Design Exploration for Pipelined IDCT

Daniel D. Gajski
Peter Grun
Wenwei Pan
Smita Bakshi

Technical Report #96-41
September 12, 1996

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
(714) 824-7063

gajski@ics.uci.edu
pgrun@ics.uci.edu
wpan@ics.uci.edu
sbakshi@ics.uci.edu

Abstract

ASICs for video compression systems have stringent timing requirements. For example, according to the MPEG standard, the throughput of the MPEG decoder is 4115ns. This performance cannot be achieved without efficient pipelining. In this report, we explore the pipelined designs for the Inverse Discrete Cosine Transform (IDCT) which is a critical part of the MPEG decoder. We also transform the algorithm to minimize the memory requirement. We have implemented both the original and memory-optimized algorithms at the RT level, using our realistic library. The behavioral descriptions for both designs are given in the appendix.

Contents

List of Figures

Design Exploration for Pipelined IDCT

Daniel D. Gajski, Peter Grun, Wenwei Pan, Smita Bakshi
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

Abstract

ASICs for video compression systems have stringent timing requirements. For example, according to the MPEG standard, the throughput of the MPEG decoder is 4115ns. This performance cannot be achieved without efficient pipelining. In this report, we explore the pipelined designs for the Inverse Discrete Cosine Transform (IDCT) which is a critical part of the MPEG decoder. We also transform the algorithm to minimize the memory requirement. We have implemented both the original and memory-optimized algorithms at the RT level, using our realistic library. The behavioral descriptions for both designs are given in the appendix.

1 Introduction

In this report, we give a detailed example showing how to design pipelined digital systems from behavioral description. The example is a custom ASIC for Inverse Discrete Cosine Transform (IDCT), which is used in the MPEG decoder. Details of the design as well as the source listing of VHDL code are given in the following sections.

First, we transform the behavioral level specification of the algorithm in order to optimize the memory size, obtaining an improvement in the total cost of the design. By pipelining the designs in different ways, we obtain a variety of cost/performance trade-offs.

Section 2 presents the specification of the IDCT example, section 3 describes the library components needed. Next in section 4 we show the memory optimization, and in section 5 we present the pipelining exploration, by comparing several design alternatives. After the conclusions in section 6, in the appendix we show the VHDL code for our fastest pipelined designs.

2 Example Description

IDCT is often a critical part in both still and motion picture compression. It takes as input an N x N image block, and creates an output matrix which is subsequently used in the next stages of the computation.

The definition of IDCT for a block of N x N image is:

$$O[u, v] = \frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} I[m, n]\cos\frac{(2m+1)u\pi}{2N}\cos\frac{(2n+1)v\pi}{2N}$$

where:
u, v = discrete frequency variables $0 \le u, v \le N-1$.
I[m,n] = input, representing the gray level of a pixel at position (m,n) of the N x N image $0 \le m, n \le N-1$.
O[u,v] = output, representing the coefficient of point(u,v) in spatial frequency domain.

The mathematical definition for IDCT can be thought of as a pair of matrix multiplications. I is the input matrix, COS is a coefficient matrix, and O is the output matrix.

In the floating point domain, we define a coefficients matrix C as:

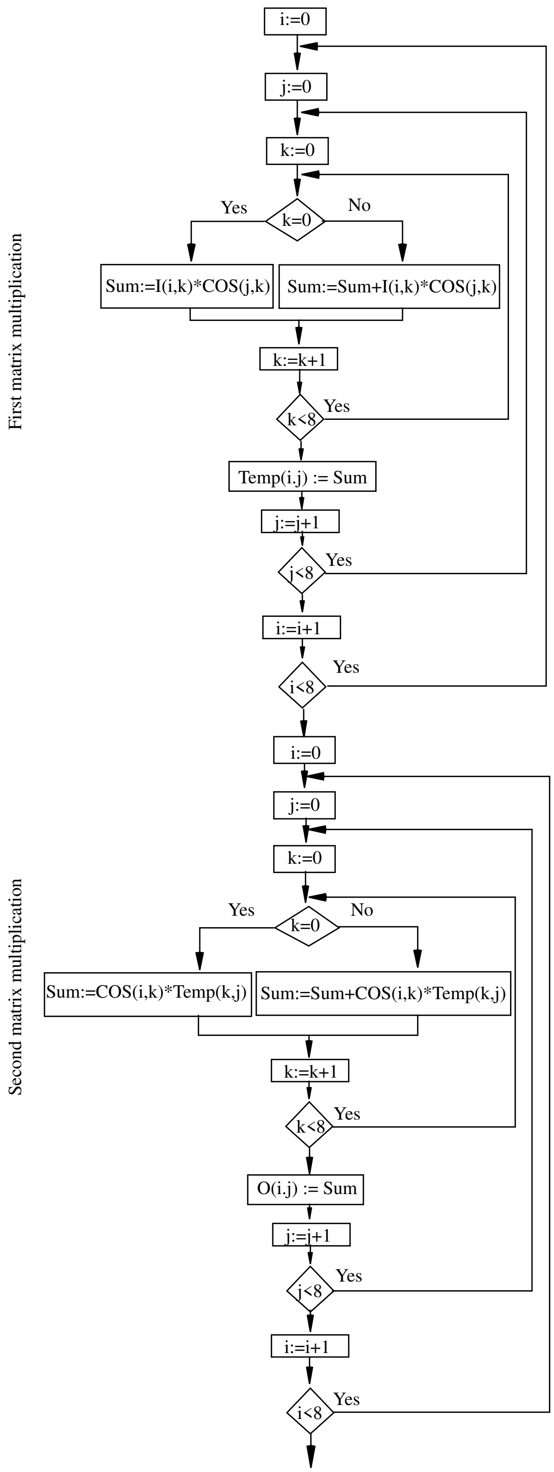

Figure 1: IDCT algorithm.

$$C[u,v] = \frac{1}{N}\cos\frac{(2n+1)u\pi}{2N}$$

Based on C, we define the integer coefficients matrix COS as:

$$\text{COS} = round(factor * \text{C})$$

and COS^T as it's transposed.

Let I be the N x N input block of image, and O be the output N x N matrix. Then,

$$\text{O} = \text{COS} \times \text{I} \times \text{COS}^T$$

Or,

$$\text{Temp} = \text{I} \times \text{COS}^T$$
$$\text{O} = \text{COS} \times \text{Temp}$$

In the MPEG standard, N is 8.

Figure 1 Shows the CDFG of the IDCT algorithm.

3 Library Components

During design space exploration, we use different components to implement the operations and storage elements from the IDCT algorithm, to obtain a variety of cost/performance trade-offs. In Table 1 we show the component library used.

No	Component	Delay (ns)	Cost (trans.)
1	16 bit selector	0.4	224
2	32 bit selector	0.4	448
3	16 bit CLA adder	2.1	1074
4	32 bit CLA adder	2.9	2148
5	8 bit booth multiplier	5.6	3562

Table 1: Components library.

No	Component	Delay (ns)	Cost (trans.)
6	16 bit booth multiplier	8.8	11220
7	8 bit booth multiplier (2 stage pipe)	3.5	4210
8	16 bit booth multiplier (2 stage pipe)	5.4	12624
9	8 bit booth multiplier (4 stage pipe)	2.7	5218
10	16 bit booth multiplier (4 stage pipe)	3.5	15036
11	8 bit register	0.4	256
12	9 bit register	0.4	272
13	16 bit register	0.4	512
14	32 bit register	0.4	1024
15	9 bit counter	2.5	414
16	64x16 RAM	3.5	6144
17	64x8 ROM	3.5	2048

Table 1: Components library.

In column three we show the delays of the components. We use the worse case delays for single signal change from any input to any output.

For the pipelined components, the delay represents the delay of the longest stage. For the storage components it represents the average between the reading and writing time.

In column four we show the cost of each component. We use 1 literal/2 transistors estimation for the basic gates (*nand, nor, inverter*), which we use subsequently in the computation of the cost of the rest of components, in a bottom-up fashion.

4 Memory Optimization

As previously mentioned, the IDCT algorithm consists of two consecutive matrix multiplications. A new input sample of 64 bytes arrives every 4115 ns. The input is stored in a 64x8 input RAM. After computation, the output is stored in a 64x8 output RAM. Our goal is: given a component library, to design an IDCT chip which has the lowest area, while still satisfying the time constraint.

Experiments have shown that in most video and speech processing applications, more than 50% of the chip area is occupied by memory units [4]. Therefore, by minimizing the storage requirements, we can obtain more area improvement than by other datapath components optimization. Hence, to drastically decrease the cost of the chip, we need to start by optimizing the memory usage.

Starting from the initial behavioral description of the IDCT (see appendix), we show how to transform the algorithm in order to lower the memory requirements.

As shown in section 2, the algorithm first multiplies the input matrix with the transposed cosine matrix generating a temporary matrix, and then multiplies the cosine matrix with the temporary to obtain the final result. Therefore, for this behavioral description, besides the input and output memories, we need another storage for the temporary result:

$$\mathrm{Temp} = \mathrm{I} \times \mathrm{COS}^{\mathrm{T}}$$
$$\mathrm{O} = \mathrm{COS} \times \mathrm{Temp}$$

One way to reduce the memory requirements is to eliminate the temporary memory from the computation. In order to do this, the order in which the operations in the algorithm are done has to be changed.

In the initial description, the first matrix multiplication computes all the elements of the temporary matrix, and the second matrix multiplication uses them to create the output matrix. If we could schedule all the uses of each element from the temporary matrix immediately after they are created, we would need only one word as temporary storage instead of the whole matrix.

In the initial algorithm, the creations and uses of the elements of Temp are interleaved, which implies that their lifetimes overlap. Thus, they cannot use the same memory location. On the other hand, if we manage to transform the algorithm so that the creations and uses of these elements are not interleaved, their lifetimes will not overlap, and only one memory location will be needed.

Instead of creating the whole Temp matrix, and then using it, we create one element at a time, and use it wherever it is needed before creating a new element. Therefore the new element can share the same memory location with the previous one, and instead of the whole Temp matrix we only need one storage location.

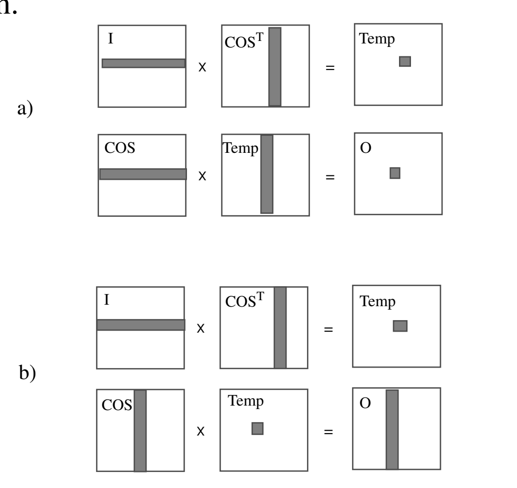

Figure 2: Memory minimization technique a) original algorithm; b) optimized algorithm.

In Figure 2 a) we show the matrix multiplications for the original algorithm. Here we first compute all the elements from the Temp matrix, and then use them column by column in the second matrix multiplication. In Figure 2 b) we show the optimized version of the algorithm. Here we first compute one element of the Temp matrix, and then use it immediately in the second matrix multiplication to accumulate the partial sums for the corresponding column from the O matrix.

In the appendix we show the behavioral VHDL description of both the original and the optimized algorithms.

These two algorithms represent the starting point in our design process. In the next sections we show several pipelined implementations of these algorithms, generating different cost/performance trade-offs.

5 Pipelining exploration

Most digital circuits today are working under certain timing constraints. For example, according to the MPEG standard, the IDCT stage should be done within 4115 ns.

Pipelining is an efficient way to greatly improve the performance of a digital system, without significantly increasing the cost.

To explore a large variety of cost/performance trade-offs, different levels of pipelining can be used. Intuitively, if the design is more pipelined, a performance gain is obtained, against an increase in complexity and a small degradation of the cost of the design.

Any computation composed of a set of tasks which operate in a sequential order on data arriving repetitively, can be pipelined by considering each task a different stage. Depending on what we consider as the computation and it's constituent tasks, we obtain pipelining at different levels.

The original IDCT algorithm is composed of two loops representing the two matrix multiplications. The memory-optimized algorithm is composed of a single loop consisting of two inner loops. In the following we present the different levels of pipelining for each of these algorithms.

5.1 Pipelining Levels.

5.1.1 Original algorithm.

5.1.1.1 Process pipelining

```
for i:=0 to 7 loop
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := I(i,k); B := COS(j,k);
            P := A * B;
            SUM := SUM + P;
            Temp(i,j) := SUM;
        end loop;
    end loop;
end loop;
                                        stage 1
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
for i:=0 to 7 loop                      stage 2
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := COS(i,k); B := Temp(k,j);
            P := A * B;
            SUM := SUM + P;
            O(i,j) := SUM;
        end loop;
    end loop;
end loop;
```

Figure 3: Process pipelining

If we consider the whole IDCT algorithm as a computation, and the two matrix multiplications as the constituent tasks, we obtain the first level pipelining. We call this process pipelining.

The two matrix multiplications represent the two stages of the pipeline, as shown in Figure 3.

5.1.1.2 Loop body pipelining

Each matrix multiplication consists of a loop. If we consider the body of this loop as a computation, and the different operations (such as the memory read, memory write, multiplication and addition) as it's constituent tasks, we obtain the second level pipelining. We call this loop body pipelining.

```
for i:=0 to 7 loop
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := I(i,k); B := COS(j,k);      stage 1
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            P := A * B;                      stage 2
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            SUM := SUM + P;                  stage 3
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            Temp(i,j) := SUM;                stage 4
        end loop;
    end loop;
end loop;
```

(a)

```
for i:=0 to 7 loop
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := COS(i,k); B := Temp(k,j);   stage 1
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            P := A * B;                      stage 2
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            SUM := SUM + P;                  stage 3
            \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
            O(i,j) := SUM;                   stage 4
        end loop;
    end loop;
end loop;
```

(b)

Figure 4: Loop body pipelining for a) the first matrix multiplication and b) the second matrix multiplication

Figure 4 shows the pipe stages in the loop body pipelining.

5.1.1.3 Functional unit pipelining

If we consider only one operation, such as the multiplication, as a computation, we can divide it into different tasks, and consider each task as a different pipe stage. This results in the third level pipelining, which we call functional unit pipelining.

Figure 5 shows the functional unit pipelining. In this case the multiplier is divided into 4 stages, while the other operations are non-pipelined.

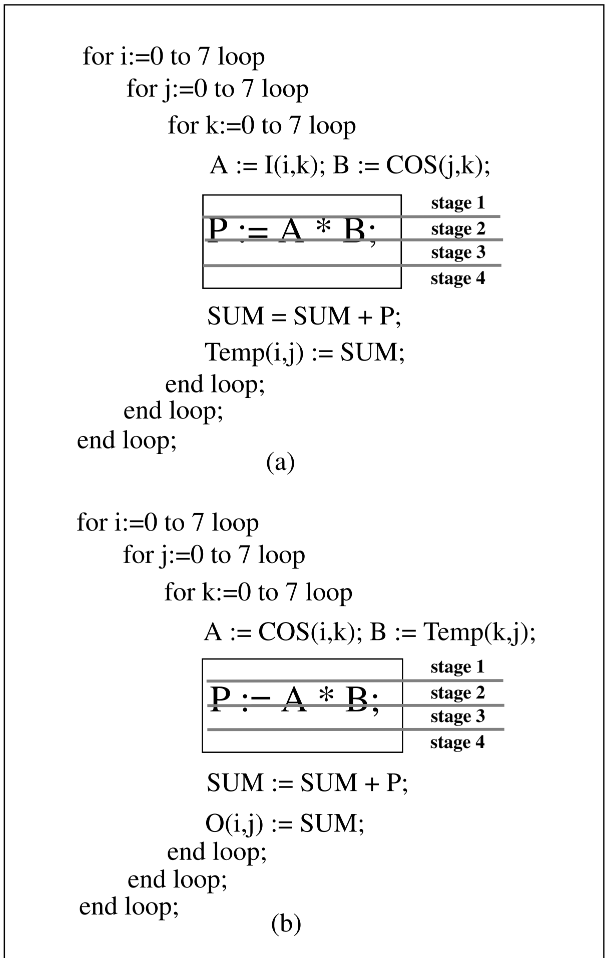

Figure 5: Functional unit pipelining for a) first matrix multiplication and b) second matrix multiplication

5.1.2 Memory-optimized algorithm.

The memory-optimized algorithm is comprised of an outer loop, which consists at it's turn of two inner loops.

If we consider the whole algorithm as a computation, we should divide it into smaller tasks. Since there is only one outer loop in the algorithm, we cannot split it into different parts without changing the specification. Therefore, we cannot pipeline it at the process level.

On the other hand, if we consider the body of this loop as a computation and the two inner loops as the constituent tasks, we obtain a first loop body pipelining.

By considering the body of each of the innermost loops as a computation comprised of the tasks of memory read, multiplication, addition and memory write, a second loop body pipelining is obtained.

As in the case of the original algorithm, each operation, such as multiplication, can be considered as a computation consisting of several parts. Each part is a different task, representing the stages of the pipeline.

5.1.2.1 First loop body pipelining.

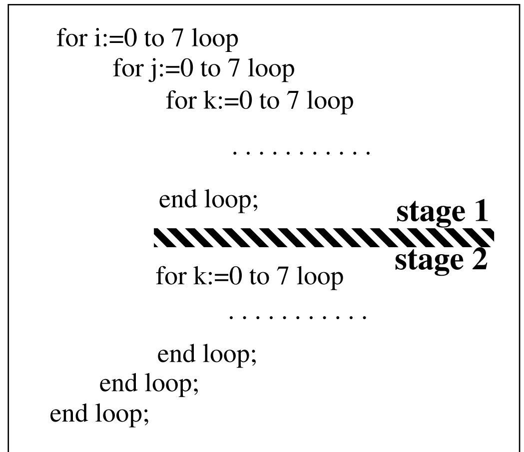

Figure 6: First loop body pipelining.

By pipelining the body of the j loop we obtain a two-stage pipeline, as shown in Figure 6. In this case, each stage consists of an inner loop executing 8 times.

5.1.2.2 Second loop body pipelining.

Figure 7 shows the innermost loop pipelining. Here we consider the body of the k loops as the computation to be pipelined, and the memory read, multiplication, addition and memory write tasks represent the pipe stages.

```
for i:=0 to 7 loop
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := I(i,k);   B := COS(j,k);    stage 1
            P := A * B;                      stage 2
            SUM := SUM + P;                  stage 3
        end loop;
        Temp := Sum;
        for k:=0 to 7 loop
            C := O(k,j);  D := COS(k,i);     stage 1
            P := D * Temp;                   stage 2
            SUM := C + P;                    stage 3
            O(k,j) := SUM;                   stage 4
        end loop;
    end loop;
end loop;
```

Figure 7: Second loop body pipelining for the two innermost loops.

5.1.2.3 Functional unit pipelining.

The functional units implementing operations from the algorithm can be also pipelined. The computation can be divided in several tasks, each task to represent a pipe stage, as shown in Figure 8.

```
for i:=0 to 7 loop
    for j:=0 to 7 loop
        for k:=0 to 7 loop
            A := I(i,k);   B := COS(j,k);
            P := A * B;          stage 1
                                 stage 2
                                 stage 3
                                 stage 4
            SUM := SUM + P;
        end loop;
        Temp := Sum;
        for k:=0 to 7 loop
            C := O(k,j);  D := COS(k,i);
            P := D * Temp;       stage 1
                                 stage 2
                                 stage 3
                                 stage 4
            SUM := C + P;
            O(k,j) := SUM;
        end loop;
    end loop;
end loop;
```

Figure 8: Functional unit pipelining.

5.2 Performance Comparison.

In a time constrained design flow, the goal is to obtain the lowest cost implementation while still satisfying the time constraint. In the IDCT example the computation has to be executed in 4115 ns.

By pipelining the design, we can increase the performance, without a significant impact on the cost. In the previous section we presented the different pipelining options we have for the IDCT algorithm. In the final design, these options can be combined in any way, to obtain different cost/performance trade-offs. For exemplification, we show a comparison between several alternatives.

No	Pipelining	Throughput (ns)	Cost (trans.)
1.1	non-pipelined	36044.8	22198
1.2	functional unit pipelining	25088	26014

Table 2: Original algorithm pipelining

No	Pipelining	Throughput (ns)	Cost (trans.)
1.3	process pipelining	18022.4	41486
1.4	loop body pipe-lining	9046	24550
1.5	process + loop + functional unit pipelining	1813	53822

Table 2: Original algorithm pipelining

No	Pipelining	Throughput (ns)	Cost (trans.)
2.1	non-pipelined	40550	16054
2.2	functional unit pipelining	23296	19870
2.3	first loop body pipelining	18022	29646
2.4	second loop body pipelining	11827	17590
2.5	first loop + second loop + functional unit pipelining	1844	42350

Table 3: Optimized algorithm pipelining

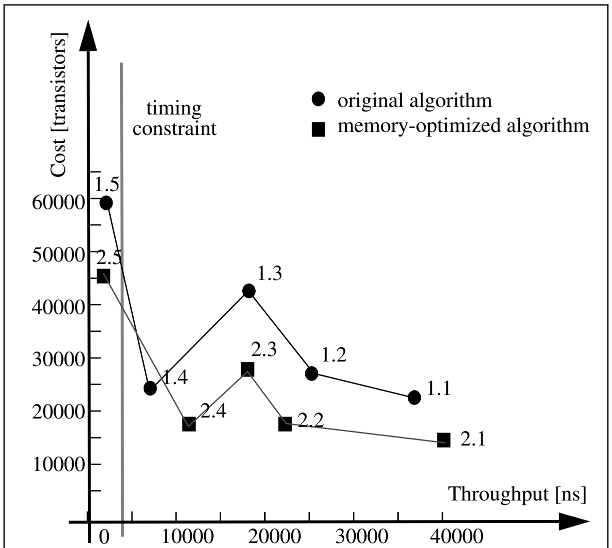

Figure 9: Pipelining exploration.

Tables 2 and 3 show the performance and cost of different implementations using the two algorithms for different combinations of pipelining. We present the non-pipelined (1.1, 2.1), partially pipelined (1.2, 1.3, 1.4, 2.2, 2.3, 2.4) and completely pipelined (1.5, 2.5) designs.

In Figure 9 we present the design alternatives in the cost/throughput space. The points denoted 1.1-1.5 represent different implementations of the original algorithm, whereas the points 2.1-2.5 represent implementations of the memory-optimized algorithm. By comparing the points on each curve, we see that the more pipelined the design, the higher the performance. The points 1.5 and 2.5 represent the most highly pipelined versions, combining all the three pipelining levels. As expected, they have the highest performance and highest cost. The position of the points 1.2, 1.3, 1.4 shows the relationship between the different pipelining alternatives. The most performance gain is obtained by the loop body pipelining, whereas the lowest gain is generated by the functional unit pipelining. A similar relationship is obtained for the memory-optimized algorithm.

By comparing the two curves representing the original and the memory-optimized algorithm, we see that the overall cost of the optimized algorithm is generally lower than the cost of the original one.

6 VHDL Models Hierarchy.

All the VHDL models are developed hierarchically in a bottom up fashion, as shown in Figure 10.

(1). The 1st level of hierarchy consists of the basic gates, *muxes* and *flip flops*. All the VHDL models in this level have only behavioral description. All the higher level components are composed of these basic entities.

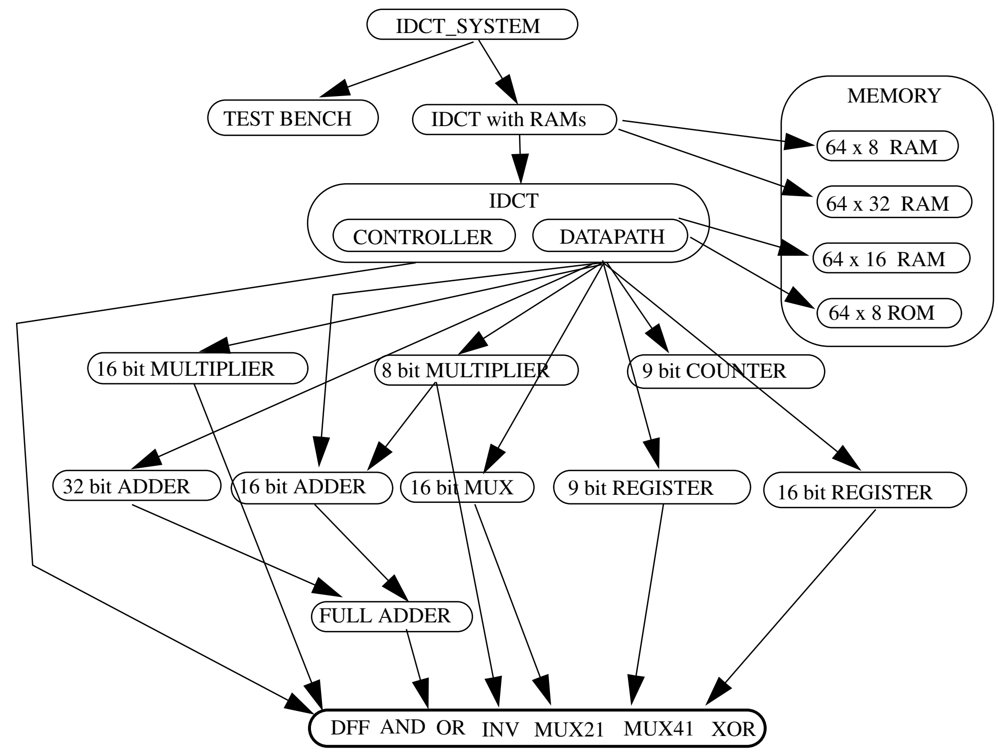

Figure 10 : VHDL modeling hierarchy

The delay information for these gates and flip flops are shown in [3] (due to current technology we scaled them down by 10).

(2). The 2nd level of hierarchy consists of the 16-bit and 32-bit adders, selectors, multipliers, and registers. They appear as RT level components in the datapath. All the VHDL models in this level have both the behavioral and structural description.

(3). Another level of hierarchy is the memory. The 64x8 ROM is used for storing the COS matrix. The 64x8 RAM and 64x32 RAM are used as input and output buffers for the IDCT as interface with other stages in the MPEG decoder. In the implementation of the original algorithm, 64x16 RAM is also used as a buffer between the two matrix multiplications.

(4). The 4th level of hierarchy consists of the datapath and controller. The datapath model is an RT level structural model. The controller is behavioral model consisting of next-state logic, output logic and state registers.

(5). The 5th level of hierarchy comprises the IDCT computation along with the input and output memories (the IDCT with RAM entity). It also includes the Test Bench entity,

(6). The 6th level of hierarchy simply incorporates the IDCT with RAM model and the testbench to be simulated.

7 Conclusions.

This report presents techniques for pipelining exploration and memory optimization. We first lower the total cost of the design by transforming the algorithm. Given a timing constraint, we traverse the design space by using different pipelining alternatives. Finally we compare the different designs created, and show what are the implications of the early design decisions on the final implementation.

We obtain a large spectrum of cost/performance trade-offs, providing a good starting point for the next levels of synthesis.

8 References.

[1] D. D. Gajski, “Principles of Digital Design”, Prentice Hall 1996.

[2] D. D. Gajski, N. Dutt, A. Wu, and S. Lin, “High Level Synthesis: Introduction to Chip and System Design”, Kluwer Academic Publishers, 1992.

[3] W. Pan, P. Grun, D.D. Gajski, “Behavioral Exploration with RTL Library”, Technical Report, UCI ICS #96-34, July 29, 1996.

[4] F. Catthoor, W. Geurts, H. De Man, “Loop Transformation Methodology for Fixed Rate Video, Image and Telecom Processing Applications”, Proc. Int. Conf. on Applic. Spec. Array Processors, San Francisco, CA, Aug. 1994.

9 Appendix.

9.1 IDCT System.

```
-- represents the whole system which includes
-- the test bench, the input and output memory,
-- and the IDCT computation.


library ieee;
   use ieee.std_logic_1164.all;
   use ieee.std_logic_misc.all;
   use ieee.std_logic_arith.all;

entity idct_system is
end idct_system;

architecture test of idct_system is
   component idct_with_ram
      port (    clk : in    std_logic;
             rd_addr : in    std_logic_vector(5 downto 0);
               start : in    std_logic;
             wr_addr : in    std_logic_vector(5 downto 0);
             wr_data : in    std_logic_vector(7 downto 0);
                done : out   std_logic;
             rd_data : out   std_logic_vector(31 downto
0) );
   end component;


    signal clk, start, done  : std_logic;
    signal wr_addr,rd_addr :
                    std_logic_vector(5 downto 0);
    signal wr_data : std_logic_vector(7 downto 0);
    signal rd_data : std_logic_vector(31 downto 0);
begin

    u1 : idct_with_ram port map(clk,rd_addr,start,
                    wr_addr,wr_data,done,rd_data);

    process
        variable clk_value : std_logic := ‘1’;
    begin

        clk_value := not clk_value;
        clk <= clk_value;
        wait for 2 ns;
    end process;


    start <= ‘0’ after 0 ns,
           ‘1’ after 5 ns,
```

```
        '0' after 10 ns;

    process
        type rf is array ( 0 to 7, 0 to 7 ) of integer;

        variable result : rf := (
        ( 88710930, -18305430, 22913790, -1664130,
14721150, 3968310, 10368810, 7296570 ),
        ( -5478165, 1130415, -1414995, 102765, -
909075, -245055, -640305, -450585 ),
        ( 68742135, -14184885, 17755905, -1289535,
11407425, 3075045, 8034795, 5654115 ),
        ( -5654880, 1166880, -1460640, 106080, -
938400, -252960, -660960, -465120 ),
        ( -7422030, 1531530, -1917090, 139230, -
1231650, -332010, -867510, -610470 ),
          ( -530145, 109395, -136935, 9945, -
87975, -23715, -61965, -43605 ),
          ( 25623675, -5287425, 6618525, -480675,
4252125, 1146225, 2994975, 2107575 ),
          ( 12723480, -2625480, 3286440, -238680,
2111400, 569160, 1487160, 1046520 )
                    );

variable data: integer;
    begin
        ----------------------------------------
        --  feed data -- sandwich case
        ----------------------------------------
        for i in 0 to 23 loop
           wr_addr <=conv_std_logic_vector(i,6);
           wr_data <= conv_std_logic_vector(255,8);
           wait for 0 ns;
        end loop;

        for i in 24 to 39 loop
             wr_addr <=conv_std_logic_vector(i,6);
             wr_data <= conv_std_logic_vector(0,8);
wait for 0 ns;
        end loop;

        for i in 40 to 63 loop
           wr_addr <=conv_std_logic_vector(i,6);
           wr_data <= conv_std_logic_vector(255,8);
wait for 0 ns;
        end loop;

        ---------------------------------
        --  handshaking
        ---------------------------------
        wait until start =  '1';


        ---------------------------------
        --  handshaking
        ---------------------------------
        wait until done = '1';
        wait until clk = '1';

        --------------------------------
        --  verify
        --------------------------------
        for i in 0 to 7 loop
        for j in 0 to 7 loop
            rd_addr <= conv_std_logic_vector(i,3) &
                    conv_std_logic_vector(j,3);
            wait for 4 ns;
           data := conv_integer(signed(rd_data));
            assert ( data = result( i, j ) ) report "error"
severity warning;
        end loop;
        end loop;


    end process;

end test;


configuration cfg_idct_system of idct_system is
  for test
  end for;
end cfg_idct_system;
```

9.2 IDCT with RAM.

-- includes the IDCT computation and the
-- input and output memories.

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_misc.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_components.all;

entity idct_with_ram is
    port (     clk : in    std_logic;
           rd_addr : in    std_logic_vector(5 downto 0);
             start : in    std_logic;
           wr_addr : in    std_logic_vector(5 downto 0);
           wr_data : in    std_logic_vector(7 downto 0);
              done : out   std_logic;
           rd_data : out   std_logic_vector(31 downto 0) );
end idct_with_ram;

architecture schematic of idct_with_ram is
```

```
  signal    data3 : std_logic_vector(31 downto 0);
  signal    addr3 : std_logic_vector(5 downto 0);
  signal    data2 : std_logic_vector(31 downto 0);
  signal    data1 : std_logic_vector(15 downto 0);
  signal    addr2 : std_logic_vector(5 downto 0);
  signal    addr1 : std_logic_vector(5 downto 0);

  component ram2
    port ( rd_addr : in    std_logic_vector(5 downto 0);
           rd_addr2 : in    std_logic_vector(5 downto 0);
           wr_addr : in    std_logic_vector(5 downto 0);
            wr_data : in    std_logic_vector(31 downto 0);
             rd_data : out   std_logic_vector(31 downto 0);
            rd_data2 : out   std_logic_vector(31 downto 0)
);
  end component;

  component ram
    port ( rd_addr : in    std_logic_vector(5 downto 0);
           wr_addr : in    std_logic_vector(5 downto 0);
            wr_data : in    std_logic_vector(7 downto 0);
              rd_data : out   std_logic_vector(15 downto 0)
);
  end component;

  component idct
    port (       clk : in    std_logic;
             rd_data : in    std_logic_vector(15 downto 0);
           rd_data2 : in    std_logic_vector(31 downto 0);
               start : in    std_logic;
               done : out   std_logic;
           rd_addr : out   std_logic_vector(5 downto 0);
          rd_addr2 : out   std_logic_vector(5 downto 0);
          wr_addr : out   std_logic_vector(5 downto 0);
           wr_data : out   std_logic_vector(31 downto 0) );
  end component;
  for all: idct use entity work.idct(schematic);

begin

  m2 : ram2
    port map ( rd_addr=>rd_addr, rd_addr2=>addr3,
               wr_addr=>addr2,wr_data=>data2,
               rd_data=>rd_data, rd_data2=>data3 );
  m1 : ram
   port map ( rd_addr=>addr1, wr_addr=>wr_addr,
              wr_data=>wr_data, rd_data=>data1 );
  idct_i : idct
    port map ( clk=>clk, rd_data=>data1, rd_data2=>
               data3, start=>start, done=>done,
               rd_addr=>addr1, rd_addr2=>addr3,
               wr_addr=>addr2, wr_data=>data2 );
end schematic;
```

9.3 IDCT computation.

-- includes the IDCT computation

9.3.1 IDCT original algorithm behavior.

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_misc.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_components.all;


entity idct is
    port (     clk : in    std_logic;
           rd_data : in    std_logic_vector(15 downto 0);
          rd_data2 : in    std_logic_vector(31 downto 0);
              start : in    std_logic;
              done : out   std_logic;
           rd_addr : out   std_logic_vector(5 downto 0);
          rd_addr2 : out   std_logic_vector(5 downto 0);
           wr_addr : out   std_logic_vector(5 downto 0);
           wr_data : out   std_logic_vector(31 downto 0) );
end idct;


architecture behavioral1 of idct is
begin
process
    type rf is array ( 0 to 7, 0 to 7 ) of integer;
    variable cosblock: rf;
    variable tempblock: rf;
    variable a, b, p, sum: integer;

begin

    -------------------------------
    --intialize parameter matrix
    -------------------------------
    cosblock := (
     ( 125,  122,    115,    103,    88,    69,    47,    24  ),
     ( 125,  103,    47,    -24,    -88,   -122,  -115,   -69  ),
     ( 125,  69,    -47,   -122,   -88,    24,    115,   103  ),
     ( 125,  24,    -115,   -69,    88,    103,   -47,   -122
),
     ( 125,  -24,   -115,   69,     88,    -103,  -47,   122  ),
     ( 125,  -69,   -47,    122,    -88,   -24,   115,   -103 ),
     ( 125,  -103,  47,     24,     -88,   122,   -115,   69  ),
     ( 125,  -122,  115,    -103,   88,    -69,   47,    -24  )
);
```

```
-------------------------------
-- handshaking
-------------------------------
wait until start = '1';
done <= '0';

--------------------------------
--matrix multiplication 1
--------------------------------
for i in 0 to 7 loop
for j in 0 to 7 loop
for k in 0 to 7 loop
 -- a := m1(i,k);
  rd_addr <= conv_std_logic_vector(i,3) &
  conv_std_logic_vector(k,3);
  wait for 4 ns;
  a := conv_integer(unsigned(rd_data));
  b := cosblock( j, k );
  p := a * b;

  if( k = 0 ) then
        sum := p;
  else
        sum := sum + p;
  end if;

  if( k = 7 ) then
       tempblock( i, j ) := sum;
  end if;
end loop;
end loop;
end loop;

--------------------------------
--matrix multiplication 2
--------------------------------
for i in 0 to 7 loop
for j in 0 to 7 loop
for k in 0 to 7 loop
      a := tempblock( k, j );
      b := cosblock( i, k );
      p := a * b;

      if( k = 0 ) then
            sum := p;
      else
            sum := sum + p;
      end if;
      if( k = 7 ) then
           -- m2(i,j) := sum;
            wr_addr <= conv_std_logic_vector(i,3) &
            conv_std_logic_vector(j,3);
            wr_data <= conv_std_logic_vector(sum,32);
            wait for 0 ns;
       end if;
      end loop;
      end loop;
      end loop;


      --------------------------------
      --handshaking
      --------------------------------
       wait until clk = '1';
       done <= '1';

end process;

end behavioral1;
```

9.3.2 IDCT Memory-Optimized Algorithm Behavior.

```
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.std_logic_misc.all;
   use ieee.std_logic_arith.all;
   use ieee.std_logic_components.all;



entity idct is
      port (      clk : in    std_logic;
             rd_data : in    std_logic_vector(15 downto 0);
             rd_data2 : in    std_logic_vector(31 downto 0);
               start : in    std_logic;
                done : out   std_logic;
             rd_addr : out   std_logic_vector(5 downto 0);
             rd_addr2 : out   std_logic_vector(5 downto 0);
             wr_addr : out   std_logic_vector(5 downto 0);
             wr_data : out   std_logic_vector(31 downto 0) );
end idct;


architecture behavioral2 of idct is
begin
process
     type rf is array(0 to 7,0 to 7) of integer;

     variable i,j,k: integer;
      variable a,b,c,d,p,s,sum,temp: integer;
      variable cos : rf;
begin
      cos := (
        ( 125,  122,    115,    103,    88,     69,     47,     24  ),
```

```
    ( 125,  103,   47,   -24,   -88,   -122,  -115,  -69  ),
    ( 125,  69,    -47,  -122,  -88,   24,    115,   103  ),
    ( 125,  24,    -115, -69,   88,    103,   -47,   -122
),
    ( 125,  -24,   -115, 69,    88,    -103,  -47,   122  ),
    ( 125,  -69,   -47,  122,   -88,   -24,   115,   -103 ),
    ( 125,  -103,  47,   24,    -88,   122,   -115,  69   ),
    ( 125,  -122,  115,  -103,  88,    -69,   47,    -24  )
);

    -------------------------------
    -- handshaking
    -------------------------------
    wait until start = '1';
    done <= '0';

    -------------------------------
    -- matrix 1 and matrix 2
    -------------------------------
    for i in 0 to 7 loop
    for j in 0 to 7 loop
    for k in 0 to 7 loop
        -- a := m1(i,k);-- reading m1
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);

        wait for 4 ns;
        a := conv_integer(unsigned(rd_data));
        b := cos(j,k);
        p := a * b;
        if (k=0) then
            s := p;
        else
            s := s + p;
        end if;
      end loop;
      temp := s;
      for k in 0 to 7 loop
          if (i/=0) then
              -- c := m2(k,j);
              rd_addr2 <= conv_std_logic_vector(k,3) &
conv_std_logic_vector(j,3);

              wait for 4 ns;
              c := conv_integer(signed(rd_data2));
          end if;
          d := cos(k,i);
          p := d * temp;
          if (i=0) then
              sum := p;
          else
          sum := c + p;
        end if;
        -- m2(k,j) := sum;
        wr_addr <= conv_std_logic_vector(k,3) &
conv_std_logic_vector(j,3);

        wr_data <= conv_std_logic_vector(sum, 32);
        wait for 4 ns;
    end loop;
    end loop;
    end loop;

    --------------------------------
    --handshaking
    --------------------------------
    wait until clk = '1';
    done <= '1';

end process;
end behavioral2;
```

9.3.3 IDCT Original Algorithm Behavior of Pipelined Design.

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_misc.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_components.all;

-- includes the controller and datapath processes for
matrix multiplication 1 and 2.

entity idct is
    port (    clk : in    std_logic;
            rd_data : in    std_logic_vector(7 downto 0);
            rd_data2 : in    std_logic_vector(31 downto 0);
              start : in    std_logic;
               done : out   std_logic;
            rd_addr : out   std_logic_vector(5 downto 0);
            rd_addr2 : out   std_logic_vector(5 downto 0);
            wr_addr : out   std_logic_vector(5 downto 0);
            wr_data : out   std_logic_vector(31 downto 0) );
end idct;

architecture behavioral1_pipe15 of idct is
  type rf is array ( 0 to 7, 0 to 7 ) of integer;
  signal cosblock: rf;
  signal tempblock1,tempblock2: rf;
  signal state1, state2: integer:=0;
  signal a, b, p1,p2,p3,p4, sum: integer;
   signal a_2, b_2, p1_2,p2_2,p3_2,p4_2, sum_2: inte-
ger;
```

```
  signal count, count2 : unsigned ( 9 downto 0 ) :=
"0000000000";
  signal done1, done2 : std_logic;
  signal memswap : std_logic := '0';


begin
    ------------------------------------
    --intialize parameter matrix
    ------------------------------------
    cosblock <= (
     ( 125, 122,   115,   103,   88,    69,    47,    24 ),
     ( 125, 103,   47,    -24,   -88,   -122,  -115,  -69 ),
     ( 125, 69,    -47,   -122,  -88,   24,    115,   103 ),
     ( 125, 24,    -115,  -69,   88,    103,   -47,   -122
),
     ( 125, -24,   -115,  69,    88,    -103,  -47,   122 ),
     ( 125, -69,   -47,   122,   -88,   -24,   115,   -103 ),
     ( 125, -103,  47,    24,    -88,   122,   -115,  69 ),
     ( 125, -122,  115,   -103,  88,    -69,   47,    -24 )
);

    ------------------------------------
    --matrix multiplication 1
    ------------------------------------

    -- controller process for matrix multiplication 1

    process(clk)
        variable varcount : unsigned ( 9 downto 0 ) :=
"0000000000";
       variable curstate1, nextstate1 : integer := 0;
       variable vardone1 : std_logic;
    begin
      if (clk='1' and clk'event) then
        case curstate1 is
            when 0 =>
                if start = '1' then
                    nextstate1 := 1;
                    varcount := "0000000000";
                else
                    nextstate1 := 0;
                end if;
                vardone1 := '0';
            when 1 =>
                nextstate1 := 2;
            when 2 =>
                nextstate1 := 3;
            when 3 =>
                nextstate1 := 4;
            when 4 =>
                nextstate1 := 5;
            when 5 =>
                nextstate1 := 6;
            when 6 =>
                nextstate1 := 7;
            when 7 =>
                if ( varcount < 512 ) then
                    nextstate1 := 7;
                else
                    nextstate1 := 8;
                end if;
            when 8 =>
                nextstate1 := 9;
            when 9 =>
                nextstate1 := 10;
            when 10 =>
                nextstate1 := 11;
            when 11 =>
                nextstate1 := 12;
            when 12 =>
                nextstate1 := 13;
            when 13 =>
                nextstate1 := 0;
                vardone1 := '1';
            when others =>
        end case;
        state1 <= curstate1;
        count <= varcount;
        if( curstate1 > 0) then
            varcount := varcount + 1;
        end if;
        curstate1 := nextstate1;
        done1 <= vardone1;
      end if;
end process;


    -- datapath process for matrix multiplication 1
    process
        variable s1 : integer := 0;
        variable i, k, j, i2, k2, j2 : integer range 0 to 7;
      variable varcount, varcount2 : unsigned ( 9 downto
0 ) := "0000000000";
    begin

    -- getting the correct indexes for memory fetch
    -- and memory write

        wait until clk'event and clk = '1';
        s1 := state1;
        varcount := count;
        varcount2 := varcount - 6;
        i := conv_integer( varcount( 8 downto 6 ) );
        j := conv_integer( varcount( 5 downto 3 ) );
        k := conv_integer( varcount( 2 downto 0 ) );
        i2 := conv_integer( varcount2( 8 downto 6 ) );
        j2 := conv_integer( varcount2( 5 downto 3 ) );
```

```
k2 := conv_integer( varcount2( 2 downto 0 ) );

case s1 is
    when 0 =>

    when 1 =>
        -- executing pipe stages 1
        -- a := m1(i,k);
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
       wait for 4 ns;
        a <= conv_integer(unsigned(rd_data));
        b <= cosblock( j, k );

    when 2 =>
        -- executing pipe stages 1,2
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
      wait for 4 ns;
      a <= conv_integer(unsigned(rd_data));
      b <= cosblock( j, k );
      p1 <= a * b;

    when 3 =>
        -- executing pipe stages 1,2,3
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
      wait for 4 ns;
      a <= conv_integer(unsigned(rd_data));
      b <= cosblock( j, k );
      p1 <= a * b;
      p2 <= p1;

    when 4 =>
        -- executing pipe stages 1,2,3,4
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
      wait for 4 ns;
      a <= conv_integer(unsigned(rd_data));
      b <= cosblock( j, k );
      p1 <= a * b;
      p2 <= p1;
      p3 <= p2;

    when 5 =>
        -- executing pipe stages 1,2,3,4,5
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
      wait for 4 ns;
      a <= conv_integer(unsigned(rd_data));
      b <= cosblock( j, k );
      p1 <= a * b;
      p2 <= p1;
      p3 <= p2;
      p4 <= p3;
        -- executing pipe stages 1,2,3,4,5,6
    when 6 =>
        rd_addr <= conv_std_logic_vector(i,3) &
        conv_std_logic_vector(k,3);
      wait for 4 ns;
      a <= conv_integer(unsigned(rd_data));
      b <= cosblock( j, k );
      p1 <= a * b;
      p2 <= p1;
      p3 <= p2;
      p4 <= p3;
      if( k = 5 ) then
          sum <= p4;
      else
          sum <= sum + p4;
      end if;

      when 7 =>
        -- executing pipe stages 1,2,3,4,5,6,7
          rd_addr <= conv_std_logic_vector(i,3)
& conv_std_logic_vector(k,3);
        wait for 4 ns;
          a <= conv_integer(unsigned(rd_data));
        b <= cosblock( j, k );
        p1 <= a * b;
        p2 <= p1;
        p3 <= p2;
        p4 <= p3;
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

    when 8 =>
        -- executing pipe stages 2,3,4,5,6,7
        p1 <= a * b;
        p2 <= p1;
        p3 <= p2;
        p4 <= p3;
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
```

```
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

    when 9 =>
        -- executing pipe stages 3,4,5,6,7
        p2 <= p1;
        p3 <= p2;
        p4 <= p3;
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

    when 10 =>
        -- executing pipe stages 4,5,6,7
        p3 <= p2;
        p4 <= p3;
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

    when 11 =>
        -- executing pipe stages 5,6,7
        p4 <= p3;
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

        when 12 =>
        -- executing pipe stages 6,7
        if( k = 5 ) then
            sum <= p4;
        else
            sum <= sum + p4;
        end if;
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;

    when 13 =>
        -- executing pipe stages 7
        if( k = 5 and varcount /= 5 ) then
            if(memswap = '0') then
                tempblock1( i2, j2 ) <= sum;
            elsif(memswap = '1') then
                tempblock2( i2, j2 ) <= sum;
            end if;
        end if;
        when others =>
    end case;
end process;


-- process for controlling the temp memory
-- swapping
process
     variable countnum : unsigned ( 9 downto 0 ) :=
"0000000000";
    variable swap : std_logic := '0';
begin
    wait on count;
    countnum := count;
    if (countnum = 518) then
        swap := not swap;
    end if;
    memswap <= swap;
end process;


-------------------------------------
--matrix multiplication 2
-------------------------------------

-- controller process for matrix multiplication 2
```

```
-- synchronized with matrix multiplication 1 by
-- waiting the signal Done1
process(clk)
      variable varcount : unsigned ( 9 downto 0 ) :=
"0000000000";
    variable curstate2, nextstate2 : integer := 0;
    variable vardone2 : std_logic;
begin
    if (clk='1' and clk'event) then

      case curstate2 is
        when 0 =>
            if done1 = '1' then
                nextstate2 := 1;
                varcount := "0000000000";
                vardone2 := '0';
            else
                nextstate2 := 0;
            end if;
            vardone2 := '0';
        when 1 =>
            nextstate2 := 2;
        when 2 =>
            nextstate2 := 3;
        when 3 =>
            nextstate2 := 4;
        when 4 =>
            nextstate2 := 5;
        when 5 =>
            nextstate2 := 6;
        when 6 =>
            nextstate2 := 7;
        when 7 =>
            if ( varcount < 512 ) then
                nextstate2 := 7;
            else
                nextstate2 := 8;
            end if;
        when 8 =>
            nextstate2 := 9;
        when 9 =>
            nextstate2 := 10;
        when 10 =>
            nextstate2 := 11;
        when 11 =>
            nextstate2 := 12;
        when 12 =>
            nextstate2 := 13;
        when 13 =>
            nextstate2 := 0;
            vardone2 := '1';
        when others =>
      end case;
      state2 <= curstate2;
          count2 <= varcount;
          if( curstate2 > 0) then
              varcount := varcount + 1;
          end if;
          curstate2 := nextstate2;
          done2 <= vardone2;
        end if;
    end process;


    -- datapath process for matrix multiplication 2
    -- pipelining execution is the same as matrix
    -- multiplication1

    process
        variable s1 : integer := 0;
        variable i, k, j, i2, k2, j2 : integer range 0 to 7;
      variable varcount, varcount2 : unsigned ( 9 downto
0 ) := "0000000000";
    begin
        wait until clk'event and clk = '1';
        s1 := state2;
        varcount := count2;
        varcount2 := varcount - 6;
        i := conv_integer( varcount( 8 downto 6 ) );
        j := conv_integer( varcount( 5 downto 3 ) );
        k := conv_integer( varcount( 2 downto 0 ) );
        i2 := conv_integer( varcount2( 8 downto 6 ) );
        j2 := conv_integer( varcount2( 5 downto 3 ) );
        k2 := conv_integer( varcount2( 2 downto 0 ) );

       case s1 is
           when 0 =>

           when 1 =>
               if (memswap = '1') then
                   a_2 <= tempblock1( k, j );
               elsif (memswap = '0') then
                   a_2 <= tempblock2( k, j );
               end if;
               b_2 <= cosblock( i, k );

           when 2 =>
               if (memswap = '1') then
                   a_2 <= tempblock1( k, j );
               elsif (memswap = '0') then
                   a_2 <= tempblock2( k, j );
               end if;
               b_2 <= cosblock( i, k );
               p1_2 <= a_2 * b_2;

           when 3 =>
               if (memswap = '1') then
                   a_2 <= tempblock1( k, j );
```

```
    elsif (memswap = '0') then
        a_2 <= tempblock2( k, j );
    end if;
    b_2 <= cosblock( i, k );
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;

when 4 =>
    if (memswap = '1') then
        a_2 <= tempblock1( k, j );
    elsif (memswap = '0') then
        a_2 <= tempblock2( k, j );
    end if;
    b_2 <= cosblock( i, k );
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;
    p3_2 <= p2_2;

when 5 =>
    if (memswap = '1') then
        a_2 <= tempblock1( k, j );
    elsif (memswap = '0') then
        a_2 <= tempblock2( k, j );
    end if;
    b_2 <= cosblock( i, k );
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;
    p3_2 <= p2_2;
    p4_2 <= p3_2;

when 6 =>
    if (memswap = '1') then
        a_2 <= tempblock1( k, j );
    elsif (memswap = '0') then
        a_2 <= tempblock2( k, j );
    end if;
    b_2 <= cosblock( i, k );
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;
    p3_2 <= p2_2;
    p4_2 <= p3_2;
    if( k = 5 ) then
        sum_2 <= p4_2;
    else
        sum_2 <= sum_2 + p4_2;
    end if;

when 7 =>
    if (memswap = '1') then
        a_2 <= tempblock1( k, j );
    elsif (memswap = '0') then
        a_2 <= tempblock2( k, j );
    end if;
    b_2 <= cosblock( i, k );
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;
    p3_2 <= p2_2;
    p4_2 <= p3_2;
    if( k = 5 ) then
        sum_2 <= p4_2;
    else
        sum_2 <= sum_2 + p4_2;
    end if;
    if( k = 5 and varcount /= 5 ) then
        wr_addr <= conv_std_logic_vector(i2,3) & conv_std_logic_vector(j2,3);
        wr_data <= conv_std_logic_vector(sum_2,32);
        wait for 4 ns;
    end if;

when 8 =>
    p1_2 <= a_2 * b_2;
    p2_2 <= p1_2;
    p3_2 <= p2_2;
    p4_2 <= p3_2;
    if( k = 5 ) then
        sum_2 <= p4_2;
    else
        sum_2 <= sum_2 + p4_2;
    end if;
    if( k = 5 and varcount /= 5 ) then
        wr_addr <= conv_std_logic_vector(i2,3) & conv_std_logic_vector(j2,3);
        wr_data <= conv_std_logic_vector(sum_2,32);
        wait for 4 ns;
    end if;
when 9 =>

    p2_2 <= p1_2;
    p3_2 <= p2_2;
    p4_2 <= p3_2;
    if( k = 5 ) then
        sum_2 <= p4_2;
    else
        sum_2 <= sum_2 + p4_2;
    end if;
    if( k = 5 and varcount /= 5 ) then
        wr_addr <= conv_std_logic_vector(i2,3) & conv_std_logic_vector(j2,3);
        wr_data <= conv_std_logic_vector(sum_2,32);
        wait for 4 ns;
    end if;
```

```
          when 10 =>
               p3_2 <= p2_2;
               p4_2 <= p3_2;
               if( k = 5 ) then
                    sum_2 <= p4_2;
               else
                    sum_2 <= sum_2 + p4_2;
               end if;
               if( k = 5 and varcount /= 5 ) then
                wr_addr <= conv_std_logic_vector(i2,3)
& conv_std_logic_vector(j2,3);
                                                  wr_data  <=
conv_std_logic_vector(sum_2,32);
                    wait for 4 ns;
               end if;

          when 11 =>
               p4_2 <= p3_2;
               if( k = 5 ) then
                    sum_2 <= p4_2;
               else
                    sum_2 <= sum_2 + p4_2;
               end if;
               if( k = 5 and varcount /= 5 ) then
                                                  wr_addr  <=
conv_std_logic_vector(i2,3)                              &
conv_std_logic_vector(j2,3);
                                                  wr_data  <=
conv_std_logic_vector(sum_2,32);
                    wait for 4 ns;
               end if;

          when 12 =>
               if( k = 5 ) then
                    sum_2 <= p4_2;
               else
                    sum_2 <= sum_2 + p4_2;
               end if;
               if( k = 5 and varcount /= 5 ) then
                                                  wr_addr  <=
conv_std_logic_vector(i2,3)                              &
conv_std_logic_vector(j2,3);
                                                  wr_data  <=
conv_std_logic_vector(sum_2,32);
                    wait for 4 ns;
               end if;

          when 13 =>
               if( k = 5 and varcount /= 5 ) then
                                                  wr_addr  <=
conv_std_logic_vector(i2,3)                              &
conv_std_logic_vector(j2,3);
                                                  wr_data  <=
conv_std_logic_vector(sum_2,32);
                         wait for 4 ns;
                    end if;
               when others =>
          end case;
     end process;

     done <= done2;

end behavioral1_pipe15;
```

9.3.4 IDCT Optimized Algorithm Behavior of Pipelined Design.

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_misc.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_components.all;



entity idct is
     port (      clk : in    std_logic;
            rd_data : in    std_logic_vector(15 downto 0);
           rd_data2 : in    std_logic_vector(31 downto 0);
              start : in    std_logic;
              done : out   std_logic;
           rd_addr : out   std_logic_vector(5 downto 0);
           rd_addr2 : out   std_logic_vector(5 downto 0);
           wr_addr : out   std_logic_vector(5 downto 0);
           wr_data : out   std_logic_vector(31 downto 0) );
end idct;


architecture behavioral2_pipe2 of idct is
     type rf is array(0 to 7,0 to 7) of integer;

     signal a,b,c,c1,c2,c3,c4,d,p1,p2,p3,p4,p5,p6,p7,p8,
            s,sum,temp,temp1: integer:=0;
     signal i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,
            j1,j2,j3,j4,j5,j6,j7,j8,j9,j10,j11,j12,j13,
             k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13:
              integer;
     signal cos : rf;
     constant period: time := 4 ns;
          signal          en:          std_logic_vector(0to
12):=”0000000000000”;
     signal start1,start2: std_logic := ‘0’;
```

```
    signal state1,state2,state: integer:=0;
begin
    cos <= (
     ( 125,  122,    115,    103,    88,     69,     47,     24  ),
     ( 125,  103,    47,     -24,    -88,    -122,   -115,   -69 ),
     ( 125,  69,     -47,    -122,   -88,    24,     115,    103 ),
     ( 125,  24,     -115,   -69,    88,     103,    -47,    -122
),
     ( 125,  -24,    -115,   69,     88,     -103,   -47,    122 ),
     ( 125,  -69,    -47,    122,    -88,    -24,    115,    -103 ),
     ( 125,  -103,   47,     24,     -88,    122,    -115,   69 ),
     ( 125,  -122,   115,    -103,   88,     -69,    47,     -24 )
);

    -- top level controller (controlling the two pipeline
    -- controllers)

    process(clk)
        variable l:integer:=0;
    begin
        if clk='1' then
          case state is
            when 0 =>
                if start='1' then
                    state <= 1;
                    start1 <='1';
                    start2 <='0';
                    l := 1;
                else
                    start1 <= '0';
                    start2 <= '0';
                    l := 0;
                end if;

            when 1 =>
                if l<13 then
                    start1 <= '0';
                    start2 <= '0';
                    l := l + 1;
                else
                    start1 <='0';
                    start2 <='1';
                    state <= 0;
                end if;

            when others =>
          end case;
        end if;
    end process;


    -- pipeline contrller 1
    process(clk)
        variable l: integer := 0;
    begin
        if clk='1' then
          case state1 is
            when 0 =>
                if start1 = '1' then
                    state1 <= 1;
                    en(5) <= en(4) after period;
                    en(4) <= en(3) after period;
                    en(3) <= en(2) after period;
                    en(2) <= en(1) after period;
                    en(1) <= en(0) after period;
                    en(0) <= '1' after period;
                    i1 <= l / 64 after period;
                    j1 <= (l mod 64) / 8 after period;
                    k1 <= l mod 8 after period;
                    l := l+1;
                else
                    en(5) <= en(4) after period;
                    en(4) <= en(3) after period;
                    en(3) <= en(2) after period;
                    en(2) <= en(1) after period;
                    en(1) <= en(0) after period;
                    en(0) <= '0' after period;
                    l := 0;
                    i1 <= l / 64 after period;
                    j1 <= (l mod 64) / 8 after period;
                    k1 <= l mod 8 after period;
                    end if;

            when 1 =>
                if l<512 then
                    en(5) <= en(4) after period;
                    en(4) <= en(3) after period;
                    en(3) <= en(2) after period;
                    en(2) <= en(1) after period;
                    en(1) <= en(0) after period;
                    en(0) <= '1' after period;
                    i1 <= l / 64 after period;
                    j1 <= (l mod 64) / 8 after period;
                    k1 <= l mod 8 after period;
                    l := l+1;
                else
                    en(5) <= en(4) after period;
                    en(4) <= en(3) after period;
                    en(3) <= en(2) after period;
                    en(2) <= en(1) after period;
                    en(1) <= en(0) after period;
                    en(0) <= '0' after period;
                    l := 0;
                    i1 <= l / 64 after period;
                    j1 <= (l mod 64) / 8 after period;
                    k1 <= l mod 8 after period;
                    state1 <= 0;
                end if;
```

```
            when others =>
          end case;
       end if;
end process;

-- pipeline controller 2
process(clk)
     variable l: integer := 0;
begin
     if clk = '1' then
        case state2 is
          when 0 =>
               if start2 = '1' then
                    state2 <= 1;
                    en(12) <= en(11) after period;
                    en(11) <= en(10) after period;
                    en(10) <= en(9) after period;
                    en(9) <= en(8) after period;
                    en(8) <= en(7) after period;
                    en(7) <= en(6) after period;
                    en(6) <= '1' after period;
                    i7 <= l / 64 after period;
                    j7 <= (l mod 64) / 8 after period;
                    k7 <= l mod 8 after period;
                    l := l+1;
               else
                    en(12) <= en(11) after period;
                    en(11) <= en(10) after period;
                    en(10) <= en(9) after period;
                    en(9) <= en(8) after period;
                    en(8) <= en(7) after period;
                    en(7) <= en(6) after period;
                    en(6) <= '0' after period;
                    l := 0;
                    i7 <= l / 64 after period;
                    j7 <= (l mod 64) / 8 after period;
                    k7 <= l mod 8 after period;
               end if;

          when 1 =>
               if l<512 then
                    en(12) <= en(11) after period;
                    en(11) <= en(10) after period;
                    en(10) <= en(9) after period;
                    en(9) <= en(8) after period;
                    en(8) <= en(7) after period;
                    en(7) <= en(6) after period;
                    en(6) <= '1' after period;
                    i7 <= l / 64 after period;
                    j7 <= (l mod 64) / 8 after period;
                    k7 <= l mod 8 after period;
                    l := l+1;
               else
                    en(12) <= en(11) after period;
                    en(11) <= en(10) after period;
                    en(10) <= en(9) after period;
                    en(9) <= en(8) after period;
                    en(8) <= en(7) after period;
                    en(7) <= en(6) after period;
                    en(6) <= '0' after period;
                    l := 0;
                    i7 <= l / 64 after period;
                    j7 <= (l mod 64) / 8 after period;
                    k7 <= l mod 8 after period;
                    state2 <= 0;
               end if;

            when others =>
          end case;
       end if;
end process;



----------------
-- matrix 1
----------------


process
begin
     wait on clk;
     if (clk='1') and (en(0) ='1') then
          i2 <= i1 after period;
          j2 <= j1 after period;
          k2 <= k1 after period;
          b <= cos(j1,k1) after period;
          -- a <= m1(i,k);-- reading m1
          rd_addr <= conv_std_logic_vector(i1,3) &
conv_std_logic_vector(k1,3);

          wait for period;
          a <= conv_integer(unsigned(rd_data));
     end if;
end process;

process(clk)
begin
     if (clk='1') and (en(1) ='1') then
          p1 <= a * b after period;
          i3 <= i2 after period;
          j3 <= j2 after period;
          k3 <= k2 after period;
     end if;
end process;

process(clk)
begin
     if (clk='1') and (en(2) ='1') then
          p2 <= p1 after period;
```

```
            i4 <= i3 after period;
            j4 <= j3 after period;
            k4 <= k3 after period;
        end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(3) ='1') then
            p3 <= p2 after period;
            i5 <= i4 after period;
            j5 <= j4 after period;
            k5 <= k4 after period;
         end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(4) ='1') then
            p4 <= p3 after period;
            i6 <= i5 after period;
            j6 <= j5 after period;
            k6 <= k5 after period;
        end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(5) ='1') then
            if (k6=0) then
                s <= p4 after period;
            else
                s <= s + p4 after period;
                if (k6=7) then
                    temp <= s + p4 after period;
                end if;
            end if;
        end if;
    end process;

    --------------------
    -- matrix 2
    --------------------

    process
    begin
        wait on clk;
        if (clk='1') and (en(6) ='1') then
            i8 <= i7 after period;
            j8 <= j7 after period;
            k8 <= k7 after period;
            temp1 <= temp after period;
            d <= cos(k7,i7) after period;
            if (i7/=0) then
                -- c <= m2(k,j);
                rd_addr2 <= conv_std_logic_vector(k7,3)
& conv_std_logic_vector(j7,3);

                wait for period;
                c <= conv_integer(signed(rd_data2));
            end if;
        end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(7) ='1') then
            p5 <= d * temp1 after period;
            i9 <= i8 after period;
            j9 <= j8 after period;
            k9 <= k8 after period;
            c1 <= c after period;
        end if;
    end process;

    process(clk)
    begin
            if (clk='1') and (en(8) ='1') then
                p6 <= p5 after period;
                i10 <= i9 after period;
                j10 <= j9 after period;
                k10 <= k9 after period;
                c2 <= c1 after period;
            end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(9) ='1') then
            p7 <= p6 after period;
            i11 <= i10 after period;
            j11 <= j10 after period;
            k11 <= k10 after period;
            c3 <= c2 after period;
        end if;
    end process;

    process(clk)
    begin
        if (clk='1') and (en(10) ='1') then
            p8 <= p7 after period;
            i12 <= i11 after period;
            j12 <= j11 after period;
            k12 <= k11 after period;
            c4 <= c3 after period;
        end if;
    end process;
```

```
      process(clk)
      begin
          if (clk='1') and (en(11) ='1') then
              if (i12=0) then
                  sum <= p8 after period;
              else
                  sum <= c4 + p8 after period;
              end if;
              i13 <= i12 after period;
              j13 <= j12 after period;
              k13 <= k12 after period;
          end if;
      end process;

      process(clk)
      begin
          if (clk='1') and (start='1') then done <= '0';
          end if;
          if (clk='1') and (en(12) ='1') then
              -- m2(k,j) <= sum;
              wr_addr <= conv_std_logic_vector(k13,3) &
conv_std_logic_vector(j13,3) after period;

               wr_data <= conv_std_logic_vector(sum, 32)
after period;
          if (i13=7) and (j13=7) and (k13=7) then
                  done <='1' after 2*period, '0' after 3 *
period;
          end if;
        end if;
      end process;
end behavioral2_pipe2;
```