Fast Fourier Transform except that we only use real arithmetic operations. The flow graph is given in Figure 1.1. The network consists

of 5 stages, each of which has 4 elements. It requires 12 multipliers and 29 adders. We use this network to compute the 1-D DCT because it needs the smallest number of multipliers.

Our overall architecture has three main components. The first component consists of the network described above. Its purpose is to compute the DCT of the columns successively in a fully pipelined fashion. The second component, called the Transpose, consists of a 8×8 array of shift registers. Its purpose is to transpose the input matrix after passing through the first DCT network. After this operation, the columns will be ready to be processed through another one dimensional DCT network forming the third main component of our architecture. The design of all the three components has to fully support the pipelining of the incoming data. A block diagram depicting the overall architecture of the chip is shown in Figure 1.2

A description of the overall device specifications are as follows:

1. The device is fully pipelined.

2. The input matrix is fed in bit serially, LSB first, one row at a time.

3. During any clock period, 8 bits, corresponding to the eight elements in a row are input.

4. During any clock period, 8 bits corresponding to the eight elements of a column of the output, are available at the output pins.

5. The device handles input data at the rate of 300 Mbps.

6. An internal precision of 18 bits is maintained throughout the chip.

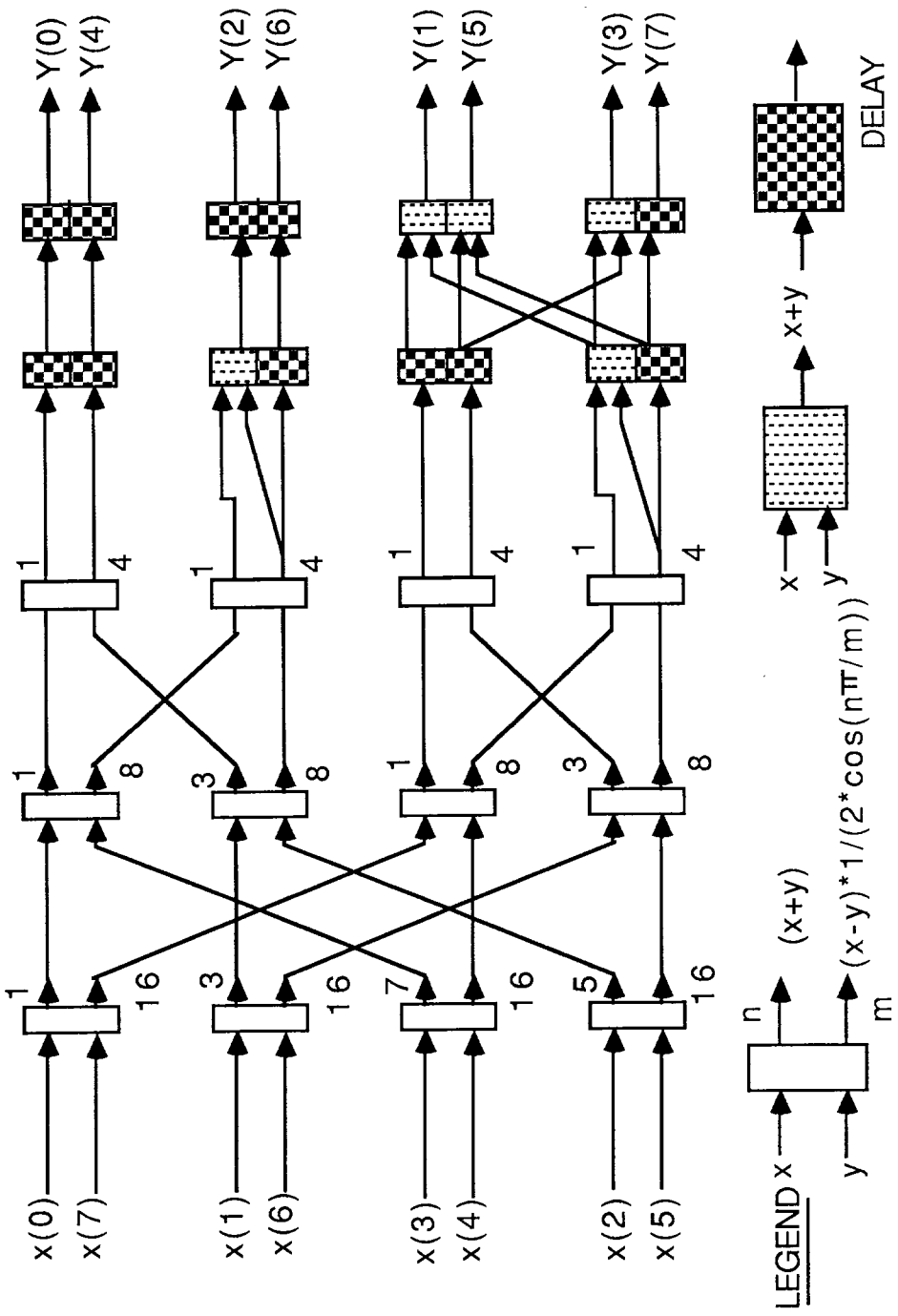


Figure 1.1: Flow Graph for the DCT

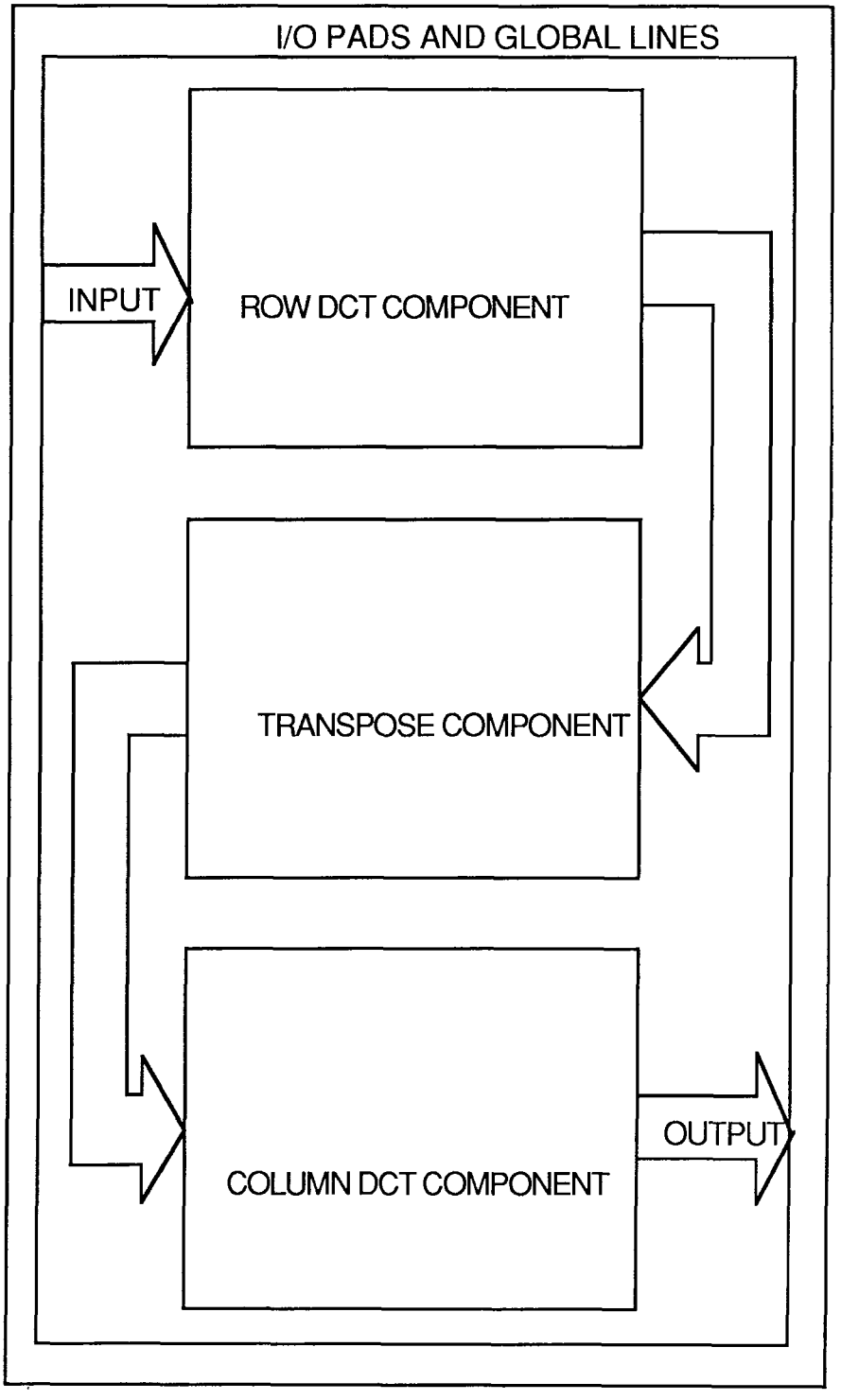


Figure 1.2: Block Diagram of DCT chip

7. Two's complement arithmetic is used.

8. Only two external controls are required.

9. In order to avoid problems due to clock skew and to provide better isolation between subcomponents, a two phase non overlapping scheme is used.

In the next chapter we describe the design, implementation, simulation and testing of the basic subcomponents used in the chip.

CHAPTER
TWO

DESCRIPTION OF THE BASIC COMPONENTS

I Introduction

As described earlier four basic components are required to form the DCT chip. These are:

1. One Bit serial adder.
2. One Bit serial subtractor.
3. One Bit serial multiplier.
4. One Bit Dynamic shift register.

The design, simulation and implementation of these basic components is the subject of this chapter.

II One Bit serial adder

The most significant circuit used in the design of this chip is the bit-serial adder. Its design determines both the speed of operation and the size of the chip. Based

Since the adder must support pipelining at the bit level, it should

on these considerations, we decided to use the circuit introduced by Ohwada et al [8].

Full adders are usually composed of an exclusive-OR gate or a reduced majority function of their inputs. The exclusive-OR type has the drawback of a large delay time in the carry output, and the majority-function type circuit has the drawback of a large delay time in the sum output. An exclusive-OR type adder with transfer gates (TG) is used here. This circuit shown in Fig 2.1, operates with two input signals, and the exclusive-OR is in parallel to improve the speed. When $A \oplus b$ is at low level, TG2 and TG4 are open and input C appears on the sum output, and input A or B appears on the carry output. When $A \oplus B$ is at a high level, TG1 and TG3 are open, $\overline{C}$ appears on the sum output, and C appears on the carry output. There are a total of 24 transistors, including inverters for polarity reversal, which is a smaller number than that used for the majority-function type full addder. The calculating speed of this adder is faster than both the exclusive-OR type or the majority-function type.

As mentioned earlier we have used 2's complement arithmetic. Control signal R, identifies the Most Significant bit of the input. This control input is low when the bit being input is the most significant bit and high otherwise. This control is used by the various subcomponents in different ways. In 2's complement arithmetic, any carry generated while adding the bits in the nth position, i.e , while adding the most significant bits must be neglected. The one bit adder uses the control R to accomplish this.

Since the adder must support pipelining at the bit level, it should

1. Discard any carry generated when the msb's of the two words are added.

2. Clear the carry register before the lsb's of the next word are input.

This is easily accomplished. The carry generated by the addition, is 'ANDED' with the control R and the result is stored in the dynamic shift register.

Keeping this in mind a behavioral description of the one bit adder can be given with reference to the block diagram of the adder in Figure 2.2. The inputs A and B are the bits to be added. C is the carry generated by the previous addition which has been ANDed with the control R and delayed by one clock period. This delay is accomplished by feeding the previous carry to the dynamic shift register delay. The design and implementation of this shift register is explained in section V of this chapter. The output of the shift register is ANDed with the control R. This ensures that the carry input is 0 when least significant bits are being summed. The sum and carry outputs are available at the respective output lines.

The Boolean equations representing the operation of the adder are as follows:

$$SUM_t = A_t \oplus B_t \oplus C_{t-1} \cdot R_t$$

$$CARRY_t = A_t \cdot B_t \cdot C_{t-1} \cdot R_t$$

where the running variable t denotes time. C_{t-1} denotes, for example, the carry generated at time $t-1$, i.e. during the previous clock period.

The truth table for the one bit serial adder is shown in Table 2.1.

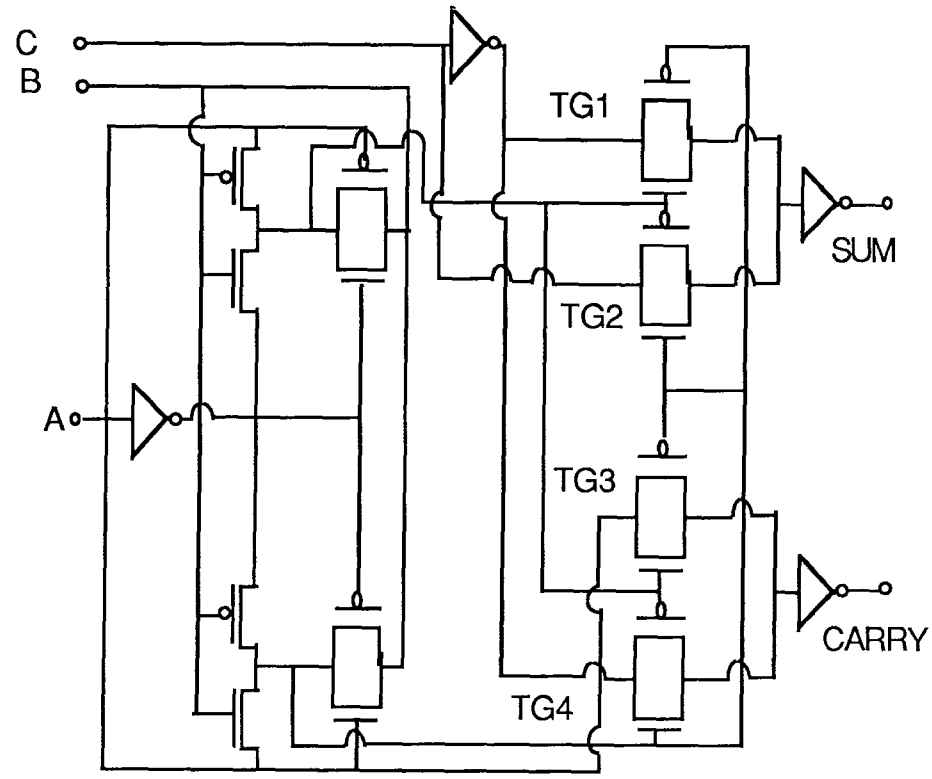


Figure 2.1: Circuit Diagram of the Serial Adder

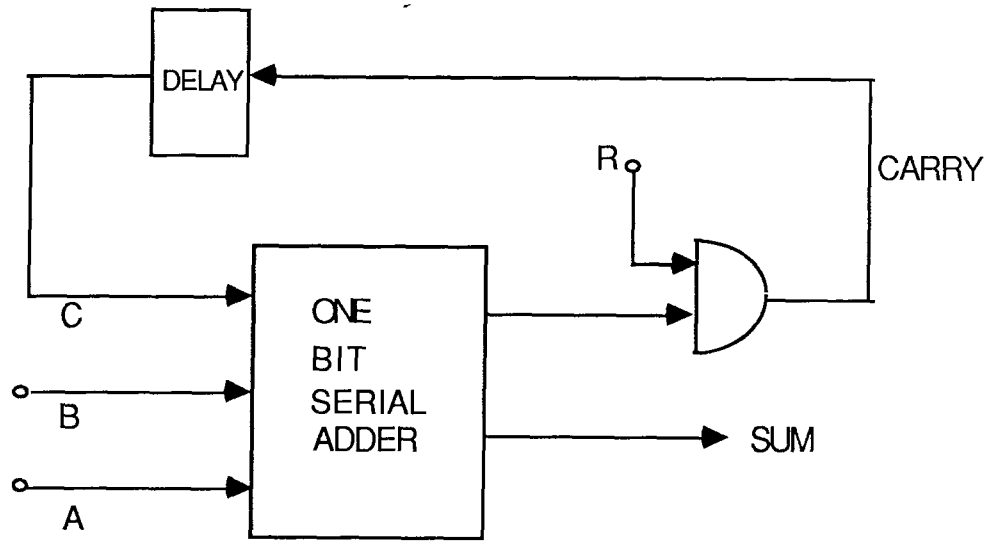


Figure 2.2: Block Diagram of the Serial Adder

A_t	B_t	C_{t-1}	R_t	SUM	$CARRY_t$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	0
0	0	0	1	0	0
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	1

Table 2.1: Truth Table of One Bit Serial Adder

III One Bit Serial Subtractor

The design of this subcomponent is based on that of the One Bit Adder. Since we are performing 2's Complement arithmetic, the operation of subtraction of two numbers can be accomplished serially based on the following algorithm: The algorithm can be described with reference to The Block diagram of the One Bit Subtractor is shown in Figure 2.3. We need to subtract B from A. Input A bit serially to the adder. Invert B before it is input to the adder. Take the carry generated by the previous subtraction operation and NAND it with the control R. Take the output of the NAND gate, delay it by one clock period and feed it as the carry of the adder. Since R is 0 when the most significant bits of the previous two words are added this operation ensures that the carry to the adder is a 1 whenever the least significant bits are added. In short we invert B and add it to A while simultaneously ensuring that the carry is a 1 when the least significant bits are added. In 2's complement arithmetic this performs the subtraction operation.

The truth table for the Subtractor is shown in Table 2.2.

IV One Bit Pipelined Multiplier

In this section we describe the design of the One Bit Pipelined Multiplier. It should be noted that from the flow graph of the DCT shown in Figure 1.1, we know precisely the coefficients of all the multipliers used in the chip. In this section we will see that this simplifies the design of the multiplier, and decrease the area that it occupies. The multiplier used in our design was first proposed by R.F.Lyon [7].

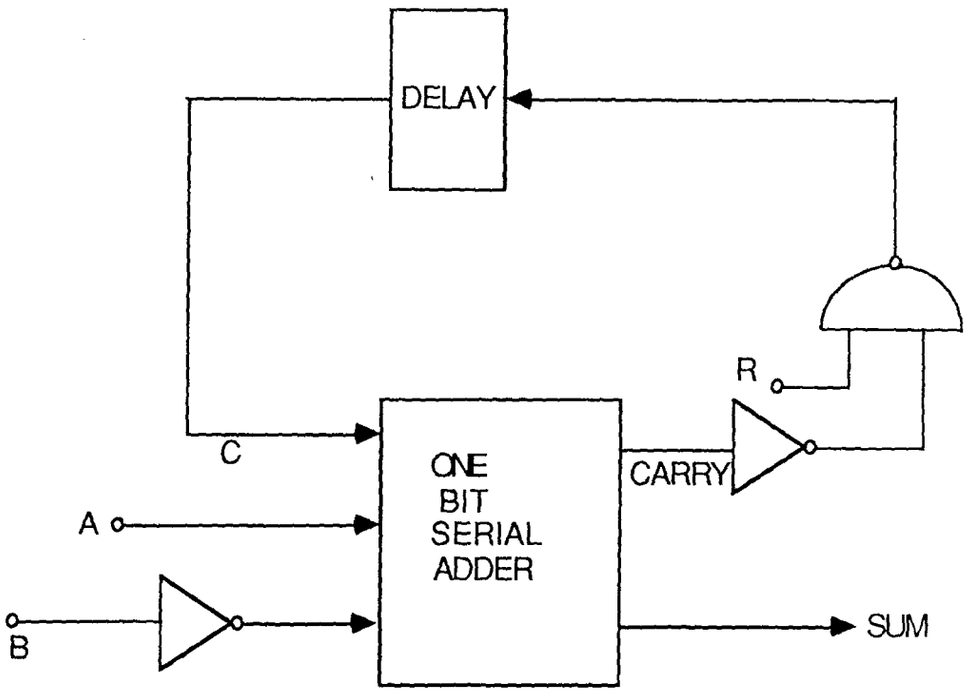


Figure 2.3: Block Diagram of the One Bit Subtractor

A_t	B_t	C_{t-1}	R_t	SUB	$CARRY_t$
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	0	1
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	1	0	1

Table 2.2: Truth Table of One Bit Serial Subtractor

We have modified Lyon's design because the coefficients of our multipliers are fixed and known in advance. In the following two subsections we review, the basic theory of the multiplier as explained by Lyon [7] in his paper. The information is taken directly from Lyon's paper. We then present the modifications that we have made to this design.

IV.1 The Basic Pipeline Multiplier

A serial-parallel multiplier takes at least N + K bit times (clock counts) to form the N + K bit product of an N bit multiplicand (data word) and a K bit multiplier (coefficient); this restricts the operation rate to one multiplication every N + K bit times, even if the result is rounded or truncated to the N most significant bits. The pipeline multiiplier also requires at least N + K bit times per operation, but can do one operation every N bit times by starting one operation before finishing the previous operation, without additional hardware. The term 'pipeline' describes a system which has operation period less than its operation delay, without parallel processing. In order to achieve one multiplication per N bit times, the use of any single part (say gate, or flip-flop) of the circuit must be restricted to no more than N (consecutive) bit times per multiplication. Since all data are serial, this means that every partial product (data word times one bit of coefficient) and every partial product sum must be restricted to N bits. This is not possible when the first partial product to enter the sum is the most significant one, unless the low order parts of all partial products are discarded, resulting in large errors.

The pipeline multiplier accumulates partial product sums starting with the least

significant partial product. After each addition, the result is an N-bit number which is truncated to $N-1$ bits before the next partial product (twice as significant) is added. Of course, overflow is possible if the data word is not restricted to N-1 bits, instead of the possible N. Hence one extra bit time separation of words may be required. To eliminate errors due to overflow, we have padded the input with extra bits of 0's. One bit is added for every multiplier that the data flows through. Hence, the input to the chip is represented as 8 bits with extra 0 bits added to eliminate errors due to overflow. Since we maintain an internal precision of 18 bits, we add 10 padding bits to the input data, and represent it as 18 bits.

The implementation of the pipeline multiplier is illustrated in Figures 2.4, 2.5 and 2.6. It consists of K modules, with serial interconnections for the data word and the partial product sum. The coefficient bits are applied by parallel connections, with the least significant bit corresponding to the first (leftmost) module. The coefficient bits in our case are embedded in the multiplier itself. This is accomplished as follows. With reference to Figure 2.5 we see that if y_0 is 0, i.e. the coefficient is 0, then PPS_0 is always equal to PPS_{-1}. Hence the adder is unnecessary if the corresponding coefficient bit is a 0. On the other hand we see that if the coefficient y_0 is a 1 then PP_0 is always equal to X_0 and hence the AND gate is unnecessary. These two considerations make the multiplier cell with a coefficient of 0 different from and smaller than its counterpart with a coefficient of 1 embedded in it. The timing signals that control the truncation are applied serially in step with the data word. Since the first module has an unused partial product input, this line is grounded.

Several minor additions to the circuit improve its practicality. First, notice that there is a path of combinatorial logic from the PPS_{-1} input through all the adders to the output. The propagation delay along this path limits the allowable bit rate. By inserting clocked delays on all the serial interconnections, we allow an increased bit rate, and hence an increased operation rate, though we double the delay (filling time of the pipeline) through the multiplier.

IV.2 Modification to allow TWO's Complement Data Words

If data words are represented in two's complement notation, the circuit described will not work when the data are negative, i.e. when sign bit (SB) is 1. But the algorithm of adding partial products to the partial product sum should still work if the additions are done properly. the key is that two's complement numbers implicitly have sign bits extended infinitely to the left.

Notice in Figure 2.6 that each addition involved the insertion of a zero bit at the left end of the partial product sum. This is done by the AND gate and control signal, which equivalently truncates the low order bit of the partial product sum in the next multiplication. For proper addition, a sign extension, not a zero, should be inserted.

One other modification required is a provision for clearing the carry flip-flop after each multiplication, i.e. after the most significant data bit has been processed; previously with positive numbers and no overflow, the clearing was automatic.

The resulting modules for a multiplier with a coefficient 1 embedded and one with a coefficient 0 are shown in Figures 2.8 and 2.7 respectively. It can be seen

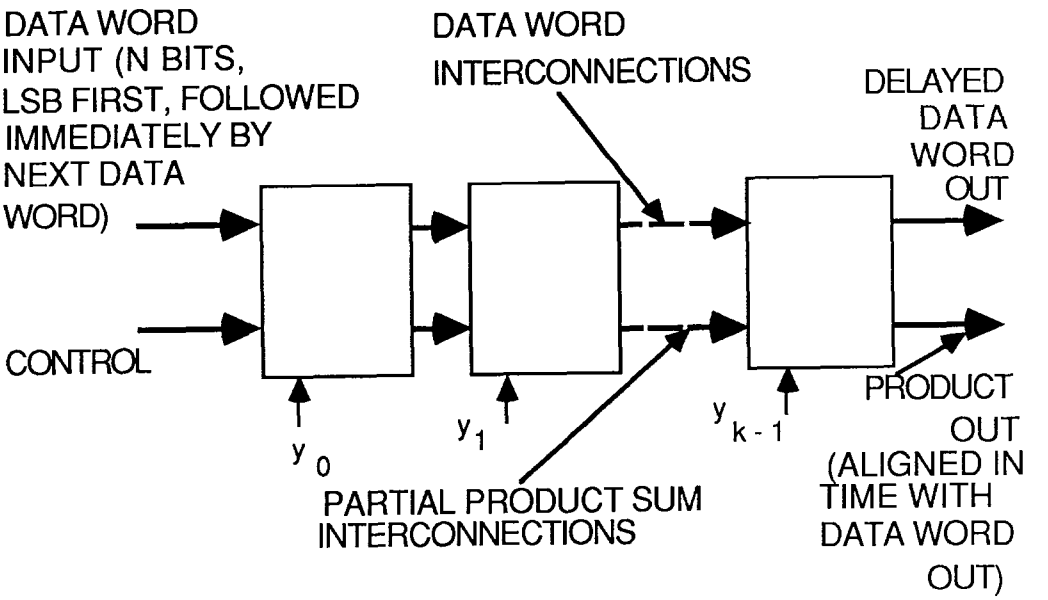


Figure 2.4: Block diagram of Serial Pipeline Multiplier

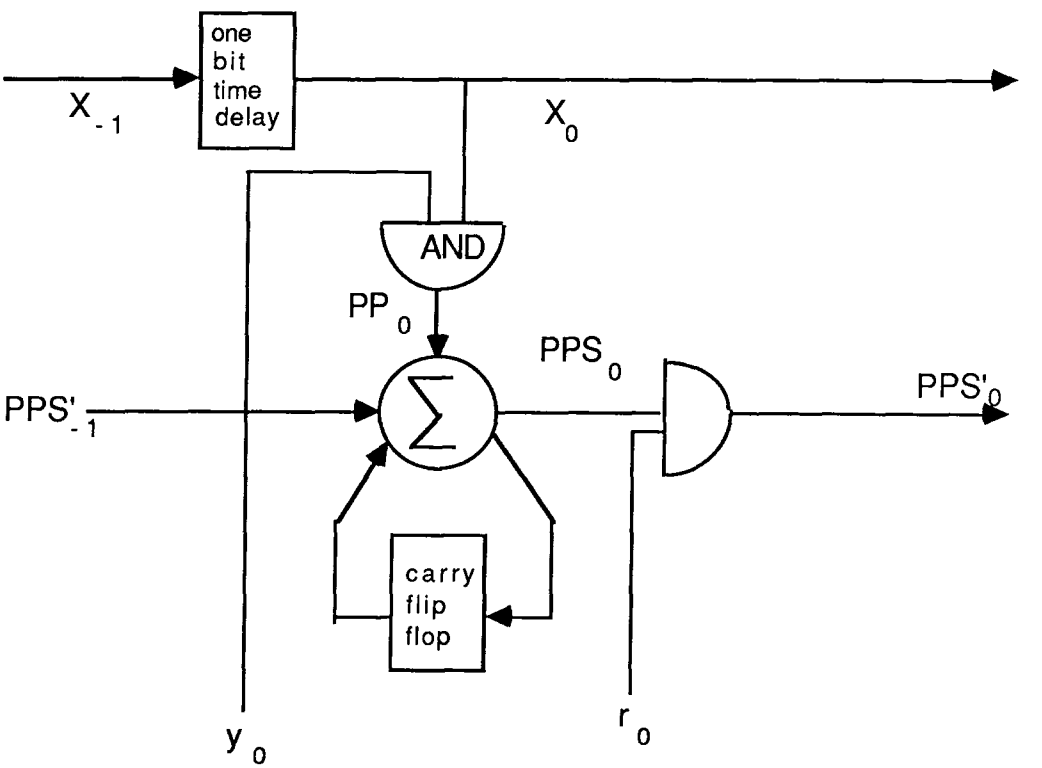


Figure 2.5: Detailed diagram of Blocks

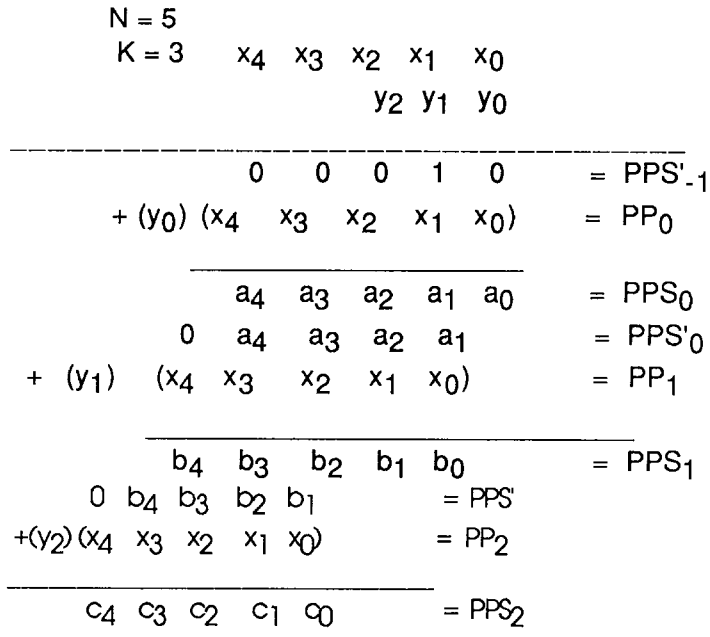


Figure 2.6: Numeric interpretation of logic of Multiplier

from these two figures that we have introduced a delay of one time unit in the path for the partial product and in order to make up for this, an additional delay has been introduced in the data path. The control R described in the previous sections is used as the control here. R is at a low level when the most significant bit arrives. This signal is delayed by one time unit before being input as the control to the multiplier. This ensures that R is at a low level when the least significant bits are processed. Also since the delay along the paths for the control and the data are the same, R is in step with the data that flows through the multiplier. Hence considering any one bitcell of the multiplier, the control R is low only when the data bit being processed by that cell is the least significant bit. The least significant bit of the product is available at the partial product ouput of the last multiplier cell one clock period after the control output R of that cell is low. In effect this means that during the clock period when the control output R of the last multiplier cell is a 0, the msb of the previous product is available at the partial product output of that cell.

Referring again to Figs 2.8 and 2.7 the following should be noted:

1. In both the figures the two switches at the partial product outputs are controlled by R. When R is 0, the previous partial product is output from the switch. If R is 1, the present partial product is output. The previous partial product is delayed by one clock period and fed to the switch. The circuit diagram of the switch is shown in Figure 2.9. This ensures that a sign extension is inserted when negative data words are being multiplied by the coefficient.

If the data word is positive and no overflow is produced, a zero is inserted.

2. The control that is input to the AND gate that feeds the carry flip flop in Fig 2.8 is low when the data bit being multiplied is the most significant bit. Hence, the carry flip flop is cleared when the most significant bit is being processed. This ensures that the carry is 0 when the least significant bits are processed.

V One Bit Dynamic Shift Register

This bitcell performs the following dual functions:

1. Synchronizes data flow throughout the chip.
2. Provides isolation between subcomponents.
3. It forms the basic building block of the transpose component.

V.1 Clocks and Timing issues

In order to provide perfect isolation between components and to ensure that the deleterious effects of clock skew are minimized, we have chosen a two phase non overlapping scheme. The two phases of the clock are denoted as CL1 and CL2. Since the entire design is in CMOS, we have to route two additional clock signals throughout the chip. These are the inverted versions of CL1 and CL2, which will be denoted as CL1B and CL2B. A graphical representation of the clocks is shown in Figure 2.10.

With reference to Figure 2.10 the following should be noted:

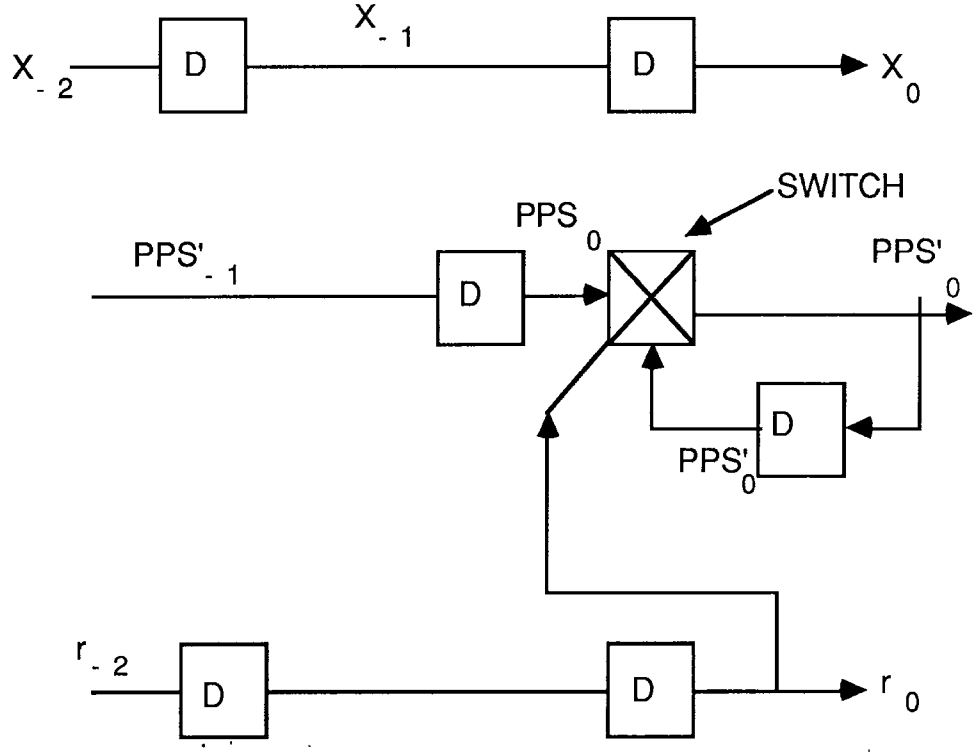


Figure 2.7: Multiplier with Coefficient 0

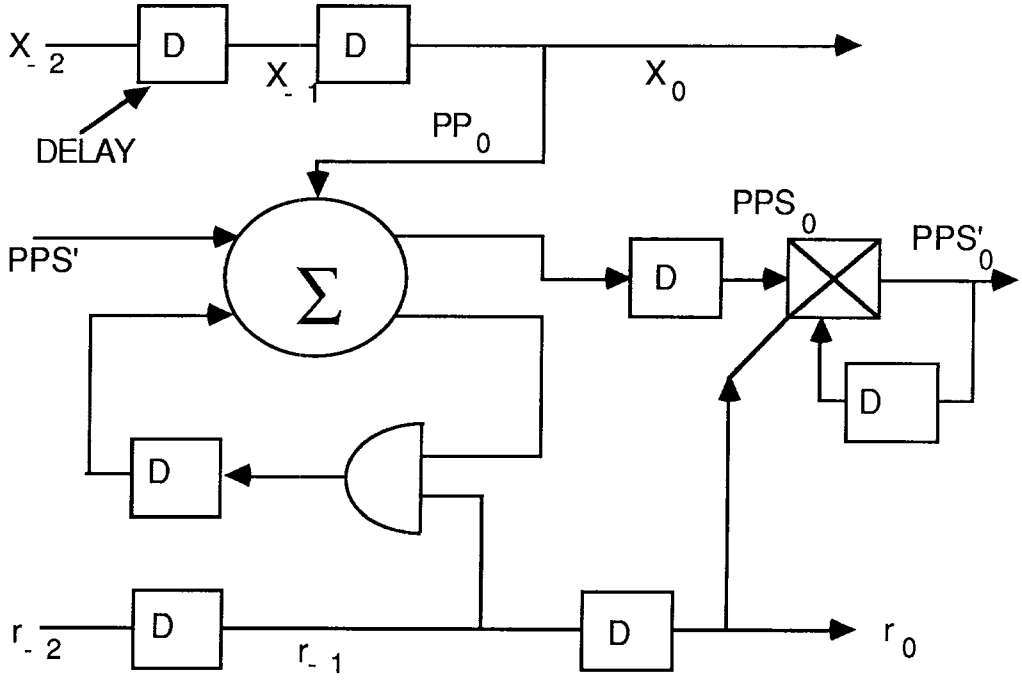


Figure 2.8: Multiplier with Coefficient 1

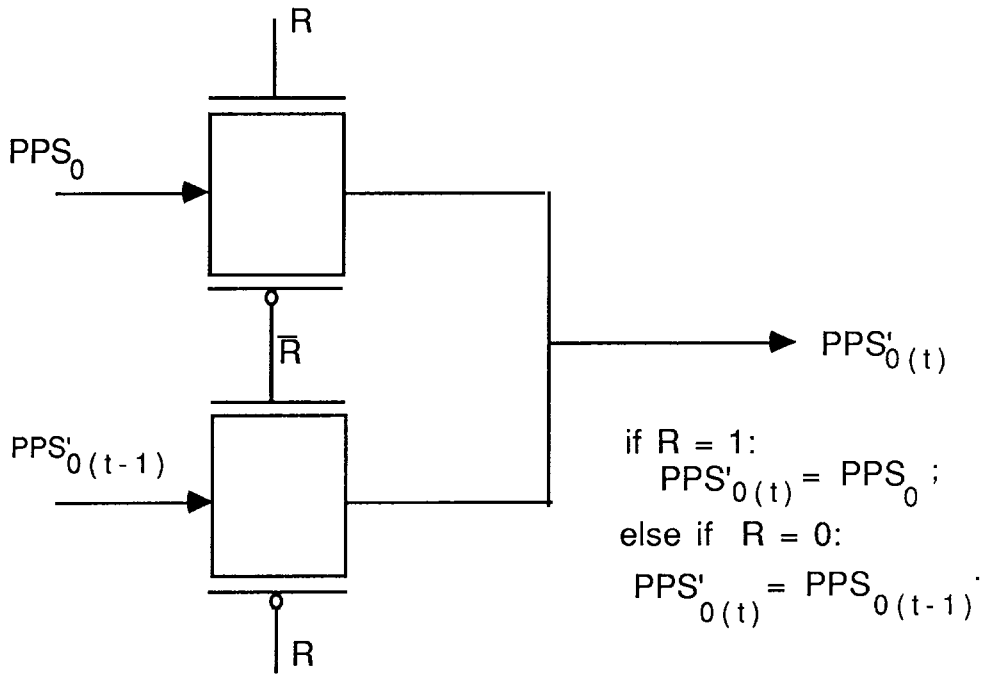


Figure 2.9: Switch used in Multiplier

1. Input Data is valid between any two logic high phases of CL2.

2. Output Data is picked up from any subcomponent when CL1 is high.

This gives rise to the following factors that determine the clock period. All the subcomponents are fed new inputs when CL2 goes high. Once CL2 goes low, the data that is input to any subcomponent is locked in by means of charge storage on the gate capacitances of the transistors that feed the subcomponent. From then on, the outputs of all subcomponents have to settle down before CL1 goes high. Once CL1 goes high, the outputs of all subcomponents are latched into shift registers connected to their outputs (all the subcomponents are fed by Dynamic Shift Registers and all the subcomponents have such registers connected to their outputs). These shift registers in turn transmit the latched data as inputs to

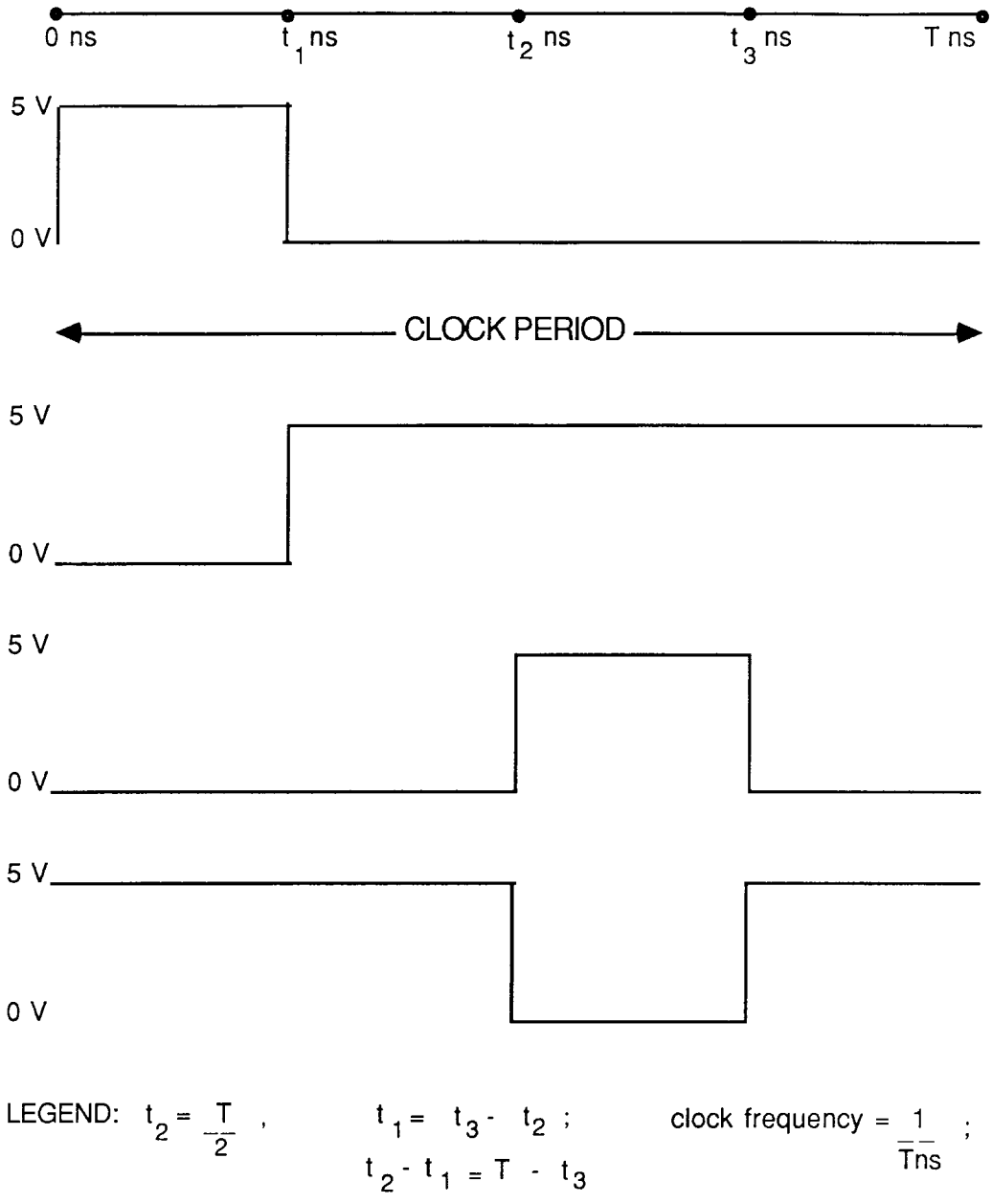


Figure 2.10: Clocks of the DCT chip

next subcomponent during the next high phase of CL2. Hence all operations in the basic cells take place between the rising edge of CL2 and the rising edge CL1. The time lag between these two occurences should be larger than the propogation delay of the slowest subcomponent,in our case the One Bit Multiplier with a coefficient of 1.

V.2 Design of the Shift Register

The circuit diagram of the One Bit Dynamic Shift Register is shown in Figure 2.11. When CL1 is high, node n1 will be conditionally pulled high or low by the p or n data transistors, respectively. When CL1 is low, this value is stored, the clocked transistors being turned off. The second stage operates similarly. Thus, when CL1 is high the data at node 'in' is latched into the shift register and is output at node 'out' when CL2 rises. The register provides a delay of one clock period and also serves to isolate node 'in' from node 'out'. It is interesting to note that if the clocked transistors are placed between the inverter and supply rails, severe problems arise due to charge redistribution.

VI Layout and Design Issues

Before the actual layout of the different subcomponents was commenced the following decisions were made:

1. The POWER and GROUND rails will be run horizontally. The power rail will run at the top of each cell and the ground rail will run at the bottom.

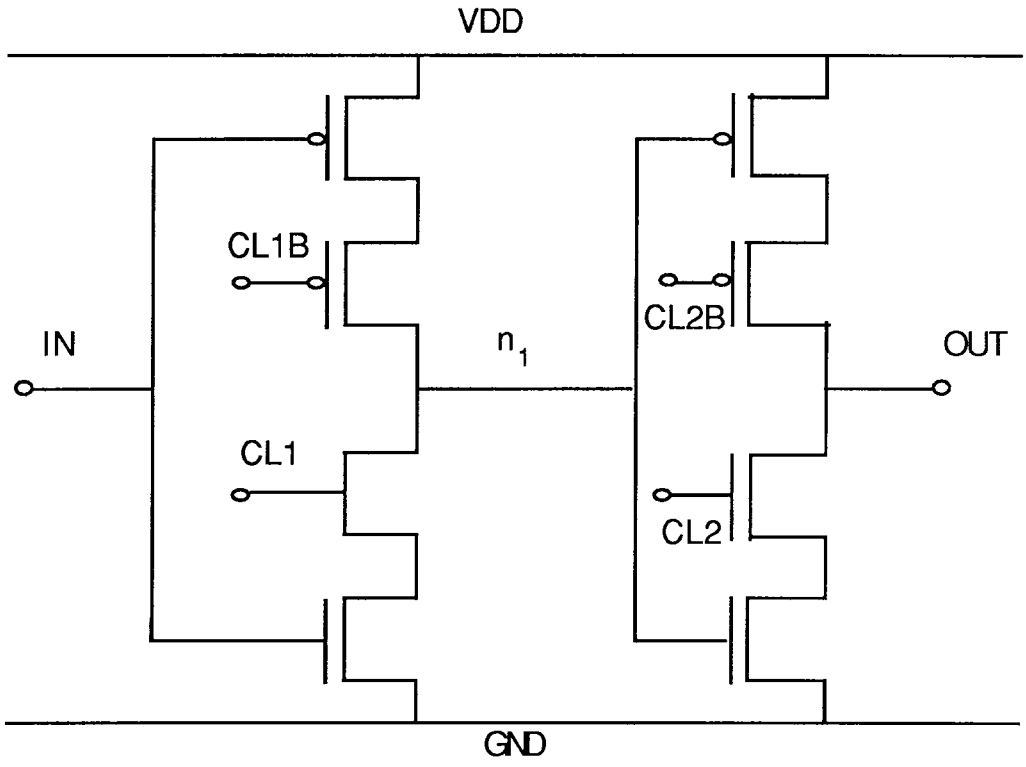


Figure 2.11: Dynamic Shift Register

2. It was decided that the four clock lines will be routed horizontally both on the top and the bottom of each cell, above and below the power and ground rails, so that the cells above and below each set of clock lines can tap the clock signals from these lines. Thus each set of clock lines is shared by the cells above it and below it, a scheme that saves area and reduces the delay along the clock lines.

3. All these global rails will be run in metal2, since the resistance and capacitance of this layer are half that of metal1. Within each cell the width of the clock rails will be 4λ and to ensure sufficient current carrying capability, the power and ground rails will be 8λ wide.

In order to facilitate the abutting of bitcells to form the large components of the DCT chip, a constant height of 56λ was maintained between the power and ground rails in all the bitcells. The clock lines were separated from these rails by 4λ and a spacing of 4λ was maintained between the clock rails. Thus different bitcells could be placed abutting each other. The transistors forming the circuits were placed, as far as possible, between the power and ground rails. p-type transistors were placed near the VDD line and the n-type transistors were placed near the GND line.

The resulting layouts for the One Bit Adder, One Bit Subtractor, One Bit Multiplier with a coefficient 0 and a One Bit Multiplier with a coefficient 1 are shown in Figures 2.12 through 2.15. The layout of the Dynamic Shift Register is shown in Figure 2.16.

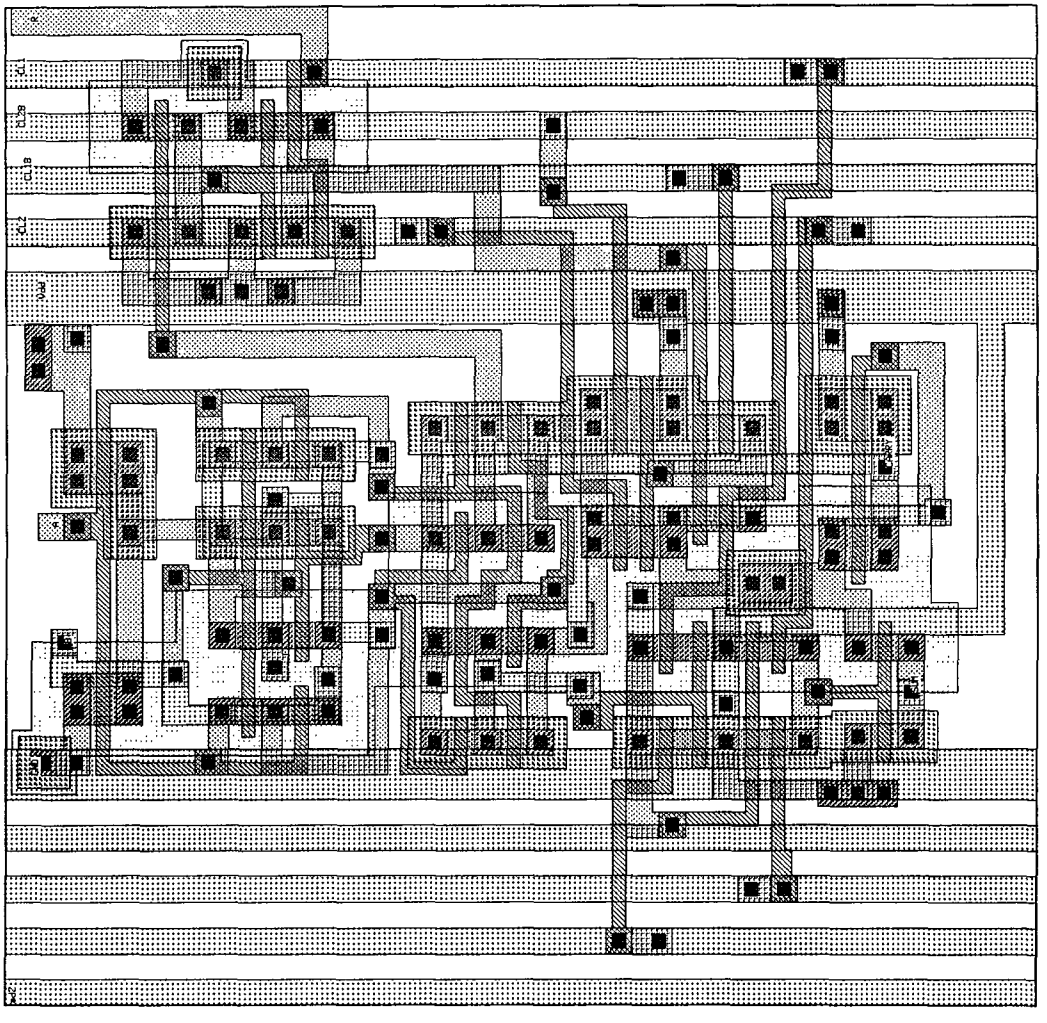


Figure 2.12: Layout of the One Bit Serial Adder

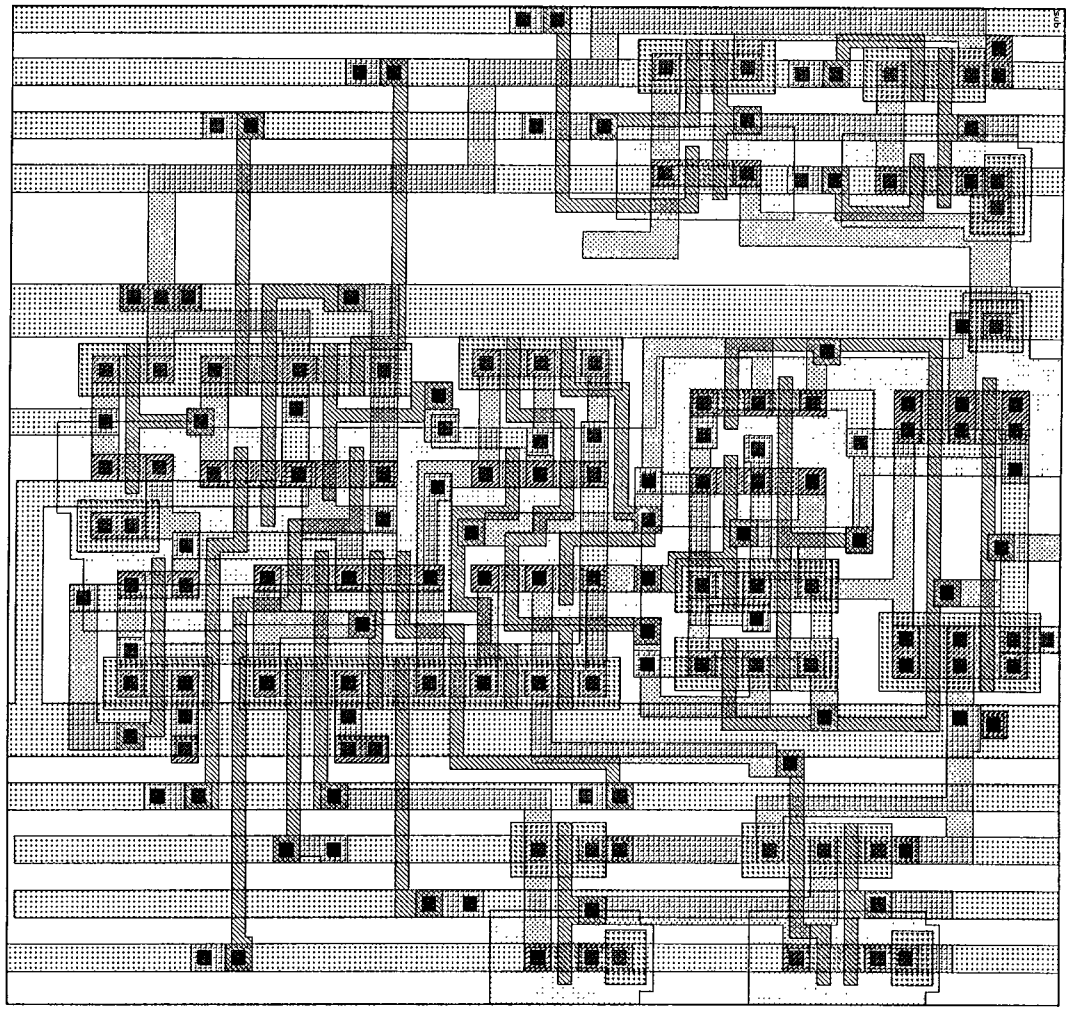


Figure 2.13: Layout of the One Bit Serial Subtractor

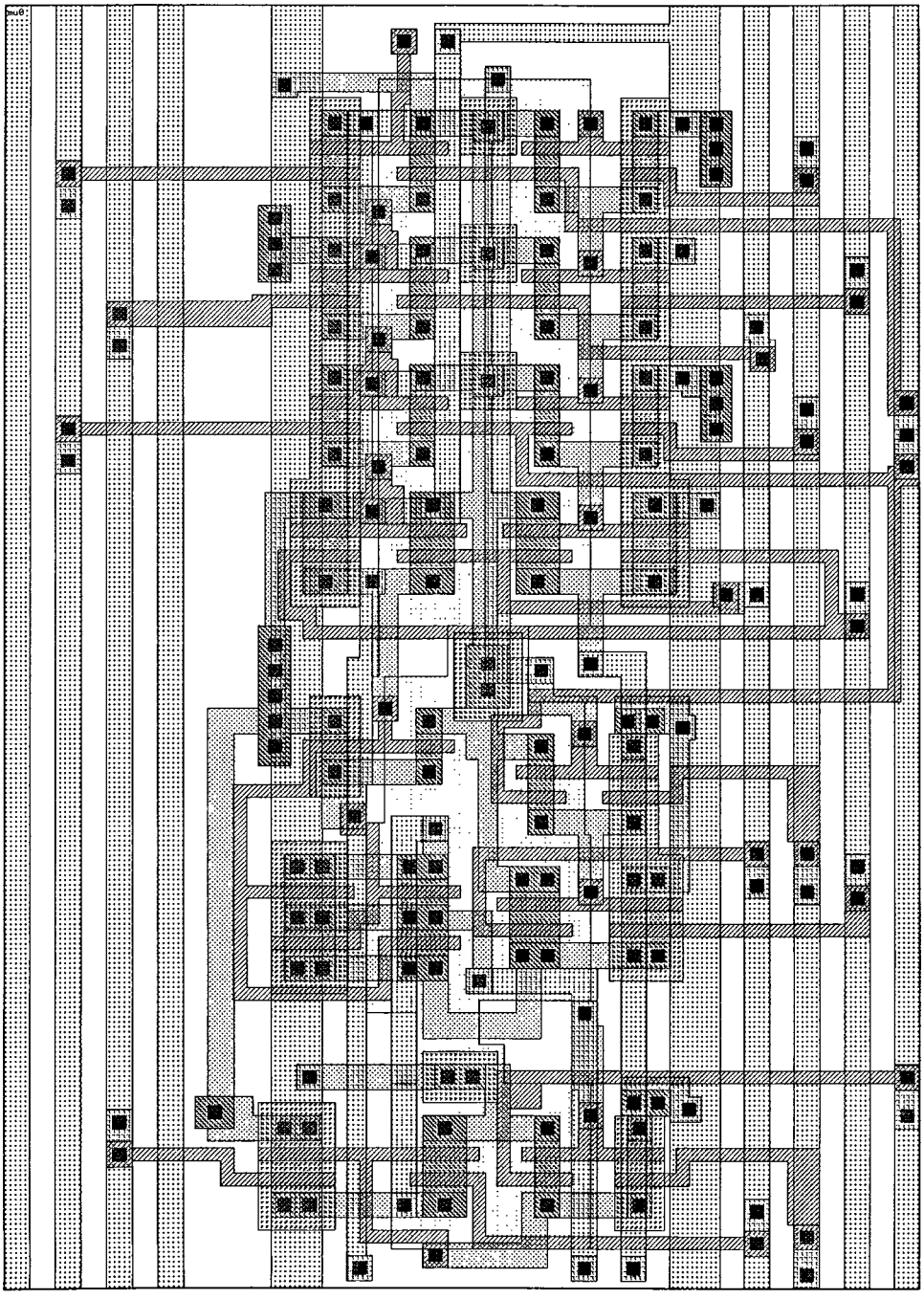


Figure 2.14: Layout of the One Bit Multiplier with coeff 0

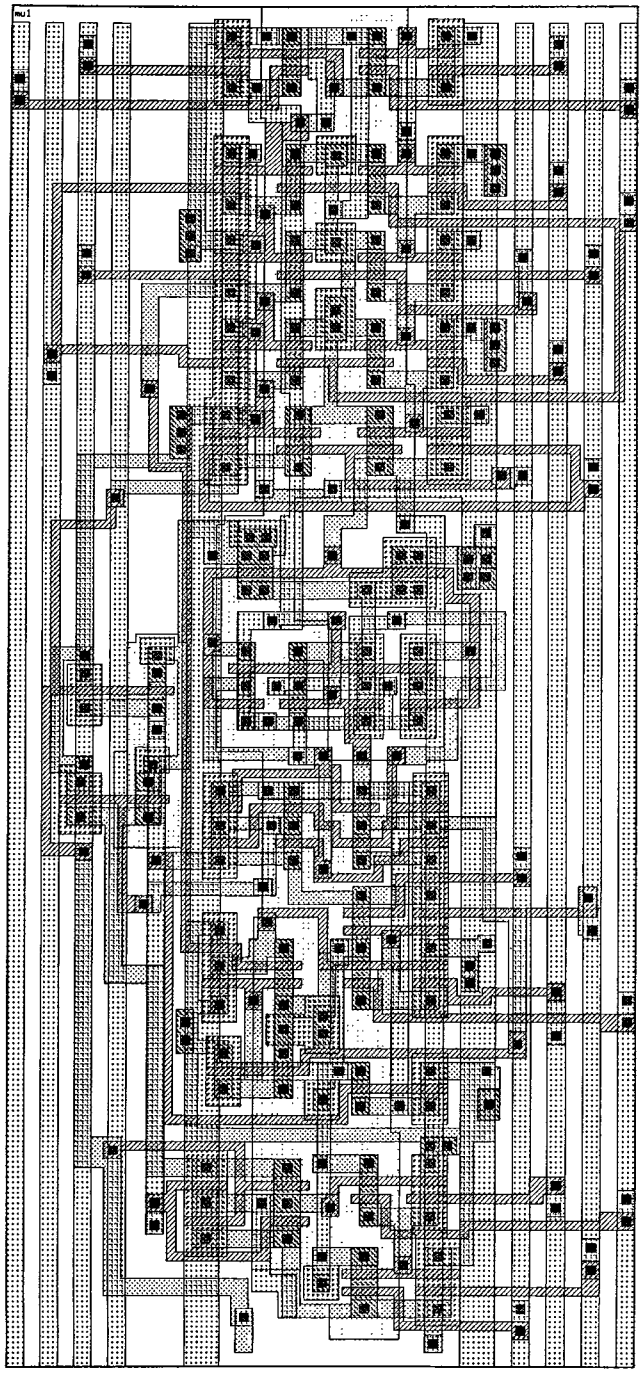


Figure 2.15: Layout of the One Bit Multiplier with coeff 1

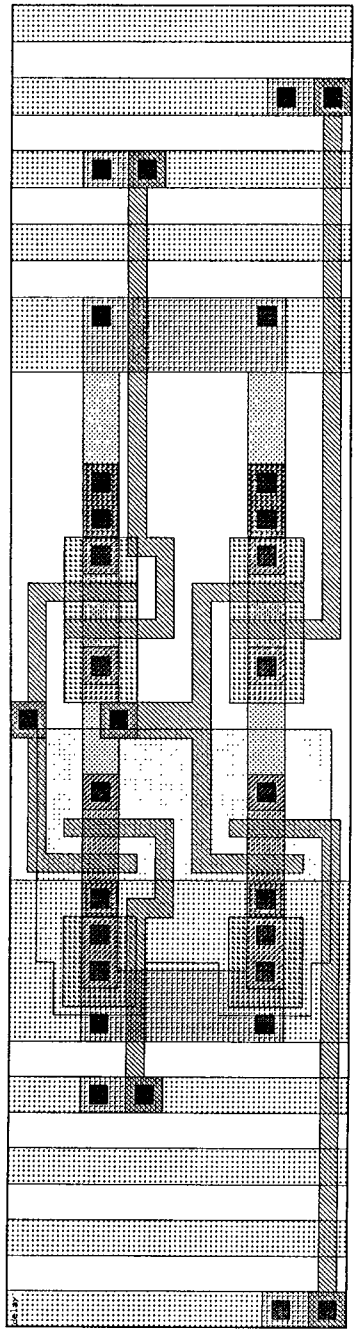


Figure 2.16: Layout of the Dynamic Shift Register

for various k,l. The algorithm computes the DCT to within a constant factor. Since we know the values of the multiplication factors for the various coefficients, this can be taken care of when designing the scalar-quantizer or additional multipliers can be introduced on chip to make up for this deficiency.

2. The flow graph for the algorithm (shown in Figure 1.1) requires that data coming out of the multipliers be in synchronism with data coming out of the adders. Additional circuitry has been included in the design to ensure this.

II Basic components

A perusal of the flow graph shows that the entire computation can be performed using 3 types of cells. The One bit adder, the Dynamic shift register and a cell that comprises of an adder, subtractor and a multiplier. We shall refer to the third cell as the *basic dct* cell and consider its design in detail here. A block diagram of this cell is shown in Figure 3.1. It is made up of an adder the output of which is fed to an array of shift registers. The subtractor output is fed to the multiplier. We have represented the coefficients of the multipliers as 12 bit numbers. Hence each multiplier is made up of 12 bitcells, each of which can be a multiplier bit cell with a coefficient of 0 or a 1 embedded. The data inputs to the bitcell are represented as A and B. The control input is R. R identifies the most significant bit input to the adder and the subtractor. It is then delayed and acts as the control for the multiplier where it denotes the processing of the least significant bit.

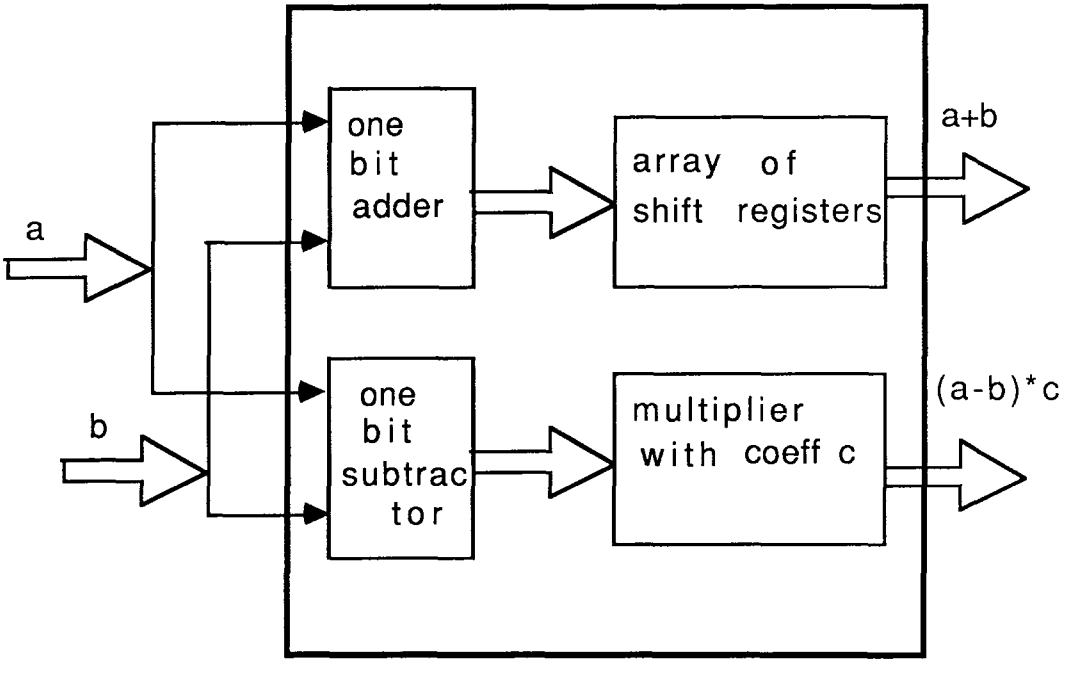


Figure 3.1: Basic cell of the DCT component

The output of the subtractor which is $(A - B)$, is fed to the multiplier where it is multiplied by the coefficient C embedded in the multiplier. The multiplied output is available after $(12 \times 2) + 1$, i.e. after 25 clock cycles. Hence the output of the adder should be delayed by 24 clock cycles if the two outputs are to be in synchronism. 25 shift registers are included to perform this operation. We thus have ensured that the outputs A+B and $(A - B) \times (C)$ leave the bitcell in synchronism. It should be noted that we have not considered misalignment of data due to truncation by the multiplier. This is considered in the next section.

III Truncation of Data by the Multiplier

In the following discussion we assume that data is represented in such a way that problems due to overflow do not exist. Let us assume that the input to the multiplier, denoted as (A - B), is represented as 18 bits, i.e. 14 integer bits and 4 fractional bits. Let the coefficient of the multiplier C be represented as 12 bits, i.e. as 3.9 bits. Then without truncation the output of the multiplier is (18 + 12), i.e. 30 bits long. It can be represented by (14 + 3) = 17 integer bits and (4 + 9) = 13 fractional bits (as 17.13 bits). The pipelined multiplier truncates this 30 bit output to 18 bits, i.e. it discards the least significant 12 bits. Hence the output of the multiplier of the basic dct cell denoted as $(A - B) \times (C)$ can now be represented as 17.1 bits. In the previous section, we said that this output should be in synchronism with the output of the adder denoted as (A + B) and represented as 14.4 bits. In that case, the stage after this multiplier will be processing the least significant bit of (A + B) which has a binary weight of 2^{-4} along with the

lsb of $(A - B) \times C$ which has a binary weightof 2^{-1}. This is clearly a problem of misalignment of data at the outputs.

We have overcome this problem by modifying the last bitcell of the multiplier and by changing the number of clock periods by which the output of the adder is delayed before it is output by the bitcell. We add three 0 bits to the multiplied output and truncate its three most significant bits. Before data to be multiplied is input to the multiplier, we introduce three pad bits in the most significant positions of the data. Thus the truncation of these three most significant bits at the output of the multiplier does not affect the precision of the adder. With reference to the above example, let us assume that we have two numbers to be input to the bitcell. Also assume that considering both overflow at the added outputs and overflow due to the multiplication the numbers can be represented by 11.4 bits. Before the data is input to the bit cell we pad the data with three 0 bits and now represent it as 14.4 bits. Since overflow has been considered, the output of the adder has a most significant bit with a binary weight of 2^{13} and a lsb whose binary weight is 2^{-4}. On the other hand, the multiplied output has a msb with a binary weight 2^{16} and a lsb with a binary weight of 2^{-1}. If we now add three 0 bits to the lsb positions of the multiplied output and truncate the three msbs', the outputs of the adder and the multiplier are aligned.

We accomplish the alignment operation as follows: It was mentioned in the previous chapter that whenever the control R is a 0, the multiplier bit cell truncates the partial product output and instead, it produces a sign extension. The control circuitry of the partial product outputs of the multiplier are designed so that instead

of producing sign extensions in the three most significant positions, it introduces three 0 bits which are the pad bits. Previously, the control output R is low when the msb of the product is output. Now R is low three cycles after the previous msb has been output, i.e. now the lsb of the product is available at the output of the last multiplier cell, 22 cycles after data was input to the multiplier. The only thing left for us to do, so that the outputs of the adder are aligned with that of the multiplier, is to reduce the number of shift registers, delaying the sum to 22.

Another factor that we used to our advantage can be found by inspecting the flow graph in Figure 1.1. We see that the data from each column of cells has to be in synchronism. If we consider each column as a stage of the DCT, the coefficients of the multipliers in each of the stages is listed below:

Ist stage:

$$\frac{1}{2cos\frac{\pi}{16}} = 0.5097955$$

$$\frac{1}{2cos\frac{3\pi}{16}} = 0.6013448$$

$$\frac{1}{2cos\frac{5\pi}{16}} = 0.8999762$$

$$\frac{1}{2cos\frac{7\pi}{16}} = 2.5629154$$

IInd stage:

$$\frac{1}{2cos\frac{\pi}{8}} = 0.5411961$$

$$\frac{1}{2cos\frac{3\pi}{8}} = 1.306563$$

IIIrd stage:

$$\frac{1}{2cos\frac{\pi}{4}} = 0.7071067$$

For the first stage, the largest coefficient is 2.5629154. Two integer bits are enough to represent the integer parts of all the coefficients in the first stage. Hence, we represent all the coefficients in this stage using 2.10 bits. Thus, to solve the issue of misalignment of data output from this stage, all we need are two pad bits. For the first stage the number of delays in the path of the adder output is now 23. Extending this, we have represented the coefficients in the second stage as 1.11 bits. This stage requires only one pad bit and 24 delays in each bit cell comprising this stage. All the coefficients in the last stage are less than 1. They are represented as 0.12 bits. No pad bits are required for this stage and the data output from the adder will be aligned with the product after 25 delays.

IV Layout and Design issues

In this section we consider the aspects of laying the cells together to form the Row/Column DCT component. Before we proceed to layout the component, we have to route the control R through the component. We explain this in the following subsection.

IV.1 Control for the component

The signal R is the only external control applied to the chip for the DCT component. The other control signal controls the transpose component. External to the chip, R is low whenever the most significant bit is input to the chip. The routing of this control through the DCT component can be explained with reference to Figure 1.1. It can be seen that the input data is first fed to the adders and subtractors in

the basic dct cells that comprise the first stage. Since R is low in synchronism with the msb, we can apply the external control R directly as the control for the adders and the subtractors in the first stage. After the msb's are processed they are fed to the multipliers of the first stage after a delay of one clock period, which is the delay through the adders and the subtractors. Since the msb's are immediately followed by the lsb's after one clock period delay, we can delay the control by one clock period with respect to the data and use it as the control for the multipliers (the control for the multipliers should indicate the presence of the lsb). This solves the problem of control for the first stage.

In the previous section it was mentioned that due to truncation of the two most significant bits of the product, the control R at the output of the last bit cell of a multiplier in the first stage is low when the bit immediately following the lsb of the product is output. With reference to Figures 2.4 and 2.8 we notice that the control is delayed twice through each multiplier bit cell. Hence, when the control output of the last but one bit cell of a multiplier in the first stage is low, the data output from the last bit cell (i.e. from the multiplier itself) is the most significant bit of the previous product. This can be used to control the adders and subtractors of the next stage. Similarly the control R_{-1} of the last bit cell can be used as the control for the multipliers of the next stage. In this way we can derive the controls for every stage from the control outputs of the previous stages.

IV.2 Layout of the DCT component

The final layout of the DCT component is shown in Figure 3.2. The layout is a mapping of the flow graph of Figure 1.1 onto silicon. The component has aproximately 15,000 transistors and occupies an area of 2822λ by 5285λ.

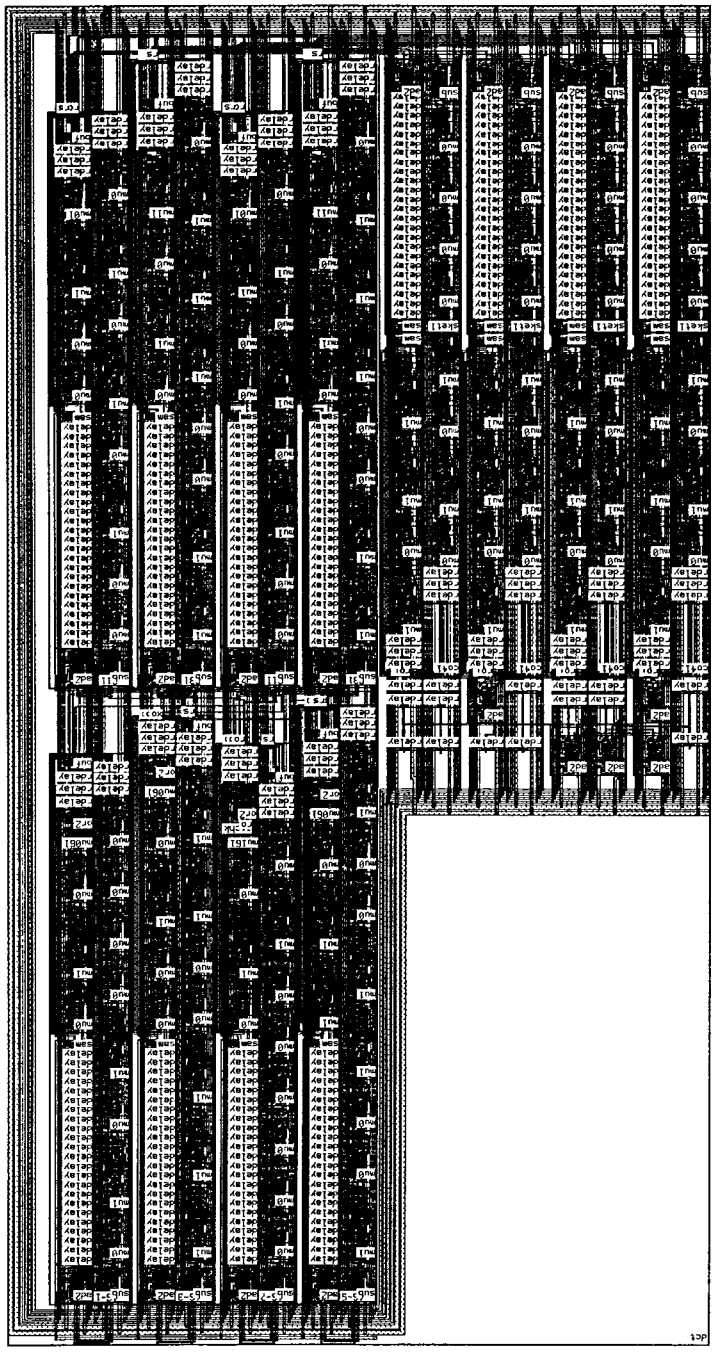


Figure 3.2: Layout of the DCT component

CHAPTER
FOUR

The Transpose Component

I Introduction

In this chapter, we discuss the design and the implementation of the *Transpose* component. We have used a novel aproach in the design and implementation of this component, an aproach that is both simple and easy to implement. This component is made up of two basic cells, a one bit dynamic shift register and a switch (similar to the switch used in the multiplier). It is perfectly pipelined and needs only one control signal. In the following sections, we present the theory of operation of this component and discuss the design and layout issues.

II Principle of Operation

The Transpose component is an array of 8×8, 18-bit shift registers. We can explain the operation of this component with reference to the block diagram of Figure 4.1.

Let us consider any shift register A not on the periphery of the array (by shift register we mean an array of 15 one bit shift registers). As shown in Figure 4.2, it has two other shift registers adjacent to it, shift register B immediately above it and shift register C immediately behind it (towards the inputs). The input to each

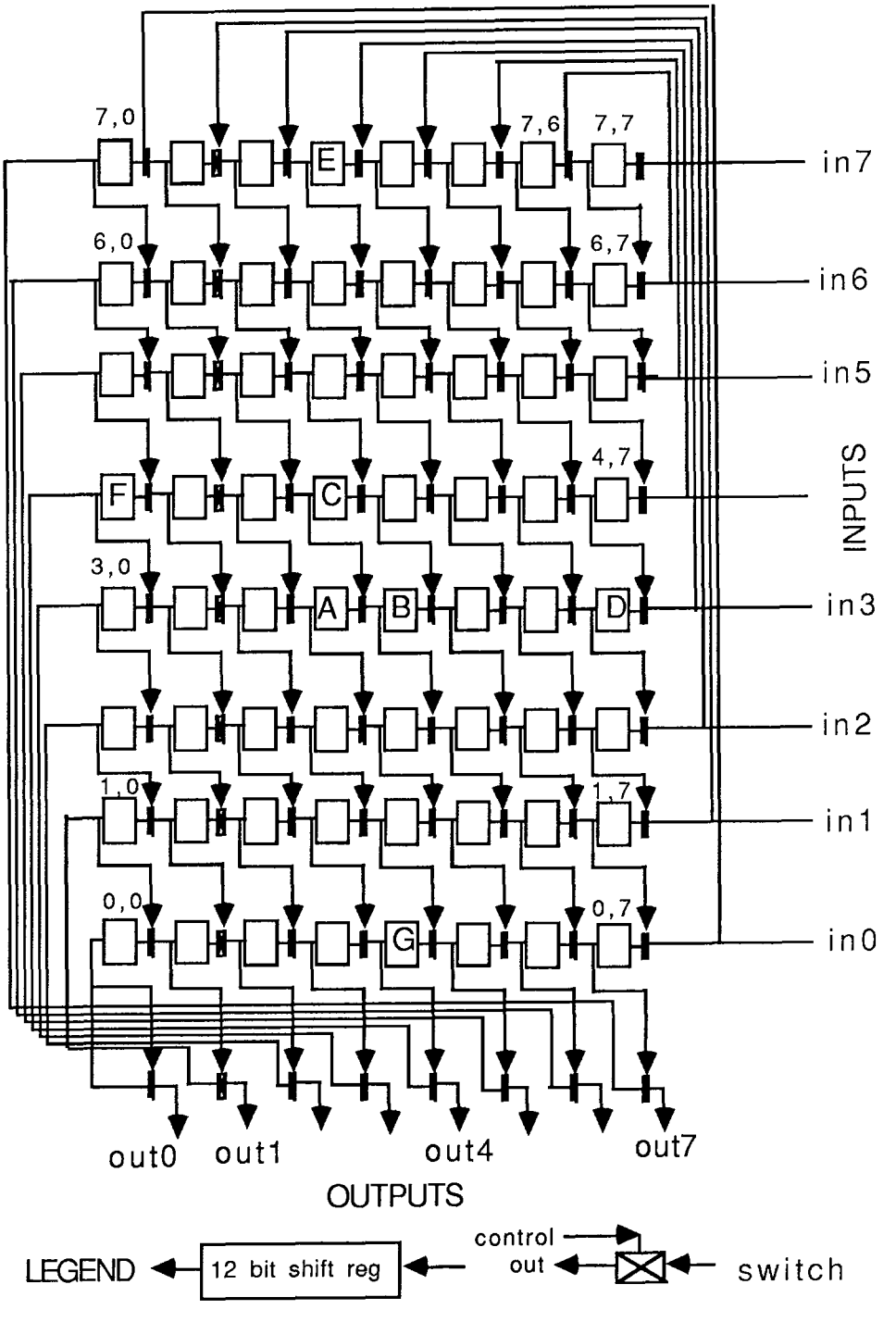


Figure 4.1: Block Diagram of the Transpose Component

shift register like the one we are considering, comes from a switch. The switch is similar to the switch used in the multiplier. It performs the function of a two input one ouput multiplexer. There are 64 such switches in this component and each of these switches is controlled by one control signal which we will refer to, henceforth, as control T. The switch, whose output is the input to A, has its inputs tied to the outputs of of the shift registers B and C. When the control T is a 1, the output from C is channeled to the input of A. If the control T is a 0, the output from B is now the input to A. The output of any shift register like A is connected to the switches that feed the shift register immediately in front of it and the one immediately below it. Next we consider the shift registers on the periphery of the array. In Figure 4.1, these are the shift registers D, E, F, G. The shift registers along the seventh column and along the seventh row of the array are the input shift registers. Inputs enter the array through these shift registers. Likewise the shift registers along the zeroth row and column are the output shift registers. We now consider the input and output shift registers.

One input to the switches that feed the input shift registers is connected to an external input line. Thus the switches of shift registers E and D have one of their input lines connected to the input line 'in3'. The other input to these switches is connected to the outputs of the shift registers immediately above or behind the switch depending upon its location. Thus one input to the switch of E is connected to the output of the shift register immediately behind it and one input to the switch of D is connected to the output of the shift register immediately above D. When T is a 1, the external input line ,'in3', feeds D while E is fed from the shift register

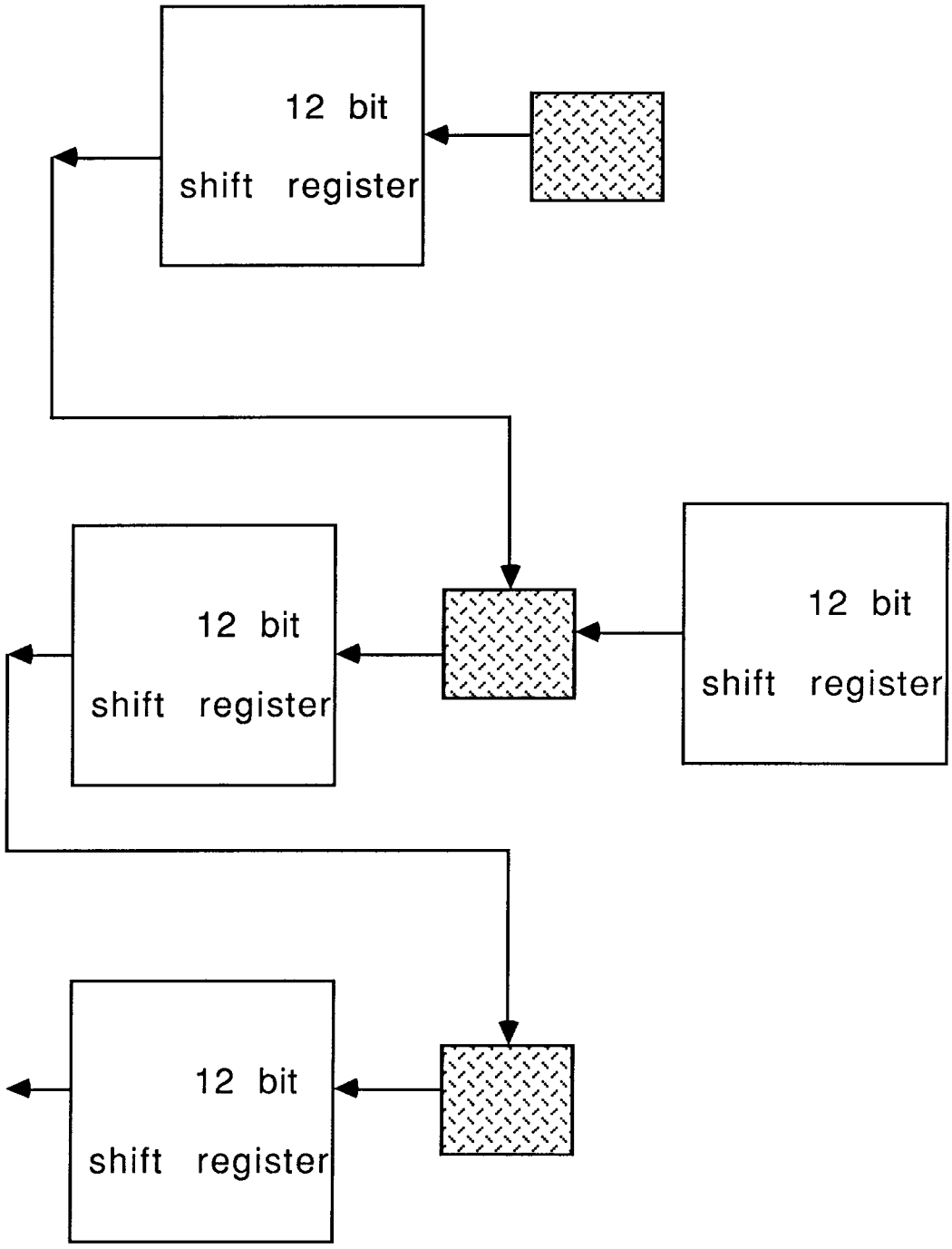


Figure 4.2: Configuration of the Transpose Unit

behind it. When T is a 0, 'in3' feeds E and D is fed from the shift register above it.

Let us now look at the output shift registers F and G. All such shift registers have their outputs connected to a switch that feeds an output line (out4 in the case of F and G) and to a switch that feeds, depending on their positions, either the shift register immediately below or the following one. When T is a 1, the output of F is connected to the output line, 'out4', while the output of G is connected to the input of the shift register next to it. When T is a 0, the output of F feeds the shift register immediately below it while the output of G is connected to the output line 'out4'.

From the above discussion, and with reference to Fig 4.1, we see that when T is 1, data flows horizontally into and through the array. When T is 0, it flows vertically. Let us consider each input matrix, i.e., each set of 8×8, 18 bit numbers as one block. Initially, T is set to 1 and the block is fed into the component. After $8 \times 8 \times 18$ clock cycles, the entire block has been input to the array and is latched onto the outputs of the shift registers. We now set T to 0. Henceforth, data flows vertically through the array. The transposed data is now available bit serially, at the output lines. Also, after the first block has been input to the array, the second block is available at the input lines, and this enters the array vertically. After one block cycle, i.e., after $8 \times 8 \times 18$ clock cycles, the transpose of the first block has been output and the second block has filled the array in a vertical fashion. We now set the control T to 1. Henceforth, the transpose of the second block is available at the output lines, the data flows horizontally through the component and the third

block enters the array horizontally.

Thus by switching the control from one state to another every block cycle, we accomplish the transpose operation in a perfectly pipelined fashion. It should be noted that, as long as the switching takes place every block cycle, the initial state of the control does not affect the operation of the component. Only shift registers and switches have been used in this design. It is much simpler and easier to implement than the traditional RAM array type of transpose component.

III Layout and Design issues

The control T for the transpose component is derived externally. The control has been routed through the component in metal1. The size of the transpose component is 2000λ by 3000λ. It has a total of about 9000 transistors and is designed to operate at 10Mhz. A plot of the layout is shown in Figure 4.3.

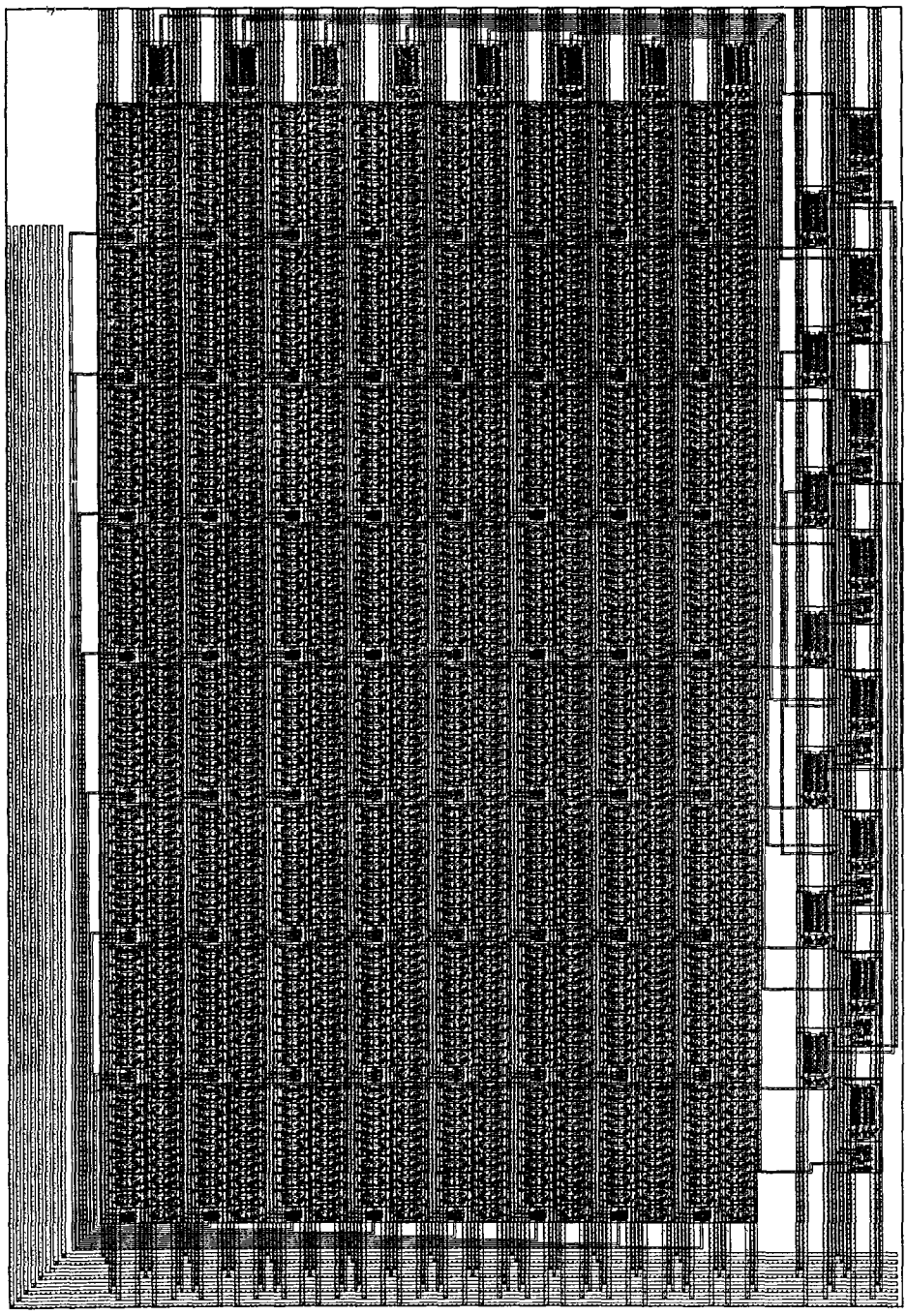


Figure 4.3: Layout of the Transpose Component

CHAPTER
FIVE

SIMULATION

I Introduction

After the basic cells have been designed, we have to ensure that they will, on silicon, meet the desired specifications and more importantly we have to ensure that the design is logically correct. Various CAD tools are available that will help us do this. We have used the circuit simulators, SPICE and SPLICE to simulate the designs on a circuit level. CRYSTAL was used for timing verification. Logic simulation at the Behavioral level and Gate level was done using HILO. Switch level simlation was done using the simulator IRSIM. Chronologically, first the Behavioral level simulation and Gate level simulations were done. After this the basic cells were laid out. Circuit simulation of the basic cells was done using SPICE and SPLICE. Then the basic cells were put together to form the various components. The components themselves were simulated on a switch or transistor level using IRSIM. Timing simulation of the components was performed using CRYSTAL. One of the main issues in simulation is the design of test vectors. These are the vectors that will be used to test the design as well as to test the chips after fabrication. In the following sections, we present the test vectors used in simulations, as well as

the results of testing.

II One Bit Serial Adder and Subtractor

The design of test vectors for these component are easy. Basically, we have to make sure that all eight combinations for three inputs (A, B, CARRY) are exercised. We have to also exercise the carry flip-flop. Simulations show that the maximum propogation delay through the Adder and the Subtractor is 3ns.

III One Bit Dynamic Shift Register

Crystal was used to determine that the maximum propogation delay through this component is 1ns.

IV One Bit Multipliers

In the case of this component, we have to exercise the delay flip-flops in the path of the data, control and carry. We also have to exercise the adder and control circuitry. Circuit simulations using SPICE predict a maximum propogation delay of 3ns through the multiplier with a coefficeint of 0, and 5ns through the multiplier with a coefficient of 1 embedded in it.

V Transpose Component

As said in the previous chapter, this component is made up of an array of shift registers and switches. We have to ensure that these cells are operating properly and that the operation of switching data flow is proper. Since this component has a

large number of transistors, the switch level simulator IRSIM was used to simulate it. The test vectors used were 8 X 8, 18 bit arrays of data elements. The test vectors are enumerated below:

1. A matrix consisting of only zeros.

2. A matrix consisting of only ones.

3. A matrix consisting of alternating zeros and ones.

4. Randomly generated matrices.

Different combinations of the above mentioned test matrices were fed to the chip alternately. Switching of the control is done after each block is fed in and the outputs are monitored.

Simulations show that this component can operate at a maximum clock speed of 10MHz. The limiting factor in the speed of operation of this component is the capacitance and resistance of the long lines that feed the output switches.

VI Row/Column DCT

This component computes the 1-D DCT of an 8 element vector. A large number of random vectors were used as test vectors. A few of these are listed below. The outputs from a 'C' program that computes the DCT without truncation and the outputs from the simulations are listed along with the inputs. With reference to the flow graph of Figure 1.1, the following should be noted about the tables shown in the following pages: inputs are listed in order starting with x0, x1, ..., x7,

outputs are listed in order as X0, X1, X2, ..., X7. The tables also give an idea of the error produced in the 1-D DCT computation due to the inherent truncation in the multipliers and due to error in representing the coefficients.

INPUT	CORRECT OUTPUT	OUTPUT FROM CHIP
48	993.0000000000	993.0000
123	-95.0318984985	-95.75
89	-136.5531311035	-136.7500
175	-126.4350204468	-126.6875
234	212.8391571045	212.8125
56	-165.6450500488	-165.8125
78	-117.1761169434	-117.2500
190	32.6013336182	32.3750
1	308.0000000000	308.0000
5	-88.8332366943	-89.625
67	-62.0359115601	-62.2500
2	8.9865064621	8.5625
98	-65.0538253784	-65.0625
3	47.0255050659	46.8125
125	-90.6396408081	-90.6250
7	212.8673095703	212.625

Table 5.1: Test vectors for the DCT component

INPUT	CORRECT OUTPUT	OUTPUT FROM CHIP
67	599.0000000000	599.0000
89	52.9811859131	52.1875
155	57.0928878784	56.9375
43	-142.5351104736	-142.625
27	-47.3761520386	-47.3125
89	-95.5557708740	-95.375
0	191.4194335938	191.3125
129	-22.3569240570	-22.3125
57	668.0000000000	668.0000
23	-110.3275833130	-110.6250
51	-169.1290130615	-169.1250
93	146.8634643555	-146.6250
187	5.6568512917	5.6250
67	112.6426925659	112.2500
189	-171.80003845215	-171.7500
1	182.0400238037	181.7500

Table 5.2: Test vectors for the DCT component

INPUT	CORRECT OUTPUT	OUTPUT FROM CHIP
178	543.0000000000	543.0000
45	69.1196136475	68.6250
39	179.6697082520	179.3125
52	89.9775009155	89.8750
12	85.5599212646	85.5000
78	78.4635848999	78.3750
49	99.3166503906	99.1875
90	-52.2684936523	-52.3750
48	780.0000000000	780.0000
56	-192.2438201904	-192.875
120	117.8692398071	117.6875
87	-203.7928466797	-204.0625
65	18.3847789764	18.3750
34	57.8238716125	57.4375
167	-25.8620243073	-25.8125
203	81.3584060669	81.1250

Table 5.3: Test vectors for the DCT component

INPUT	CORRECT OUTPUT	OUTPUT FROM CHIP
23	393.0000000000	393.0000
32	-101.68026733340	-102.3125
44	39.5251770020	39.3125
45	-63.0471076965	-63.2500
47	10.6066026688	10.5625
23	22.6518669128	22.5000
90	-43.1597023010	-43.1875
89	38.7695465088	38.6875
54	1024.0000000000	1024.0000
67	-253.9251098633	-254.8125
90	-46.6901168823	-46.8125
128	-85.4985427856	-85.8125
235	162.6345520020	162.5625
50	-47.1960029602	-47.5625
190	-145.9795684814	-145.9375
210	176.1038665771	175.6875

Table 5.4: Test vectors for the DCT component

CHAPTER
SIX

TESTING

I Introduction

In this chapter we describe the tester and the results of testing various test chips fabricated through MOSIS, for the purpose of characterizing the basic cells. While testing the different components we have used the same test vectors that we have used in the simulations described in the previous chapter.

II IMS-HS2000 Logic Master System

The IMS-HS2000 is an interactive test and verification system for digital designs. It provides a structured environment for prototype analysis, and allows real-time comparison of the design's performance against expected data output.

The Logic Master is controlled from a IBM PC over a GPIB interface by using either the Screen Interface or the Command Language Interface.

There are five basic steps to testing using this workstation:

1. Verify System Configuration: Verfy that the system contains the I/O card and pods to match the current test requirements.

2. Name Channels and Create Groups: Assign device signals to pod channels and organize channels into logical groups. Organizing channels into groups provides a short-hand method for setting test parameters for multiple channels at once.

3. Wire the Verification Station for the Device: Once channels are assigned to groups, it is easy to see which device pins should be wired to which pods.

4. Specify Data Formats, Timing and Voltage: Set thresholds, display radix, format types, delay, and comparison times.

5. Enter and Run the Pattern: Enter the test vectors, run the test and compare the test resultswith your expected data.

We now present the layouts of the test chips and the test results.

III Basic Cells

This chip, fabricated using 2um, CMOS double metal technology contained an adder, a subtractor and a multiplier with a coefficient of 1010 and a set of shift registers. We had it fabricated to ensure that the custom designed basic cells were functioning properly. A plot of the chip is shown in Figure 6.1. The results of the testing are as follows:

1. Max Clock Speed: 12.5Mhz.

2. Min Spacing between clocks: 5ns

III.1 I/O pads

In all the chips that we designed the I/O pads used were part of the standard library supplied by MOSIS. The input pads do not have any input buffers while the output pads do have a buffer. The delay through these pads is about 5 ns.

IV Transpose

A plot of this chip is shown in Figure 6.2. The chip was tested using the test vectors described in the previous chapter. It was found that the shift registers on the chip were functioning properly. However, the input and output switches were found to be glitched into specific states. These switches always channel one of their two inputs to the output, irrespective of the state of the control. In order to fix this problem the switches may have to be redesigned.

V 1st Stage of the DCT

This chip consisted of eight DCT cells. A block diagram of the chip is shown in Figure 6.3. It was also fabricated using 2um CMOS process. While testing the chip a buffering problem was noticed. This problem can be explained as follows: While designing the 12 bit multipliers, in order to save space, the multiplier cells were coiled around. It was laid out as two rows of multiplier bit cells and metal lines were used to channel data coming out of the lower level bit cells to the ones on the top level. It was realized that additional buffering is required to drive these lines. This problem was rectified and the newly designed multipliers were used in

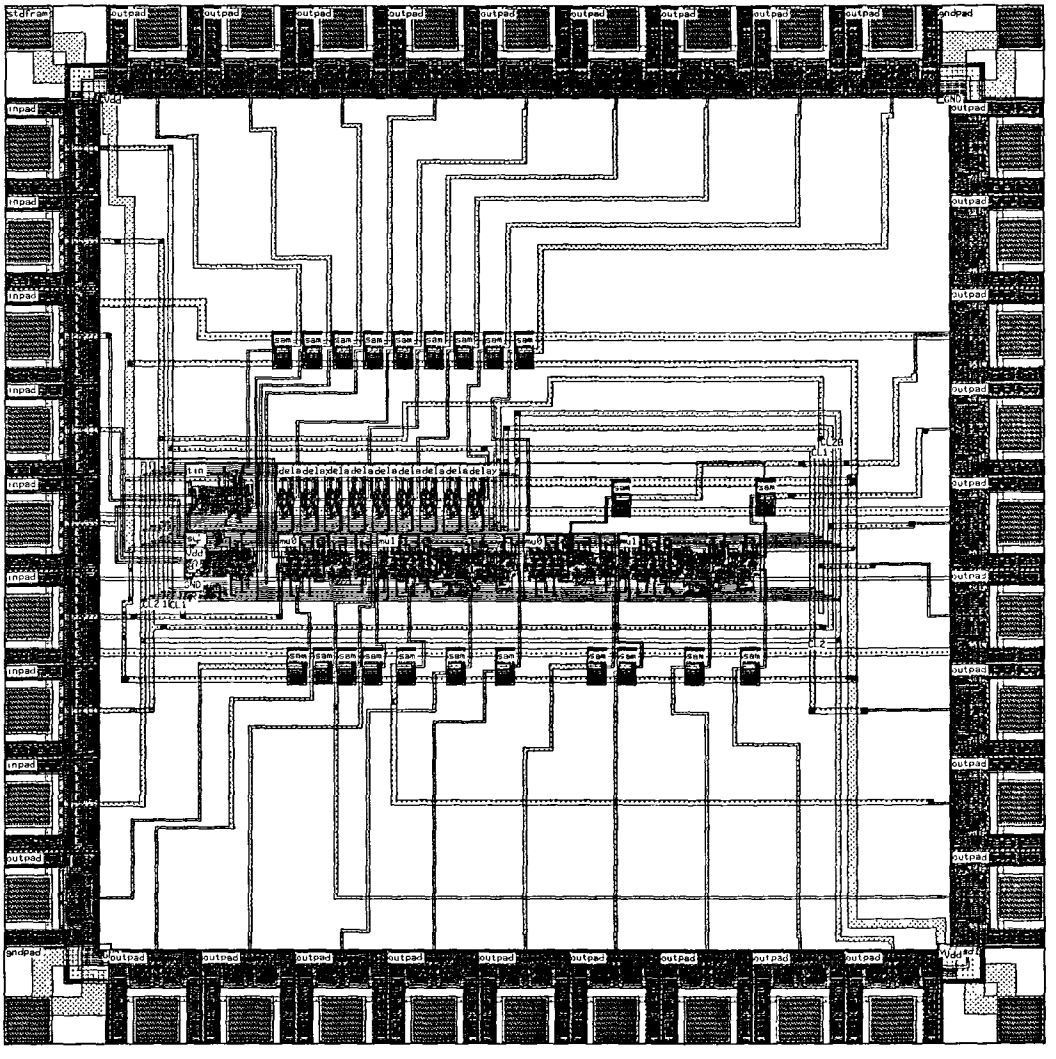


Figure 6.1: Test Chip for the Basic Cells

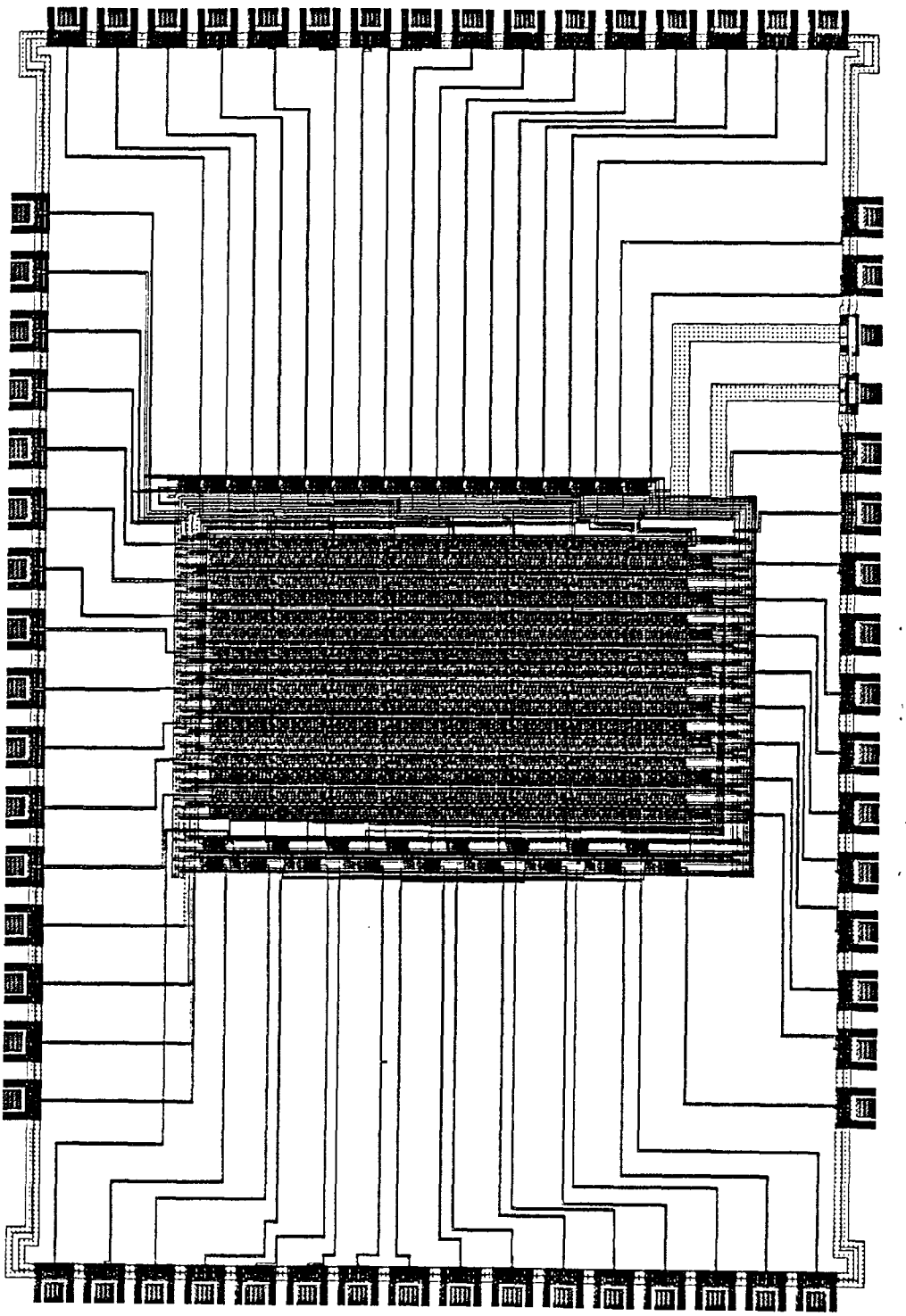


Figure 6.2: Layout of the 8 X 8 Transpose

the Row/Column DCT component. A layout of the chip is shown in Figure 6.4.

VI Row and Column DCT chip

A plot of the chip is shown in Figure 6.5. The test vectors used to test this chip were explained in the previous chapter. This component was found to be operational at upto 12.5 Mhz.

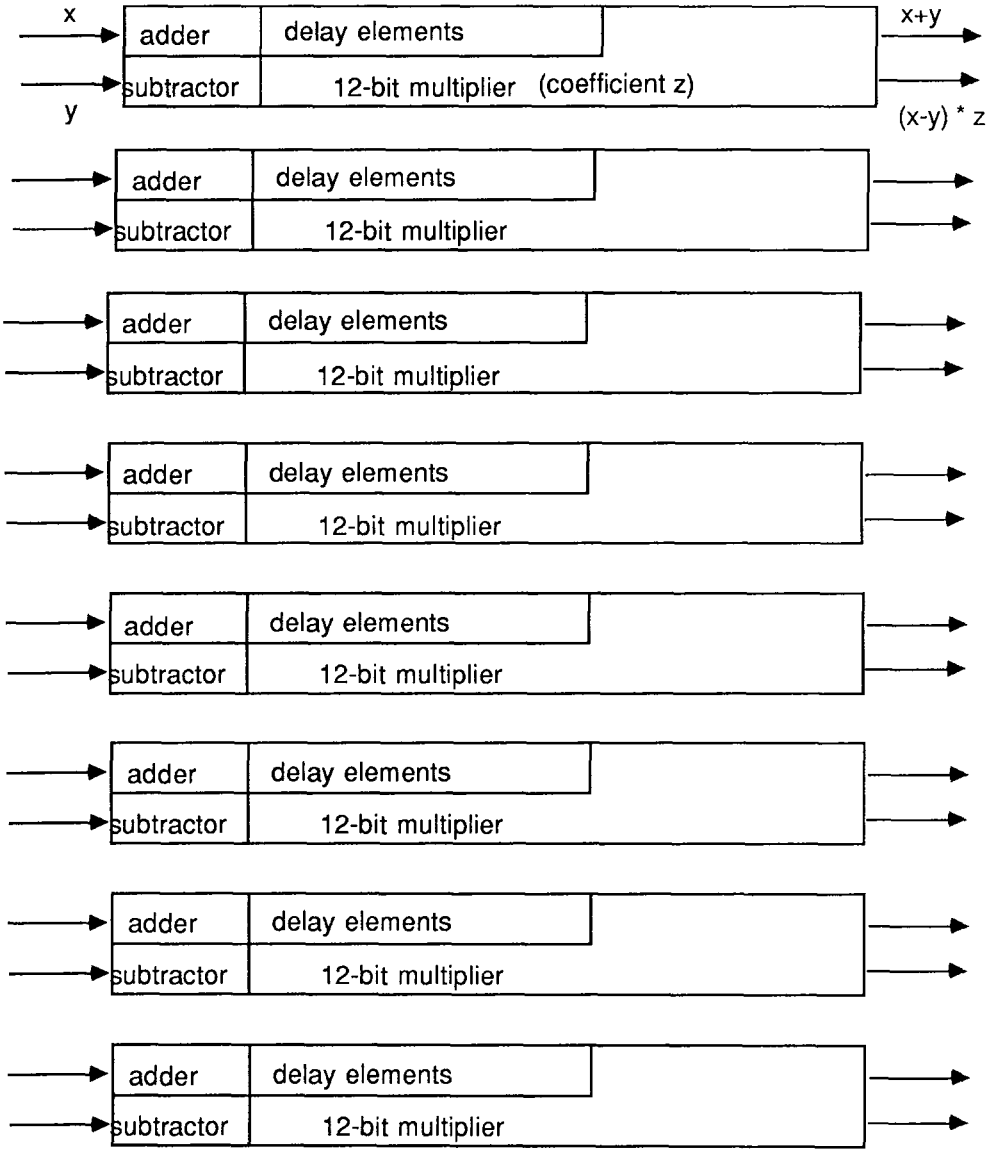


Figure 6.3: 1st Stage Block Diagram

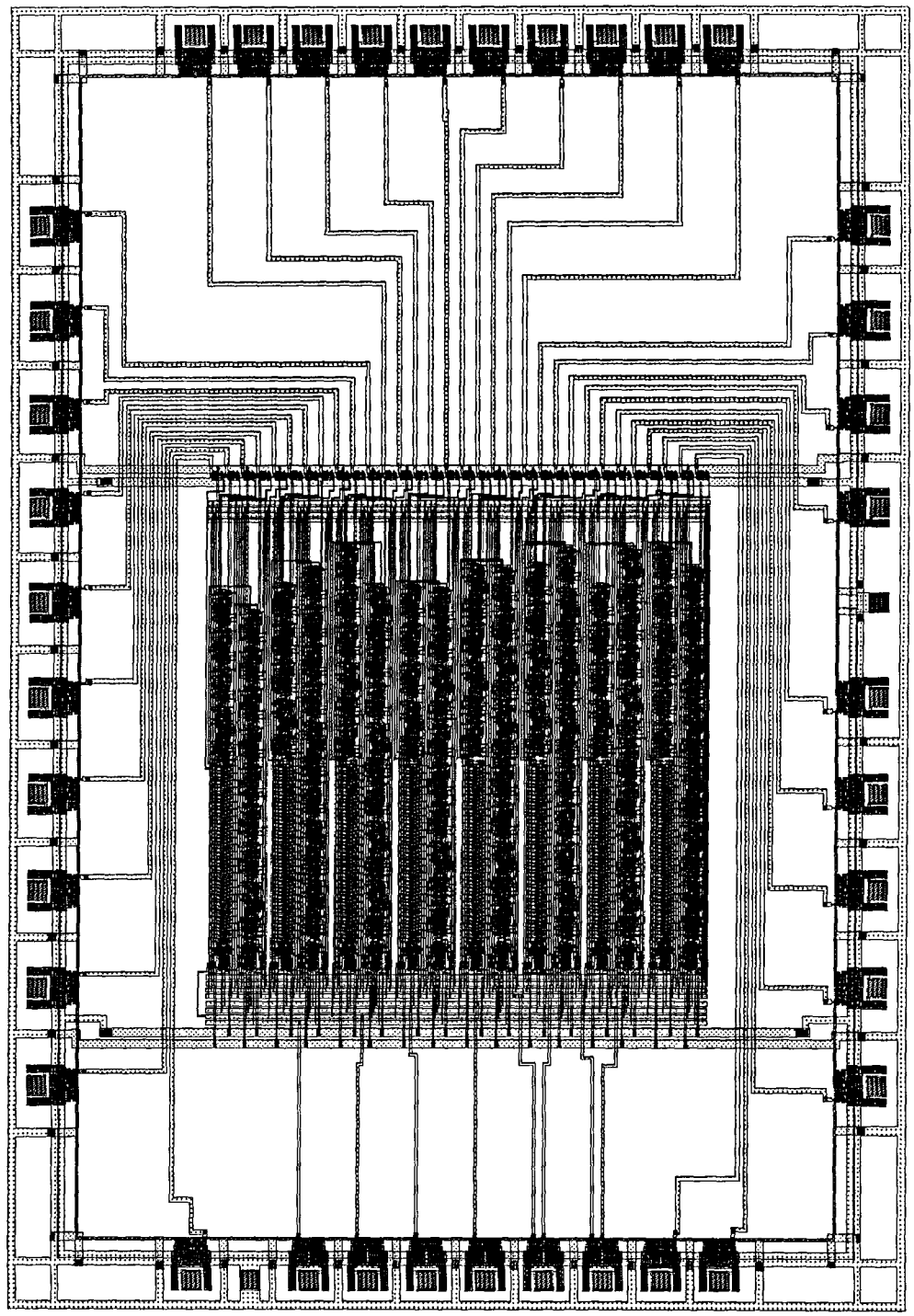


Figure 6.4: Layout of the 1st Stage

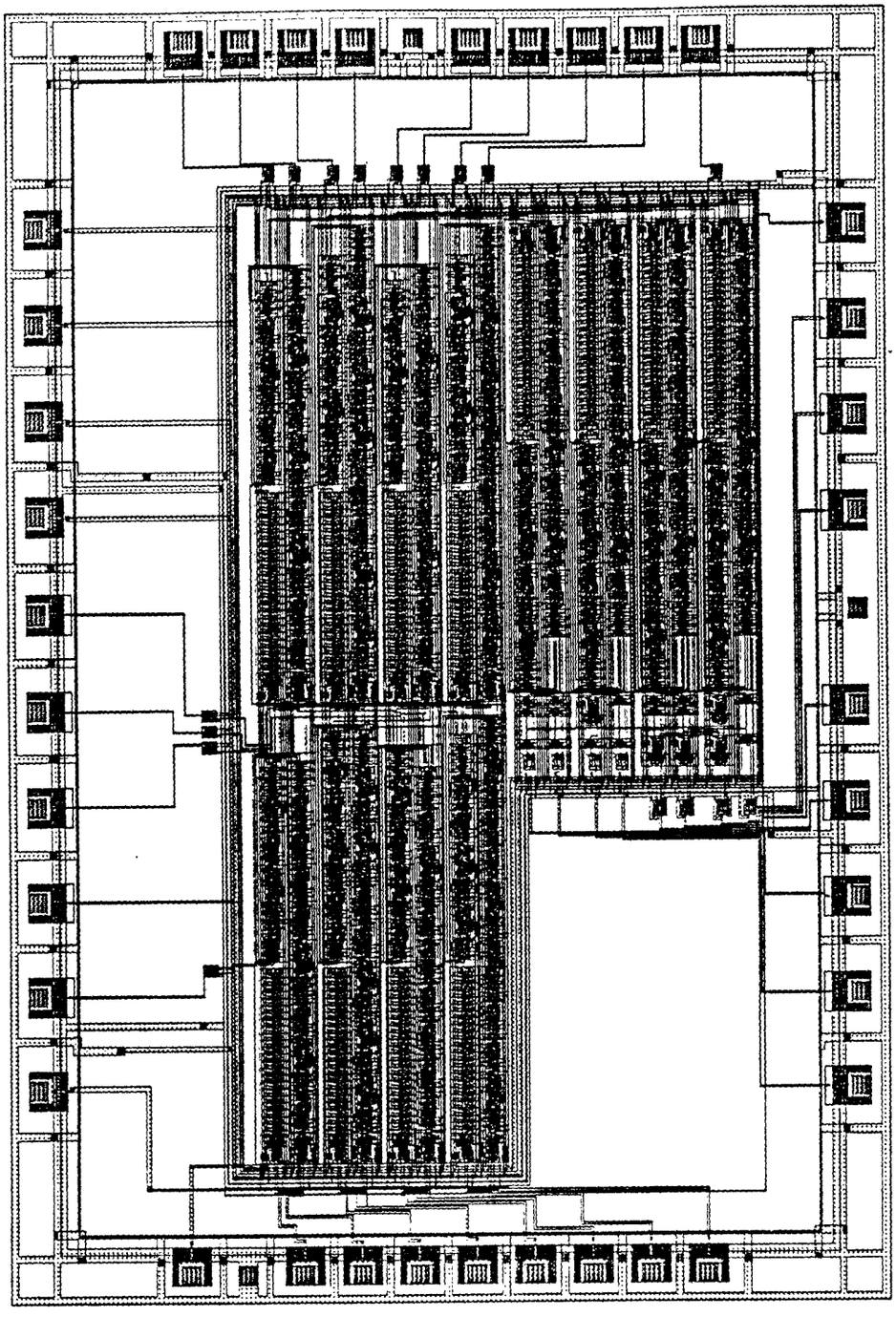


Figure 6.5: Layout of the Row and Column DCT

CHAPTER
SEVEN

CONCLUDING REMARKS

In this thesis we have presented the implementation of a chip that computes the 2-D Discrete-Cosine-Transform of an 8×8 array. Parts of the chip have been fabricated using 2um CMOS process. The overall areas of the two major components have also been presented. It can be seen that the whole chip coresponding to the block diagram shown in Fig 1.2, can be put together on a chip of about 9000λ by 9000λ.

The whole chip is implemented by putting together, just four basic cells. All the cells, except the shift register are based on the transmission gate adder. The basic cells have propogation delays less that 5ns. All the basic cells have been exhaustively simulated using SPICE. We have verified by actual testing that the cells behave as predicted by SPICE.

Next we have presented the detailed design of the Transpose and DCT components. We have elaborated on the design of the pipelined multipliers and the problem of truncation inherent in these multipliers. We have explained the method we have used to overcome this problem. We have also presented the design of the transpose component. Both these components require only one external control and have been simulated using switch level simulators. Timing simulations pre-

dicted that these components can operate at a maximum clock speed of 10Mhz. Both these components have been individually fabricated and tested. The detailed results of the testing has also been presented. Test results show that these components can operate at 10Mhz. Thus, the chip can, at an operating speed of 10Mhz, process aproximately 60 frames of 256×256 pixels per second. This is sufficient for real time applications.

We would like to mention the following as directions for future research:

1. All the components have been fabricated using 2.0um CMOS technology. It is possible to fabricate the chip using 1.2um CMOS technology. This would result in a reduction in the area occupied by the chip and possibly, an increase in the maximum operating speed. It would then be possible to put together a chip that would compress images in blocks of 16×16 pixels.

2. In the chip that we have described we have represented the coefficients for the multipliers as 12 bit numbers. If a higher degree of precision is required we can easily increase the precision with which the coefficients are represented. It is also an easy task to increase the internal precision that we maintain within the chip, as this would only require an increase in the number of shift registers that make up the transpose component.

Thus the design that we have presented is not only simple, but also offers faster, fully pipelined image compression as opposed to some of the existing DCT processors that require complicated arithmetic and control units. Using this design we can achieve a high throughput and successive inputs can be processed with no

delay.

BIBLIOGRAPHY

[1] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete Cosine Transform. *IEEE Trans. Comput*, C-23: pp. 90-94, Jan, 1974.

[2] M.R. Haralick. A storage efficient way; to implement the discrete cosine transform. *IEEE Trans. Comput*, C-25, pp. 764-765, July, 1976.

[3] B.D. Tseng and W.C. Miller. On computing the discrete cosine transform. *IEEE Trans Comput*, C-27, pp. 966-968, Oct, 1978.

[4] W.H. Chen, C.H. Smith, and S.C. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Trans. Commun*, COM-25, pp. 1004-1009, Sept, 1977.

[5] M.J. Narasimha and A.M. Paterson. On the computation of the discrete cosine transform. *IEEE Trans. Commun*, COM-26, pp 934-936, Jun, 1978.

[6] J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-28, pp. 27-34, Feb, 1980.

[7] R.F. Lyon. Two's complement pipeline multipliers. *IEEE Trans. Commun*, Apr, 1976.

[8] Norihiko Ohwada, Tadakatsu Kimura, and Masanobu Doken. LSI's for digital signal processing. *IEEE Journal of Solid State Circuits*, SC-14, No. 2, pp. 214-220, April, 1979.

[9] P. Denyer and D. Renshaw. *VLSI Signal Processing: A Bit-Serial Approach.* Addison-Wesley Publishing Co., 1985.

[10] Glasser,L.W. and Dobberpuhl,D.W. *The Design and Analysis of VLSI Circuits.* Addison Wesley Systems Series, 1985.

[11] Hwang,K. *Computer Arithmetic, Principles, Architecture and Design.* John Wiley and Sons, 1979.

[12] Kung,H.T. Why Systolic Architectures? *IEEE Trans Comput* pp. 37-46, Jan, 1982.

[13] Mead,C. and Conway,L. *Introduction to VLSI Systems.* Addison–Wesley Series in Computer Science, 1980.

[14] Mukherjee, A. *Introduction to nMOS and CMOS VLSI System Design.* Prentice-Hall, 1986.

[15] Pucknell, D.A. *Basic VLSI Design, Principles & Applications* Prentice-Hall of Australia Pty Ltd, 1985.

[16] Weste,N. and Eshraghian,K. *Principles of CMOS VLSI Design – A Systems Perspective.* Addison–Wesley VLSI Systems Series, 1985.

[17] Jeffrey D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[17] Jeffrey D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.