THESIS REPORT

Master's Degree

ASIC Design of Bit-Serial and Bit-Parallel Discrete Cosine Transform Processors

by V. Karunakaran
Advisor: K.J.R. Liu

M.S. 94-3

INSTITUTE FOR SYSTEMS RESEARCH

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

ASIC DESIGN OF BIT-SERIAL AND BIT-PARALLEL DISCRETE COSINE TRANSFORM PROCESSORS

by

Vignarajah Karunakaran

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1994

ABSTRACT

Title of thesis: ASIC DESIGN OF BIT-SERIAL AND BIT-PARALLEL DISCRETE COSINE TRANSFORM PROCESSORS

Name of degree candidate: Vignarajah Karunakaran

Degree and Year: Master of Science, 1994

Thesis directed by: Professor K.J. Ray Liu, Ph.D., Electrical Engineering

Designs of the bit-serial and bit-parallel versions of the Discrete Cosine Transform Processor using the universal IIR filter module are presented, with emphasis on the bit-serial design. A bit-serial cell mini-library was created. The designs were performed with the AlliedSignal Aerospace Microelectronics Center's 1.2 micron double metal p-well CMOS standard cell library. The core of the bit-serial design is the 18-bit data x 8-bit coefficient bit-serial multiplier, whose design is also presented in detail; the multiplier is capable of handling negative data and negative coefficients, and has an accuracy of $o(2^{-16})$. The 8-point 18-bit bit-serial DCT has a maximum clock speed of 139.0 MHz and 55.6 MHz under best and worst case conditions respectively. Two bit-parallel design implementations are presented, one with straight bit-parallel multiplier cells and the other with ROM multipliers using distributed arithmetic. The bit-parallel designs are also 8-point, but have an 8-bit wide input and a 12-bit wide output, thereby calculating with much less precision. The parallel multiplier chip's maximum speed under best and worst case conditions is 28.4 MHz and 11.4 MHz respectively, whereas the ROM multiplier chip's is 36.3 MHz and 14.5 MHz respectively. All three designs have a throughput of one clock cycle, with respect to their data input rates. The latencies for the bit-serial and bit-parallel designs are 38 and 5 cycles respectively.

DEDICATION

To my Family

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASA	AlliedSignal Aerospace
ASIC	Application Specific Integrated Circuits
CMOS	Complementary Metal Oxide Semiconductor
DC1.2	Double metal CMOS 1.2 micron
DCT	Discrete Cosine Transform
DHT	Discrete Hartley Transform
DSP	Digital Signal Processing
DST	Discrete Sine Transform
ERC	Electronic Rules Check
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
HDTV	High Definition Television
IIR	Infinite Impulse Response
I/O	Input/Output
KLT	Karhunen-Loeve Transform
LSB	Least Significant Bit
MEC	Microelectronics Center
MGC	Mentor Graphics Corporation
MHz	mega hertz
MSB	Most Significant Bit
SIPO	Serial-In Parallel Out
SNR	Signal to Noise Ratio
VLSI	Very Large Scale Integration

Chapter 1 - The IIR Filter Structure for the Discrete Cosine Transform

Introduction

The Discrete Cosine Transform (DCT) is a near optimal transform, as it statistically approaches the optimal Karhunen-Loeve Transform (KLT); the KLT minimizes the mean square error of the system, but is difficult to employ because of its computational complexity, which includes requiring data from the future. The DCT has a very high energy compaction capability. The 2-dimensional (2-D) DCT is widely recognized as the most effective tool in speech and image data compression, enabling a very high bit-rate reduction. It is suitable to be used in high speed video processing applications such as High Definition Television (HDTV), among a variety of other signal processing applications.

K.J. Ray Liu and C.T. Chiu proposed an architecture to compute the 2-D DCT from a frame-recursive point of view [2]. This approach lead to two real-time parallel lattice structures for successive frame and block 2-D DCT. The DCT and DST can be dually generated from the same structure. They showed that these structures are fully pipelined with throughput of N clock cycles for an NxN successive input data frame. The number of multipliers required is a linear function of the transform size N. There is no constraint on N. These structures are modular, regular and locally connected; a d-dimensional architecture would need only d 1-D DCT blocks, without transposition. Liu and Chiu went on to present unified parallel lattice structures for time-recursive DCT, DST and DHT, which was an extension to the above mentioned two architectures. This architecture had a throughput of o(1).

K.J. Ray Liu, C.T. Chiu, R.K. Kolagotla and J.F. Ja' Ja' extended the lattice architectures to optimal unified lattice filter structures and IIR filter modules [1]. These filter structures preserve the advantages of the lattice structures while reducing the hardware complexity in half; they merge the buffering and transform operation to reduce the hardware complexity to o(N) -- reduced from the previous low of o(N.logN) [6]; these not only deliver block transforms, but also generate time-recursive transforms, i.e., the transform of the N-point sequence [x(t+1), x(t+2),......x(t+N)] is generated one clock cycle after the transform of [x(t), x(t+1),......x(t+N-1)] is generated; the traditional time required is o(logN). Table 1.1 gives a summarized comparison between a few top DCT algorithms by Liu-Chiu [2], Chen [8], Lee [9] and Hou [10].

DCT IIR Filter Architecture

Input data arrive serially in most real-time signal processing applications. The DCT is capable of handling these data without any buffering, unlike in traditional FFT based algorithms. Let us consider the N-point data sequence [x(t), x(t+1),......x(t+N-1); t = 0,1,2,....]. The 1-D DCT and 1-D DST for this sequence is defined as follows [4]:

$$Xc(k,t) = C(k)\sqrt{\frac{2}{N}}\sum_{t}^{t+N-1} x(n)\cos\left(\left(n-t+\frac{1}{2}\right)k\frac{\pi}{N}\right) \qquad \text{for } k = 0,1,...N\text{-}1$$

$$Xs(k,t) = C(k)\sqrt{\frac{2}{N}}\sum_{t}^{t+N-1} x(n)\sin\left(\left(n-t+\frac{1}{2}\right)k\frac{\pi}{N}\right) \qquad \text{for } k = 1,2,...N$$

where
$$C(k) = \frac{1}{\sqrt{2}} \quad \text{if } k=0 \text{ or } N$$
$$= 1 \quad \text{otherwise}$$

Here the time index in Xc(k,t) and Xs(k,t) denotes that the transform is for data starting at x(t), and not the DCT at time t. Now, the time-recursive relationship between Xc(k,t) and Xc(k,t+1) can be obtained by eliminating the effect of the first term of the previous sequence and updating the effect of the last term of the current sequence [1].

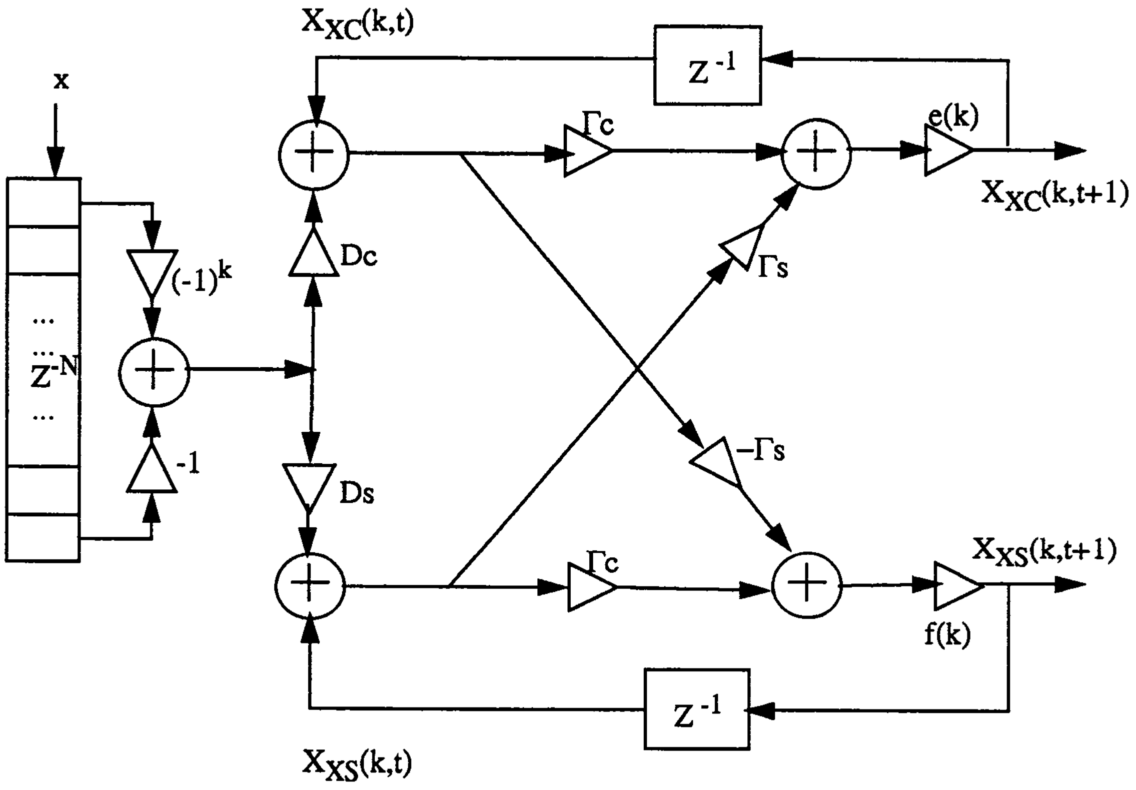
We would like to compute the time-recursive DCT through filter structures. This can be achieved if we can view the transform operation as linear shift invariant system that transforms the input sample sequence into transform coefficients. This can be done by looking at the transfer function of the DCT [1]. The transfer function is derived and expressed in a Z transform format:

$$Hc(z) = \sqrt{\frac{2}{N}}C(k)\left((-1)^k - z^{-N}\right)\cos\left(\pi\frac{k}{2N}\right)\frac{(1-z^{-1})}{\left(1-2\cos\left(\pi\frac{k}{N}\right)z^{-1}+z^{-2}\right)}$$

This Hc(z) is an FIR, since the poles in the denominator are cancelled by the zeros of $((-1)^k - z^{-N})$ in the numerator. If we factor out the updating vector $(1-z^{-N})$, we can see that the basic structure of all transforms is composed of an FIR and an IIR filter with a second order denominator and a first order numerator; this means that we are using an IIR filter to realize an FIR filter [1]. As mentioned before, the hardware complexity is greatly reduced due this IIR filter realization. The DCT can thus be realized with a shift array register and a second order IIR filter.

Fig. 1.2 illustrates the universal IIR filter module. Fig. 1.3 is the universal structure customized for the DCT and DST, with expressions for coefficients, etc.; Table 1.3 lists the coefficients. Table 1.2 lists the coefficients for the universal lattice modules presented in Fig. 1.1. Fig. 1.4 presents the parallel IIR filter structure for the 1-D DCT and DST. DXT is a common name for different discrete sinusoidal transforms such as DCT, DST (Discrete Sine Transform), DHT (Discrete Hartley Transform) and DFT (Discrete Fourier Transform).

Fig. 1.1: The Universal Lattice Module



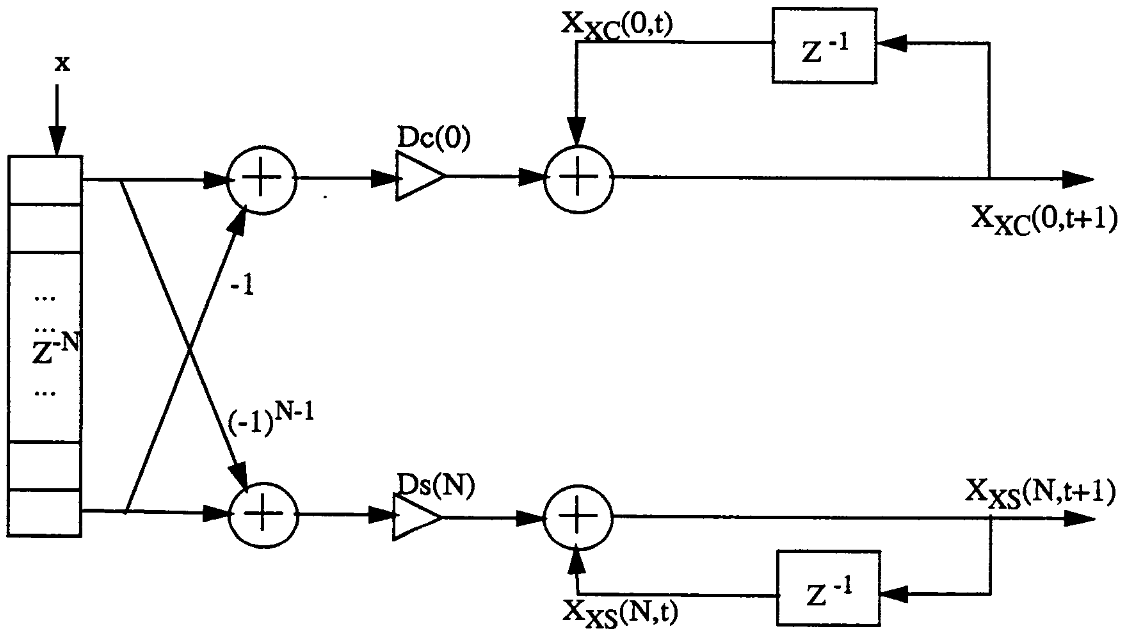


Fig. 1.2: The Universal IIR Filter Module

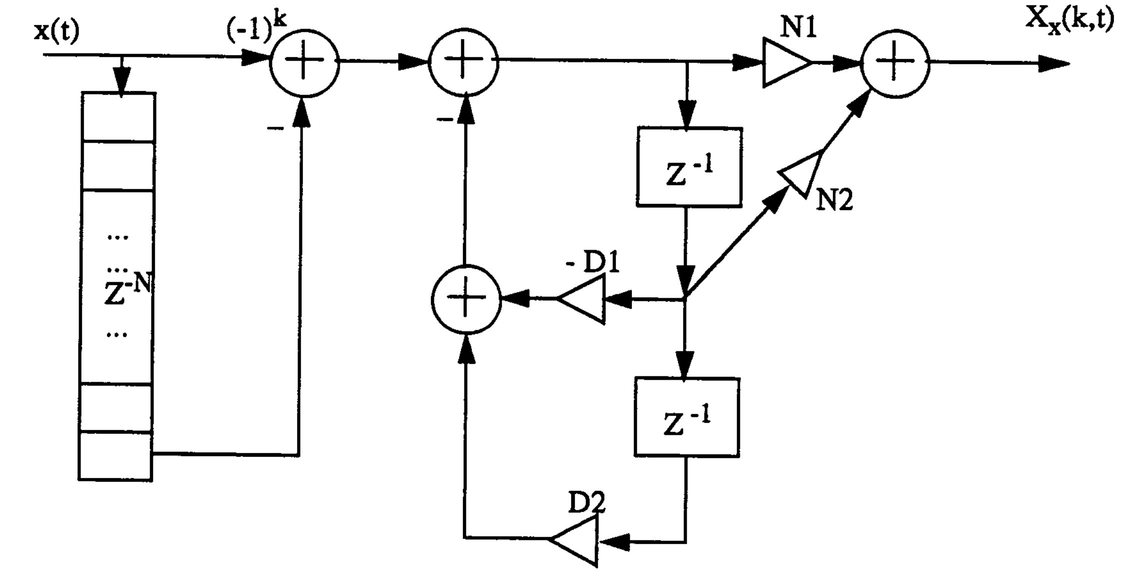


Fig. 1.3: The IIR Filter Structure for the DCT and DST

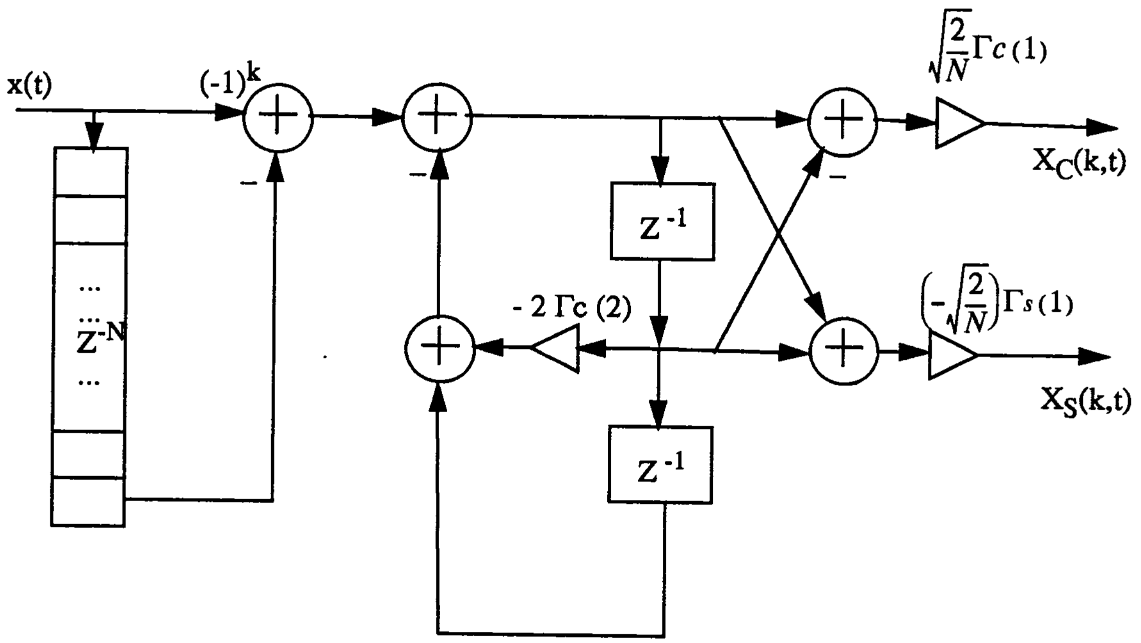


Fig. 1.4: The Parallel IIR Filter Structure for 1-D DXT

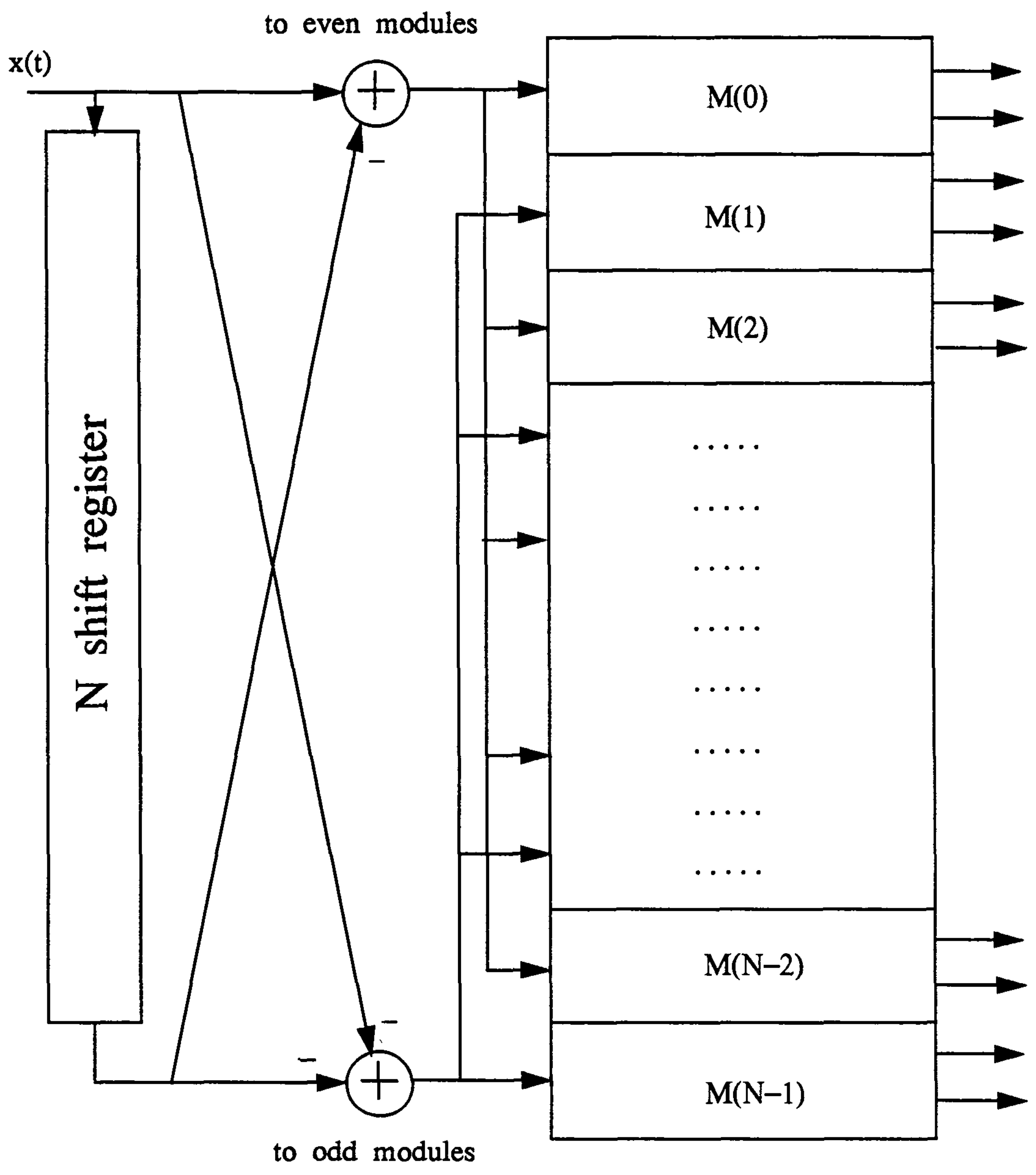


Table 1.1: Comparison of different DCT algorithms

	Liu-Chiu1	Liu-Chiu2	chen *et. al.*	Lee	Hou
No. of Multipliers	$6N-4$	$4N$	$N ln(N)$ $-3N/2+4$	$(N/2)\ln(N)$	$N-1$
latency	N	$2N$	$N/2$	$[\ln(N)(\ln(N)-1)]/2$	$3N/2$(order)
limitation on transform size N	no	no	powerof 2	power of 2	power of 2
communication	local	local	global	global	global
I/O operation	*SIPO*	*SISO*	*PIPO*	*PIPO*	*SIPO*

Table 1.2: Coefficients of the Lattice structure for the DXT

	Γ_c	Γ_s	D_c	D_s	$e(k)$	$f(k)$
DCT/DST	$\cos(\pi k/N)$	$\sin(\pi k/N)$	$C(k)\sqrt{\frac{2}{N}}$ $\cos(\pi k/2N)$	$C(k)\sqrt{\frac{2}{N}}$ $\sin(\pi k/2N)$	1	1
DHT/DFT	$\cos(2\pi k/N)$	$\sin(2\pi k/N)$	$\sqrt{\frac{1}{N}}$	0	1	1
LOT/CLT	$\cos(\pi/2N)$	$\sin(\pi/2N)$	$\sqrt{\frac{1}{N}}(-1)^k j\cdot$ $\exp -j\theta_k$ $\sin(\pi/4N)$	$\sqrt{\frac{1}{N}}(-1)^k j\cdot$ $\exp -j\theta_k$ $\sin(\pi/4N)$	$\exp j2\theta_k$	$\exp j2\theta_k$

Table 1.3: Coefficients of the universal IIR filter structure for the DXT

	k	n	$D1$	$D2$	$N1$	$N2$
DCT	k	N	$2\cos(\pi k/N)$	1	$C(k)\sqrt{\frac{2}{N}}$ $\cos(\pi k/2N)$	$-C(k)\sqrt{\frac{2}{N}}$ $\cos(\pi k/2N)$
DST	k	N	$2\cos(\pi k/N)$	1	$-C(k)\sqrt{\frac{2}{N}}$ $\sin(\pi k/2N)$	$-C(k)\sqrt{\frac{2}{N}}$ $\sin(\pi k/2N)$
DHT	0	N	$2\cos(2\pi k/N)$	1	$\sqrt{\frac{1}{N}}[\cos(2\pi k/N)$ $-\sin(2\pi k/N)]$	$\sqrt{\frac{1}{N}}$
DFT	0	N	$2\cos(2\pi k/N)$	1	$\sqrt{\frac{1}{N}}[\cos(2\pi k/N)$ $+j\sin(2\pi k/N)]$	$\sqrt{\frac{1}{N}}$
CLT	0	$2N$	$\exp j2\theta_k$ $2\cos(\pi/2N)$	$\exp j4\theta_k$	$\sin(\pi/4N)^2$ $\exp j\theta_k$	$(-1)^k\sin(\pi/4N)$ $\cos(\pi/4N)\exp j4\theta_k$

Chapter 2 - Design Background and Tools

Fractional Two's Complement Arithmetic

In general, any real number represented in two's complement form, with m bits of integer and n bits of fraction, can be evaluated as follows:

$$a_{m-1}a_{m-2}\ldots\ldots a_2a_1a_0.a_{-1}a_{-2}\ldots\ldots a_{-n} = -a_{m-1}.2^{m-1} + \Sigma\, a_k.2^k \quad \text{for } k = -n \text{ to } m-2$$

Example: 1011.1101 = -8+0+2+1+0.5+0.25+0+0.0625 = -4.1875

Conversion from a binary representation of a real number to a two's complement number goes as follows:

(1) if the number is positive, the two's complement representation is the same as the number itself; (2) if the number is negative, simply flip/invert the bits and add a 1 to the lsb.

Example: -4.1875 = -0100.0011 ----> 1011.1101

Conversion from two's complement to binary representation of the real number is the reverse procedure: (1) if the msb is a 0, the number is positive, and therefore the two's complement representation is the same as the number itself; (2) if the msb is a 1, the number is negative, so simply prepend the negative sign after flipping/inverting the bits and adding a 1 to the lsb.

Example: .1100 ----> -.0100 = -.25

ASIC Design Using ASA MEC DC1.2 Library

The entire design was performed using the AlliedSignal Aerospace Double Metal CMOS 1.2μ P-well standard cell library. It is one of the most advanced silicon gate libraries available. Its performance over the full military environment and small feature size makes it a valuable asset for those looking for high speed, high density and system-on-chip applications. The library consists of 683 standard cells and 70 macro design cells (themselves made from the standard cells) for a total of 753 cells. The library carries an extensive set of digital logic, including families of on-chip ROM's, static RAM's, dual port RAM's and FIFO memory elements, microprocessor building blocks called bus cells, arithmetic operation cells, a full set of scan testable cells, input and output pads, scan input and output pads, testability modules, non-bus scan memory and scan bus elements.

The cells have already been laid out in 1.2μ double metal CMOS P-well technology. Spice simulations have already been performed on these cells along with data obtained from the layout of the cells. The worst case propagation delays, setup and hold times, from pin to pin, and minimum (and maximum, if applicable) pusle widths of signals are specified in the data sheets for these standard cells. The simulation models for each cell contain functionality and all of the cell-specific timing.

The timing numbers are extracted for best case and worst case conditions, corresponding to a temperature range of -55 C to 125 C, process variables and a supply voltage range of 5.5 V to 4.5 V. The transition from best to worst case simulation is facilitated through a derating factor, which is set to 0.4 for best and 1.0 for worst. A derating factor of anywhere between 0.4 and 1.0 is specified to obtain conditions in between best and worst. The derating factor, D(T), has the following relationship with temperature, T, and any constant To (all temperature expressed in Centigrade):

$$D(T) = D(To) \times \frac{(273+T)}{(273+To)}$$

If we normalize the derating factor at 125 C to 1.0, then we have the following equation, which would enable us to figure out the derating factor at any given temperature, or the corresponding temperature for any given derating factor:

$$D(T) = \frac{(273+T)}{398}$$

A simple ASIC Design Process Flow would entail the following:

1. Design specifications and/or requirements.
2. Conceptual design.
3. Detailed design.
4. Schematic capture using standard cells and hierarchical modules (designed from standard cells).
5. Preliminary layout.
6. Test plan.
7. Preliminary design review.
8. Generation of simulation test vectors.
9. Functional and timing simulation of schematic.
10. Block place and route, using "black box" cells for routing.
11. Back Annotation.
12. Timing simulation.
13. Chip preparation, reticles and fabrication.
14. Critical design review (CDR).
15. Assembly and test.

The DC1.2 library is available with the Mentor Graphics IDEA Station 8.2 software; i.e., the simulation models are written with this software. The designs were performed using the MGC v8.2 software tool suite; Design Architect was used for schematic capture, Design Viewpoint Editor for design viewpointing and QucikSimII for functional and timing simulations, and the Design Manager for design management. The Mentor Graphics language AMPLE was used for most programming.

ASA MEC is in the process of developing a SOI 1.6μ standard cell library. This library is scheduled to contain approximately 150 cells and be available as an autosynthesis library, the cells being very small. It is intended to capture the two DCT designs in this new library, using VHDL and AutoLogic for design capture, autosynthesis and optimization. The DCT designs are to be fabricated along with the test chip for the library.

Unfortunately, the DC1.2 library does not contain any bit-serial standard cells. Therefore, a mini library of bit-serial hierarchical cells was necessary; this bit-serial library is presented in the next chapter.

Our DCT designs were performed up to step 9 as above.

Functionality and Timing Verification

QuickSimII was used for functionality as well as timing simulations, as mentioned in the previous section. An integral part of the Mentor Graphics v8.2 tool suite, QuickSimII is a powerful multi-level simulator and debugging tool. QuickSimII provides pinpoint accuracy, fast iterations and a large capacity for submicron ASICs and dense board-level designs. QuickSimII allows the designer to use compiled logic quickpart models, VHDL models, behavioral models and hardware models in any combination.

QuickSimII simulates with 12 states: 3 logic levels by 4 strengths. The logic levels are 0, 1 and X, and the strengths are S, R, Z and I (strong, resistive, high impedance and indeterminate), the precedence of strengths being in the given order; these combine to produce 0s, 1s, Xs, 0r, 1r, Xr, 0z, 1z, Xz, 0i, 1i and Xi; the states 0, 1 and X, by default, refer to 0s, 1s and Xs respectively.

Timing, which includes propagation delays, setup/hold/pulse width violations, pin capacitances, pin and net loading, backannotation capability, etc., for the simulation models is attached through technology files, equation files and back annotation files. The equation files are sufficiently detailed to allow for pre-layout (of chip) estimates of interconnect capacitance and derating for process/voltage/temperature variations. These estimates are based on default values assigned to simulation model properties and variables. Properties derived from extractions and back annotated to a viewpoint, will override the default values. MEC implements the following equation to compute the delays through a cell:

$$\begin{aligned}\text{Delay} = \; & \text{De\_Rating} * (\text{TP}_{\text{intrinsic}}) \\ & + \text{De\_Rating} * (\text{Drive} * (\text{Net\_Cap} + \text{Input\_Cap})) \\ & + \text{netdelay}_{\text{max}}\end{aligned}$$

"De_Rating" is the derating factor mentioned in the previous section. "$\text{TP}_{\text{intrinsic}}$" is the intrinsic propagation delay through the cell, obtained through spice simulations of the cell layout, measured in nano seconds (ns). "Drive" is the drive strength of the output pin measured in ns/pf. "Net_Cap" is the total net capacitance, measured in pf; its value is either a pre-backannotation estimate based on assumed worst case layout or the actual parasitic capacitance extracted from the layout. "Input_Cap" is the total input capacitance of the pins that the output is driving; for tristate outputs, this includes the tristated (non-driving) pin's capacitance; this is also measured in pf. The equation file contains two basic sets of equations, one for normal (non-tristated) outputs, and one for tristated outputs. In the tristate equation the output capacitance of the node driving the net is subtracted from the summation of all pin capacitance on the net, because it has already been accounted for in the timing numbers derived from SPICE.

The interconnect delay time extracted from the layout process, "netdelay", is one more item which is included in the overall propagation delay of a signal. This delay time is not a simulation model property and therefore cannot be included in the equation file. Instead QuickSimII evaluates this delay prior to performing setup/hold time check, and prior to determining the propagation delay through a cell. Because this net delay is pre-processed the effect of the delay will not be visible on the incoming signal net. However, its effect will be visible in both functionality and in the apparent "overall" propagation delay through a cell. Each input to a cell will have its own netdelay value; the simulator will use the largest netdelay value in calculating the propagation time. In the case of a register, the netdelay of the clock is used for a clock to output transition. Any netdelay on other inputs will be taken into account while performing setup or hold time checks.

The QuickSimII model symbols and functional models are technology-independent. The specific technology is specified through a unique label ("DC1" for double metal CMOS 1.2µ) in the "viewpoint" for the simulation. The derating factor could also be specified in the viewpoint, but defaults to 1.0.

Block Place & Route, Layout and Extraction of Timing and Parasitics

The netlist conversion was performed by the "dfitosl" tool, which is an interface tool between MGC v8.2 EDDM database and Silvar Lisco v3.218 layout tool. Place and route, compaction, layout, and extraction of timing were performed with Silvar Lisco v3.218. The extracted parasitics and timing information were then backannotated into the design viewpoints in MGC QuickSimII and simulations were rerun to obtain final timing. Then the layouts were brought into the MGC ICEnvironment v8.2 set of tools in order to perform a layout versus schematic (LVS) check, DRC check and verify extracted parasitics. The LVS checks report any shorts or missing instances, cells or nets, etc, and are performed in MGC ICTrace. Then the DRC rules are run in MGC ICRules to verify that there are no violations. The extraction of lumped capacitances and distributed delays are performed in MGC ICExtract; lumped capacitance is the total parasitic capacitance on the whole net, even if the net has multiple fanout, whereas the distributed delays give an estimate of the delay through a particular net segment from one output pin into an input pin. Once timing is verified after backannotation the designs are ready for chip preparation, CDR and then fabrication.

Chip Area Calculation

The area of each cell is provided in the technology files in square microns. Therefore, it is possible to calculate the total cell area, even before a preliminary layout, once you have a stable design after functional (and possibly timing) simulations. The total chip area cannot be done until a preliminary layout (or place and route) is performed. However, MEC estimates the total chip area to be, on average, twice the total cell area, and thus giving an initial estimate of die size, so that the prototype and/or production packages could be ordered.

An Ample program was written to calculate the total cell area and the estimated chip area from the design hierarchy file (extracted from the design simulation database) and an input file with all cells being used and the associated cell areas; these two files serve as the inputs to the "calc_area" program along with the name of the output file for placing the calculated areas.

There are two other vehicles to find out the total cell area prior to layout. One is through the Electronic Rules Checker program written by MEC and the other is through the pre-route processor, scbuild, which is a feature of the Silvar Lisco block, place and route tool.

SNR

The minimum signal to noise ratio (SNR) required for most digital imaging applications is 40 dB. SNR of a system is defined as

$$SNR = 20\log\left(\left|\frac{Signal}{Noise}\right|\right) = 20\log\left(\left|\frac{I}{O-I}\right|\right)$$

where I denotes the input signal magnitude and O denotes the output signal magnitude. The difference in magnitude between output and input is the noise. For the purpose of our designs, we could define the expected result as the input and the simulated result as the output.

Hardware and Computer System

All design activity was performed on a 65 Mip Sun Sparc Station 10 machine with an enormous 4 giga byte local disk space. The operating system was SunOS 4.1.3. All programs written in Ample and C were used with double precision floating point arithmetic.

Testing

A program "dct_out.amp" was written in Ample to calculate the DCT of an input sequence. The program is specific to the bit-serial design; however, it was modified slightly, to a new program by the name "dctparll_out.amp", to run with the bit-parallel design. The "dct_out.amp" program takes the following as inputs: the output file to which calculations are to be written, the number of bits in one data word, the size of the transform (eg: 8-point), the clock period and the latency of the design. The program specifies the calculated DCT output for each clock cycle, taking into account the latency of the design. The program is given below. Since it is tedious to go through approximately 2^{18} cycle simulation for an 18 bit, 8-point DCT, the input data sequence used is 00....0, 00....1, 001....1, 01....0, 01....1, 011....1, 10....0, 10....1, 101....1, 11....0, 11....1, 111....1.

```
function dct_out_flr(outfile: string{default = "dct_out.S2_flr"},
                     nbits         : integer{default = 18},
                     N             : integer{default = 8},
                     ck_period: integer{default = 20},
                latency          : integer{default = 38} ) // = 20 cyc + 18 bits
{
  local file1 ;
  local t = 0, xval, incr ;
  local i, j, k ;
  local date = $date() ;
  local DCT = ["NONE",void], C = ["NONE",void];
  local lastN = ["NONE",void] ;
  local c1 = sqrt(2/N) ;
  local max_incr, msbs ;
  local prevDCT0 ;

  file1 = $open_file($free_stream_id(), $strcat("/home/kvk/dct_test/",outfile),@write);

//      WRITE HEADER INFO TO FILE
  $writes_file(file1,"\/ Outputs of DCT for \'complt_S2\' input DATA\n") ;
  $writes_file(file1,"\/ Clock period  = ",ck_period," ns\n") ;
//      INITIALIZATIONS
  for (k = 0; k <= N-1 ; k=k+1)
      {
      C = $create_vector(N,C) ;
      C[k] = 1 ;
      DCT = $create_vector(N,DCT) ;
      DCT[k] = 0 ;
      lastN = $create_vector(N,lastN) ;
      lastN[k] = 0 ;
      }
  C[0] = 1.414213562 ;
  max_incr = pow(2,nbits-2);

  for (j = 0; j <= 3; j = j+1)
   {
   if (j==0)    msbs = 0 ;      // Data begins at 000....0
   else if (j==1)msbs = max_incr ;// Data begins at 010....0
   else if (j==2)msbs = 2 * max_incr;// Data begins at 100....0
   else         msbs = msbs + max_incr;// Data begins at 110....0

   incr = 0;
   while (incr < max_incr)
       {
       xval = msbs + incr;
       lastN = shift(lastN,N,xval);// Store the last N data
       $writeln($strcat("Processing  Sequence = ",lastN));
       $writeln($strcat("Processing  DATA = ",xval));
       $writes_file(file1,t,". At Time = ",(latency+nbits*t)*ck_period,
                                  " for DATA = ",lastN,"  ==>\n");
       prevDCT0 = DCT[0];
       for (k = 0; k < N; k = k+1)
         {
             DCT[k] = C[k] * c1 * xform_basis(lastN,t,k,N) ;
             DCT[k] = floor(DCT[k]) ;
             if (k > 0)
                   $writes_file(file1,"  DCT(",k,") = ",DCT[k]," ;");
             else          // The previous DCT[0] applies, not current one
```

```
                        $writes_file(file1," DCT(",k,") = ",prevDCT0," ;");
                if (k == N/2-1)
                        $writes_file(file1,"\n");
           }
         $writes_file(file1,"\n");
         incr = 2 * incr + 1 ;// Increments are 0,01,011,0111,01111,....,01...1
         t = t + 1;
         }
    }
  $close_file(file1) ;
}

function xform_basis(xt,t,k,M),invisible
{
        local n, xbf = 0, pi = 3.141592654 ;
        for (n = t; n <= t+M-1; n=n+1)
        {
         xbf = xbf + xt[n-t] * cos((n - t + 0.5) * k * pi/M) ;
        }
        return xbf;
}

function shift(ary,m,newval),invisible
{
        local j ;
        for (j = 0; j < m-1; j=j+1)
                ary[j] = ary[j+1];
        ary[m-1] = newval;
      return ary ;
}
```

An extract from the output of the program, run for an 8-point, 18 bit, 20 ns clock and 38 cycle latency simulation of a bit-serial design, is given below:

```
// Outputs of DCT for 'complt_S2' input DATA
// Clock period  = 20 ns
0.  At Time = 760  for DATA = [0, 0, 0, 0, 0, 0, 0, 0]  ==>
  DCT(0) = 0 ;  DCT(1) = 0 ;  DCT(2) = 0 ;  DCT(3) = 0 ;
  DCT(4) = 0 ;  DCT(5) = 0 ;  DCT(6) = 0 ;  DCT(7) = 0 ;
1.  At Time = 1120  for DATA = [0, 0, 0, 0, 0, 0, 0, 1]  ==>
  DCT(0) = 0 ;  DCT(1) = -1 ;  DCT(2) = 0 ;  DCT(3) = -1 ;
  DCT(4) = 0 ;  DCT(5) = -1 ;  DCT(6) = 0 ;  DCT(7) = -1 ;
2.  At Time = 1480  for DATA = [0, 0, 0, 0, 0, 0, 1, 3]  ==>
  DCT(0) = 2 ;  DCT(1) = -2 ;  DCT(2) = 1 ;  DCT(3) = -2 ;
  DCT(4) = 0 ;  DCT(5) = -1 ;  DCT(6) = 0 ;  DCT(7) = -1 ;
3.  At Time = 1840  for DATA = [0, 0, 0, 0, 0, 1, 3, 7]  ==>
  DCT(0) = 7 ;  DCT(1) = -5 ;  DCT(2) = 3 ;  DCT(3) = -3 ;
  DCT(4) = 1 ;  DCT(5) = -1 ;  DCT(6) = 0 ;  DCT(7) = -1 ;
4.  At Time = 2200  for DATA = [0, 0, 0, 0, 1, 3, 7, 15]  ==>
  DCT(0) = 18 ;  DCT(1) = -12 ;  DCT(2) = 7 ;  DCT(3) = -4 ;
```

DCT(4) = 2 ; DCT(5) = -2 ; DCT(6) = 0 ; DCT(7) = -1 ;
5. At Time = 2560 for DATA = [0, 0, 0, 1, 3, 7, 15, 31] ==>
DCT(0) = 40 ; DCT(1) = -24 ; DCT(2) = 14 ; DCT(3) = -8 ;
DCT(4) = 4 ; DCT(5) = -3 ; DCT(6) = 1 ; DCT(7) = -1 ;
6. At Time = 2920 for DATA = [0, 0, 1, 3, 7, 15, 31, 63] ==>
DCT(0) = 84 ; DCT(1) = -49 ; DCT(2) = 27 ; DCT(3) = -16 ;
DCT(4) = 9 ; DCT(5) = -6 ; DCT(6) = 3 ; DCT(7) = -2 ;
7. At Time = 3280 for DATA = [0, 1, 3, 7, 15, 31, 63, 127] ==>
DCT(0) = 174 ; DCT(1) = -97 ; DCT(2) = 54 ; DCT(3) = -31 ;
DCT(4) = 18 ; DCT(5) = -11 ; DCT(6) = 6 ; DCT(7) = -3 ;
8. At Time = 3640 for DATA = [1, 3, 7, 15, 31, 63, 127, 255] ==>
DCT(0) = 354 ; DCT(1) = -194 ; DCT(2) = 108 ; DCT(3) = -62 ;
DCT(4) = 36 ; DCT(5) = -22 ; DCT(6) = 12 ; DCT(7) = -6 ;
9. At Time = 4000 for DATA = [3, 7, 15, 31, 63, 127, 255, 511] ==>
DCT(0) = 715 ; DCT(1) = -387 ; DCT(2) = 216 ; DCT(3) = -124 ;
DCT(4) = 72 ; DCT(5) = -44 ; DCT(6) = 24 ; DCT(7) = -12 ;
10. At Time = 4360 for DATA = [7, 15, 31, 63, 127, 255, 511, 1023] ==>
DCT(0) = 1436 ; DCT(1) = -773 ; DCT(2) = 433 ; DCT(3) = -247 ;
DCT(4) = 144 ; DCT(5) = -88 ; DCT(6) = 49 ; DCT(7) = -24 ;
11. At Time = 4720 for DATA = [15, 31, 63, 127, 255, 511, 1023, 2047] ==>
DCT(0) = 2879 ; DCT(1) = -1546 ; DCT(2) = 867 ; DCT(3) = -493 ;
DCT(4) = 288 ; DCT(5) = -175 ; DCT(6) = 99 ; DCT(7) = -47 ;
12. At Time = 5080 for DATA = [31, 63, 127, 255, 511, 1023, 2047, 4095] ==>
DCT(0) = 5764 ; DCT(1) = -3092 ; DCT(2) = 1735 ; DCT(3) = -986 ;
DCT(4) = 576 ; DCT(5) = -350 ; DCT(6) = 199 ; DCT(7) = -93 ;
13. At Time = 5440 for DATA = [63, 127, 255, 511, 1023, 2047, 4095, 8191] ==>
DCT(0) = 11534 ; DCT(1) = -6184 ; DCT(2) = 3471 ; DCT(3) = -1972 ;
DCT(4) = 1153 ; DCT(5) = -700 ; DCT(6) = 398 ; DCT(7) = -185 ;
14. At Time = 5800 for DATA = [127, 255, 511, 1023, 2047, 4095, 8191, 16383] ==>
DCT(0) = 23074 ; DCT(1) = -12367 ; DCT(2) = 6943 ; DCT(3) = -3943 ;
DCT(4) = 2307 ; DCT(5) = -1400 ; DCT(6) = 797 ; DCT(7) = -370 ;
15. At Time = 6160 for DATA = [255, 511, 1023, 2047, 4095, 8191, 16383, 32767] ==>
DCT(0) = 46154 ; DCT(1) = -24734 ; DCT(2) = 13886 ; DCT(3) = -7885 ;
DCT(4) = 4615 ; DCT(5) = -2799 ; DCT(6) = 1595 ; DCT(7) = -739 ;
16. At Time = 6520 for DATA = [511, 1023, 2047, 4095, 8191, 16383, 32767, 65535]
==>
DCT(0) = 92314 ; DCT(1) = -49467 ; DCT(2) = 27772 ; DCT(3) = -15769 ;
DCT(4) = 9231 ; DCT(5) = -5597 ; DCT(6) = 3191 ; DCT(7) = -1477 ;
17. At Time = 6880 for DATA = [1023, 2047, 4095, 8191, 16383, 32767, 65535, 65536]
==>
DCT(0) = 138293 ; DCT(1) = -66795 ; DCT(2) = 25272 ; DCT(3) = -4292 ;
DCT(4) = -4707 ; DCT(5) = 7010 ; DCT(6) = -6158 ; DCT(7) = 3440 ;
18. At Time = 7240 for DATA = [2047, 4095, 8191, 16383, 32767, 65535, 65536, 65537]
==>
DCT(0) = 183912 ; DCT(1) = -74206 ; DCT(2) = 7731 ; DCT(3) = 12270 ;
DCT(4) = -9413 ; DCT(5) = 88 ; DCT(6) = 5418 ; DCT(7) = -4932 ;

Note that since the new DATA at time t is nearly twice the DATA word at time t-1 (i.e., DATA(t) = 2 * DATA(t-1)) until step 16 above, the DCT output at time t should also be nearly twice the DCT output at time t-1 for steps 1 to 16.

Performance Definitions

The *throughput* of a DCT processor is defined as the number of transform words per clock cycle per DCT output pin.

The *latency* of a DCT processor is defined as the number of clock cycles required to see the DCT output from the time of inputting the data. For a bit-serial implementation, it could be interpreted as the number of cycles from the first bit of the data to the first bit of the transform.

The *pipeline depth* of a DCT processor is defined as the number of clock cycles between two transform words. For a bit-serial implementation, it could be interpreted as the number of cycles from the first bit of one transform to the first bit of the next transform. This is the inverse of the throughput.

The *data rate* of a DCT processor is defined as the number of transform words from the whole processor per second. The data rate can be determined by dividing the clock frequency by the pipeline depth.

Chapter 3 - Bit-Serial Cell Library

Motivation for Bit-Serial Design

It was already mentioned that data arrives serially for most DSP applications. To utilize the bandwidth better or due to bandwidth limitations, one may want to send the data bit by bit (bit-serial). A high clock rate would also be desirable. The area of the chip may be a concern. It was desired to see what the differences, advantages and disadvantages of the bit-serial approach to a design were, as opposed to a normal bit-parallel design.

Synchronous Design

The bit-serial DCT is a highly synchronous design. Every cell of the design is synchronous. This is necessary because the bits come in one at a time. The methodology for this design was to bring the least significant bit (lsb) first; the most significant bit (msb) will come in last. There is no sign extension bit in this design as introduced in previous bit-serial designs; thus the redundancy of the sign extension is eliminated. The lsb is accompanied by a control signal which flags the fact that the data being clocked in is the lsb of a new data word.

All data are in two's complement representation. The design is an 8-point 18 bit data with an 8 bit coefficient. The minimum required number of bits for data is 18 for an 8 bit coefficient; this is due to the second order IIR filter, which has a bit-serial multiplier in one of the two loops feeding back -- the multiplier requires 2b clock cycles to multiply for a b bit coefficient. The details of the multiplier designs are presented in the next chapter.

Due to the bit-serial approach, the chip area and packaging are expected to be very small.

The layout is expected to be very dense. There is only one control signal LSB, one data signal DATA, a CK signal and CLR signal as inputs; there are only eight outputs plus LSBOUT, the delayed version of the LSB control signal.

The entire design was created and simulated with the AlliedSignal Aerospace Microelectronics Center's double metal CMOS 1.2 micron (DC1.2) library. The tool kit used from schematic capture through simulation was Mentor Graphics IDEA Station version 8.2_1.

Pipelined Data and Control Methodology

Without loss of generality, let us say that the DCT is N-point, the data is b_0 bits and the coefficient is b_1 bits. The DCT data are real integers. The multiplier coefficients are fractions m, where $-0.5 < m < 0.5$. The output data of all cells, including the multipliers and the DCT processor itself, should be a b_0 bit integer.

The single bit data signal DATA carries in the input data bit by bit. The lsb arrives first. To indicate that DATA has the lsb of a new b_0 bit word, the control signal LSB goes active low. The LSB control signal serves to alert the cell that a new word is beginning; for the multiplier, it also serves as a flag to indicate that the previous bit clocked in was the MSB, in order to enable the multiplier to sample the sign information, etc. For a b_0 bit two's complement number, the MSB could serve as the sign bit.

The cell library, and therefore the design components as they stand alone, are in a sense memoryless. The control signal LSB tells the cells which is the beginning of a word, but there is no way for the cells to know which word they are processing, and for that matter, which bit they are processing. These all augur for a fully pipelined design. The results are computed back-to-back; there is no buffering involved. This enables the cells, and thereby the whole design, to have a throughput of 1 bit per clock cycle. The latency is b_0 for each cell, except for the multiplier small cells that require 2 clocks, which possess a latency of 2, and the bigger multipliers, which posses latencies of 8 for the 8x8 multiplier that requires one clock cycle, 16 for the 8x8 multiplier that requires two clock cycles and 18 for the 18x8 multiplier. These cell designs are described in the following sections. Note that the throughput is increased due to not employing a sign extension bit in the design.

The cell library is presented in the sections to follow. The clock CK is positive edge-triggered and the clear signal CLR is active low in all cells.

Cell BITADD

The schematic for the BITADD cell is given in Fig 3.2. The DC1.2 library standard cells T008M (2-input AND gate), T082M1 (full adder) and T273M3 (high speed 3 bit D-register) are used. Fig. 3.1 is a sheet with all the bit-serial cell library symbols. The BITADD is the very first. Active levels are depicted on these symbols, as well as DC1.2 library symbols.

Signals A and B are the data inputs to be added. The control signal is LSB. LSBDEL is the signal one clock cycle behind LSB, which feeds into the next cell's LSB control. SUM is the summation result. The carryout from the previous cycle's data carries back into the full adder. However, when the LSB control is asserted, the carryin gets cancelled; this prevents the previous data's carryout affecting the current data.

Cell BITSUB

The schematic for the BITSUB cell is given in Fig 3.3. The DC1.2 library standard cells T004M (simple inverter), T032M (2-input OR gate), T082M1 (full adder) and T273M3

Fig. 3.1: Bit-Serial Cell Library Symbols

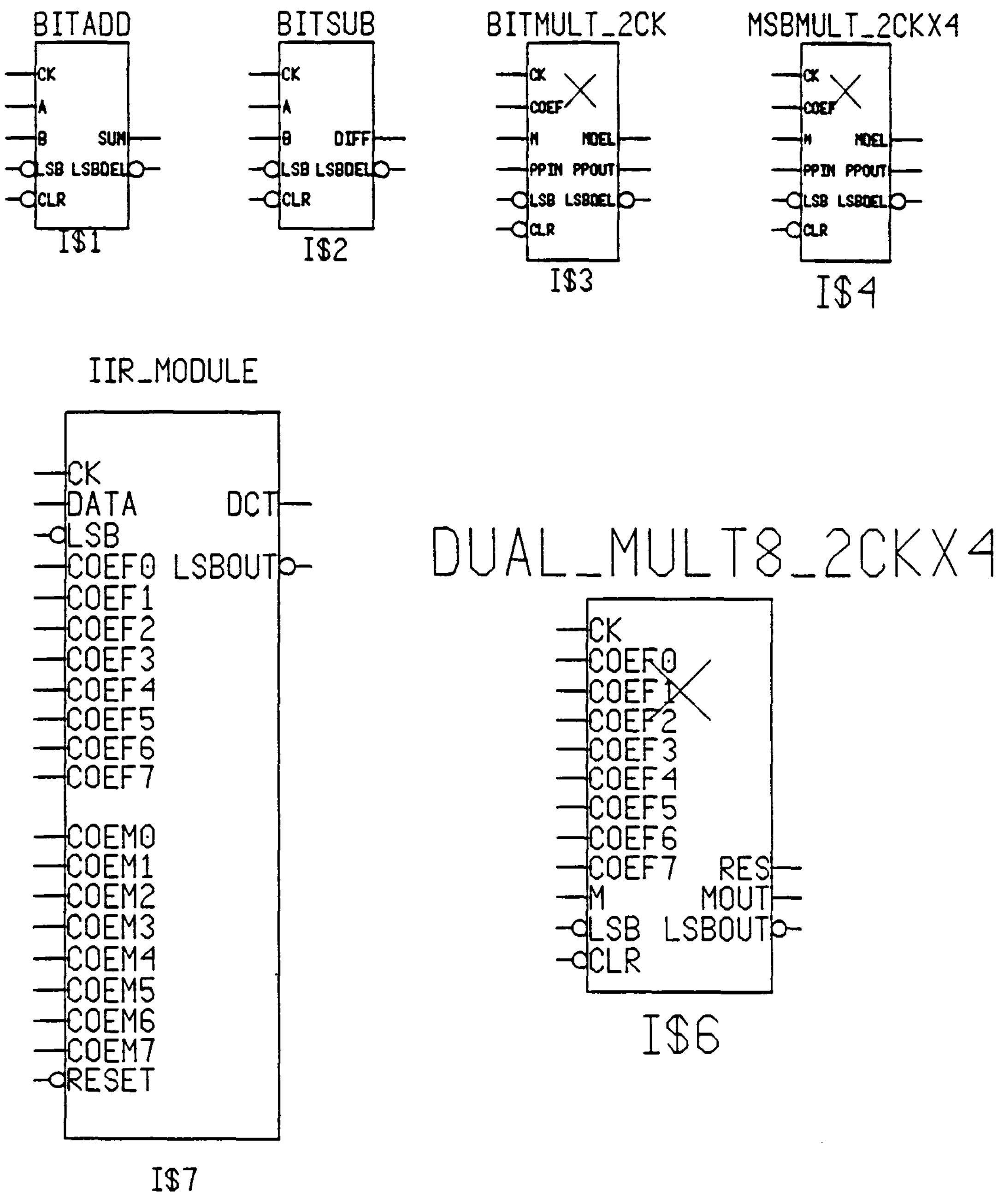


Fig. 3.2: BITADD Schematic

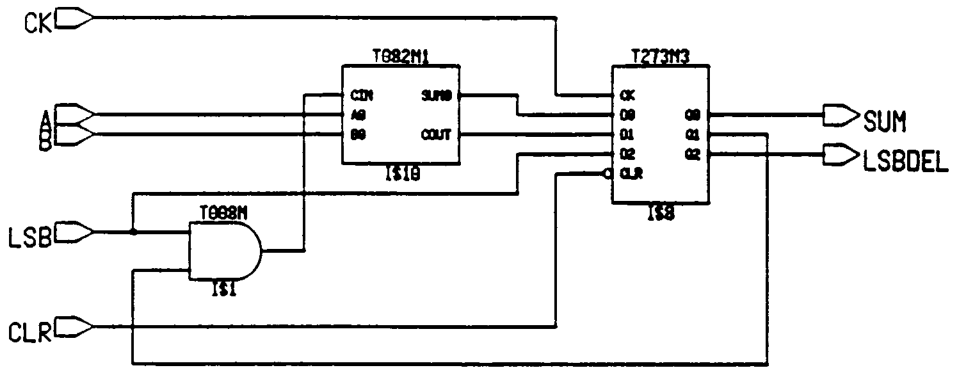
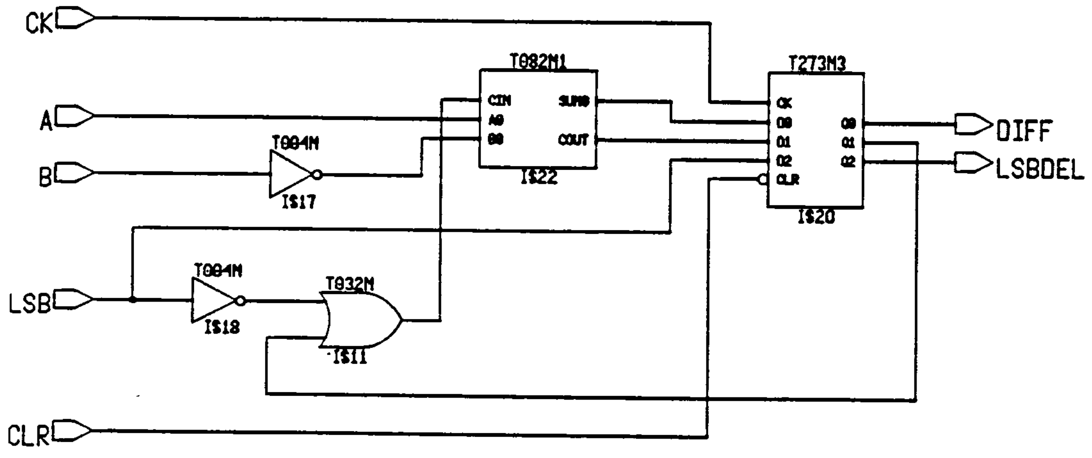


Fig. 3.3: BITSUB Schematic



(high speed 3 bit D-register) are used. The BITSUB symbol is the second from the top left corner in Fig. 3.1.

Signals A and B are the data inputs to be subtracted. The control signal is LSB. LSBDEL is the signal one clock cycle behind LSB, which feeds into the next cell's LSB control. DIFF is the difference between A and B,i.e., DIFF = A-B (the difference). The carryout from the previous cycle's data is fed back into the full adder. However, when the LSB control is asserted, the carryin gets forced to a 1; this allows for the conversion of B to two's complement, explained in the next paragraph.

One may wonder how a subtraction could produce a carryout or how the subtraction can be performed through addition. The subtraction is performed by adding the two's complement of B to A. The two's complement is obtained by inverting the signal and adding a 1 to it. The T004M takes care of the inversion with ease. The 1 is added to the data's lsb (when LSB=0).

Cell BITMULT_2CK

The schematic for the BITMULT_2CK cell is given in Fig 3.4. The DC1.2 library standard cells T004M (simple inverter), T008M (2-input AND gate), T082M1 (full adder), T157M1 (1 bit 2-1 multiplexer) and T273M3 (high speed 3 bit D-register) are used. The BITMULT_2CK symbol is the third from the top left corner in Fig. 3.1.

Signals M and COEF are the multiplicand and coefficient respectively; M is the b_0 bit data passing through, with the lsb coming in first. Signal PPIN is the partial product coming in from the previous BITMULT_2CK stage. A b_0 x b_1 bit bit-serial multiplier will require b_1-1 stages of BITMULT_2CK cells, hooked up in series, and a MSBMULT_2CKx4 at the last stage. At the first stage the PPIN is grounded. Output signal PPOUT is the partial product going to the BITMULT_2CK of the next stage. The PPOUT coming out of MSB-MULT_2CK is the multiplied result.

The control signal is LSB. LSBDEL is the signal two clock cycles behind LSB, which feeds into the next stage's LSB control. MDEL is the signal two clock cycles behind M, which feeds into the next stage's M input. The carryout from the previous cycle's data carries back into the full adder. However, when the LSB control is asserted, the carryin gets cancelled; this prevents the previous data's carryout affecting the current data.

If the COEF is a 0, the data M into the full adder is zeroed out; thus the PPOUT would only be the carryout from the previous cycle (ANDed with LSB) plus the PPIN. If the COEF is a 1, the data in M is added in getting the PPOUT. If the LSB is a 0, the PPOUT is set to its previous value, i.e., it is sign-extended.

It is necessary that the cell have two levels of registering, because the PPOUT has to go out to the next stage one cycle before the data (or multiplicand) M goes there, and the PPOUT requires registering as well. This is why PPOUT is registered once and M and LSBDEL are registered twice. Initially, a cell BITMULT was created with just one level of registering. But this BITMULT design created havoc in the simulation of an 8x8 multiplier MULT8 which used 7 BITMULT and 1 MSBMULT cells hooked up in series; there were numerous spikes and the delay for the multiplier result to settle down after each clock cycle was intolerably high -- 105.2 ns worst case obtained with a maximum clock speed of 9.5 MHZ -- defeating the purpose of the bit-serial approach; the long delay for the result to stabilize was due to the PPOUT signal being totally asynchronous throughout the b_1 stages of the multiplier. The largest delay for the result of an 8x8 multiplier MULT8_2CK result, which used 7 BITMULT_2CK and 1 MSBMULT_2CK cells, to settle down was 3.5 ns worst case obtained with a maximum clock speed of 55.6 MHz. The dual registering does not affect the throughput -- it is the latency that is doubled.

Fig. 3.4: BITMULT_2CK Schematic

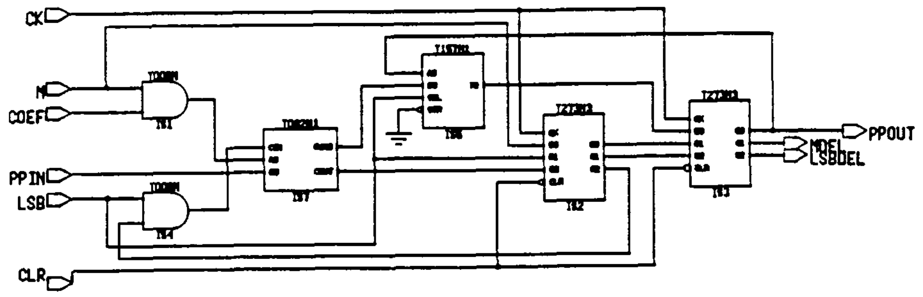


Fig. 3.5: MSBMULT_2CKx4 Schematic

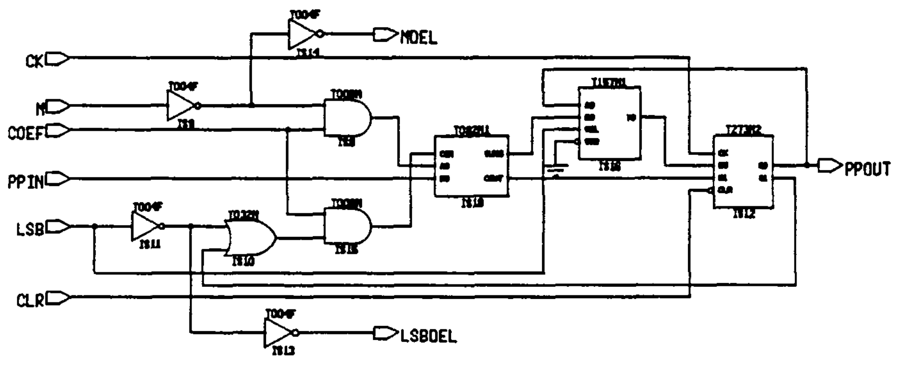
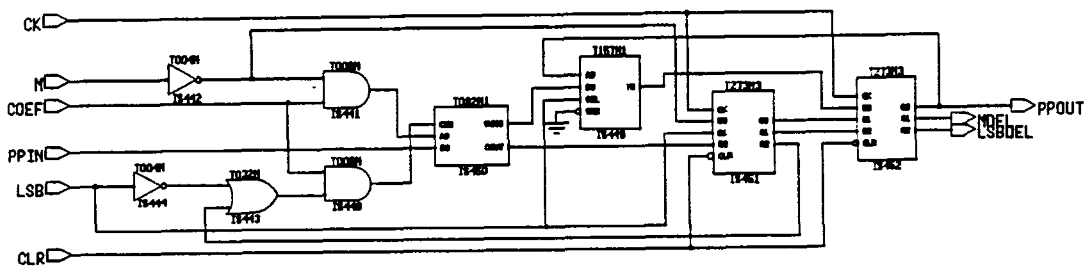


Fig. 3.6: MSBMULT_2CK Schematic



One would expect a b_0 x b_1 multiplication to produce a b_0+b_1 bit result. However, since there are feedback loops in the IIR filter, where the multiplier will be employed, it is necessary to preserve the data at b_0 bits; i.e., b_1 bits of the multiplied result need to be truncated. This is a serious problem. The problem was overcome because the b_1 bits represent the fractional part of the result, since the coefficient is a fraction less than magnitude 0.5, the result can never exceed the value of the input data. The implementation to handle the truncation was again to use LSB, which is what is done when the carryout from the previous cycle is erased by the LSB.

Cell MSBMULT_2CKx4

The MSBMULT_2CKx4 is very similar to the BITMULT_2CK, except that it is geared to handle the msb of the coefficient and also multiplies the result by 4, because the coefficients supplied to the multiplier are 4 times smaller than the actual DCT coefficients. For example, if the actual coefficient is -2, one has to supply -0.5 to the multiplier, or in binary, .1000 0000; the decimal point is implied. This is because the DCT coefficients have magnitudes of up to 2, whereas this multiplier can handle magnitudes of up to 0.5.

The schematic for the MSBMULT_2CKx4 cell is given in Fig 3.5. The DC1.2 library standard cells T004M (simple inverter), T008M (2-input AND), T032M (2-input OR), T082M1 (full adder), T157M1 (1 bit 2-1 multiplexer) and T273M2 (high speed 2 bit D-register) are used. The MSBMULT_2CKx4 symbol is the last from the top left corner in Fig. 3.1.

Signals M and COEF are the multiplicand and coefficient respectively; M is the b_0 bit data passing through, with the lsb coming in first. Signal PPIN is the partial product coming in from the previous BITMULT_2CK stage. A b_0 x b_1 bit bit-serial multiplier will require

b_1-1 stages of BITMULT_2CK cells, hooked up in series, and a MSBMULT_2CKx4 at the last stage. PPOUT is the multiplied result.

The multiplication by 4 is achieved in this cell through a tricky methodology, which fools the next cell into thinking that it is receiving an lsb. The control signal is LSB. LSBDEL is the signal virtually two clock cycles behind LSB, which feeds into the next cell's LSB control; however, in reality, LSBDEL is just a buffered version of LSB, i.e., the LSBDEL skips the two levels of registering. This is done in order to advance LSBDEL so that the output looks like it was shifted twice, which is equivalent to multiplying by 4. The error produced by this shift is of $o(2^{-(b_0-2)})$.

MDEL is the signal, again virtually, two clock cycles behind M, which feeds into the next cell's data input. The carryout from the previous cycle's data carries back into the full adder. However, when the LSB control is asserted, the carryin gets forced to 1; this is how a 1 is added to M to make it a two's complement number.

If the COEF is a 0, the data M and the carryin into the full adder are zeroed out; thus the PPOUT would only be equal to the PPIN. If the COEF is a 1, the data inverted M is added in getting the PPOUT.

In this cell, as opposed to the BITMULT_2CK cell, two levels of registering is implied, but bypassed by the fact that LSBDEL and MDEL are advanced by two cycles. However, the PPOUT signal still needs to be registered before coming out. Initially, along with the cell BITMULT, an MSBMULTx4 was created with just one implied level of registering; but this failed for the same reasons given in the BITMULT_2CK section.

It it quite obvious that this cell has the largest critical path delay when compared to any of the bit-serial cells. An estimate of the critical path delays through this cell would be the addition of the propagation delays through the following cells between signals LSB and PPOUT: t004f, t032m, t008m, t082m1, t157m1 and t273m2. We shall calculate this delay as follows (worst case):

$$tP_{critical} = 0.5 + 0.2 * (0.9 + 0.15) + 1.7 + 1.1 * 0.15 + 1.4 + 0.9 * 0.3 + 2.8 + 0.9 * 0.1 + 1.7 + 0.9 * 0.25 + 0.0 + 3.3 + 0.5 * 0.25 = 12.485 \text{ ns}$$

Note that the t273m2 register does not require a setup time. This estimate is without any netdelay or back annotation calculations. The best case delay would be approximately 5.0 ns. Even if we estimate a 40% increase in delay due to backannotation, we have approximately 17.5 ns worst case and 7.0 ns best case critical path delay. This would accommodate a worst clock speed of 57 MHz and best case clock speed of 143 MHz for any size multiplier.

Cell MSBMULT_2CK

The MSBMULT_2CK is identical to the MSBMULT_2CKx4, except that it does not multiply the result by 4, i.e., it registers the LSB and M signal twice to produce LSBDEL and MDEL respectively. The schematic for the cell is given in Fig. 3.6. This cell is not used in the DCT design.

Chapter 4 - Bit-Serial Multiplier

Multiplier Algorithm

In the previous chapter we went over the designs of the bit-serial multiplier cells, as they were part of the bit-serial library. We also went over how a multiplier is designed using these cells. But, we did not go over the algorithm for bit-serial multiplication. The trickiest part in designing a bit-serial multiplier is implementing the sign extension during multiplication when a you have a negative multiplicand. An example of a bit-serial multiplication is given below:

Let us consider a 4x4 multiplication of -1 x -3 = +3. The result obtained through usual means is:

```
            1 1 1 1        =        -1
          x 1 1 0 1        =      x -3
  ------------------             ------
  1 1 1 1/1 1 1 1                   +3
  0 0 0/0 0 0 0                  =====
  1 1/1 1 1 1
  0/0 0 0 1             <------ two's complement of 1111 since 1101 is negative
  ------------------
  0 0 0 0 0 0 1 1          =        +3
  ==========
```

The sign extensions are those shown in the triangle. It is this implementation that is difficult in a bit-serial design. Previous designs [3] were able to handle only negative coefficients but not negative multiplicands; this is because, as you can see in the example, the sign extension is required only when the multiplicand is negative. The effect of the coefficient being negative is that to add the two's complement of the data, instead of the data itself, in the last stage, which is pinpointed in the example; this can easily be implemented, as done in the MSBMULT_2CKx4 (and MSBMULT_2CK) cells where the data signal M is inverted and a 1 added when control signal LSB is asserted -- only when COEF = 1.

Let us look at an example of the methodology for a bit-serial multiplication; this example has the same -1 x -3 multiplication used in the previous example. In this example, the multiplicand bits come down vertically, one bit per cycle, lsb first and msb last, passing through the coefficients; the coefficients are at the right in a vertical column, lsb at the top and msb at the bottom; each bit of the coefficients represent one stage of BITMULT_2CK, and the msb of the coefficient represents an MSBMULT_2CK:

```
                        1               I
                      1 1               I  direction of
                    1 1 1               I  multiplicand
                  1 1 1 1               I
---------------------------             V
                1 1 1 1         1       I
              1 1 1 1           0       I
            1 1 1 1             1       I
          1 1 1 1               1       V
---------------------------
                1 1 1 1
              0 0 0 0
            1 1 1 1
          0 0 0 1
---------------------------
        0 0 0 0 0 0.1 1
===========================
```

In the above example, the msb of the multiplicand is always on the left most column of a row. The partial products are computed and they trickle down to the last stage, unless there is an lsb somewhere in the middle of the BITMULT_2CK stages, where the running partial product gets cancelled out by the control signal LSB = 0; this prevents partial products from the current multiplicand word affecting the multiplication of the multiplicand word ahead.

DUAL_MULT8_2CKx4 Design Module

The DUAL_MULT8_2CKx4 module is an 18 bit multiplicand by 8 bit coefficient multiplier. The multiplier, in effect, shifts the multiplied result twice in order to multiply it further by 4. It is capable of handling positive and negative multiplicands and coefficients. Because it multiplies the result by 4 to provide the final answer, it is able to accept coefficients of magnitude up to 2 (as opposed to 0.5); the coefficients should be divided by four and supplied to the multiplier. For example, if the actual coefficient is -2, one has to supply -0.5 to the multiplier, or in binary .1000 0000; the decimal point is implied.

Restrictions on this multiplier are: (1) coefficients magnitude should be less than 0.5; (2) multiplicand M has to fall within the range 1110 0000 0000 0000 <= M < 0010 0000 0000 0000; if not, the most significant bit will be truncated by the x4 operation, and thus the sign information will be lost. Keep in mind that the largest, in effect, the multiplier can multiply by 2 at the most -- maximum coefficient magnitude is 0.5 and the double shift (equivalent to x4) give a resultant of x2.

The symbol for the module is shown in Fig. 3.1 and the schematic for this module is given in Fig. 4.1. As shown in the schematic, the multiplier has two paths of 8 bit coefficient multipliers. One path is to evaluate the result if the data is positive and the other is to evaluate the result if the data is negative. Two paths are necessary as the multiplier does not know whether the multiplicand is positive or negative until the msb arrives. When the msb arrives, the lsb of the result is ready to go in the next cycle. The sign of the multiplicand is necessary for determining whether the sign extension, as described in the examples at the beginning of the chapter, is to be performed or not. As such, the two paths independently calculate the result and when the msb arrives, it selects the result through a 2-1 multiplexer.

What are these two independent calculations? One is to assume the multiplicand is positive and perform the multiplication. The other is to assume it is negative and perform the multiplication. When the multiplicand is assumed negative, it is converted to two's complement (this is equivalent to negating it), then multiplied by the coefficients and then, once again, the multiplication result is converted to two's complement, i.e., negated, to obtain the final result. In effect, a double negation is performed so that the result would correspond to the actual multiplicand M. The negation is necessary so that at least the 8 coefficient multiplier path thinks that the multiplicand is negative. The negation is performed by sending the data through the B input of a BITSUB cell and having the A input grounded.

The simulation of this module was tested with worst and best case parameters. The maximum clock speed in worst case was 55.6 MHz; for best case, 139.0 MHz. An Ample program "chk_res.amp" was written to extract the stimulus and results, which were in bit-serial order, and convert to a hex integer and compared the results against established numbers. The accuracy was excellent; an error of $o(2^{-(b_0-2)})$, where $b_0 = 18$, was seen as expected. The throughput of the multiplier is 1 bit per cycle. The latency of the multiplier is $2b_1 = 16$ cycles.

The multiplier was rigorously tested and verified with an Ample program "chk_res.ample", since it is extremely difficult to decipher the output from a bit-serial multiplier report. Due to the length of the simulations and report, these results are not included here.

MULT8_2CKx4 Design Module

The MULT8_2CKx4 module is an 8 bit coefficient multiplier. It is superior to the DUAL_MULT8_2CKx4 in that it can handle multiplicands of sizes between 1 and 16 bits.

However, it cannot handle negative multiplicands; it can handle negative coefficients, though. The multiplier, in effect, shifts the multiplied result twice in order to multiply it further by 4. It is capable of handling positive and negative multiplicands and coefficients. Because it multiplies the result by 4 to provide the final answer, it is able to accept coefficients of magnitude up to 2 (as opposed to 0.5); the coefficients should be divided by four and supplied to the multiplier. For example, if the actual coefficient is -2, one has to supply -0.5 to the multiplier, or in binary .1000 0000; the decimal point is implied.

Restrictions on this multiplier are: (1) coefficients magnitude should be less than 0.5; (2) multiplicand M has to fall within the range 110.....0 <= M < 010.....0; if not, the two most significant bits will be truncated by the x4 operation, and thus the sign information will be lost. Keep in mind that the largest, in effect, the multiplier can multiply by 2 at the most -- maximum coefficient magnitude is 0.5 and the double shift (equivalent to x4) give a resultant of x2.

The loss of accuracy is $o(2^{-(b_0-2)})$, where $b_0 = 8$. The throughput of the multiplier is 1 bit per cycle. The latency of the multiplier is $2b_1-2 = 14$ cycles. This module is not used in the DCT. The schematic for the module is given in Fig. 4.2.

MULT8_2CK Design Module

This module is identical to the MULT8_2CKx4 module except that it does not multiply the final result by 4; the only difference in the schematic is that the MSBMULT_2CK cell is used instead of the MULT8_2CKx4. This module is not used in the DCT.

Restrictions on this multiplier are different from MULT8_2CK accordingly: (1) coefficients magnitude should be less than 0.5; (2) multiplicand M has to fall within the range 10.....0 <= M < 10.....0.

Fig. 4.1: DUALMULT8_2CKx4 Schematic

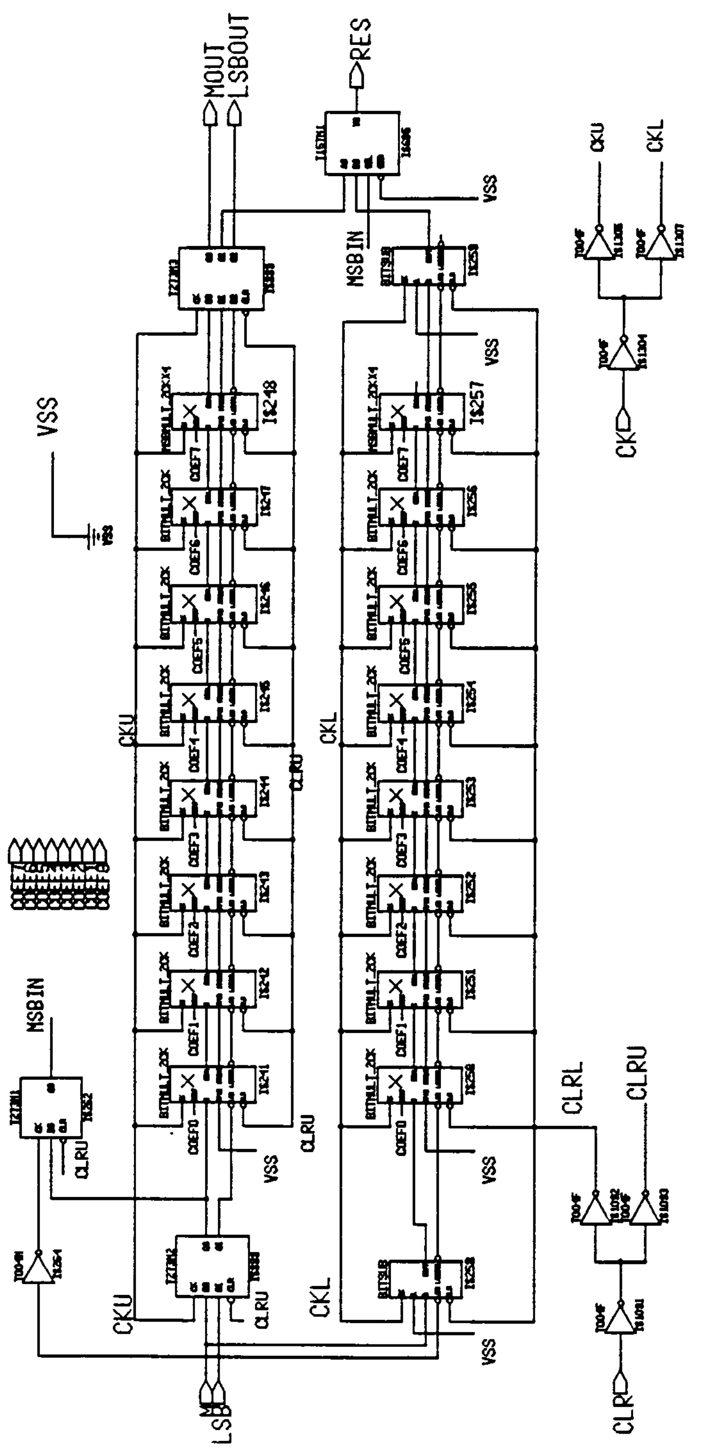
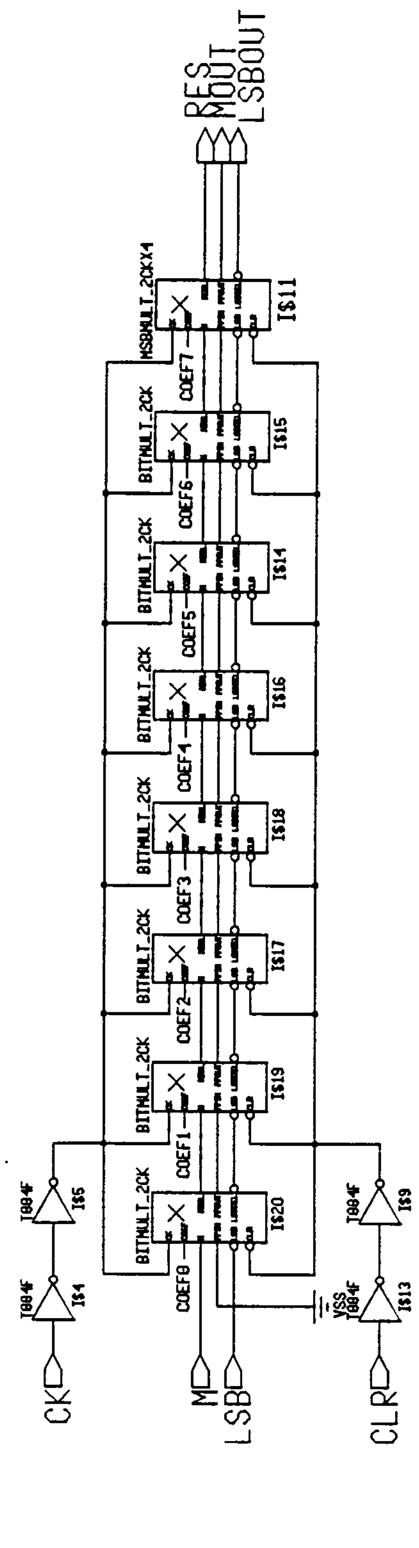


Fig. 4.2: MULT8_2CKx4 Schematic



Chapter 5 - Bit-Serial IIR Module and DCT

Calculation of Coefficients

The coefficients to the multipliers, and thereby to the DCT processor, can be calculated from Fig. 1.2 (and Fig. 1.3) and Table 1.3. The actual DCT coefficients are divided by 4, for reasons given in the description of the DUAL_MULT8_2CKx4 design, before they are hard-wired in the DCTBITSERIAL (the DCT bit-serial top level module). These coefficients are passed on to the IIRMOD8 and are named COEF(7:0) and COEM(7:0), within each IIR filter module, corresponding to -D1 and N1 of Fig. 1.2 and Table 1.3; these coefficients are in turn passed to the DUAL_MULT8_2CKx4 multipliers. We shall see how these coefficients are calculated:

$$COEF = -\frac{D1}{4} = -\frac{1}{2}\cos\left(\pi\frac{k}{N}\right) = -\frac{1}{2}\cos\left(\pi\frac{k}{8}\right) \qquad \text{for k = 0 to 7}$$

$$COEM = -\frac{N1}{4} = \frac{1}{4}C(k)\sqrt{\frac{2}{N}}\cos\left(\pi\frac{k}{2N}\right) = \frac{1}{8}C(k)\cos\left(\pi\frac{k}{16}\right) \qquad \text{for k = 1 to 7}$$

$$COEM = \frac{1}{8\sqrt{2}} \qquad \text{for k = 0}$$

The calculated binary values of the coefficients will be hard-wired in the top level schematic as mentioned earlier. The decimal point is implied. These coefficients were calculated to 17 bit precision, and then rounded to 8 bit precision in an Ample program. The following table of values was produced by the program:

Table 5.1 - Coefficients for Bit-Serial DCT Chip

k	COEF	COEM
0	-0.500000 = .10000000	0.088388 = .00010111
1	-1.847759 = .10001010	0.490393 = .00011111
2	-1.414214 = .10100101	0.461940 = .00011110

k	COEF	COEM
3	-1.530734 = .11001111	0.415735 = .00011011
4	0.000000 = .00000000	0.088388 = .00010111
5	1.530734 = .00110001	0.277785 = .00010010
6	1.414214 = .01011011	0.191342 = .00001100
7	1.847759 = .01110110	0.097545 = .00000110

IIR_MODULE Design Module

The IIR_MODULE is the bit-serial implementation of the unified IIR module, shown in Fig. 1.3, for the DCT [1]. Since the data is bit-serial and the filter is of second order (with two feedback loops), there were very crucial synchronization issues. The lsb's of successive data had to synchronize at the input of the filter. If the number of delays through a loop was to be altered, then the number of bits for the data would also have to be altered accordingly, and vice versa; they had to be the same.

The symbol for the IIR_MODULE is shown in Fig. 3.1; the schematic is given in Fig. 5.1. The latency through the first feedback loop is 18 (latency of the multiplier is 16 plus 1 each for the BITSUB and BITADD); for the second loop it is twice as much, i.e., 36 (34 shifts plus one each for the BITSUB and BITADD).

DCTBITSERIAL Design Module

This is the top level module of the DCT bit-serial design. This is an 8-point, 18 bit data and 8 bit coefficients design. This is the bit-serial implementation of the DCT parallel IIR filter structure [1] shown in Fig. 1.4.

The schematic for the DCTBITSERIAL is given in Fig. 5.2. The two columns of nine 8 bit shift registers (2x9x8 shifts) perform the z^{-N} operation, i.e., delaying the data N data words, where N = 8 here (8-point DCT). Since the data is 18 bits, the data has to be shifted 8x18 times, resulting in the shift register structure seen in Fig. 5.2.

On the right hand side of the schematic is a column of 8 IIR_MODULE filters, each corresponding to a k, where k is in between 0 and 7.

The coefficients should be divided by four and supplied to the IIR_MODULE, for reasons given in the last chapter under the DUAL_MULT8_2CKx4 multiplier description. Restrictions to the design are also governed by those of the multiplier: (1) coefficients magnitude should be less than 0.5; (2) data DATA has to fall within the range 11 0000 0000 0000 0000 <= DATA < 01 0000 0000 0000 0000; if not, the most significant bits will be truncated by the multiplier, and thus the sign information will be lost. It is highly recommended that one stays within a smaller range, as when the multiplied data feeds back into the multiplier again or sent through the other multiplier, the data could be increased in magnitude and thus the msb could be lost. A test on this issue concluded that it is safe to stay within the following range: 11 1111 0000 0000 0000 <= DATA < 00 0001 0000 0000.

There is only one clock input signal CK. To avoid clock, reset and data skews, the design has very effective inverting driver structures right down through its hierarchy. A simulation without any driving buffers resulted in a clock skew of 2000 ns!

Since there are 8 synchronized LSBOUT output signals, one coming out of each IIR-MOD8 module, the effective LSBOUT can be any one of these.

During simulation prior to layout, a worst clock speed of 56 MHz and best case clock speed of 141 MHz were measured. The throughput is 1 bit per cycle. The latency of the design is 38 cycles.

Before getting into block place and route, the design was run through the electronic rules check (ERC) program. There were no errors reported from the program. The following are a few statistics from the program:

```
***********QUICK CHECK STATISTIC REPORT****************
Total number of instances in design = 2277
Total number of pins in design = 10812
Total number of nets in design = 4043
Equivalent gate count in design = 19349.5
Total cell area in design (in sq. microns) = 33961358
*******************************************************
```

The following are a few statistics from the Silvar Lisco blocl place and route pre-processor, scbuild. Bcells refer to big cells, such as ROMs, bit-parallel multipliers, RAMs, etc. Pcells refer to I/O and power pad cells.

```
        Circuit statistics
        ******************
        Nr. of elements   :     2277
          Nr. of Cells    :     2245
          Nr. of Pcells   :       32
          Nr. of Bcells   :        0
        Nr. of Nets       :     3608
        Nr. of Pins       :    10365
        Nr. of Pins/Net   :      2.9 (average)
        Nr. of Pins/Cell  :      4.6 (average)
        Cell Area         :  8.4897e+08 2.0E-1 MICRON**2
        Bcell Area        :  0.0000e+00 2.0E-1 MICRON**2
        Pcell Area        :  5.8142e+07 2.0E-1 MICRON**2
```

The chip I/O is as follows:

<u>Inputs</u>

Data: DATA Control: LSB Clock: CK Reset: RESET

Select: MUXS2, MUXS1, MUXS0

<u>Outputs</u>

Data: DCT0, DCT1, DCT2, DCT3, DCT4, DCT5, DCT6, DCT7 Control: LSBOUT

Fig. 5.1: IIR_MODULE Schematic

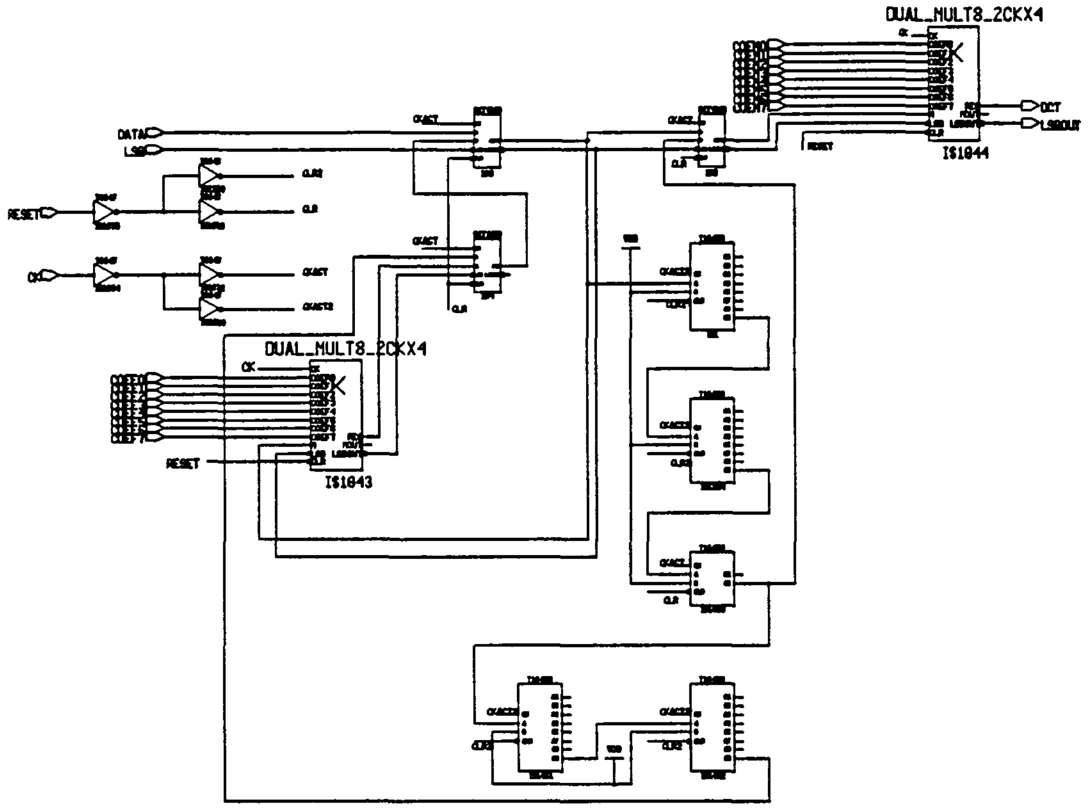
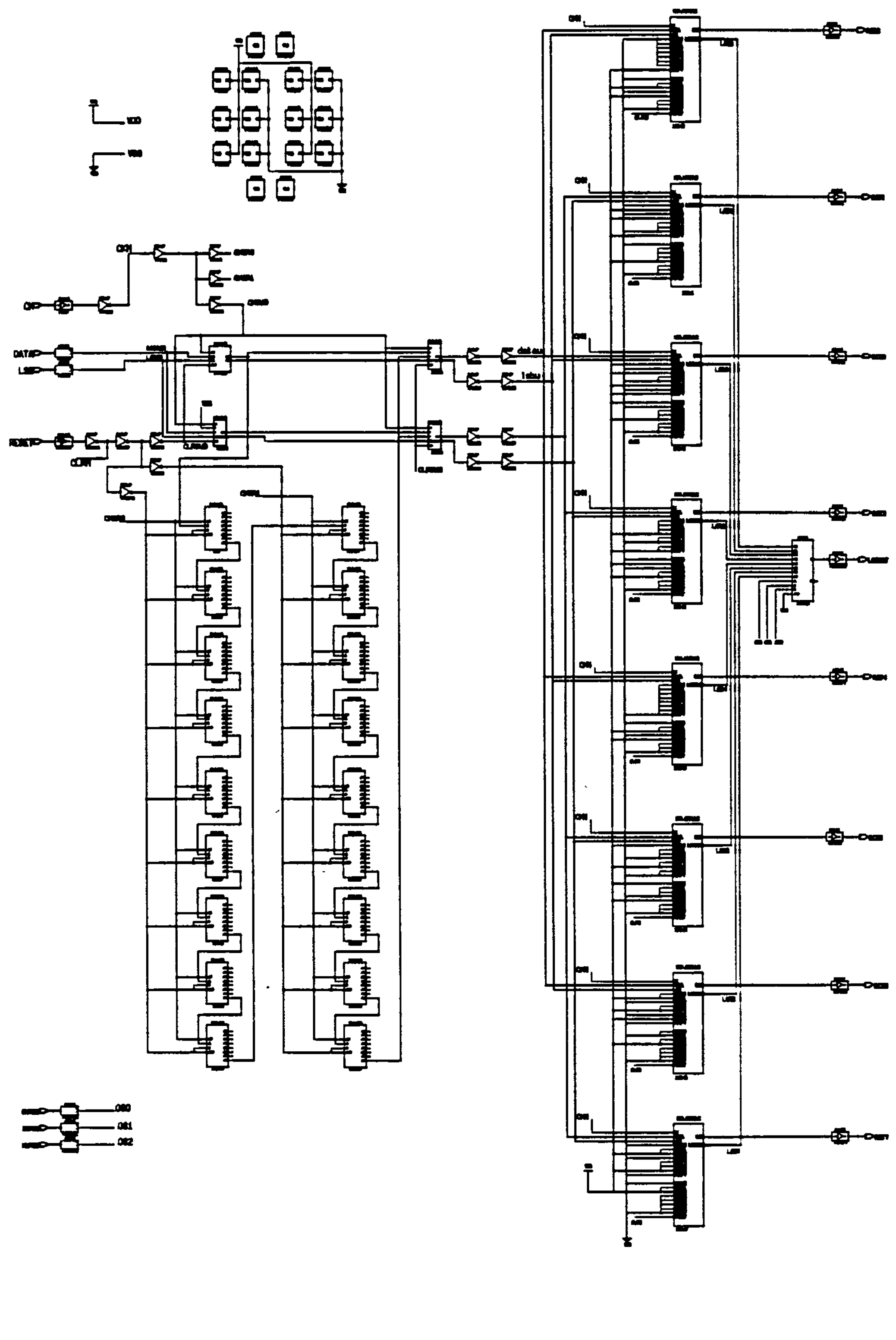


Fig. 5.2: DCTBITSERIAL Schematic



DCTBITSERIAL Layout

The netlist extraction, place and route, layout and extraction of timing were performed with Silvar Lisco v3.218. The following is a summary of the layout extracted from the tool:

```
DESIGN_NAME DCTBITSERIAL
LOGIC_NAME DCTBITSERIAL
DATE_TIME 94/04/05 18:28:41
UNIT_OF_SIZE 200 nanometer
UNIT_NAME 2.0E-1 MICRON
PLI_FILE dct_bser.sc/h34b8-2.pli
INP_WHRATIO 95
INP_RATIO 50
NR_CELLS 2245
NR_BCELLS 0
NR_PCELLS 32
NR_FEEDS_EXT 78
NR_LINKS 0
NR_SPACERS 0
NR_NETS 3606
LENGTH_ON_METAL2 3082670 2.0E-1 MICRON
LENGTH_ON_METAL1 7830874 2.0E-1 MICRON
NR_OF_VIA 12406
NR_OF_MATVIA 1665
WIDTH_OF_ACTIVE_AREA 33912 2.0E-1 MICRON
WIDTH_OF_ACTIVE_AREA 6.7824e+00  mm
WIDTH_OF_ACTIVE_AREA 2.6702e+02  mils
HEIGHT_OF_ACTIVE_AREA 38284 2.0E-1 MICRON
HEIGHT_OF_ACTIVE_AREA 7.6568e+00  mm
HEIGHT_OF_ACTIVE_AREA 3.0145e+02  mils
ACTIVE_AREA 1.2983e+09  2.0E-1 MICRON**2
ACTIVE_AREA 5.1931e+01  mm**2
ACTIVE_AREA 8.0494e+04  mils**2
WIDTH_OF_TOTAL_AREA 40682 2.0E-1 MICRON
WIDTH_OF_TOTAL_AREA 8.1364e+00  mm
WIDTH_OF_TOTAL_AREA 3.2033e+02  mils
HEIGHT_OF_TOTAL_AREA 43316 2.0E-1 MICRON
HEIGHT_OF_TOTAL_AREA 8.6632e+00  mm
HEIGHT_OF_TOTAL_AREA 3.4107e+02  mils
TOTAL_AREA 1.7622e+09  2.0E-1 MICRON**2
TOTAL_AREA 7.0487e+01  mm**2
TOTAL_AREA 1.0926e+05  mils**2
AREA_RATIO 52
```

The above summary gives us a lot of information on the layout. The layout consists of 34 horizontal rows. The number of cells is 2245; the number of pads is 32 (16 I/O and 16 power supply pads). The area_ratio of 52 (or 0.52) tells us that the layout is densely packed with the ratio of interconnect area to cell area of 0.52, i.e., the cell area is nearly twice as much as the interconnect area. The die size is 320 x 341 mils^2, or 8136.4 x 8663.2 micron^2. This yields a total area of 1092.6 x 10^2 mils^2, or 70.488 x 10^6 micron^2. In the common man's units, this is a 8.2 mm x 8.7 mm, or 70.5 mm^2 die. A plot of the layout is given in Fig. 5.3

During the layout of this chip, the prime considerations taken into account were: the total area; the interconnect area to cell area ratio; the die size -- as square a die as possible; optimal placement of I/O and power pads; optimal placement of standard cells, with minimal connecting wire lengths; minimal parasitic delays and capacitances (lumped and distributed) extracted from the layout; maximized active area.

It is worthy to note that, after the extraction of parasitic capacitances and delays, only two nets had a net load greater than 1.0pF. This is by all means remarkable for a chip of 320 x 341 mils^2. The delays due to layout (as opposed to intrinsic delays and input capacitance of the standard cells) would be less than 1.0ns on any single net, as most dc1.2 standard cells have a drive strength of 1.0ns/pF.

Testing

The design was simulated in QuickSimII after backannotation of the parasitic loads and delays. The following is a sampling of the results:

```
2900.0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1
2920.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1
```

```
  2940.0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0
  2960.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0
  2980.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0
  3000.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0
  3020.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 1 1 0 0
  3040.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 0 1 0 0
  3060.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 0 0
  3080.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 0 1 0 1 0 0
  3100.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3120.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3140.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3160.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3180.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3200.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3220.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3240.0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3260.0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3280.0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
  3300.0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Time(ns)
```

The order in which the signals appear from left to right after a time stamp is given below (refer to the schematics to locate the signal names):

"RESET", "CLRM", "CLR", "CK", "CKM", "CKSUB", "DATA", "DATAIN", "LSB", "LSBIN", "MUXS2", "MUXS1", "MUXS0", "lsbu", "datau", "I\$840/I\$1044/CLRU", "I\$840/CKACT", "I\$840/I\$1044/CKU", "LSBOUT", "DCT0", "DCT1", "DCT2", "DCT3", "DCT4", "DCT5", "DCT6", "DCT7".

Since it is extremely difficult to decipher the data and output from a bit-serial simulation report, a program was written in Ample, “chk_dct.amp”, to read the report file and give the output in a more friendly manner. The result of the program is:

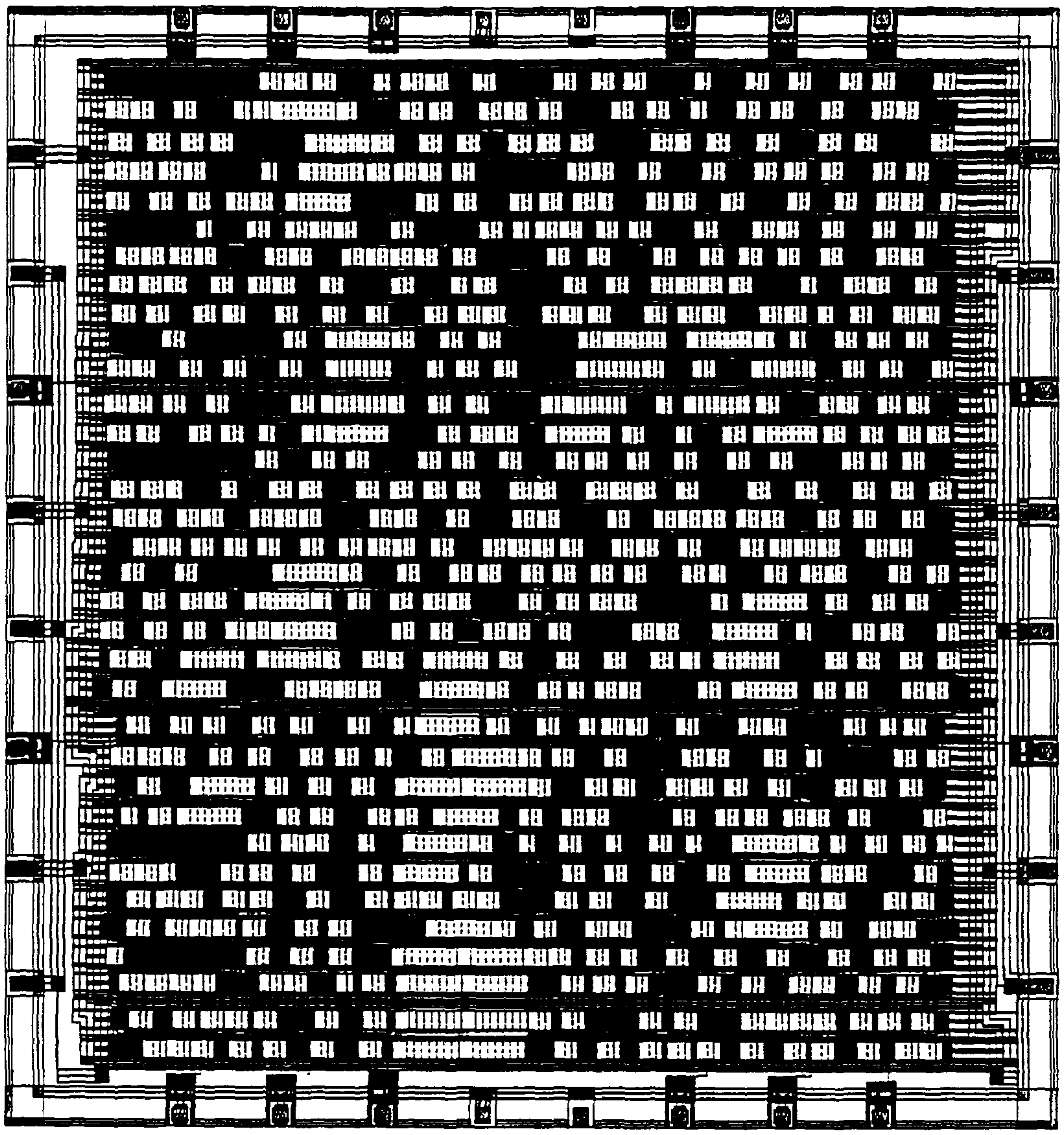
```
Simulation results of bit-serial design  dctbitserial
Input report name is complt_S2_20.rep
Output report name is complt_S2_20.chk

At Time =      760 ns  ==>
  DCT(0) = 00000 = 000000000000000000
  DCT(1) = 00000 = 000000000000000000
  DCT(2)  = 00000 = 000000000000000000
  DCT(3)  = 00000 = 000000000000000000
  DCT(4)  = 00000 = 000000000000000000
  DCT(5)  = 00000 = 000000000000000000
  DCT(6)  = 00000 = 000000000000000000
  DCT(7)  = 00000 = 000000000000000000
At Time =    1120 ns  ==>
  DCT(0)  = 00000 = 000000000000000000
  DCT(1)  = 00000 = 000000000000000000
  DCT(2)  = 00000 = 000000000000000000
  DCT(3)  = 00000 = 000000000000000000
  DCT(4)  = 00000 = 000000000000000000
  DCT(5)  = 00000 = 000000000000000000
  DCT(6)  = 00000 = 000000000000000000
  DCT(7)  = 00000 = 000000000000000000
At Time =    1480 ns  ==>
  DCT(0)  = 00000 = 000000000000000000
  DCT(1)  = 00000 = 000000000000000000
  DCT(2)  = 00000 = 000000000000000000
  DCT(3)  = 00000 = 000000000000000000
  DCT(4)  = 00000 = 000000000000000000
  DCT(5)  = 00000 = 000000000000000000
  DCT(6)  = 00000 = 000000000000000000
  DCT(7)  = 00000 = 000000000000000000
At Time =    1840 ns  ==>
  DCT(0)  = 00004 = 000000000000000100
  DCT(1)  = 3FFFC = 111111111111111100
  DCT(2)  = 00000 = 000000000000000000
  DCT(3)  = 00000 = 000000000000000000
  DCT(4)  = 00000 = 000000000000000000
  DCT(5)  = 00000 = 000000000000000000
  DCT(6)  = 3FFFC = 111111111111111100
  DCT(7)  = 00000 = 000000000000000000
At Time =    2200 ns  ==>
  DCT(0)  = 00008 = 000000000000001000
  DCT(1)  = 3FFF8 = 111111111111111000
  DCT(2)  = 00004 = 000000000000000100
  DCT(3)  = 00000 = 000000000000000000
  DCT(4)  = 00000 = 000000000000000000
  DCT(5)  = 00000 = 000000000000000000
  DCT(6)  = 00000 = 000000000000000000
  DCT(7)  = 3FFFC = 111111111111111100
At Time =    2560 ns  ==>
  DCT(0)  = 00010 = 000000000000010000
  DCT(1)  = 3FFEC = 111111111111101100
  DCT(2)  = 0000C = 000000000000001100
  DCT(3) = 3FFFC = 111111111111111100
  DCT(4) = 00004 = 000000000000000100
  DCT(5) = 00000 = 000000000000000000
  DCT(6) = 00000 = 000000000000000000
  DCT(7) = 00000 = 000000000000000000
At Time =    2920 ns  ==>
  DCT(0) = 00028 = 000000000000101000
  DCT(1) = 3FFD4 = 111111111111010100
  DCT(2) = 00018 = 000000000000011000
  DCT(3) = 3FFF4 = 111111111111110100
  DCT(4) = 00008 = 000000000000001000
  DCT(5) = 3FFFC = 111111111111111100
  DCT(6) = 00004 = 000000000000000100
  DCT(7) = 3FFFC = 111111111111111100
At Time =    3280 ns  ==>
  DCT(0) = 00050 = 000000000001010000
  DCT(1) = 3FFA8 = 111111111110101000
  DCT(2) = 00034 = 000000000000110100
  DCT(3) = 3FFE4 = 111111111111100100
  DCT(4) = 00010 = 000000000000010000
  DCT(5) = 3FFF8 = 111111111111111000
  DCT(6) = 00004 = 000000000000000100
  DCT(7) = 00000 = 000000000000000000
At Time =    3640 ns  ==>
  DCT(0) = 000A8 = 000000000010101000
  DCT(1) = 3FF4C = 111111111101001100
  DCT(2) = 00068 = 000000000001101000
  DCT(3) = 3FFC4 = 111111111111000100
  DCT(4) = 00020 = 000000000000100000
  DCT(5) = 3FFEC = 111111111111101100
  DCT(6) = 00008 = 000000000000001000
  DCT(7) = 3FFFC = 111111111111111100
```

These results were compared with the outputs obtained from the "dct_out.amp" given in chapter 2 under the "Testing" subsection. Keep in mind that what you see at, say 3640 ns was what you expected as 3280 (the previous strobe time), allowing for the propagation delay through the chip.

After the chip was functionally verified, the timing was verified. The speed was incrementally increased to figure out the maximum speed of the chip, without any glitches and setup, hold or pulse width violations which would affect the final output. It was determined that the minimum possible clock period is 7.2 ns under best case or 18.0 ns under worst case. This yields a maximum clock frequency of 139.00 MHz and 55.56 MHz under best and worst case conditions, respectively.

Fig. 5.3: DCTBITSERIAL Layout



Chapter 6 - Bit-Parallel DCT

Bit-Parallel Designs and Modules

Two ASIC chips, DCTBITPARLL and DCTROMPARLL, were designed. These designs are bit-parallel implementations of the DCT parallel IIR filter structure [1] shown in Fig. 1.4. The two implementations are -- DCTBITPARLL, with actual multipliers and DCTROMPARLL with ROM multipliers. The DCTBITPARLL is an 8-point, 8 bit input data, 16 bit coefficients, 16 bit internal bus precision and 12 bit output DCT design. The ROM multiplier implementation, DCTROMPARLL, is an 8-point, 8 bit input data, 16 bit internal bus precision and 12 bit output DCT design; the coefficients (calculated to 16 decimal places) and multiplication results are double precision numbers, and the results are programmed into the ROM to 16 bit precision; the multiplication is performed using distributed arithmetic. The designs are optimally synchronized; synchronization is limited by the IIR filter structure.

The designs use regular (i.e., bit-parallel) two's complement arithmetic. The coefficients are still fractions, so we have to deal with fractional two's complement in these two designs as well.

Calculation of Coefficients

The coefficients to the multipliers, which are also the coefficients of the DCT, can be calculated from Fig. 1.2 (and Fig. 1.3) and Table 1.3. In the DCTBITPARLL design, the coefficients are hard-wired in the top level schematic to the calculated coefficient values. These coefficients are passed on to the IIR filter modules and are named COEFk(15:0) and

COEMk(15:0), for k values of 0 to 7, corresponding to -D1 and N1 of Fig. 1.2 and Table 1.3; these coefficients finally go into the BMP1M4 16x16 multipliers. In the ROM implementation, the coefficients are used, external to the design, to calculate the results for the ROM outputs. We shall see how these coefficients are calculated:

$$COEF = -D1 = -2\cos\left(\pi\frac{k}{N}\right) = -2\cos\left(\pi\frac{k}{8}\right) \qquad \text{for k = 0 to 7}$$

$$COEM = N1 = C(k)\sqrt{\frac{2}{N}}\cos\left(\pi\frac{k}{2N}\right) = \frac{1}{2}C(k)\cos\left(\pi\frac{k}{16}\right) \qquad \text{for k = 1 to 7}$$

$$COEM = \frac{1}{2\sqrt{2}} \qquad \text{for k = 0}$$

These coefficients were calculated to 17 bit precision in an Ample program. The following table of values was produced by the program:

Table 6.1 - Coefficients for Bit-Parallel DCT Chip

k	COEF	COEM
0	-2.000000 = 10.000000000000000	0.353553 = 00.010110101000001
1	-1.847759 = 10.001001101111100	0.490393 = 00.011111011000101
2	-1.414214 = 10.100101011111011	0.461940 = 00.011101100100000
3	-1.530734 = 11.001111000001000	0.415735 = 00.011010100110110
4	0.000000 = 00.000000000000000	0.088388 = 00.010110101000001
5	1.530734 = 00.110000111110111	0.277785 = 00.010001110001110
6	1.414214 = 01.011010100000100	0.191342 = 00.001100001111101
7	1.847759 = 01.110110010000011	0.097545 = 00.000110001111100

The coefficients were then rounded to 16 bit precision when used in the DCTBITPARLL; for the DCTROMPARLL, the coefficients were used to double precision to calculate the results and program the results into the ROMs. For the DCTBITPARLL implementation, the calculated binary values of the coefficients are hard-wired in the top level schematic as mentioned earlier. The decimal point is implied. Since the coefficient magnitude could be as big as 2, the output of the 16x16 multiplier (with a 32 bit output) has to be extracted carefully such that we truncate the fractional part only. Note that these coefficients are equivalent to the coefficients of the bit-serial DCT multiplied by 4 (the decimal place of the binary value is shifted to the right by 2 digits); these, however, are the actual coefficients of the DCT IIR filter module, as the DCTBITSERIAL used the actual coefficients divided by 4 due to the internal design.

ROM Multiplication

The results obtained by multiplying the data and the coefficients of the DCT can be prestored in a ROM and accessed for each data value. The result is obtained by feeding in the multiplicand data to the address of the ROMs. The advantages of this method over direct multiplication are:

a) The largest access time of the two ROMs, RO18M3B and RO18M4B dc1.2 library standard cells, is 38.1 ns as opposed to the 64.5 propagation delay of the BMP1M4 multiplier. Therefore, it is conceivable that the speed of the chip would increase by approximately 50% when the ROM is used.

b) The cell area of the ROMs is much smaller than the BMP1M4, which has an area of 1915 x 2200 micron^2 = 4213.00 x 10^3 micron^2; the area of the RO18M3B is 2196 x 457 micron^2, and the RO18M4B has an area of 2650 x 457 micron^2, giving a total area of 2216.45 x 10^3 micron^2. Therefore the total ROM pair area is approximately half of the BMP1M4 area. Therefore, the area of the chip could be significantly less with the ROMs.

c) Built-in error reduction. In case of overflow, we can control the magnitude of the error. For example, if the result was bigger than the maximum allowable magnitude, we could program the result to be equal to the maximum magnitude. The multiplier BMP1M4 however would naively calculate an erroneous result and not even notify you of the fact. For example, if the result was smaller than the smallest allowable negative number, the multiplier would give a positive result; this can be avoided in the ROM by programming in the value of the smallest allowable negative number.

d) Greater reliability in the result. The BMP1M4 may have a greater tendency to give results off by a maybe even a single bit, depending upon switching voltage levels and slew rates and slopes. This is unlikely with the ROMs.

Distributed Arithmetic

The results obtained by multiplying the data and the coefficients of the DCT can be prestored in a ROM and accessed for each data value. However if we had 16 bit data, we would need $2^{16} = 65536$ words. We could reduce the size of the ROMs by implementing distributed arithmetic as follows:

$$Res = D(15 \leftarrow 0) \times COEF = D(15 \leftarrow 8) \times 2^8 \times COEF + D(7 \leftarrow 0) \times COEF$$

We could use two 256 word ROMs to do the same thing that one 65536 word ROM would do. We send the lower byte of the data into one ROM's address and the higher byte into the other ROM's address. For the lower byte ROM, we would calculate the result of the lower byte data times the coefficient and store the value into the ROM. Then for the higher byte ROM, we would calculate the result of the higher byte data times the coefficient times 2^8 and store the value into the ROM. When we calculated the results, it was evident that only 10 bits for the lower byte ROM's output and 16 bits for the higher byte ROM's output were required. The lower byte ROM's input is always positive, but we cannot say the same for the output. Therefore when adding with 16 bits of the higher byte ROM's output, we must sign extend the lower byte ROM's 12 bit output to 16 bits. The higher byte ROM's input could be either positive or negative and the same applies for the output, which will be in two's complement like the input. The implementation of this is the ROM_MULT16.

ROM_MULT16 Design Module

This module performs the multiplication by table lookup, i.e., the multiplied results are already stored in the ROM. This is a direct implementation of the multiplication using distributed arithmetic described in the previous sub-section. Although we require only 10 bits of precision for the lower byte ROM, since the DC1.2 ROM cells come in bus widths which are in increments of 4, we will choose 12 bits of precision anyway. The schematic for the ROM_MULT16 is given in Fig. 6.6.

ADD8 Design Module

This is a regular 8 bit adder. The symbol for the ADD8 is shown in Fig. 6.1; the schematic is given in Fig. 6.2. Signals A[7:0] and B[7:0], and carryin CIN are added together to produce the sum S[7:0] and carryout COUT. The adder has two T283M (4 bit fast carry adder) cells connected to each other. The lower 4 bits of A and B, which are A[3:0] and B[3:0], are connected to the top T283M; the higher 4 bits A[7:4] and B[7:4], are connected to the bottom T283M.

ADD16 Design Module

This is a regular 16 bit adder. Signals A[15:0] and B[15:0], and carryin CIN are added together to produce the sum S[15:0] and carryout COUT. The adder has four T283M (4 bit fast carry adder) cells connected to each other. The ADD16, in essence, is two ADD8 modules put together in one.

SUB8 Design Module

This is a regular 8 bit subtracter. The symbol for the SUB8 is shown in Fig. 6.1; the schematic is given in Fig. 6.3. The difference between the signals A[7:0] and B[7:0] is calculated to produce the difference DIF[7:0] and carryout COUT. If the module is used stand-alone as an 8 bit subtracter, the carryin CIN has to be set to a 1; if the SUB8 is used to make a subtracter with a greater width, say 16, the CIN would be set to a 1 for the lower byte and the COUT of the lower byte would be sent to the CIN of the higher byte. The subtracter has two T283M (4 bit fast carry adder) cells connected to each other to form the 8 bit adder. The lower 4 bits of A[7:0] and B[7:0], which are A[3:0] and B[3:0], are connected to the top T283M; the higher 4 bits A[7:4] and B[7:4], are connected to the bottom T283M.

The subtraction is performed in two's complement, i.e., the B[7:0] signal of the SUB8 is inverted and sent to the B inputs of the T283M cells. The 1 set on the lower 4 bits' CIN is to complete the conversion to two's complement.

SUB16 Design Module

This is a regular 16 bit subtracter. The difference between the signals A[15:0] and B[15:0] is calculated to produce the difference DIF[15:0] and carryout COUT. If the module is used stand-alone as an 16 bit subtracter, the carryin CIN has to be set to a 1. The SUB16, in essence, is two SUB8 modules put together in one, and therefore all logical characteristics of the SUB8 apply here too.

DELAY8 Design Module

This is simply a string of eight 8 bit D-registers (T273M8). DELAY8 delays the 8 bit data signal for 8 clock cycles, thus performing the $z^{-N} = z^{-8}$ operation of the IIR filter structure for the DCT. The symbol for the DELAY8 is shown in Fig. 6.1; the schematic is given in Fig. 6.4. The CK and CLR signals are sent through two 2X inverting buffers each to drive the inputs of the eight registers. CKDLY and CLRDLY are internal signal/net names corresponding to CK and CLR. Q[7:0] signals are the D[7:0] signals delayed by eight clock cycles.

IIR_16INTx16C Design Module

The IIR_16INTx16C is the bit-parallel multiplier implementation (using actual multiplier cells) of the unified IIR module, shown in Fig. 1.3, for the DCT [1]. The symbol for the IIR_16INTx16C is shown in Fig. 6.1; the schematic is given in Fig. 6.5.

The multiplier BMP1M4 is 16x16, producing a 32 bit result. However, because of the feedback loop, we cannot take all 32 bits. The data are 16 bit integers; the coefficients are 16 bit fractional numbers with maximum magnitude of 2. Therefore, the data D has to be within the range 1100 0000 0000 0000 < D < 0100 0000 0000 0000. The coefficient has a two digit integer part and 14 bits of fractional part. Therefore, out of the outputs P[31:0] of the multiplier BMP1M4, considering the range and the data and coefficient formats, what we should be interested in is P[31] (for sign) and P[28:14], truncating the fractional

14 bits of the result; the two msb bits P[30:29] are also truncated hoping that we stay within the specified range. The sign could also be derived from P[29] or P[30], instead of P[31], if the data D stays within the range.

An internal bus precision of 16 is maintained; the 8 bit input to the IIR_16INTx16C is sign extended to 16 bits; the 16 bit output of the BMP1M4, at the output stage, is truncated (the 4 most significant bits) to 12 bits; these 12 bits are registered and sent to the output of the module and straight to the output pads of the chip.

Synchronization is a sticky issue for the IIR filter module. Since the maximum and required delay in the Z^{-1} and Z^{-2} delay loops is 1 and 2 respectively, only two registers can be used in the feedback loops. Registers are used as necessary outside the IIR filter feedback loops.

IIR_ROM16x16 Design Module

The IIR_ROM16x16 is the ROM multiplier implementation of the unified IIR module, shown in Fig. 1.3, for the DCT [1]. The symbol for the IIR_ROM16x16 is shown in Fig. 6.1; the schematic is given in Fig. 6.7.

The multiplier ROM_MULT16 is programmed to perform multiplication on a 16 bit input and produce a 16 bit result. The 16 bit result is because of the feedback loop; we cannot take all 32 bits results. The data are 16 bit integers; the coefficients are fractional numbers with maximum magnitude of 2. Therefore, the data D has to be within the range 1100 0000 0000 0000 < D < 0100 0000 0000 0000.

An internal bus precision of 16 is maintained; the 8 bit input to the IIR_ROM16x16 is sign extended to 16 bits; the 16 bit output of the ROM_MULT16 at the output stage is truncated (the 4 most significant bits) to 12 bits; these 12 bits are registered and sent to the output of the module and straight to the output pads of the chip.

Synchronization is a sticky issue for the IIR filter module. Since the maximum and required delay in the Z^{-1} and Z^{-2} delay loops is 1 and 2 respectively, only two registers can be used in the feedback loops. Registers are used as necessary outside the IIR filter feedback loops.

DCTBITPARLL Design Module

This is the top level module of the DCTBITPARLL design. The schematic for the DCTBITPARLL is given in Fig. 6.8. The schematics for the four modules used in the design -- SUB8, DELAY8 and IIR_16INTx16C are given in Figs. 6.3, 6.4 and 6.5 respectively, and their symbols are given in Fig. 6.1.

The DELAY8 module (8 clock cycle delays) performs the Z^{-N} operation, i.e., delaying the data N data words, where N = 8 here (8-point DCT).

On the right hand side of the schematic is a column of eight IIR_16INTx16C filter modules, each corresponding to a k, where k is in between 0 and 7.

There is only one clock input signal CK. To avoid clock, reset and data skews, the design has very effective inverting driver structures right down through its hierarchy.

The processor is designed to be as synchronous as possible. All asynchronous components and modules are followed by a register, except in the IIR_16INTx16C feedback loops.

The critical path of the DCTBITPARLL is clearly the Z^{-1} feedback loop of the IIR filter.

The critical path delay would then be the addition of the propagation delays through the BMP1M4, ADD16, SUB16 and REG16. The delay through the ADD16 and SUB16 is the delay through the four T283M carryouts. The delay through REG16 is the same as the delay through a T273M8. We shall calculate the critical path delay as follows (worst case --- this implies cases of overflow and other error conditions, 125 C and worst case process variables):

$$\begin{aligned} tP_{critical} &= \text{delay(BMP1M4)} + \text{delay(ADD16)} + \text{delay(SUB16)} + \text{delay(T273M8)} \\ &= (64.5 + 0.2 * 0.25) + 2 * (3.7 + 2 * 6.9 + 7.0 + 1.1 * 0.25) + (4.5 + 0.7 * 0.1) \\ &= 119 \text{ ns} \end{aligned}$$

Note that the T273M8 register does not require a setup time. This estimate is without any netdelay or back annotation calculations. The best case delay would be approximately (assuming no overflow or carryout, -55 C and best case process variables):

$$\begin{aligned} tP_{critical} &= 0.4 * [(64.5 + 0.2 * 0.25) + 2 * (3.7 + 1.1 * 0.25) + (4.5 + 0.7 * 0.1)] \\ &= 30 \text{ ns} \end{aligned}$$

If we estimate a 20% increase in delay due to backannotation, we have approximately 143 ns worst case and 36 ns best case critical path delay. This would accommodate a worst clock speed of 7 MHz and best case clock speed of 28 MHz.

Prior to layout, the maximum clock speed for the whole design was 32.5 MHz at best case and 13.0 MHz at worst case (125 C and worst process). The throughput is 1 transform word per cycle. The latency of the design is 5 cycles. All testing of the design were done with worst case parameters.

Before getting into block place and route, the design was run through the electronic rules check (ERC) program. There were no errors reported from the program. The following are a few statistics from the program:

```
***********QUICK CHECK STATISTIC REPORT****************
Total number of instances in design = 687
Total number of pins in design = 4774
Total number of nets in design = 2084
Equivalent gate count in design = 45264
Total cell area in design (in sq. microns) = 78307102
*******************************************************
```

The following are a few statistics from the Silvar Lisco block place and route pre-processor, scbuild. Bcells refer to big cells, such as ROMs, bit-parallel multipliers, RAMs, etc. Pcells refer to I/O and power pad cells.

```
        Circuit statistics
        ******************
        Nr. of elements  :     687
          Nr. of Cells   :     549
          Nr. of Pcells  :     122
          Nr. of Bcells  :      16
        Nr. of Nets      :    1691
        Nr. of Pins      :    4849
        Nr. of Pins/Net  :     2.9 (average)
        Nr. of Pins/Cell :     7.1 (average)
        Cell Area        :  2.7171e+08 2.0E-1 MICRON**2
        Bcell Area       :  1.6854e+08 2.0E-1 MICRON**2
        Pcell Area       :  2.5628e+08 2.0E-1 MICRON**2
```

The chip I/O is as follows:

Inputs

Data: D7, D6, D5, D4, D3, D2, D1, D0

Clock: CK

Reset: RESET

Outputs

Data: DCT0(11:0), DCT1(11:0), DCT2(11:0), DCT3(11:0), DCT4(11:0), DCT5(11:0), DCT6(11:0), DCT7(11:0)

DCTROMPARLL Design Module

This is the top level module of the DCTROMPARLL design. The schematic for the DCTROMPARLL is given in Fig. 6.9. The schematics for the four modules used in the design -- SUB8, DELAY8 and IIRMOD8 are given in Figs. 6.3, 6.4 and 6.5 respectively, and their symbols are given in Fig. 6.1.

The DELAY8 module (8 clock cycle delays) performs the Z^{-N} operation, i.e., delaying the data N data words, where N = 8 here (8-point DCT).

On the right hand side of the schematic is a column of eight IIR_ROM16x16 filter modules, each corresponding to a k, where k is in between 0 and 7.

There is only one clock input signal CK. To avoid clock, reset and data skews, the design has very effective inverting driver structures right down through its hierarchy.

The processor is designed to be as synchronous as possible. All asynchronous components and modules are followed by a register, except in the IIR_ROM16x16 feedback loops.

The critical path of the DCTROMPARLL is clearly the Z^{-1} feedback loop of the IIR filter. The critical path delay would then be the addition of the propagation delays through the RO18M4B, ADD16's, SUB16 and REG16. The delay through the ADD16 and SUB16 is the delay through the four T283M carryouts. The delay through REG16 is the same as the delay through a T273M8. We shall calculate the critical path delay as follows (worst case, which here implies cases of overflow and other error conditions, 125 C and worst case process variables):

$$\begin{aligned} tP_{critical} &= \text{delay(RO18M4B)} + 2 * \text{delay(ADD16)} + \text{delay(SUB16)} + \text{delay(T273M8)} \\ &= (38.1 + 1.7 *0.25) + 3 * (3.7 +2 * 6.9 + 7.0 + 1.1 *0.25) + (4.5 +0.7 *0.1) \end{aligned}$$

$= 118$ ns

Note that the T273M8 register does not require a setup time. This estimate is without any netdelay or back annotation calculations. The best case delay would be approximately (assuming no overflow or carryout, -55 C and best case process variables):

$$tP_{critical} = 0.4 * [(38.1 + 1.7 * 0.25) + 3 * (3.7 + 1.1 * 0.25) + (4.5 + 0.7 * 0.1)]$$
$$= 22 \text{ ns}$$

If we estimate a 20% increase in delay due to backannotation, we have approximately 142 ns worst case and 27 ns best case critical path delay. This would accommodate a worst clock speed of 7 MHz and best case clock speed of 37 MHz.

Prior to layout, the maximum clock speed for the whole design was 38.5 MHz at best case and 15.4 MHz at worst case (125 C and worst process). The throughput is 1 transform word per cycle. The latency of the design is 5 cycles. All testing of the design were done with worst case parameters.

Before getting into block place and route, the design was run through the electronic rules check (ERC) program. There were no errors reported from the program. The following are a few statistics from the program:

```
***********QUICK CHECK STATISTIC REPORT****************
Total number of instances in design = 767
Total number of pins in design = 5318
Total number of nets in design = 2340
Equivalent gate count in design = 52096
Total cell area in design (in sq. microns) = 49249342
*******************************************************
```

The following are a few statistics from the Silvar Lisco block place and route pre-processor, scbuild. Bcells refer to big cells, such as ROMs, bit-parallel multipliers, RAMs, etc. Pcells refer to I/O and power pad cells.

```
Circuit statistics
******************
Nr. of elements  :      767
  Nr. of Cells   :      613
  Nr. of Pcells  :      122
  Nr. of Bcells  :       32
Nr. of Nets      :     2155
Nr. of Pins      :     5121
Nr. of Pins/Net  :      2.4 (average)
Nr. of Pins/Cell :      6.7 (average)
Cell Area        :  3.4526e+08 2.0E-1 MICRON**2
Bcell Area       :  8.8539e+08 2.0E-1 MICRON**2
Pcell Area       :  2.5628e+08 2.0E-1 MICRON**2
```

The chip I/O is as follows:

Inputs

Data: D7, D6, D5, D4, D3, D2, D1, D0

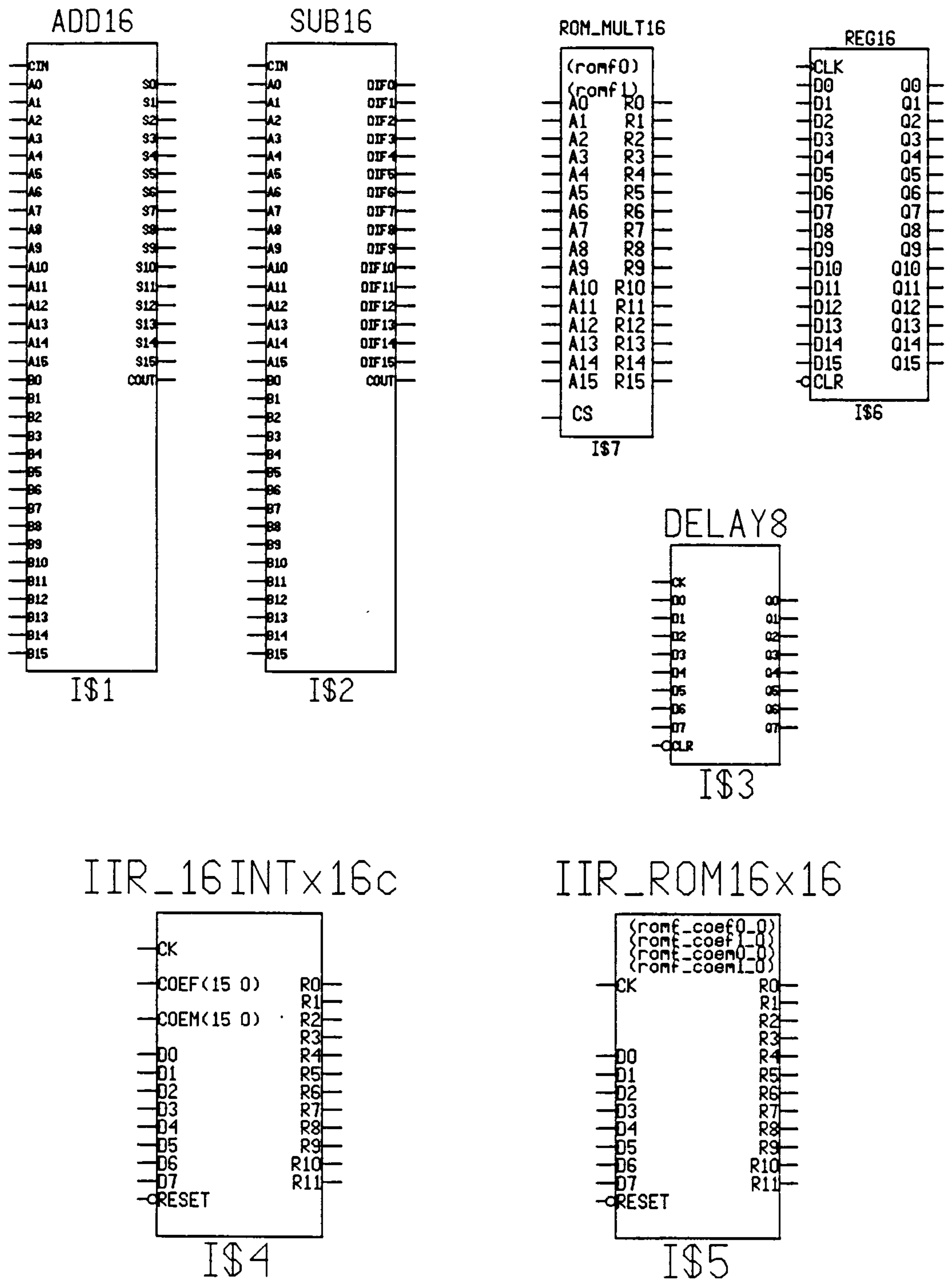
Clock: CK

Reset: RESET

Outputs

Data: DCT0(11:0), DCT1(11:0), DCT2(11:0), DCT3(11:0), DCT4(11:0), DCT5(11:0), DCT6(11:0), DCT7(11:0)

Fig. 6.1: Bit-Parallel Hierarchical Module Symbols



ADD16
CIN
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15
COUT
I$1
SUB16
CIN
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
DIF0 DIF1 DIF2 DIF3 DIF4 DIF5 DIF6 DIF7 DIF8 DIF9 DIF10 DIF11 DIF12 DIF13 DIF14 DIF15
COUT
I$2
ROM_MULT16
(romf0)
(romf1)
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
CS
I$7
REG16
CLK
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15
CLR
I$6
DELAY8
CK
D0 D1 D2 D3 D4 D5 D6 D7
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
CLR
I$3
IIR_16INTx16c
CK
COEF(15 0)
COEM(15 0)
D0 D1 D2 D3 D4 D5 D6 D7
RESET
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11
I$4
IIR_ROM16x16
(romf_coef0_0)
(romf_coef1_0)
(romf_coem0_0)
(romf_coem1_0)
CK
D0 D1 D2 D3 D4 D5 D6 D7
RESET
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11
I$5

Fig. 6.2: ADD8 Schematic

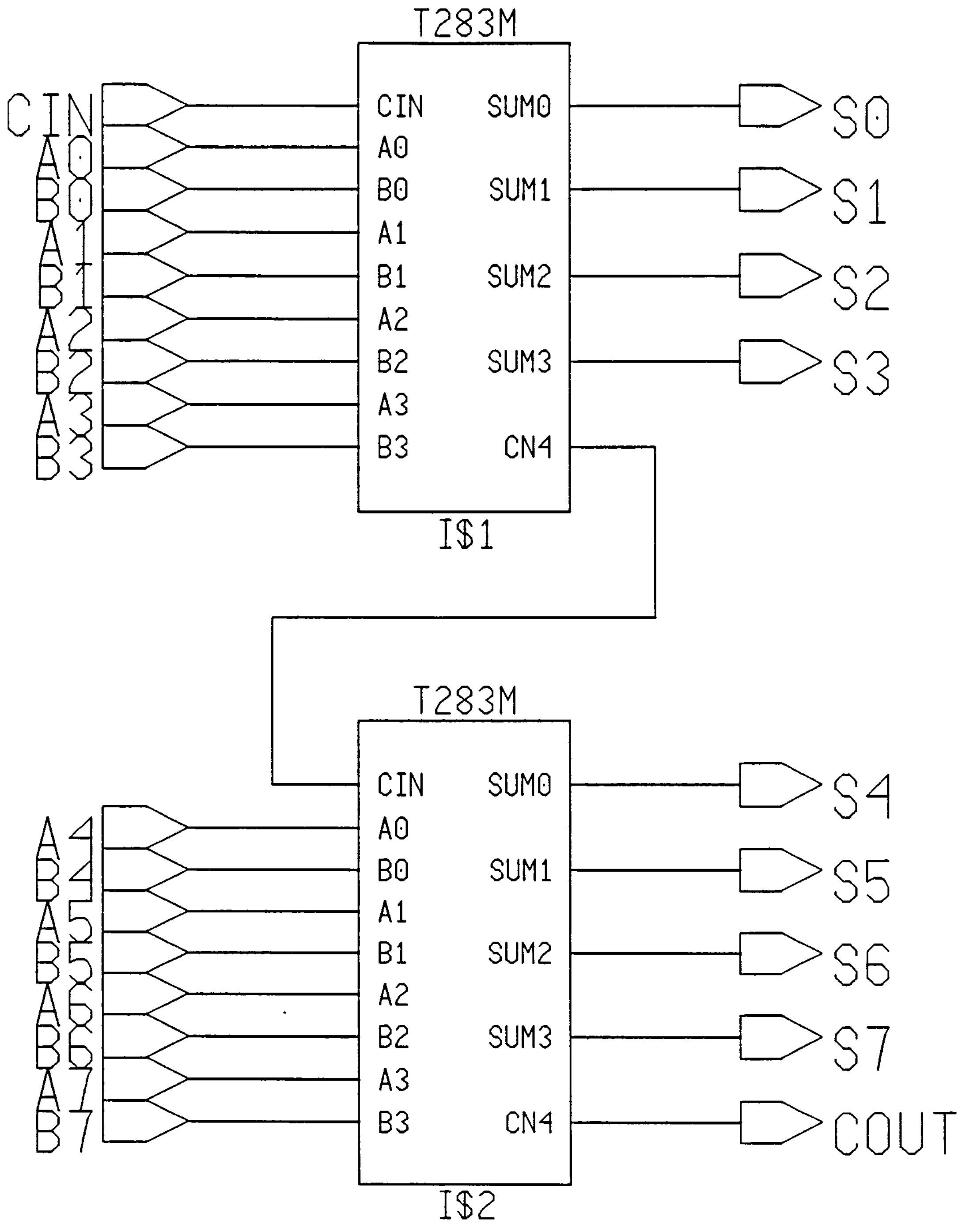


Fig. 6.3: SUB8 Schematic

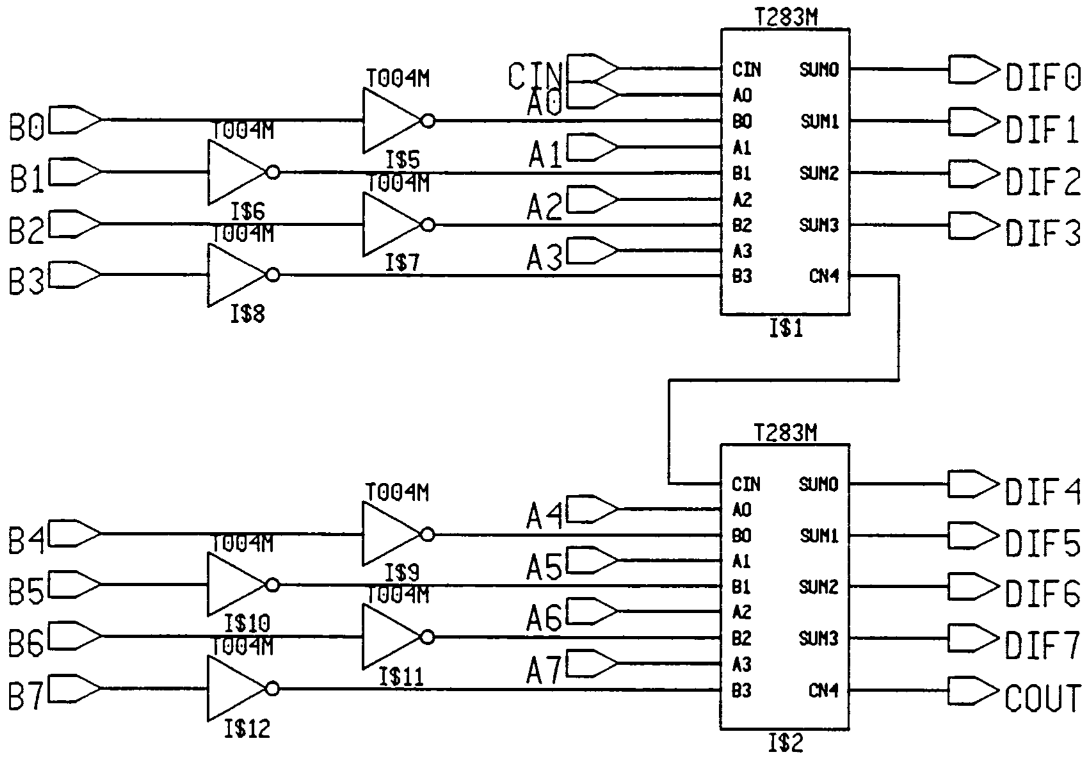


Fig. 6.4: DELAY8 Schematic

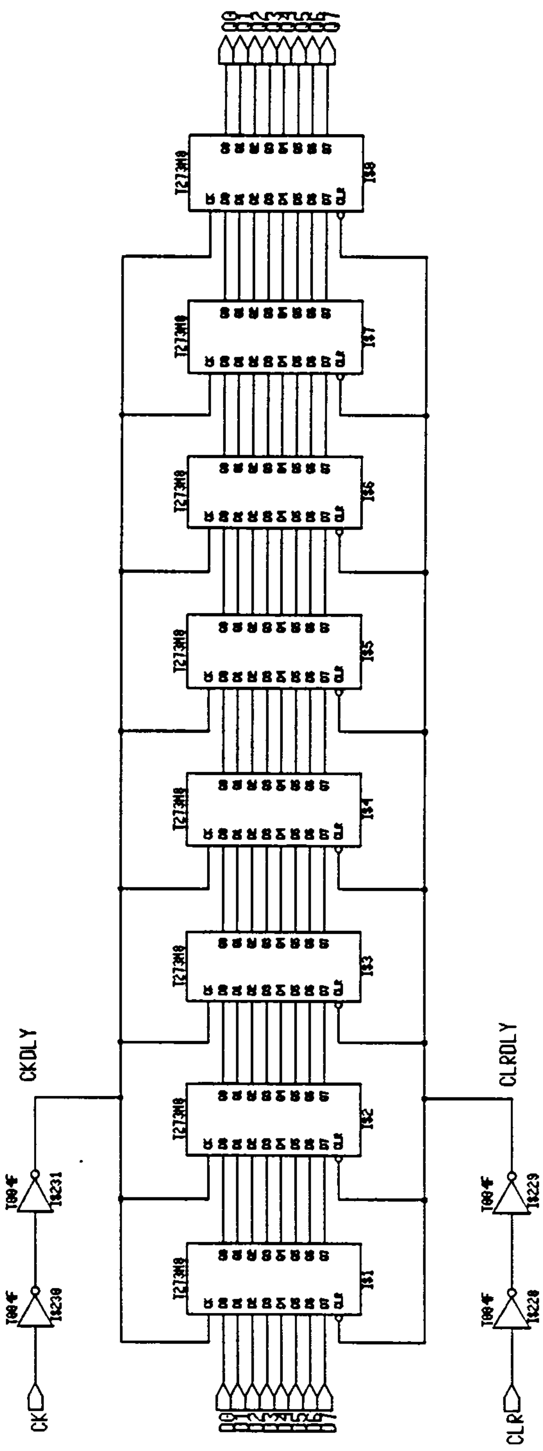


Fig. 6.5: IIR_16INTx16C Schematic

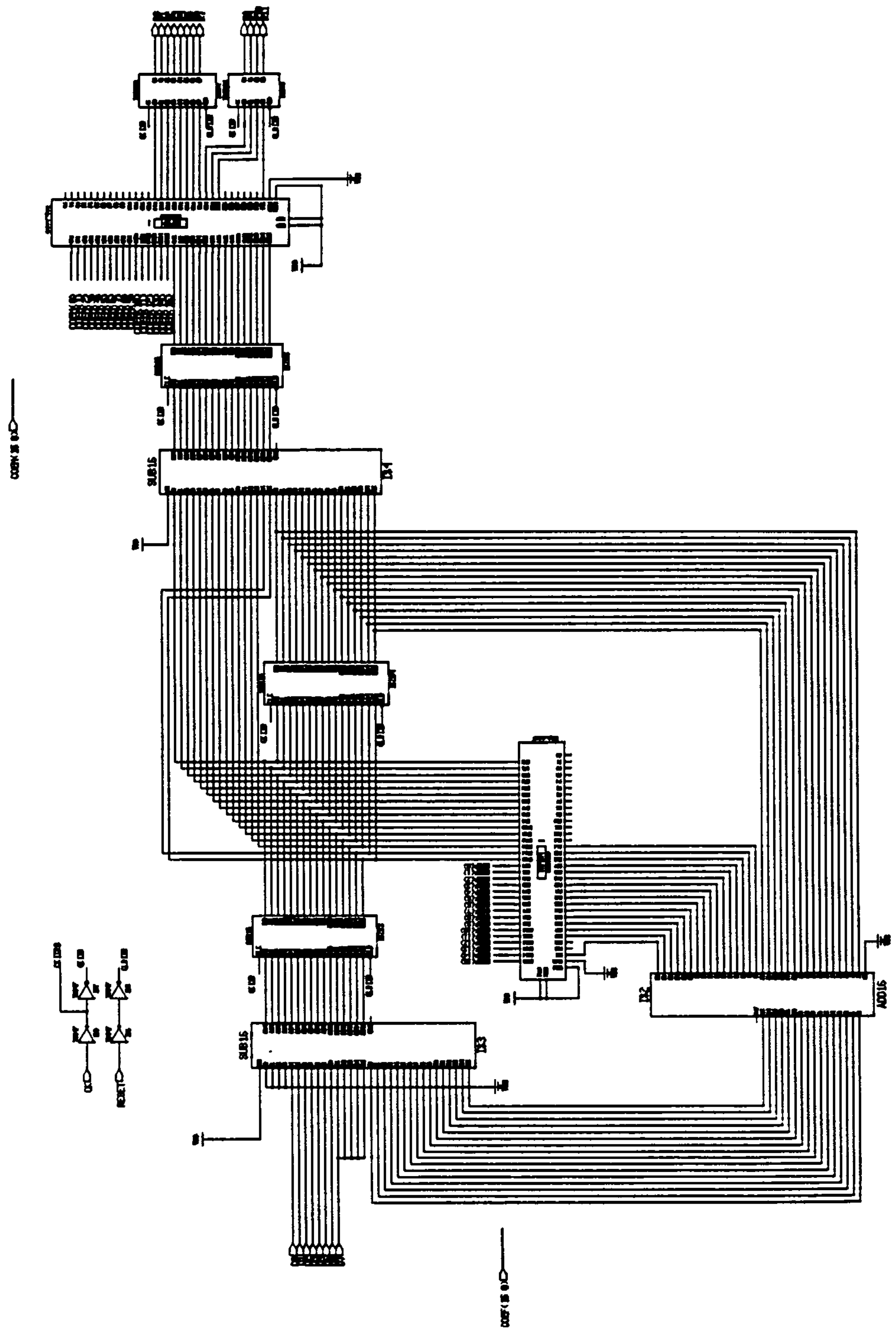


Fig. 6.6: ROM_MULT16 Schematic

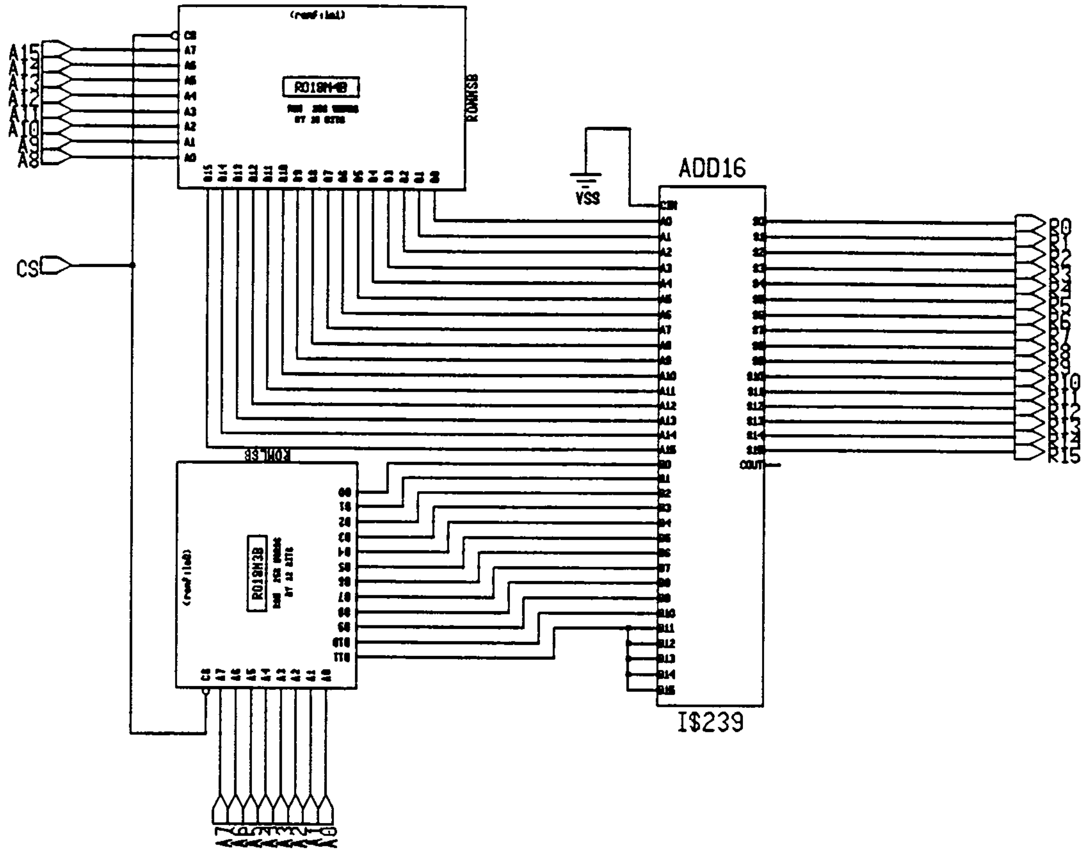


Fig. 6.7: IIR_ROM16x16 Schematic

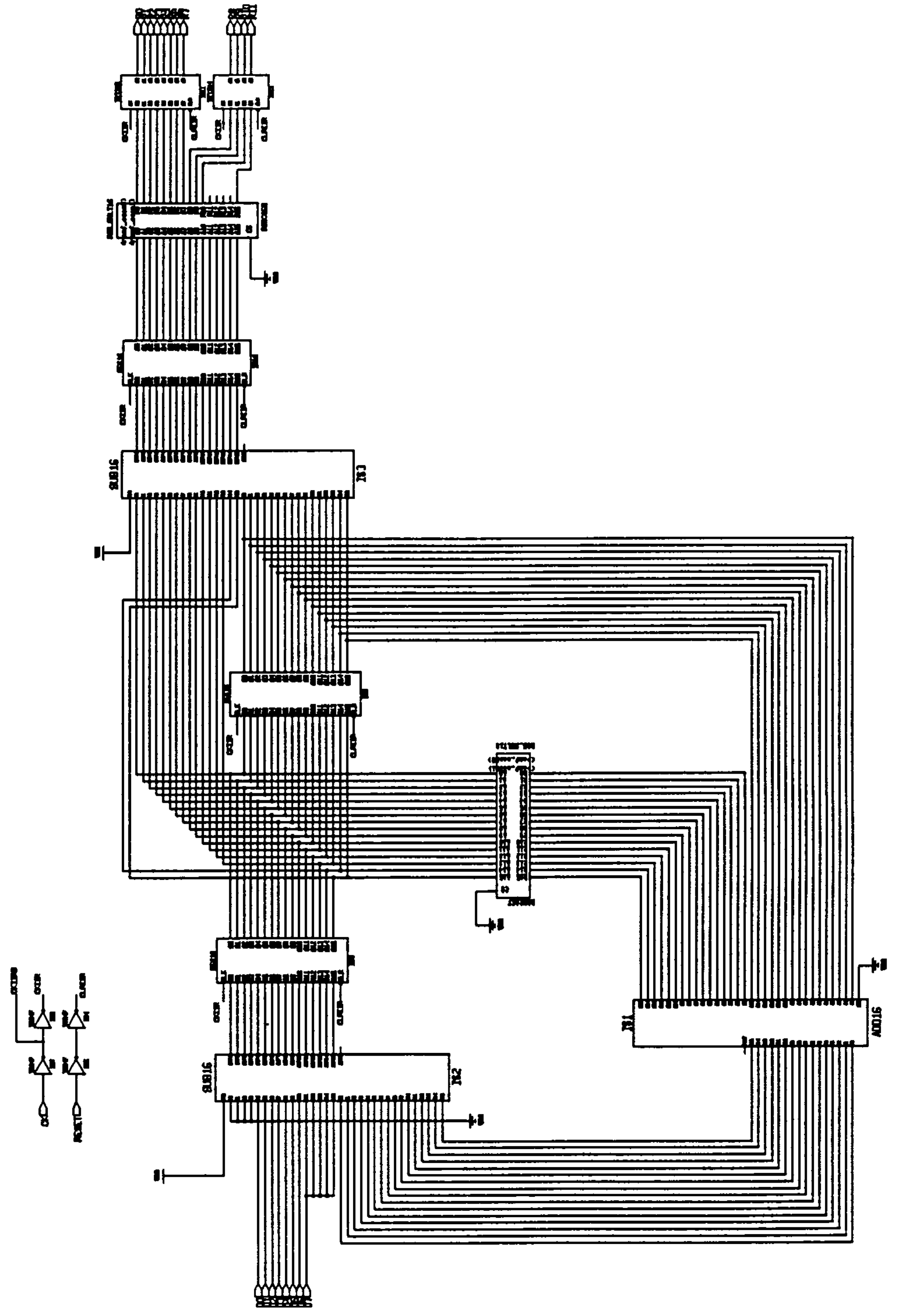


Fig. 6.8: DCTBITPARLL Schematic

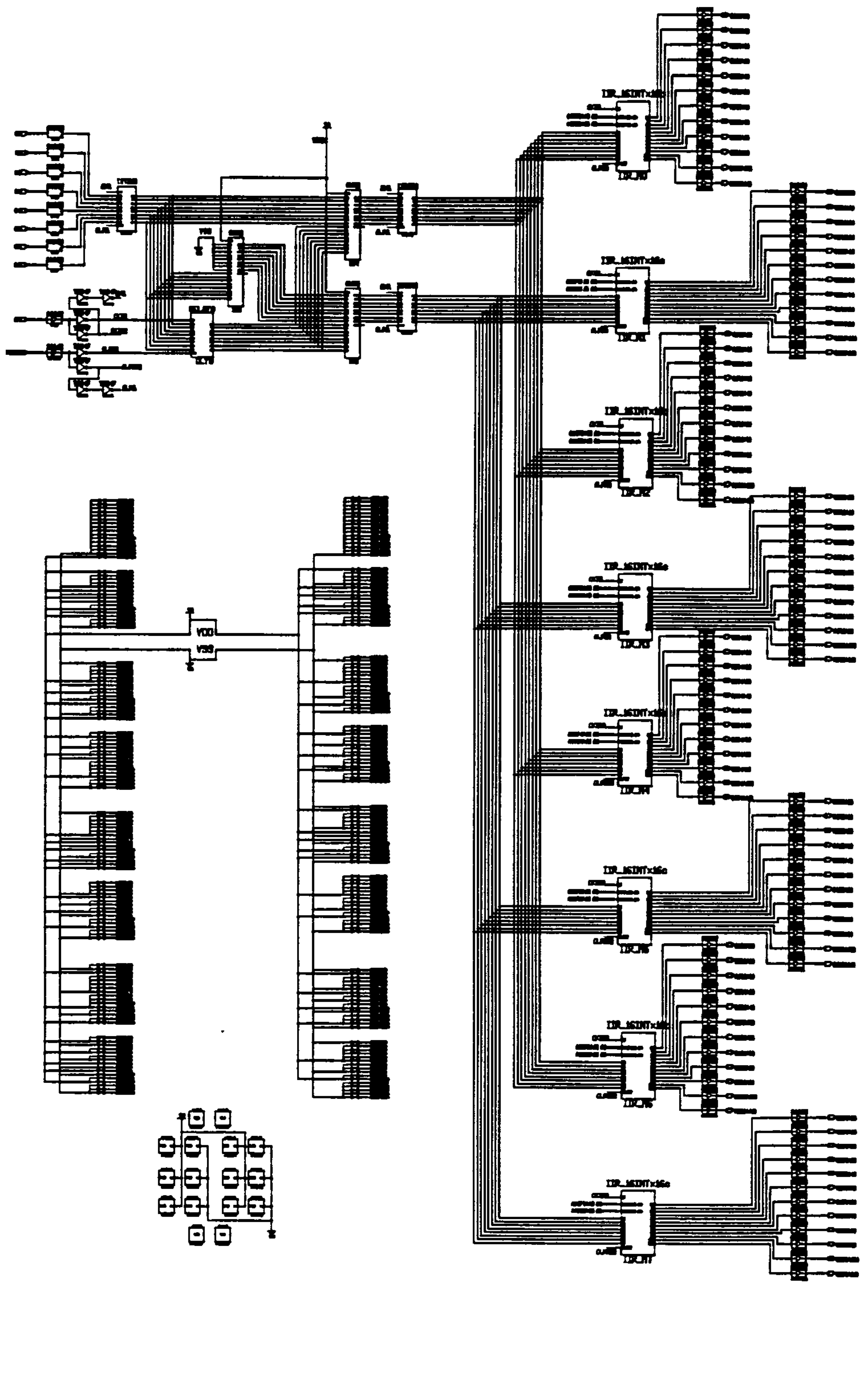
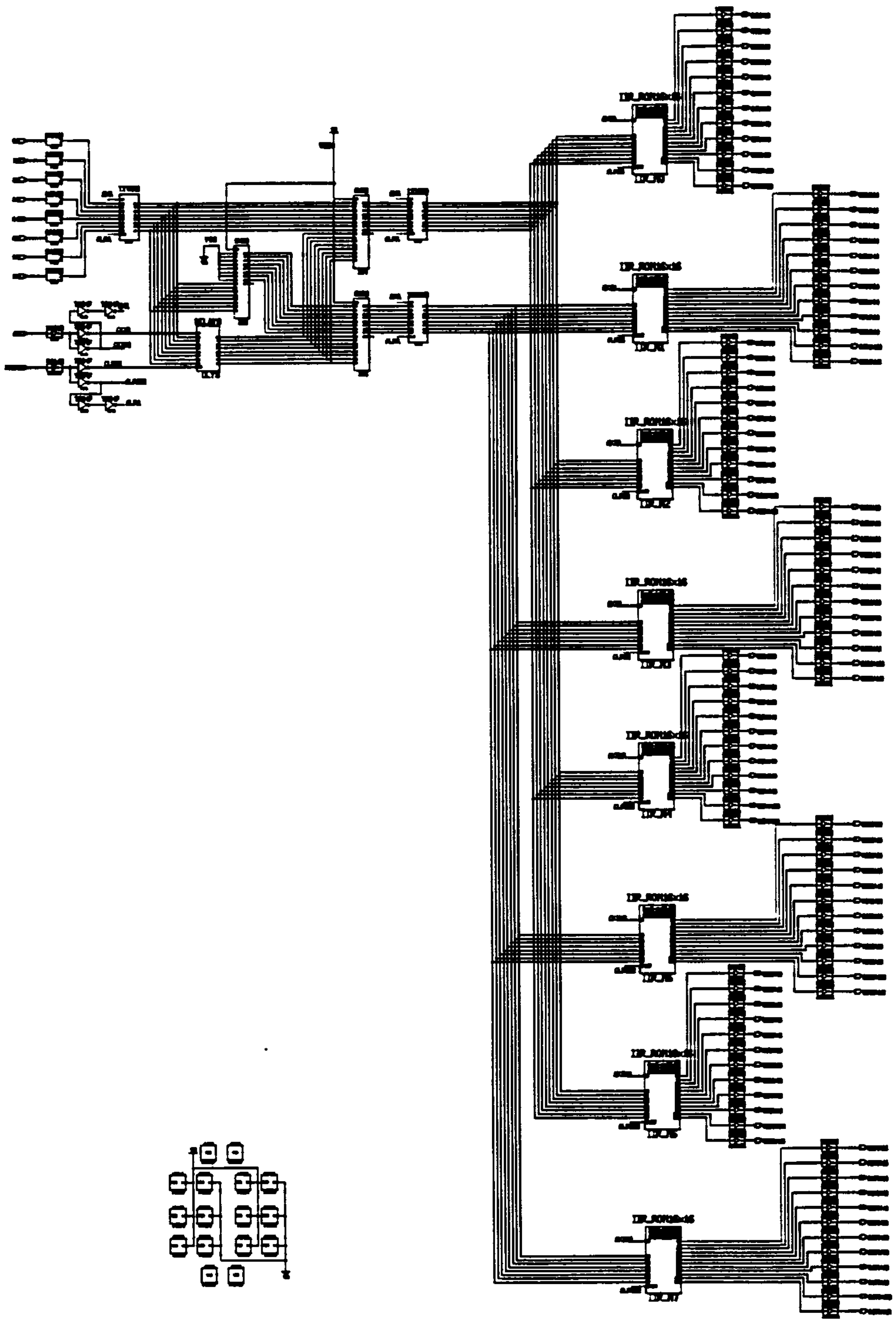


Fig. 6.9: DCTROMPARLL Schematic



DCTBITPARLL Layout

The netlist extraction, place and route, layout and extraction of timing were performed with Silvar Lisco v3.218. The following is a summary of the layout extracted from the tool:

```
DESIGN_NAME DCTBITPARLL
LOGIC_NAME DCTBITPARLL
DATE_TIME 94/04/13 23:31:48
UNIT_OF_SIZE 200 nanometer
UNIT_NAME 2.0E-1 MICRON
PLI_FILE dct_parll.sc/v4x4h8b8-2.pli
INP_WHRATIO 120
INP_RATIO 50
NR_CELLS 549
NR_BCELLS 16
NR_PCELLS 122
NR_FEEDS_EXT 142
NR_LINKS 0
NR_SPACERS 0
NR_NETS 1689
LENGTH_ON_METAL2 11570419 2.0E-1 MICRON
LENGTH_ON_METAL1 26130279 2.0E-1 MICRON
NR_OF_VIA 10980
NR_OF_MATVIA 828
WIDTH_OF_ACTIVE_AREA 49910 2.0E-1 MICRON
WIDTH_OF_ACTIVE_AREA 9.9820e+00  mm
WIDTH_OF_ACTIVE_AREA 3.9299e+02  mils
HEIGHT_OF_ACTIVE_AREA 62594 2.0E-1 MICRON
HEIGHT_OF_ACTIVE_AREA 1.2519e+01  mm
HEIGHT_OF_ACTIVE_AREA 4.9287e+02  mils
ACTIVE_AREA 3.1241e+09  2.0E-1 MICRON**2
ACTIVE_AREA 1.2496e+02  mm**2
ACTIVE_AREA 1.9369e+05  mils**2
WIDTH_OF_TOTAL_AREA 58652 2.0E-1 MICRON
WIDTH_OF_TOTAL_AREA 1.1730e+01  mm
WIDTH_OF_TOTAL_AREA 4.6183e+02  mils
HEIGHT_OF_TOTAL_AREA 67950 2.0E-1 MICRON
HEIGHT_OF_TOTAL_AREA 1.3590e+01  mm
HEIGHT_OF_TOTAL_AREA 5.3504e+02  mils
TOTAL_AREA 3.9854e+09  2.0E-1 MICRON**2
TOTAL_AREA 1.5942e+02  mm**2
TOTAL_AREA 2.4710e+05  mils**2
AREA_RATIO 59
```

The above summary gives us a lot of information on the layout. The layout consists of 4 rows of 4 16x16 multipliers each, one pair of rows at the top and one pair of rows at the bottom; there are 8 horizontal rows sandwiched in between the two pairs of multiplier rows. The number of cells is 549 MSI and SSI cells and 16 multiplier bus cells; the number of pads is 122 (106 I/O pads and 16 power supply pads). The area_ratio of 59 (or 0.59) tells us that the layout is densely packed with the ratio of interconnect area to cell area of 0.59, i.e., the cell area is approximately 1.7 times larger than the interconnect area. The die size is 462 x 535 mils^2, or 11730.4 x 13590.0 micron^2. This yields a total area of 2471.0 x 10^2 mils^2, or 159.416 x 10^6 micron^2. In the common man's units, this is a 11.73 mm x 13.59 mm, or 159 mm^2 die. A plot of the layout is given in Fig. 6.10.

During the layout of this chip, the prime considerations taken into account were: the total area; the interconnect area to cell area ratio; the die size -- as square a die as possible; optimal placement of the big cells -- the multipliers; optimal placement of I/O and power pads; optimal placement of other cells, with minimal connecting wire lengths; minimal parasitic delays and capacitances (lumped and distributed) extracted from the layout; maximized active area.

After the extraction of parasitic capacitances and delays, only a few nets had a net load greater than 1.0pF. Therefore, delays due to layout (as opposed to intrinsic delays and input capacitance of the standard cells) would be less than 1.0ns on any single net, as most dc1.2 standard cells have a drive strength of 1.0ns/pF.

DCTROMPARLL Layout

The netlist extraction, place and route, layout and extraction of timing were performed with Silvar Lisco v3.218. The following is a summary of the layout extracted from the tool:

```
DESIGN_NAME DCTROMPARLL
LOGIC_NAME DCTROMPARLL
DATE_TIME 94/04/13 19:10:15
UNIT_OF_SIZE 200 nanometer
UNIT_NAME 2.0E-1 MICRON
PLI_FILE dct_rom.sc/v16x2h10b8-2.pli
INP_WHRATIO 120
INP_RATIO 50
NR_CELLS 613
NR_BCELLS 32
NR_PCELLS 122
NR_FEEDS_EXT 94
NR_LINKS 0
NR_SPACERS 0
NR_NETS 2153
LENGTH_ON_METAL2 13304999 2.0E-1 MICRON
LENGTH_ON_METAL1 35494592 2.0E-1 MICRON
NR_OF_VIA 11259
NR_OF_MATVIA 535
WIDTH_OF_ACTIVE_AREA 45900 2.0E-1 MICRON
WIDTH_OF_ACTIVE_AREA 9.1800e+00  mm
WIDTH_OF_ACTIVE_AREA 3.6142e+02  mils
HEIGHT_OF_ACTIVE_AREA 54552 2.0E-1 MICRON
HEIGHT_OF_ACTIVE_AREA 1.0910e+01  mm
HEIGHT_OF_ACTIVE_AREA 4.2954e+02  mils
ACTIVE_AREA 2.5039e+09  2.0E-1 MICRON**2
ACTIVE_AREA 1.0016e+02  mm**2
ACTIVE_AREA 1.5524e+05  mils**2
WIDTH_OF_TOTAL_AREA 58308 2.0E-1 MICRON
WIDTH_OF_TOTAL_AREA 1.1662e+01  mm
WIDTH_OF_TOTAL_AREA 4.5912e+02  mils
HEIGHT_OF_TOTAL_AREA 60060 2.0E-1 MICRON
HEIGHT_OF_TOTAL_AREA 1.2012e+01  mm
HEIGHT_OF_TOTAL_AREA 4.7291e+02  mils
TOTAL_AREA 3.5020e+09  2.0E-1 MICRON**2
TOTAL_AREA 1.4008e+02  mm**2
TOTAL_AREA 2.1712e+05  mils**2
AREA_RATIO 103
```

The above summary gives us a lot of information on the layout. The layout consists of 2 rows of 16 ROM cells each, one row at the top and one row at the bottom; there are 10 horizontal rows sandwiched in between the two rows of ROMs. The number of cells is

613 MSI and SSI cells and 32 ROM cells (16 are 256x16 and the other 16 are 256x12); the number of pads is 122 (106 I/O pads and 16 power supply pads). The area_ratio of 103 (or 1.03) tells us that the layout is fairly densely packed with a ratio of interconnect area to cell area of 1.03, i.e., the cell area is approximately equal to the interconnect area. The die size is 459 x 473 mils^2, or 11661.6 x 12012.0 micron^2. This yields a total area of 2171.2 x 10^2 mils^2, or 140.080 x 10^6 micron^2. In the common man's units, this is a 11.66 mm x 12.01 mm, or 140 mm^2 die.

During the layout of this chip, the prime considerations taken into account were: the total area; the interconnect area to cell area ratio; the die size -- as square a die as possible; optimal placement of the big cells -- the ROMs; optimal placement of I/O and power pads; optimal placement of other cells, with minimal connecting wire lengths; minimal parasitic delays and capacitances (lumped and distributed) extracted from the layout; maximized active area. A plot of the layout is given in Fig. 6.11.

After the extraction of parasitic capacitances and delays, only a few nets had a net load greater than 1.0pF. Therefore, delays due to layout (as opposed to intrinsic delays and input capacitance of the standard cells) would be less than 1.0ns on any single net, as most dc1.2 standard cells have a drive strength of 1.0ns/pF.

Comparison Between ROM and Multiplier

The DCTROMPARLL layout is smaller than the DCTBITPARLL as expected. One could verify this by looking at the summary for each layout; the DCTBITPARLL has an area of 159 mm^2, while the DCTROMPARLL has an area of 140 mm^2. This is a significant 12.5% reduction in area. However, the difference is not as much as expected, as there is a lot of area taken up by the interconnect net area. This accounts for the superior interconnect area to cell area ratio of 0.59 for the DCTBITPARLL as opposed to the 1.03 area ratio for the DCTROMPARLL.

Testing

The designs were simulated in QuickSimII after backannotation of the parasitic loads and delays. The following is a sampling of the results:

a) DCTBITPARLL

```
        1200.0 1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  1
1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
        1400.0 1  1  1  1  1  1  1  0  0  0  0  0  0  0  1  1
1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1
1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
0  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1
        1600.0 1  1  1  1  1  1  1  0  0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1
1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1
1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
0  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1
        1800.0 1  1  1  1  1  1  1  0  0  0  0  0  0  1  0  0
0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1
1  1  1  1  0  1  1  0  0  0  0  0  0  0  0  0  0  1  1  1  1
1  1  1  1  1  1  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1
        2000.0 1  1  1  1  1  1  1  0  0  0  0  0  0  1  0  0
0  0  1  1  0  0  0  0  0  0  0  0  1  0  0  1  1  1  1  1  1
1  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1
1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1
0  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1
```

```
      2200.0 1  1  1  1  1  1  1  0  0  0  0  0  0  1  0  0
0  1  1  1  0  0  0  0  0  0  0  1  0  1  0  0  1  1  1  1  1
1  1  0  1  0  0  0  0  0  0  0  0  0  0  1  1  1  0  1  1
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  0
0  1  1  1  1  1  1  1  1  1  1  0  1  0  0  0  0  0  0  0  0
0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1
     Time(ns)
```

Since the signal state list report above is wide, they rap around in this document for every time stamp. The order in which the signals appear from left to right after a time stamp is given below (refer to the schematics to locate the signal names). The DCTROMPARLL has the same signal names and column definitions, so this list applies to the DCTROM-PARLL as well:

"RESET", "CLRIN", "CLRIN2", "CLRA", "DLY8/CLRDLY", "IIR_M0/CLRIIR", "CK", "CKIN", "CKIN2", "CKA", "DLY8/CKDLY", "IIR_M0/CKIIR", "D7", "D6", "D5", "D4", "D3", "D2", "D1", "D0", "DCT0(11)", "DCT0(10)", "DCT0(9)", "DCT0(8)", "DCT0(7)", "DCT0(6)", "DCT0(5)", "DCT0(4)", "DCT0(3)", "DCT0(2)", "DCT0(1)", "DCT0(0)", "DCT1(11)", "DCT1(10)", "DCT1(9)", "DCT1(8)", "DCT1(7)", "DCT1(6)", "DCT1(5)", "DCT1(4)", "DCT1(3)", "DCT1(2)", "DCT1(1)", "DCT1(0)", "DCT2(11)", "DCT2(10)", "DCT2(9)", "DCT2(8)", "DCT2(7)", "DCT2(6)", "DCT2(5)", "DCT2(4)", "DCT2(3)", "DCT2(2)", "DCT2(1)", "DCT2(0)", "DCT3(11)", "DCT3(10)", "DCT3(9)", "DCT3(8)", "DCT3(7)", "DCT3(6)", "DCT3(5)", "DCT3(4)", "DCT3(3)", "DCT3(2)", "DCT3(1)", "DCT3(0)", "DCT4(11)", "DCT4(10)", "DCT4(9)", "DCT4(8)", "DCT4(7)", "DCT4(6)", "DCT4(5)", "DCT4(4)", "DCT4(3)", "DCT4(2)", "DCT4(1)", "DCT4(0)", "DCT5(11)", "DCT5(10)", "DCT5(9)", "DCT5(8)", "DCT5(7)", "DCT5(6)", "DCT5(5)", "DCT5(4)", "DCT5(3)", "DCT5(2)", "DCT5(1)", "DCT5(0)", "DCT6(11)", "DCT6(10)", "DCT6(9)", "DCT6(8)", "DCT6(7)", "DCT6(6)", "DCT6(5)", "DCT6(4)", "DCT6(3)", "DCT6(2)", "DCT6(1)", "DCT6(0)", "DCT7(11)", "DCT7(10)", "DCT7(9)", "DCT7(8)", "DCT7(7)", "DCT7(6)", "DCT7(5)", "DCT7(4)", "DCT7(3)", "DCT7(2)", "DCT7(1)", "DCT7(0)".

b) DCTROMPARLL

```
        1200.0 1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   1
1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        1400.0 1   1   1   1   1   1   1   0   0   0   0   0   0   0   1   1
1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1
1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   1   1
1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0
0   1   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0
0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1
        1600.0 1   1   1   1   1   1   1   0   0   0   0   0   0   1   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1   1
1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1
1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0
0   1   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        1800.0 1   1   1   1   1   1   1   0   0   0   0   0   0   1   0   0
0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1   1   1
1   1   1   1   0   1   1   0   0   0   0   0   0   0   0   0   0   1   1   1   1
1   1   1   1   1   1   1   1   0   1   0   0   0   0   0   0   0   0   0   0   0
1   1   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0
0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1
        2000.0 1   1   1   1   1   1   1   0   0   0   0   0   0   1   0   0
0   0   1   1   0   0   0   0   0   0   0   0   1   0   0   1   1   1   1   1   1
1   1   1   0   1   0   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1
1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   1
0   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0
0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1
        2200.0 1   1   1   1   1   1   1   0   0   0   0   0   0   1   0   0
0   1   1   1   0   0   0   0   0   0   0   1   0   1   0   0   1   1   1   1   1
1   1   0   1   0   0   0   0   0   0   0   0   0   0   0   1   1   0   1   1   1
1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0
0   1   1   1   1   1   1   1   1   1   1   0   1   0   0   0   0   0   0   0   0
0   0   0   1   1   1   1   1   1   1   1   1   1   1   1   1
      Time(ns)
```

Since it is extremely difficult to decipher the data and output from a bit-serial simulation report, a program was written in Ample, "chk_parll.amp", to read the report file and give the output in a more friendly manner. The result of the program is:

a) DCTBITPARLL

```
Simulation results of bit-parallel design  dctbitparll
Input report name is d12o_16x16.rep
Output report name is d12o_16x16.chk
Date:  4-12-1994

At Time =     1200 ns  ==>
  DCT(0)  = 000 = 000000000000 = 0
  DCT(1)  = 000 = 000000000000 = 0
  DCT(2)  = 000 = 000000000000 = 0
  DCT(3)  = 000 = 000000000000 = 0
  DCT(4)  = 000 = 000000000000 = 0
  DCT(5)  = 000 = 000000000000 = 0
  DCT(6)  = 000 = 000000000000 = 0
  DCT(7)  = 000 = 000000000000 = 0
At Time =     1400 ns  ==>
  DCT(0)  = 000 = 000000000000 = 0
  DCT(1)  = FFF = 111111111111 = -1
  DCT(2)  = 000 = 000000000000 = 0
  DCT(3)  = FFF = 111111111111 = -1
  DCT(4)  = 000 = 000000000000 = 0
  DCT(5)  = FFF = 111111111111 = -1
  DCT(6)  = 000 = 000000000000 = 0
  DCT(7)  = FFF = 111111111111 = -1
At Time =     1600 ns  ==>
  DCT(0)  = 001 = 000000000001 = 1
  DCT(1)  = FFE = 111111111110 = -2
  DCT(2)  = 001 = 000000000001 = 1
  DCT(3)  = FFE = 111111111110 = -2
  DCT(4)  = 000 = 000000000000 = 0
  DCT(5)  = FFF = 111111111111 = -1
  DCT(6)  = 000 = 000000000000 = 0
  DCT(7)  = FFF = 111111111111 = -1
At Time =     1800 ns  ==>
  DCT(0)  = 003 = 000000000011 = 3
  DCT(1)  = FFB = 111111111011 = -5
  DCT(2)  = 003 = 000000000011 = 3
  DCT(3)  = FFD = 111111111101 = -3
  DCT(4)  = 001 = 000000000001 = 1
  DCT(5)  = FFF = 111111111111 = -1
  DCT(6)  = 000 = 000000000000 = 0
  DCT(7)  = FFF = 111111111111 = -1
```

```
At Time =    2000 ns  ==>
  DCT(0) = 009 = 000000001001 = 9
  DCT(1) = FF4 = 111111110100 = -12
  DCT(2) = 007 = 000000000111 = 7
  DCT(3) = FFC = 111111111100 = -4
  DCT(4) = 002 = 000000000010 = 2
  DCT(5) = FFE = 111111111110 = -2
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = FFF = 111111111111 = -1
At Time =    2200 ns  ==>
  DCT(0) = 014 = 000000010100 = 20
  DCT(1) = FE8 = 111111101000 = -24
  DCT(2) = 00E = 000000001110 = 14
  DCT(3) = FF8 = 111111111000 = -8
  DCT(4) = 004 = 000000000100 = 4
  DCT(5) = FFD = 111111111101 = -3
  DCT(6) = 001 = 000000000001 = 1
  DCT(7) = FFF = 111111111111 = -1
```

a) DCTROMPARLL

```
Simulation results of bit-parallel design  dctromparll
Input report name is d12_rom16x16.rep
Output report name is d12_rom16x16.chk
Date:  4-12-1994

At Time =    1200 ns  ==>
  DCT(0) = 000 = 000000000000 = 0
  DCT(1) = 000 = 000000000000 = 0
  DCT(2) = 000 = 000000000000 = 0
  DCT(3) = 000 = 000000000000 = 0
  DCT(4) = 000 = 000000000000 = 0
  DCT(5) = 000 = 000000000000 = 0
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = 000 = 000000000000 = 0
At Time =    1400 ns  ==>
  DCT(0) = 000 = 000000000000 = 0
  DCT(1) = FFF = 111111111111 = -1
  DCT(2) = 000 = 000000000000 = 0
  DCT(3) = FFF = 111111111111 = -1
  DCT(4) = 000 = 000000000000 = 0
  DCT(5) = FFF = 111111111111 = -1
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = FFF = 111111111111 = -1
```

```
At Time =    1600 ns  ==>
  DCT(0) = 001 = 000000000001 = 1
  DCT(1) = FFE = 111111111110 = -2
  DCT(2) = 001 = 000000000001 = 1
  DCT(3) = FFE = 111111111110 = -2
  DCT(4) = 000 = 000000000000 = 0
  DCT(5) = FFF = 111111111111 = -1
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = 000 = 000000000000 = 0
At Time =    1800 ns  ==>
  DCT(0) = 003 = 000000000011 = 3
  DCT(1) = FFB = 111111111011 = -5
  DCT(2) = 003 = 000000000011 = 3
  DCT(3) = FFD = 111111111101 = -3
  DCT(4) = 001 = 000000000001 = 1
  DCT(5) = FFF = 111111111111 = -1
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = FFF = 111111111111 = -1
At Time =    2000 ns  ==>
  DCT(0) = 009 = 000000001001 = 9
  DCT(1) = FF4 = 111111110100 = -12
  DCT(2) = 007 = 000000000111 = 7
  DCT(3) = FFC = 111111111100 = -4
  DCT(4) = 002 = 000000000010 = 2
  DCT(5) = FFE = 111111111110 = -2
  DCT(6) = 000 = 000000000000 = 0
  DCT(7) = FFF = 111111111111 = -1
At Time =    2200 ns  ==>
  DCT(0) = 014 = 000000010100 = 20
  DCT(1) = FE8 = 111111101000 = -24
  DCT(2) = 00D = 000000001101 = 13
  DCT(3) = FF8 = 111111111000 = -8
  DCT(4) = 004 = 000000000100 = 4
  DCT(5) = FFD = 111111111101 = -3
  DCT(6) = 001 = 000000000001 = 1
  DCT(7) = FFF = 111111111111 = -1
```

These results were compared with the outputs obtained from the "dctparll_out.amp" as mentioned in chapter 2 under the "Testing" subsection. Keep in mind that what you see at, say 3640 ns was what you expected as 3280 (the previous strobe time), allowing for the propagation delay through the chip. The following are a sampling of the expected results file output by "dctparll_out.amp":

```
Outputs of DCT for 'complt_S2' input DATA
Clock period  = 200 ns
Date:  4-14-1994

0.  At Time = 1200  for DATA = [0, 0, 0, 0, 0, 0, 0, 0]  ==>
  DCT(0) = 0 ;  DCT(1) = 0 ;  DCT(2) = 0 ;  DCT(3) = 0 ;
  DCT(4) = 0 ;  DCT(5) = 0 ;  DCT(6) = 0 ;  DCT(7) = 0 ;
1.  At Time = 1400  for DATA = [0, 0, 0, 0, 0, 0, 0, 1]  ==>
  DCT(0) = 0 ;  DCT(1) = 0 ;  DCT(2) = 0 ;  DCT(3) = 0 ;
  DCT(4) = 0 ;  DCT(5) = 0 ;  DCT(6) = 0 ;  DCT(7) = 0 ;
2.  At Time = 1600  for DATA = [0, 0, 0, 0, 0, 0, 1, 3]  ==>
  DCT(0) = 1 ;  DCT(1) = -2 ;  DCT(2) = 2 ;  DCT(3) = -1 ;
  DCT(4) = 1 ;  DCT(5) = 0 ;  DCT(6) = 0 ;  DCT(7) = 0 ;
3.  At Time = 1800  for DATA = [0, 0, 0, 0, 0, 1, 3, 7]  ==>
  DCT(0) = 3 ;  DCT(1) = -5 ;  DCT(2) = 4 ;  DCT(3) = -2 ;
  DCT(4) = 1 ;  DCT(5) = -1 ;  DCT(6) = 0 ;  DCT(7) = 0 ;
4.  At Time = 2000  for DATA = [0, 0, 0, 0, 1, 3, 7, 15]  ==>
  DCT(0) = 8 ;  DCT(1) = -11 ;  DCT(2) = 7 ;  DCT(3) = -4 ;
  DCT(4) = 2 ;  DCT(5) = -1 ;  DCT(6) = 1 ;  DCT(7) = 0 ;
5.  At Time = 2200  for DATA = [0, 0, 0, 1, 3, 7, 15, 31]  ==>
  DCT(0) = 18 ;  DCT(1) = -24 ;  DCT(2) = 14 ;  DCT(3) = -7 ;
  DCT(4) = 5 ;  DCT(5) = -3 ;  DCT(6) = 1 ;  DCT(7) = -1 ;
6.  At Time = 2400  for DATA = [0, 0, 1, 3, 7, 15, 31, 63]  ==>
  DCT(0) = 40 ;  DCT(1) = -48 ;  DCT(2) = 27 ;  DCT(3) = -15 ;
  DCT(4) = 9 ;  DCT(5) = -5 ;  DCT(6) = 3 ;  DCT(7) = -1 ;
7.  At Time = 2600  for DATA = [0, 1, 3, 7, 15, 31, 63, 127]
==>
  DCT(0) = 85 ;  DCT(1) = -97 ;  DCT(2) = 54 ;  DCT(3) = -31 ;
  DCT(4) = 18 ;  DCT(5) = -11 ;  DCT(6) = 6 ;  DCT(7) = -3 ;
```

After the chips were functionally verified, the timing was verified. The speed was incrementally increased to figure out the maximum speed of the chip, without any glitches and setup, hold or pulse width violations which would affect the final output. For the DCTROMPARLL, it was determined that the minimum possible clock period is 69 ns under best case or 28 ns under worst case. This yields a maximum clock frequency of 36.25 MHz and 14.50 MHz under best and worst case conditions, respectively. For the DCTBITPARLL, it was determined that the minimum possible clock period is 88 ns under best case or 35 ns under worst case. This yields a maximum clock frequency of 28.41 MHz and 11.36 MHz under best and worst case conditions, respectively.

Fig. 6.10: DCTBITPARLL Layout

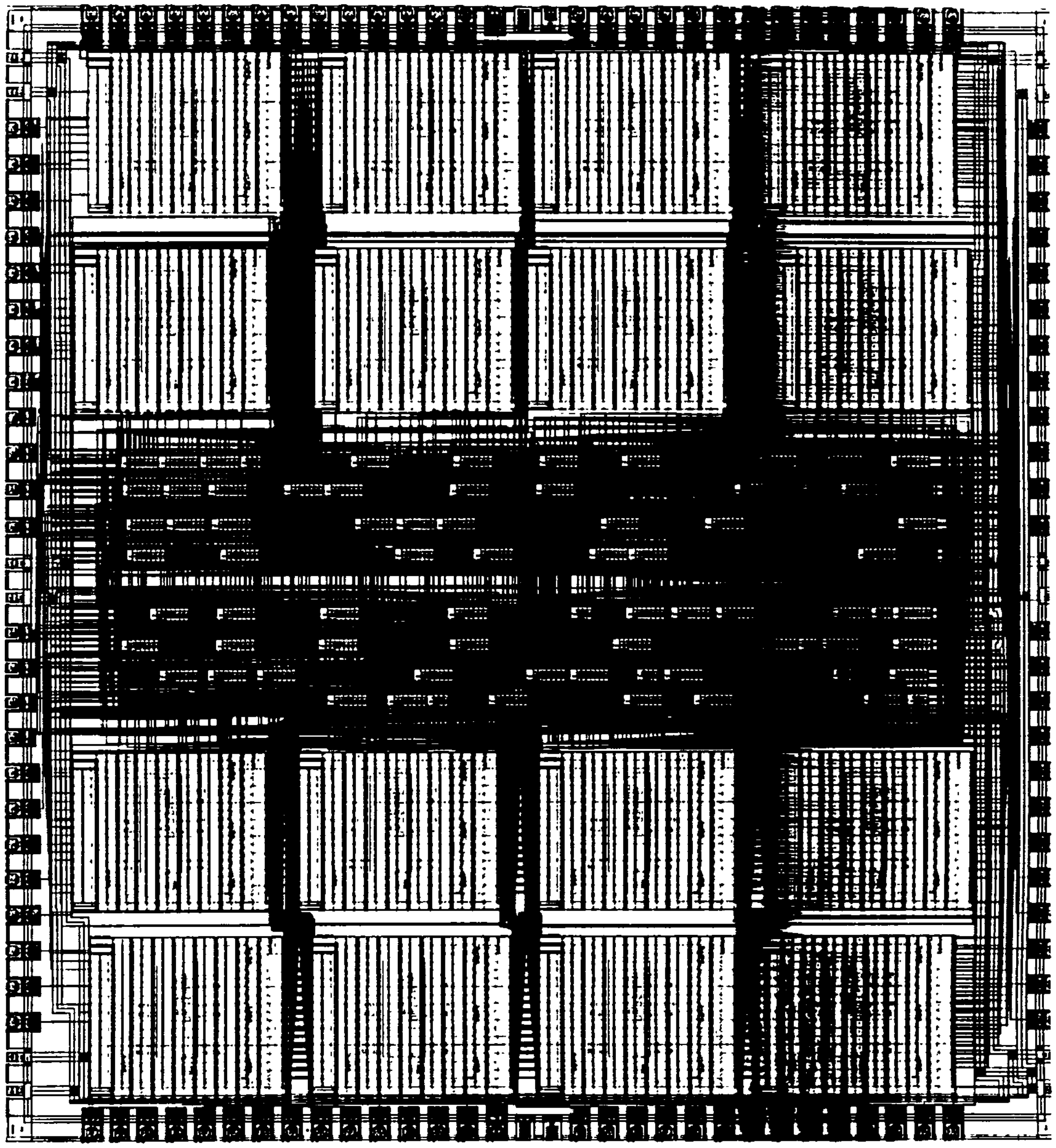
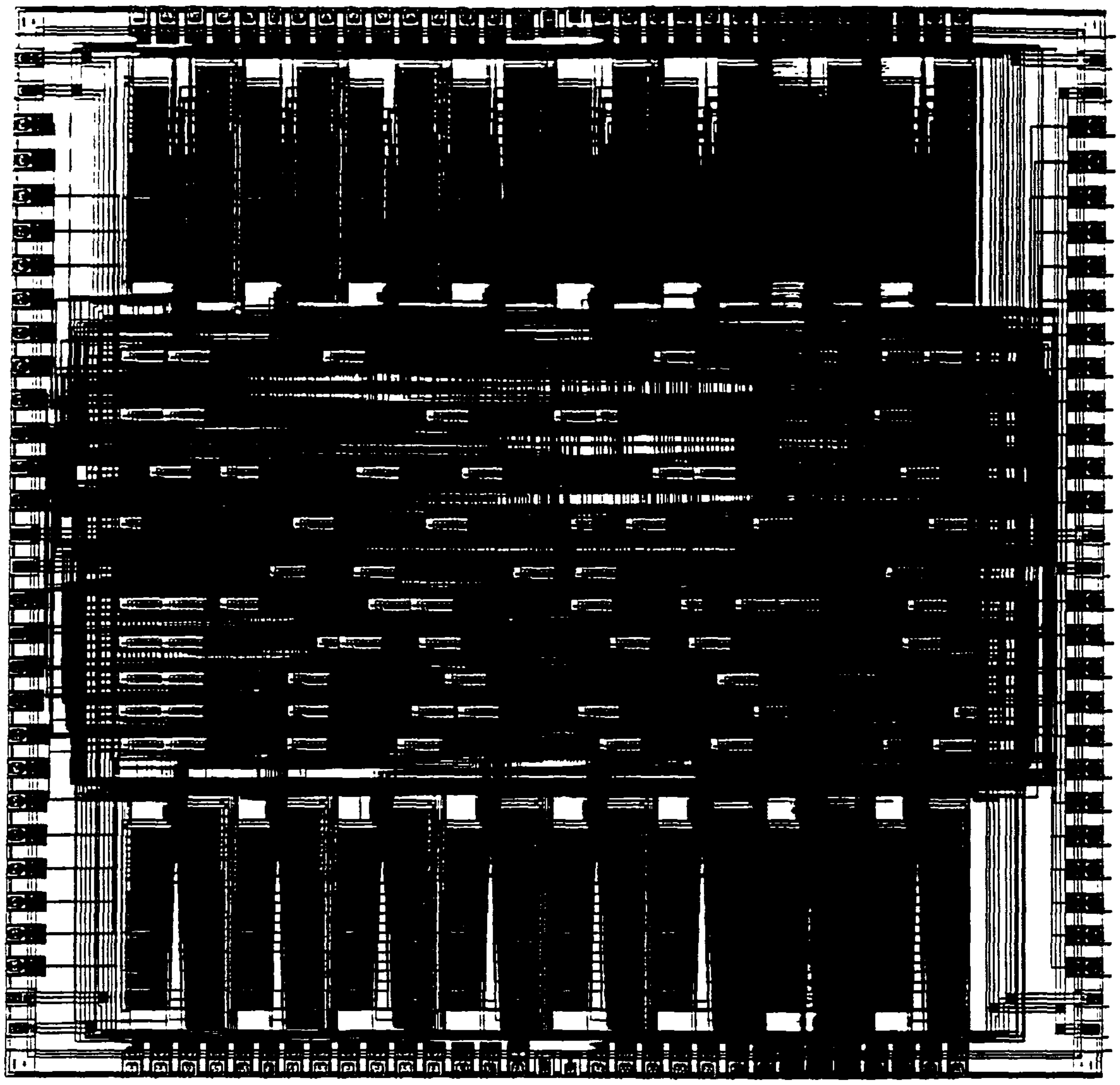


Fig. 6.11: DCTROMPARLL Layout



Chapter 7 - Comparison and Conclusion

Point-by-point Comparison

(1) The bit-serial design is much faster, in terms of clock speed, than the bit-parallel designs -- 139.00 MHz as opposed to 28.41 MHz for DCTBITPARLL and 36.25 MHz for DCTROMPARLL under best case conditions, and 55.56 MHz as opposed to 11.36 MHz for DCTBITPARLL and 14.50 MHz for DCTROMPARLL under worst case conditions.

(2) The area of the DCTBITSERIAL is much smaller, being 1092.6 x 10^2 mils^2, 70.49 x 10^6 micron^2 or 70.49 mm^2. The area of the DCTBITPARLL is 2471.0 x 10^2 mils^2, 159.42 x 10^6 micron^2 or 159.42 mm^2. The area of the DCTROMPARLL is 2171.2 x 10^2 mils^2, 140.08 x 10^6 micron^2 or 140.08 mm^2. Therefore, the DCTBITPARLL is about 2.3 times the area of the DCTBITSERIAL, while the DCTROMPARLL is a little over twice the area of the DCTBITSERIAL. This is despite the fact that the bit-serial design handles 18 bits as opposed to 8 bits input and 12 bits output for the bit-parallel designs.

(3) By definition, the bit-serial implementation requires much less bandwidth, as the bits arrive serially.

(4) The ratio of interconnect area to cell area is much lower for the DCTBITSERIAL; its ratio is 0.52. Although the DCTBITPARLL has a comparable ratio of 0.59, the ratio for the better (in terms of speed and total area) of the two bit-parallel designs, DCTROM-PARLL, is 1.03.

(5) Due to the adjacent placement of interconnected cells of the DCTBITSERIAL, the extremely short critical paths and the highly synchronized nature of the design, the additional timing derived from parasitic extraction was insignificant. Therefore, the speed or

functionality of the design was not affected by layout parasitics, and remained the same as pre-layout (with intrinsic cell delays, input pin capacitances and faux net capacitances as mentioned in chapter 2 under "Functionality and Timing Verification"). In contrast, the DCTBITPARLL and DCTROMPARLL, due to having the big cells, BMP1M4s and ROMs, pre-placed in certain sections of the chip during block place and route, have long critical paths in the layout; therefore, the interconnect nets have to be routed quite a distance to get to these big cells; this results in a sharp reduction in speed after layout.

(6) Number of pins are significantly low for the bit-serial design -- 16 I/O pins as opposed to 106 on the bit-parallel designs. All designs have 16 power pads. Three of the pins on the DCTBITSERIAL, MXS(2:0), are not really necessary, since these function to select one of the 8 LSBOUT signals; we could reduce the number of I/O pins to 13, if we use an 8-input OR, using the 8 LSBOUT signals from the 8 IIR filter modules as inputs, to figure the chip's LSBOUT. All designs have 16 power pads in addition.

(7) The package size for the DCTBITSERIAL, in terms of pin-out, is significantly smaller than both bit-parallel designs. This proves to be cost-effective. The DCTBITSERIAL could fit in a package with 32 pins (or 29 pins) or more. The package size for both bit-parallel designs would have to fit 122 pins.

(8) The throughput of the bit-parallel designs is 1 transform word per clock cycle (frame-recursive transforms). The throughput of the bit-serial design is 1 bit per clock cycle; since the data is 18 bits, it takes 18 cycles to get one transform word -- however, this is acceptable since the data is coming in at the same rate. Therefore, relative to their data incoming rates, all designs have the same throughput.

(9) Latency of the bit-serial design is much higher -- 38 clock cycles as opposed to 5 for the bit-parallel designs.

(10) Since the bit-serial is an 18 bit data design, it is has greater accuracy and precision

when compared to the 8 bit input data bit-parallel design. The accuracy of the bit-serial design lies in the DUAL_MULT8_2CKx4 bit-serial multiplier, which has an accuracy of o(2^{-16}); this is equivalent to a SNR of 96.33 dB. The accuracy of the bit-parallel multiplier depends on how large the coefficient and data are, and is therefore largely sequence-dependent. The SNR of the bit-parallel designs do not meet the criterion of 40 dB, which is required for most applications.

(11) The bit-serial multiplier handles only 8 bit coefficients. For each increment in coefficient bit width, we would increase the absolute data bit width by 2, and thereby increase the absolute throughput and latency by 2 clock cycles. However, the 8 bit coefficient width has proven to be sufficient, yielding a SNR of 96.33 dB.

(12) Equivalent gate counts for the DCTBITSERIAL, DCTBITPARLL and DCTROM-PARLL are 19349.5, 45264.0 and 52096.0 respectively. 1 equivalent gate represents a NAND gate, i.e., 4 transistors for CMOS. Therefore, multiplying these values by 4 would give the transistor count.

(13) Other miscellaneous statistics about the chips, such as number of standard cell instances, number of nets, number of pins, pins per net, pins per cell, total cell area, total active area, number of feed cells, number of link cells, etc., can be obtained from the designs' ERC reports, Silvar-Lisco's "scbuild" place and route preprocessor output summary and the layout summary from Silvar Lisco.

Desires and Suggested Improvements

Looking back on the designs and the lessons learned, one could draw a few conclusions and think of ways of improvement. The following set of "desires and suggested improvements" were not implemented due to resource and time conflicts:

(1) The second set of coefficients, COEMk(15:0) for k = 0 to 7, are all positive and of magnitude less than 0.5. Therefore, the three most significant bits are always 0. This means that the two most significant bits are redundant. Therefore, we could think of increasing the accuracy of the multiplication by ignoring the two MSBs and adding two LSBs. Then we would have to carefully extract the output, with appropriate shifts, if necessary. This could be done easily with the DCTBITPARLL design; we could supply the new COEMs (which are, in effect, twice shifted to the left) and extract the outputs twice shifted to the right, i.e., instead of taking P[29:14], we should take P[27:12]. For the DCTROMPARLL, this could help in the program to calculate the results, increasing the intermediate result precision by two bits. For the DCTBITSERIAL design, we could use a MSBMULT_2CK instead of a MSBMULT_2CKx4, increasing the accuracy by two bits; another option is to use another bit-serial multiplier, with only a 6 bit coefficient and no MSBMULT-type cells -- this would cut the 2 MSBs of the coefficient, reducing the latency and increasing the throughput of the multiplier, and thereby the design, by 4 clock cycles.

(2) We could go a step further to analyze each coefficient and come up with optimal precision widths for each IIR module. However, the downside of this is that we loose a level of modularity; each one of the eight modules would need a separate schematic.

(3) Use 12 bit bus precision for the bit-parallel multipliers. This, although may cause a loss in precision, will increase the speed of the designs as the propagation delay of a 12 bit ROM or 12 x 12 multiplier is 10% to 15% smaller than a 16 bit ROM or a 16 x16 bit multiplier, and the delay through a SUB12 is about 25% less than that through a SUB16.

(4) Use only 8 bits for the coefficient in the bit-parallel design (as it was proven with the bit-serial design that an 8-bit coefficient is sufficient). Using an 8-bit coefficient would cut down the multiplier size by 8 bits to a 16 x 8 multiplier, and thereby decrease the critical path delay and increase the speed of the chip significantly.

(5) Increasing the input data bus width, and increasing the internal bus precision and the output data bus width accordingly, for the bit-parallel versions may improve the SNR.

(6) Use custom cells to reduce the critical path delays. For example, the worst case delay of 25 ns through the SUB16 is critical to the speed of the chip. If we could design a better SUB16 custom cell ourselves, we may be able to increase the speed. The 64.5 ns delay through the BMP1M4 and the 38.1 ns delay through the RO18M4B are also critical to the bit-parallel designs. We may be able to modify the existing BMP1M4, as we use output pins P[29:14] only, and thereby reduce the size and delay of the multiplier.

(7) Power consumption needs to be estimated. As mentioned before, this capability is not available currently with the MEC DC1.2 library.

(8) For the bit-parallel multipliers, we could multiply the DCT coefficients by 2, 4, 8, 16, 32, ... (before feeding them to the design) and shift right 1, 2, 3, 4, 5, ... (effectively dividing by 2, 4, 8, 16, 32, ...) times respectively to obtain the final result. This approach would increase the precision of the coefficients, and thereby the multiplication.

Conclusion

A bit-serial implementation of the Discrete Cosine Transform processor, using the fully-pipelined, real-time, time-recursive IIR filter module was presented in this thesis; the counterpart bit-parallel implementations were also presented. All three designs maintain a high level of modularity and regularity, and are free of global communication; they were designed to be hierarchical, and thereby correct by construction. The hardware required is of o(N), where N corresponds to an N-point DCT.

Clearly, the bit-serial implementation has many advantages over the bit-parallel implementations; the advantages that the bit-parallel implementations have over the bit-serial implementation are few. This is despite the fact that the bit-serial design handles 18 bits of data (but 8 bit coefficients), whereas the bit-parallel designs can handle only 8 bit data input (but 16 bit coefficients). The major disadvantages of the bit-serial design is a slower absolute throughput, which is equivalent to the width (in bits) of the data stream, and the higher latency of minimum 38 clock cycles. Another disadvantage is the reduced precision of the coefficients, which however, proves to be insignificant as the DCTBITSERIAL possesses a SNR of 96.33 dB. Apart from these disadvantages, we saw that the DCTBITSERIAL has many advantages which by far overshadow its disadvantages.

These aspects of the bit-serial DCT processor design, presented here, are resonant with the requirements of modern day image, speech and signal processing applications, data compression, spectrum analysis, computer tomography and signal reconstruction, and specifically high-definition television (HDTV). The amazingly high speed of this device, up to 139.0 MHz, is particularly attractive.

APPENDIX

AlliedSignal Microelectronics DC1.2 Library Cells Used

The following is a list of the AlliedSignal Aerospace Microelectronic and Technology Center's DC1.2 library parts used:

BMP1M4	16 x 16 Multiplier (two's complement and sign magnitude)
C050M	Buffer
P000M	Input Pad
P001M	Inverting Input Pad
P002M	Output Pad
P006M,P007M	Power Pads
P006M1,P007M1	Auxilliary Power Pads, Metal1
P014M	Inverting Input pad with Schmitt Trigger (P014M1 is with pullup)
RO18M3B	ROM 256 words x 12 bits
RO18M4B	ROM 256 words x 16 bits
T004M	Simple Inverter
T004F	4X Inverting Driver
T008M	2-Input AND
T032M	2-Input OR
T082M1	Full Adder
T151M	1 bit, 8-to-1 Line Multiplexer
T157M1	1 bit, 2-to-1 Line Multiplexer
T164Mn	n bit SIPO Shift Register
T273Mn	n bit High Speed D-Register
T283M	4 bit Fast Carry Adder

References

[1] K.J.R. Liu, C.T. Chiu, R.K. Kolagotla, and J.F. Ja'Ja', "Optimal Unified Architectures for the Real-Time Computation of Time-Recursive Discrete Sinusoidal Transforms", Technical Research Report, pp. 1-23, University of Maryland, 1992.

[2] K.J.R. Liu, and C.T. Chiu, "Unified Parallel Lattice Structures for Time-Recursive Discrete Cosine/Sine/Hartley Transforms", IEEE Trans. Signal Processing, May 1993.

[3] T.A. Goodrich, "VLSI Design of Discrete Fourier Transform Processors", M.S. Thesis, University of Maryland, 1991.

[4] N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete Cosine Transform", IEEE Trans. Comput., vol. C-23, pp. 90-93, Jan. 1974.

[5] A. Rosenfeld, and A.C. Kak, "Digital Picture Processing", 2nd edition, Academic Press, 1982.

[6] A.V. Oppenheim and R.W. Schafer, "Discrete Time Siganal Processing", Prentice Hall, 1989.

[7] V. Srinivasan, "Full Custom VLSI Implementation of Time Recursive 2-D DCT/IDCT Chip", M.S. Thesis, University of Maryland, 1993.

[8] W.H. Chen, C.H. Smith, and S.C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Trans. Communications, vol. COM-25, pp. 1004-1009, Sept. 1977.

[9] B.G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform", IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-32, pp. 1243-1245, Dec. 1984.

[10] H.S. Hou, “A Fast Recursive Algorithm for Computing the Discrete Cosine Transform”, IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-35, pp. 1455-1461, Oct. 1987.

CURRICULUM VITAE

Name: Vignarajah Karunakaran

Permanent address: 8440 Cotoneaster Drive, #4F, Ellicott City, MD 21043

Degree and date to be conferred: Master of Science, 1994

Date of birth: May 3, 1967

Place of birth: Colombo, Sri Lanka

Secondary education: Royal College, Colombo-7, Sri Lanka, 1986

Collegiate institutions attended:

INSTITUTION	DATES ATTENDED	DEGREE	DATE OF DEGREE
University of Southern California	1987-1990	B.S.E.E.	1990
University of Maryland	1990-1994	M.S.	1994

Major: Electrical Engineering

Professional positions held:

ASIC Design Engineer, Microelectronics & Technology Center, AlliedSignal Aerospace, Columbia, MD, 1993-1994

Electronic Design Engineer, Computer Aided Engineering Center, AlliedSignal Aerospace, Columbia, MD, 1991-93