ICS 155B Spring 1998
Lab Assignment 2: Performance Optimizations (I)

Due Date: May 4, 1998 by noon

Handed out: April 20, 1998

1 Introduction

In this lab, you'll be developing a VHDL model for ASM chart of the DCT chip and explore several design alternatives.

In Lab 1, we have focused on the following issues:

1. The timing protocol of the DCT chip – how the chip communicates with the outside world.
2. The test bench – how we verify the VHDL model.
3. The behavioral model – what the DCT computes.

Note that in Lab 1 we only specified the behavior of the chip without any commitment to chip implementation. In order for the chip to be implemented in hardware, we have to add the notion of time, that is, we have to specify *when* each operation is executed.

The "WHEN" information is derived from two constraints, namely, data dependencies and resource constraints. Because of the data dependencies inherent in the description, Operations must be performed in series. Because of the resource constraints, the operations that use the same resources must be serialized, that is, they can never be executed at the same time. The determination of the "WHEN" information is generally referred as *scheduling*.

After scheduling, we can construct an ASM chart, which you have learned from ICS 151 and covered in more detail in ICS 152. For those of you who are not familiar with ASM charts, we are giving a short refreshment course below.

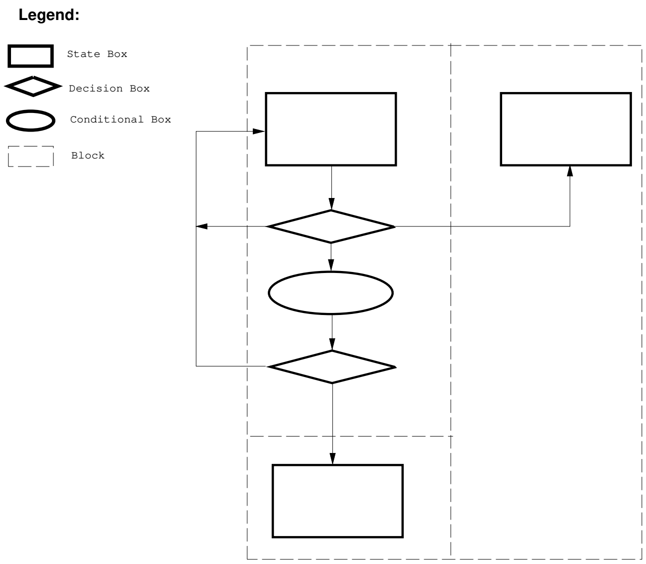

Figure 1: An ASM Chart Example

There are 4 key constructs in ASM chart (Figure 1), namely the *state box*, *decision box*, *conditional box*, and *block*. RT operations can be specified in both the state and conditional boxes. The decision boxes help to specify the conditions under which state transitions and RT operations inside the conditional boxes occur. Note that an ASM chart can be naturally partitioned into a set of blocks, in which each block contains exactly one state box, a tree of decision and conditional boxes, and several exit paths which lead to other blocks.

In this lab, we will explore several design alternatives starting with the most sequential design. The VHDL code for the sequential design is also provided below in order to illustrate how to write VHDL models for ASM charts. You are responsible for improving performance of the present ASM chart, developing a new high-performance chart and its VHDL model.

2 Sequential Design

As shown in Figure 2, mapping from behavioral model to the most sequential design is pretty straightforward. Basically, we allocate 1 clock cycle for each set of operations that do not dependent on each other.

If you look at the source listing, you can see VHDL process used to model the ASM chart. The behaviors associated with each block in ASM chart are grouped into a state. There is a variable "State" to keep track of where the current state is. The process is activated upon every rising edge of the clock. Only those operations in the current state are executed in the present clock cycle.

3 More Optimizations

As we discussed in class, several optimization techniques can be applied to the sequential design in order to speed it up:

1. loop unrolling.
2. chaining
3. multicycling

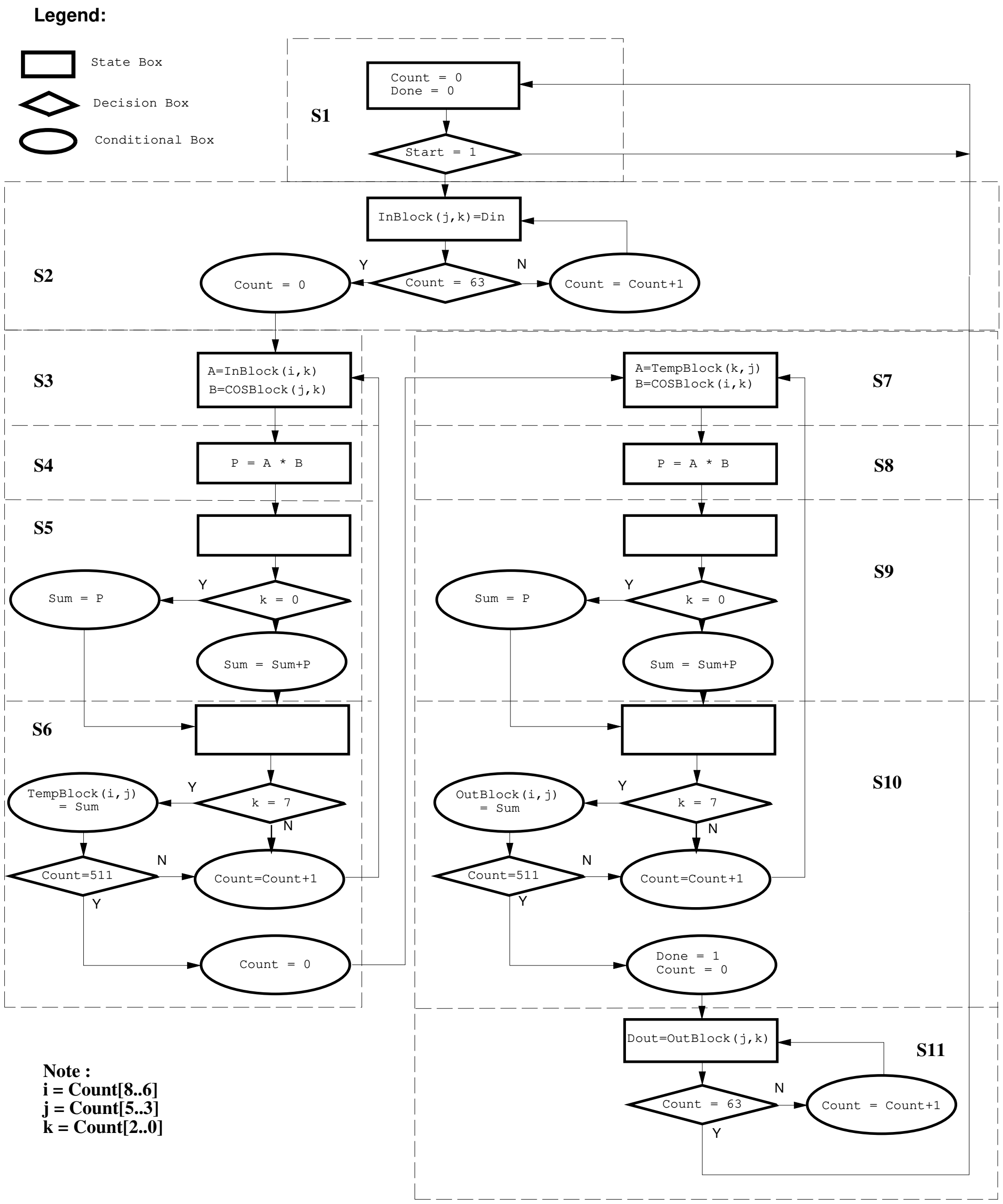

Figure 2: ASM Chart for the Sequential Design

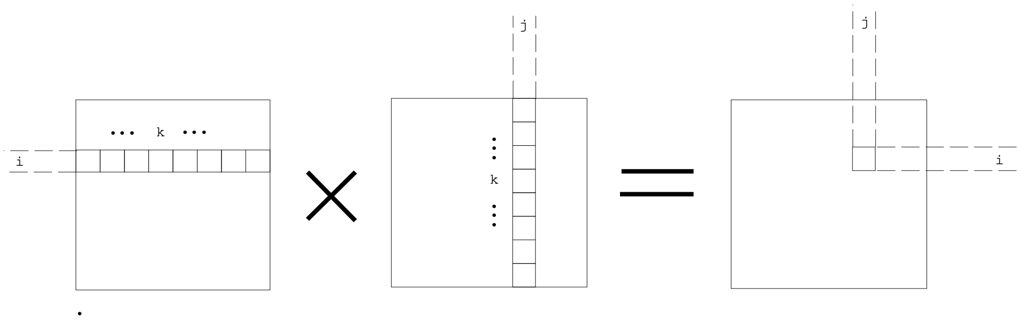

Figure 3: Illustration of Indexing

4 Assignment

1. Read this handout and copy of the lecture notes/slides (available on the class web page).
2. Study the ASM chart and the VHDL code for sequential design.
3. Redo the ASM chart by applying the following optimization techniques to obtain performance improvement: (1) loop unrolling; (2) chaining; (3) multicycling.
4. Derive the VHDL model for your ASM chart.
5. Estimate the performance in terms of number of clock cycles.
6. Your report: The write-up should include (1) Your ASM chart and a brief interpretation of the VHDL model; (2) Your complete source listing; (3) Your estimated performance and the waveform that can verify your estimated performance.
7. Give an estimate of the number of hours you spent inside and outside the lab, including the write-up.
8. Put your name and student ID on the submissions.

5 Source Listing

5.1 sequential.vhd

```
---------------------------------------------------------------------
--      File            :       sequential.vhd
--      Description     :       ASM model for a dct chip
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity dct is
        port (
                Clk :           in STD_LOGIC;
                Start :         in STD_LOGIC;
                Din :           in INTEGER;
                Done :          out STD_LOGIC;
                Dout :          out INTEGER
                );
end dct;


architecture sequential of dct is
begin

        process( Clk )
                -------------------------------------------------------
```

```
        --      S1 -- Idle State
        --      S2 -- Load Data
        --      S3-S7 -- Loop of First Matrix Multiplication
        --      S8-S11 -- Loop of Second Matrix Multiplication
        --      S12 -- Output Data
        -------------------------------------------------------
        type STATE_VALUE is ( S1, S2, S3, S4, S5, S6,
                S7, S8, S9, S10, S11 );

        type RF is array ( 0 to 7, 0 to 7 ) of INTEGER;

        variable InBlock        : RF;
        variable TempBlock      : RF;
        variable OutBlock       : RF;
        variable COSBlock       : RF := (
                ( 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 )
                );

        variable        A, B, P, Sum : INTEGER;
        variable        Count : UNSIGNED ( 8 downto 0 ) := "000000000";
        variable        i, j, k : INTEGER;
        variable        State : STATE_VALUE := S1;
        variable        VarDone : STD_LOGIC;
begin

if( Clk'event and Clk = '1' ) Then
        i := CONV_INTEGER( Count( 8 downto 6 ) );
        j := CONV_INTEGER( Count( 5 downto 3 ) );
        k := CONV_INTEGER( Count( 2 downto 0 ) );

        case State is
                --------------------------------------------
                --      State S2
                --------------------------------------------
                when S1 =>
                        Count := "000000000";
                        VarDone := '0';
                        if( Start = '1' ) then
                                State := S2;
                        else
                                State := S1;
                        end if;

                --------------------------------------------
                --      State S2
                --------------------------------------------
                when S2 =>
```

```
        if( Count = 63 ) then
                Count := "000000000";
                State := S3;
        else
                Count := Count + 1;
                State := S2;
        end if;
        InBlock( j, k ) :=  Din;

--------------------------------------------
--      State S3
--------------------------------------------
when S3 =>
        A := InBlock(i, k);
        B := COSBlock(j, k);
        State := S4;


--------------------------------------------
--      State S4
--------------------------------------------
when S4 =>
        P := A * B;
        State := S5;

--------------------------------------------
--      State S5
--------------------------------------------
when S5 =>
        if( k = 0 ) then
                Sum := P;
        else
                Sum := P + Sum;
        end if;
        State :=  S6;

--------------------------------------------
--      State S6
--------------------------------------------
when S6 =>
        if( k = 7 ) then
                TempBlock(i, j) := Sum;
        end if;
        if( Count = 511 ) then
                State := S7;
                Count := "000000000";
        else
                State := S3;
                Count := Count + 1;
        end if;


--------------------------------------------
--      State S7
```

```
--------------------------------------------
when S7 =>
        A := TempBlock(k, j);
        B := COSBlock(i, k);
        State := S8;


--------------------------------------------
--      State S8
--------------------------------------------
when S8 =>
        P := A * B;
        State := S9;

--------------------------------------------
--      State S9
--------------------------------------------
when S9 =>
        if( k = 0 ) then
                Sum := P;
        else
                Sum := P + Sum;
        end if;
        State :=  S10;

--------------------------------------------
--      State S10
--------------------------------------------
when S10 =>
        if( k = 7 ) then
                OutBlock(i, j) := Sum;
        end if;
        if( Count = 511 ) then
                State := S11;
                Count := "000000000";
                VarDone := '1';
        else
                State := S7;
                Count := Count + 1;
        end if;

--------------------------------------------
--      State S11
--------------------------------------------
when S11 =>
        VarDone := '0';
        if( Count = 63 ) then
                Count := "000000000";
                State := S1;
        else
                Count := Count + 1;
                State := S11;
        end if;
        Dout <=  OutBlock( j, k );
```

```
                end case;
                Done <= VarDone;
        end if;
        end process;
end sequential;
```

5.2 clock.vhd

```
--------------------------------------------------------------
--      File    :       clock.vhd
--                      vhdl model for a clock generator
--------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity clock_gr is
        generic ( cycle : TIME := 50 ns );
        port ( ck : out STD_LOGIC );
end clock_gr;

architecture behavioral of clock_gr is
begin
        process
                variable periodic : STD_LOGIC := '1';
        begin

                periodic := not periodic;

                ck <= periodic;

                wait for ( cycle / 2 );
        end process;
end behavioral;
```

5.3 tb.vhd

```
--------------------------------------------------------------------
--      File            :       tb.vhd
--      Description     :       Test Bench for DCT chip
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity tb is
end tb;
```

```
architecture test of tb is
        component dct
                port(
                        Clk     :       in STD_LOGIC;
                        Start   :       in STD_LOGIC;
                        Din :           in INTEGER;
                        Done :          out STD_LOGIC;
                        Dout :          out INTEGER
                        );
        end component;
        for all : dct use entity work.dct( sequential );

        component cg
                --generic ( cycle : TIME := 50 ns )
                port ( ck : out STD_LOGIC );
        end component;
        for all : cg use entity work.clock_gr( behavioral );

        signal Clk, Start, Done         : STD_LOGIC;
        signal Din, Dout                : INTEGER;
begin

        U1 : dct port map( Clk, Start, Din, Done, Dout );
        U2 : cg port map( Clk );

        Start <= '0' after 0 ns,
                 '1' after 30 ns,
                 '0' after 100 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  )
                                        );
        begin

                ---------------------------------
                --      Handshaking
                ---------------------------------
                wait until Start =  '1';

                ---------------------------------
                --      Feed Data -- Sandwich Case
                ---------------------------------
                wait until Clk = '1';
                for i in 0 to 23 loop
```

```
                wait until Clk = '0';
                Din <= 255;
        end loop;

        for i in 24 to 39 loop
                wait until Clk = '0';
                Din <= 0;
        end loop;

        for i in 40 to 63 loop
                wait until Clk = '0';
                Din <= 255;
        end loop;

        ---------------------------------
        --      Handshaking
        ---------------------------------
        wait until Done = '1';
        wait until Clk = '1';

        --------------------------------
        --      Verify
        --------------------------------
        for i in 0 to 7 loop
        for j in 0 to 7 loop
                wait until Clk = '0';
                assert ( Dout = Result( i, j ) ) REPORT "Error" SEVERITY warning;
        end loop;
        end loop;

        --------------------------------
        --      The End
        --------------------------------
        assert (FALSE) REPORT "Simulation Done." SEVERITY failure;

    end process;

end test;
```