Informatics and Mathematical Modelling
Technical University of Denmark

02207 : Advanced Digital Design Techniques

Design for Low Power by Reducing Switching Activity

LAB 2

Group *dt07*

Markku Eerola (s053739)

Rajesh Bachani (s061332)

Josep Renard (s071158)

November 19, 2007

Contents

1 Introduction

The purpose of this exercise was to estimate the power dissipation in a digital circuit due to the switching activity in the cells. Power is dissipated in a digital circuit, dynamically, in two ways; one, the power that is spent in charging or discharging the capacitance load connected to the output of the cell, and two, the power dissipated inside the cell due to short circuit currents and the internal capacitance charging or discharging. This holds for combinational cells. For sequential cells, there is extra power spent at every clock cycle, even if the output of the cell does not change. This is because there is some reaction to every clock cycle in sequential cells, which would take some power.

Static power in digital circuits is due to the internal leakage currents in CMOS. Though, in this exercise, we are particularly interested in analyzing the dynamic power dissipation.

We estimate the dynamic power in a serial to parallel converter. The converter takes in 8 bits (one byte) in every clock cycle, and gives out 32 bits (4 bytes) after every 4 clock cycles. The input byte at the first clock cycle is the most significant byte in the output, whereas the input byte in the fourth clock cycle is the lowest significant byte. The converter, thus, waits for four clock cycles to produce an output.

In the next section 2, we discuss three designs for such a converter. In section 3, we simulate the VHDL code for the designs using Modelsim, and verify that all the designs are working correctly. The VHDL code is then synthesized using Design Vision, and a switching activity is produced using VSS simulator. Based on this switching activity, a power report is presented for the synthesized design. These reports are presented in the last section 5, which also contains the VHDL code for the designs. In section 4, we discuss the results obtained.

1.1 Authors by Section

- *Markku Eerola*
- *Josep Renard*
- *Rajesh Bachani*

2 Designs for Serial to Parallel Conversion

In this section, we give an overview of the three designs for serial to parallel conversion, which are evaluated for their power consumption in this exercise.

2.1 Design A: Shift Register

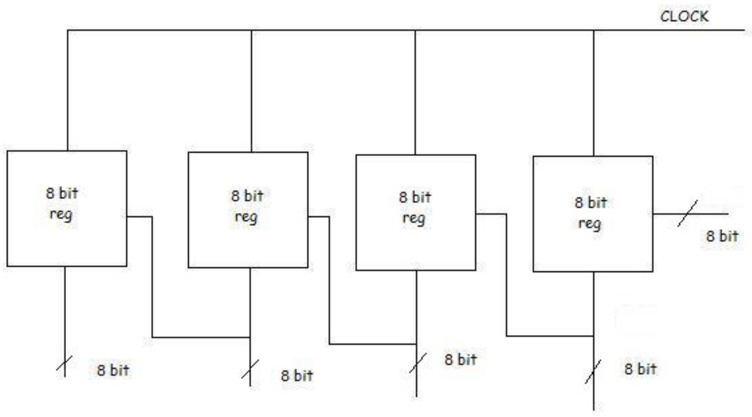


Figure 1: Converter using a 8-bit Shift Register

The shift register is implemented with a simple way, first he makes a movements of 8 bits to the next significant 8 bits and finally he stores on the eight first the input data, the shift register works until the clock stops. Here the input is only connected with the register of less important bits.

2.2 Design B: Register with Enable

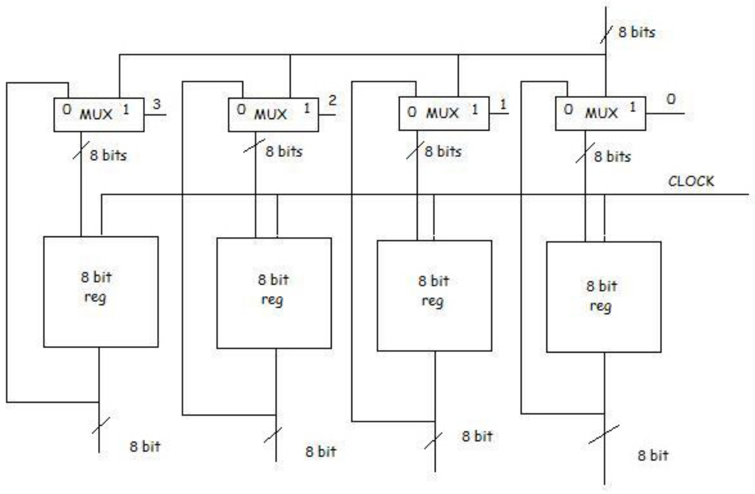


Figure 2: Converter using 8-bit Registers with Enable

For implement the shift register with an enable signal we use a multiplexor that open the door for store in a register, I mean the multiplexor is a filter that manage which register is going to store the input data, so the shift functionality is secure, the multiplexor make possible to store the first input on the most significant bits without through for the previous registers like the default shift register. In that version the input data is connected at the multiplexor and he give to the register the previous value of the register or the new input depending the value of the enable.

2.3 Design C: Register with Clock-Gating

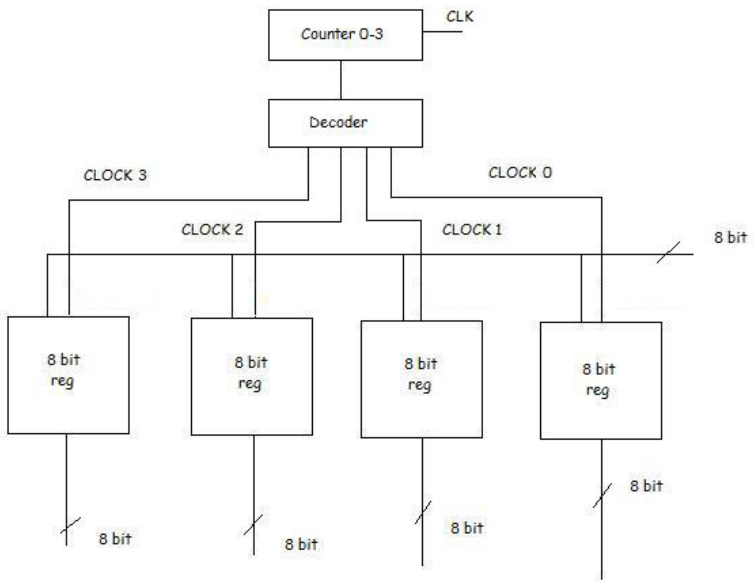


Figure 3: Converter using 8-bit Registers with Clock Gating

The shift register gated is developed using the same idea that the enable of doesn't across for all the registers to store the most significant 8 bits, but now we manage the clocks of the registers, how? very easy. We use a counter that is continuously counting 0 to 3 and depending the number of the counter the next entity, the decoder, enables a register for store the data with his clock, so the decoder open the register to store with the clock. The registers receive the input data all the time but only which is with the clock period open can store the data.

3 Simulation of the designs with Modelsim

All the three designs are simulated with Modelsim, to verify the functionality.

The following two screenshots demonstrate the working of implementation for Design A. The first screenshot id taken at 33ns while the second is taken at 43ns. It can be seen that in a new clock cycle, the 8 bit registers ripple their values to the more significant register, and the value of Qk for that clock cycle is fed into the least significant register.

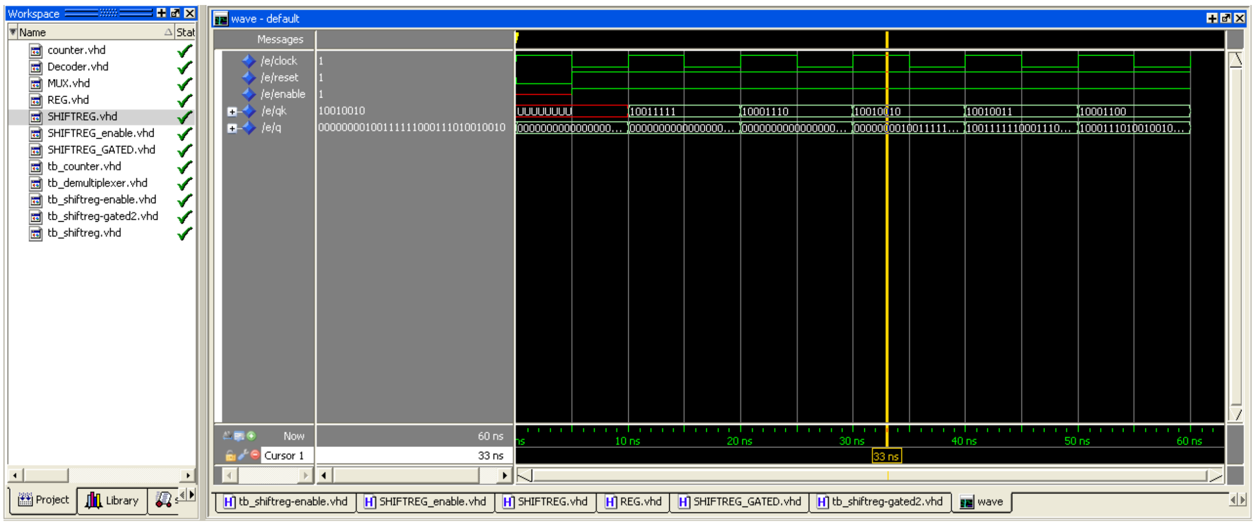


Figure 4: Simulation screenshot for Design A at 33ns

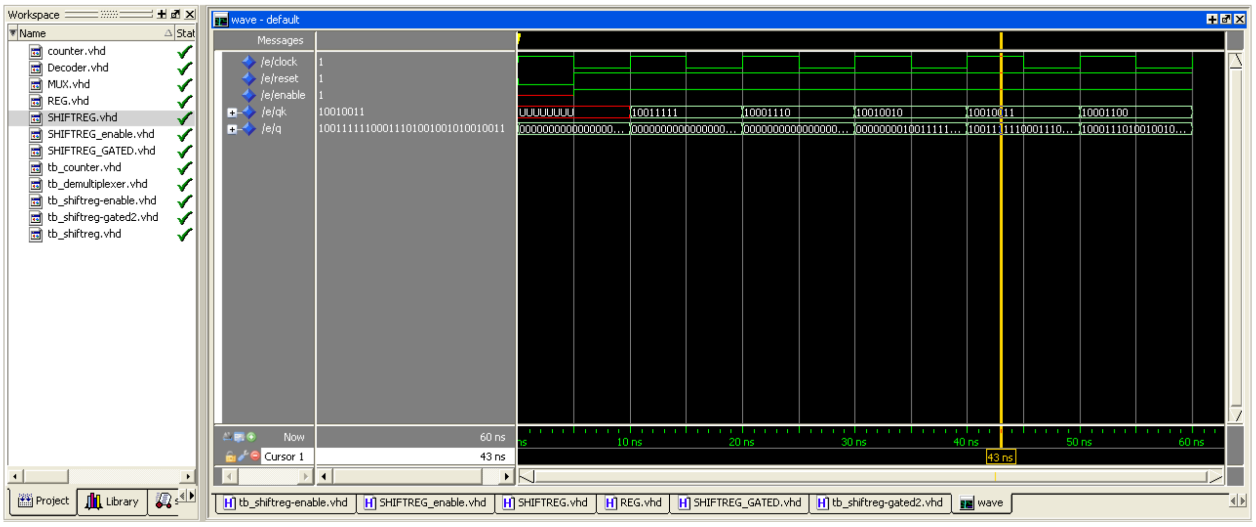
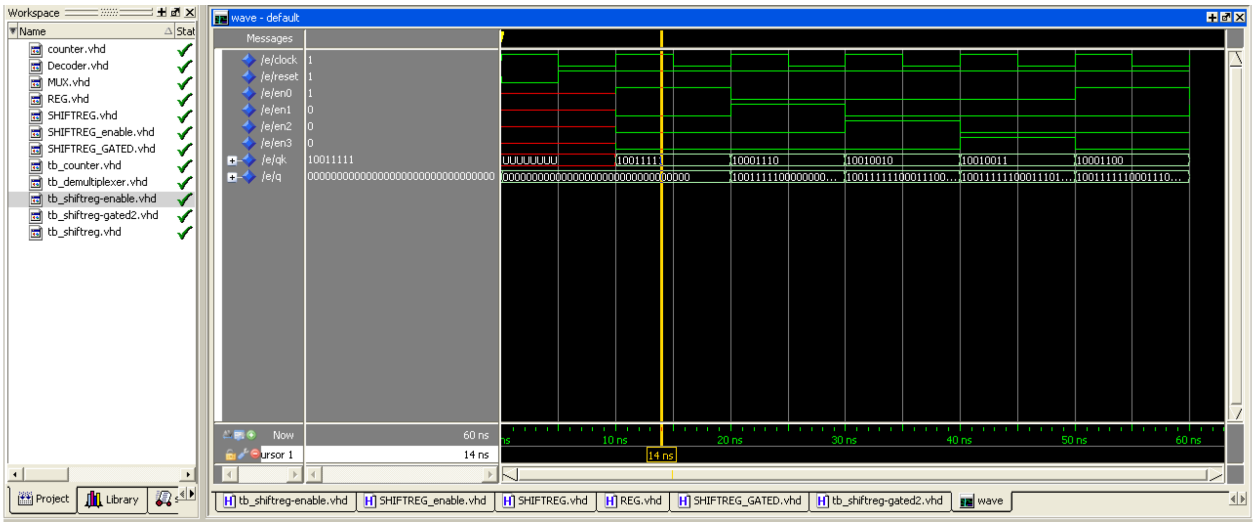


Figure 5: Simulation screenshot for Design A at 43ns

The following screenshots are from the simulation of Design B. In the first instance, which is at 14ns on the timeline, we have some value at Qk, but no value at Q. Then, at 24ns, the value of Qk in the previous clock cycle is loaded into the most significant register. Further on, at 33ns, the value of Qk in the previous clock cycle is loaded into the second most significant register. This repeats for four clock cycles.

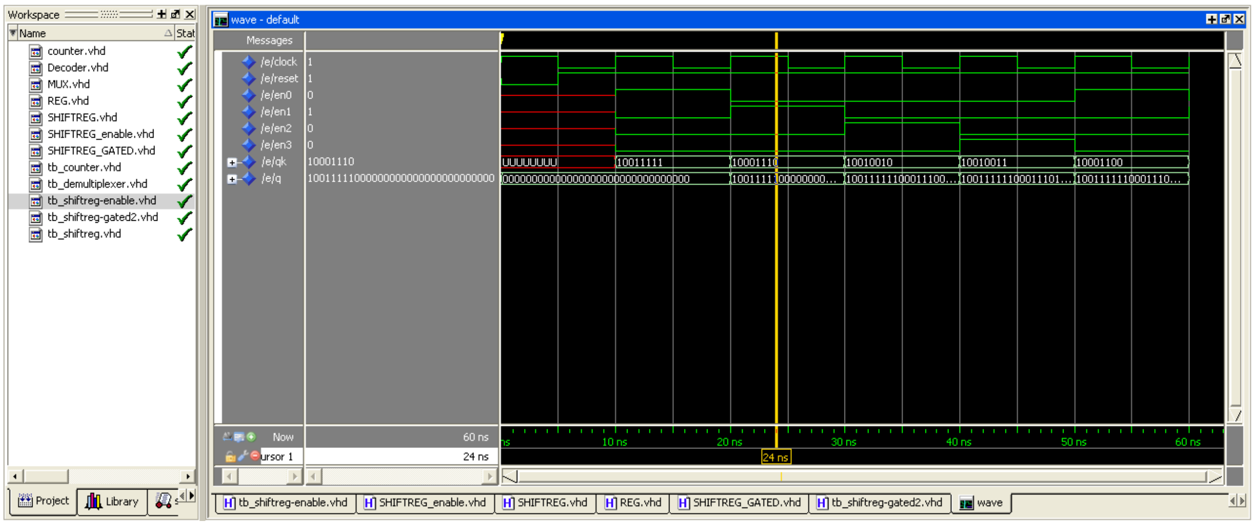


```
/e/clock   1
/e/reset   1
/e/en0     1
/e/en1     0
/e/en2     0
/e/en3     0
/e/qk      10011111
/e/q       00000000000000000000000000000000

/e/qk  UUUUUUUU | 10011111 | 10001110 | 10010010 | 10010011 | 10001100
/e/q   00000000000000000000000000000000 | 100111111000000000... | 10011111100011100... | 10011111100011101... | 100111111000111 0...

Now       60 ns
Cursor 1  14 ns
```

Figure 6: Simulation screenshot for Design B at 14ns



```
/e/clock   1
/e/reset   1
/e/en0     0
/e/en1     1
/e/en2     0
/e/en3     0
/e/qk      10001110
/e/q       10011111000000000000000000000000

Now       60 ns
Cursor 1  24 ns
```

Figure 7: Simulation screenshot for Design B at 24ns

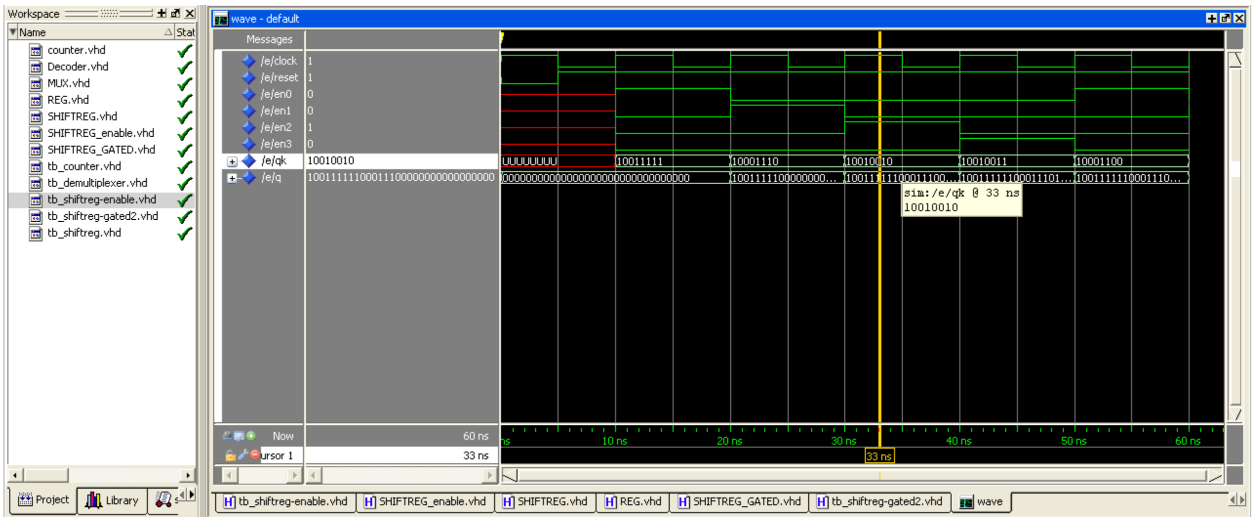


Figure 8: Simulation screenshot for Design B at 33ns

Then, for Design C, we have the following screenshots. As we can see, the values of Qk are transferred to the output to different registers. Also, the values of Qk are transferred in the same clock cycle, which was not the case with the Design B.

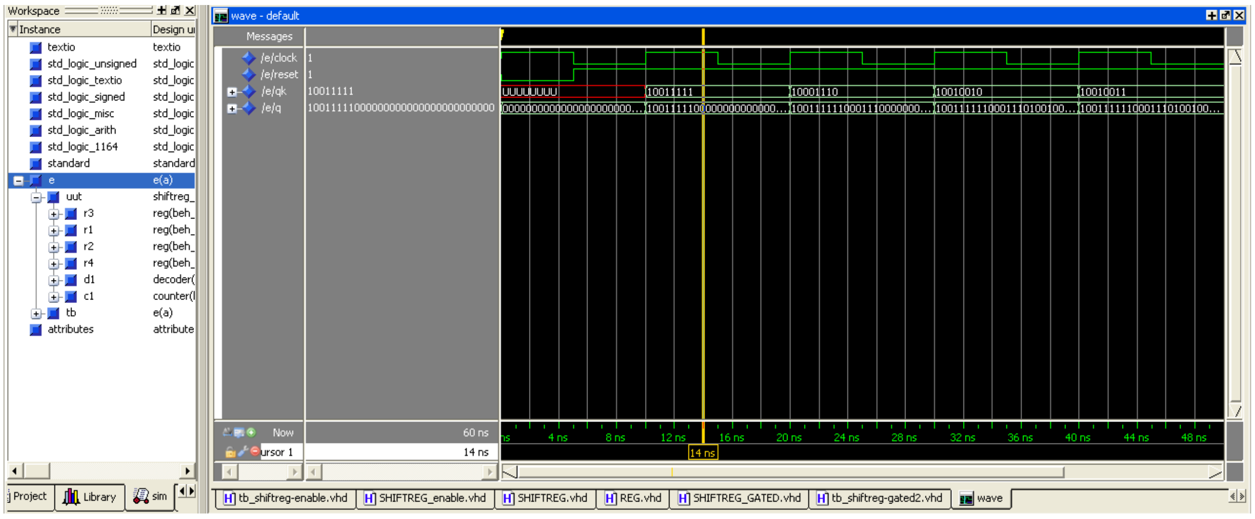


Figure 9: Simulation screenshot for Design C at 14ns

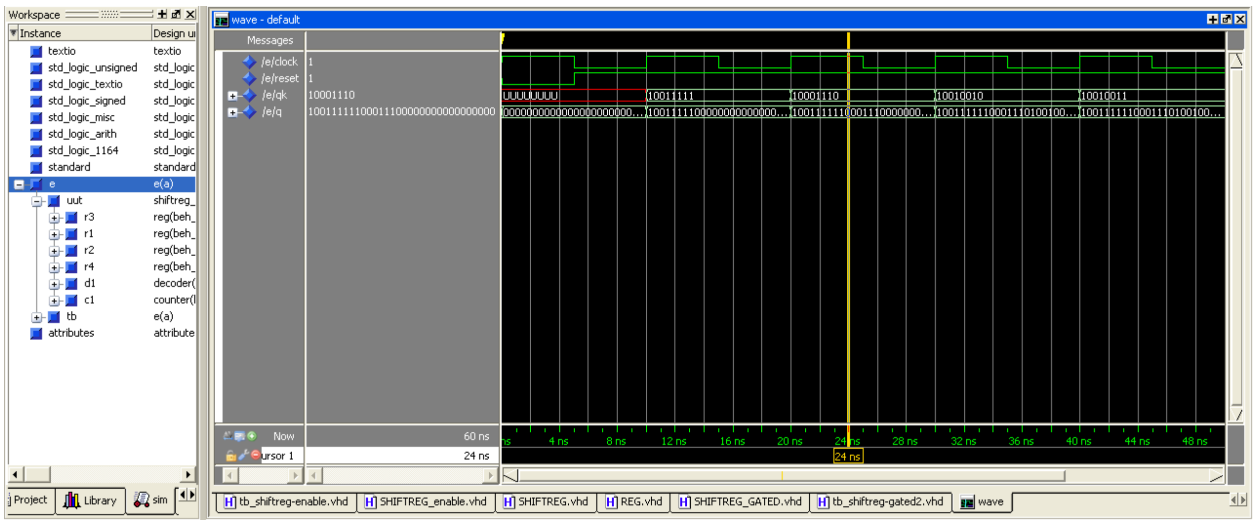


Figure 10: Simulation screenshot for Design C at 24ns

4 Power Reports and Discussion

In this section we discuss the results obtained from the simulations which were presented in section 3. For a short recap, the results can be seen in table 1:

	Total Dynamic Power	**Cell Leakage Power**
Design A	55.2 uW	773.7 nW
Design B	47 uW	800.5 nW
Design C	32.6 uW	835.0 nW

Table 1: Overview of Results from Power Reports

It is quite clear from the results that Design A consumes the most power (dynamic), while Design C is the most efficient, among the three designs. Though, this was expected at the beginning of the exercise. The static power, which comes from the internal leakage currents, is considerably low as compared to the dynamic power, in all the designs. So, we would not consider it in our analysis. Also, we would mean dynamic power when writing power, from now on.

In Design A, there is a transition at the output of each of the four registers, in every clock cycle. The input Qk is transferred from one register to the adjacent more significant register, every clock cycle, until it reaches the most significant register where it is just overwritten in the next clock cycle. This is the reason why the design is much consuming in terms of power. In every clock cycle, there is a switching activity in all the outputs of the four registers. Since switching accounts for a lot of power, for the entire time line of the simulation, we have high levels of power consumption.

Design B is efficient than Design A. Switching in Design B is controlled by the enable signals, which indicate which register should be loaded with Qk in the next clock event. If the enable signal is SET, the register is loaded with Qk, otherwise, the output is the same (rather, the previous value is reloaded into the register). What makes this design efficient than Design A is the reduction in the switching activity in terms of loading the register only when the enable is SET.

The overhead in this design, though, is the logic for the enable signals. It is interesting to note here that we have not implemented the logic for the enable signals as a separate combinational block. The enable signals are governed from the test bench, which was provided for the exercise. Thus there is no measure of the power dissipation in the enable block, which we think would be considerable. Also, from the lecture notes, we see that the ratio of power dissipated in Design A to that in Design B is 1:1.19, which indicates that Design B should consume more power. We believe this discripancy is because of we not implementing the enabling logic specifically.

Lets consider Design C now. It is quite clear that this design is the most efficient than both Design A and Design B. This design is based on clock-gating, which means that the original clock signal is not sent directly to the registers, but sent only when the register should be loaded with a new value. This has the power saving advantage when compared with Design B. When the register gets a clock cycle, it loads Qk. Otherwise, there is no internal power dissipation due to clock cycles for which the output does not change. In Design B, even if the enable for a register was RESET, which meant that the register would output would not change, still, since there was a clock cycle there was some power dissipated internally in the cell. This is avoided tremendously in Design C. And moreover, since the switching activity is directly propotional to the clock cycles arriving, reducing the clock cycles to individual registers means reducing unnecessary switching activity.

5 Implementation and Power Reports

Listing 1: SHIFTREG.vhd

```vhdl
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity SHIFTREG is
      Port (    CLOCK : In     std_logic;
                RESET : In     std_logic;
               ENABLE : In     std_logic;
                   QK : In     std_logic_vector (7 downto 0);
                    Q : InOut    std_logic_vector (31 downto 0) );
end SHIFTREG;

architecture BEHAVIORAL of SHIFTREG is

   begin
    process(RESET,CLOCK)
     variable i,j,k,l : integer;
     begin

       if ( RESET = '0' ) then
          for i in 0 to 31 loop
              q(i) <= '0';
          end loop;
       elsif ((CLOCK = '1') AND (CLOCK'EVENT)) then
                for i in 31 downto 8 loop
                    q(i) <= q(i-8);
                end loop;
                q(7 downto 0) <= qk;
       end if;

    end process;

end BEHAVIORAL;

configuration CFG_SHIFTREG_BEHAVIORAL of SHIFTREG is
   for BEHAVIORAL
   end for;

end CFG_SHIFTREG_BEHAVIORAL;
```

Listing 2: SHIFTREG_ENABLE.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity SHIFTREG_ENABLE is
   Port (
               CLOCK : In     std_logic;
               RESET : In     std_logic;
               QK : In        std_logic_vector (7 downto 0);
               Q : InOut      std_logic_vector (31 downto 0);
               en0: In std_logic;
               en1: In std_logic;
               en2: In std_logic;
               en3: In std_logic
         );
end SHIFTREG_ENABLE;

architecture BEH_SHIFTREG_ENABLE of SHIFTREG_ENABLE is
        component REG is
        port(
               D : in std_logic_vector(7 downto 0);
               Clock, Reset : in std_logic;
               Q : out std_logic_vector(7 downto 0)
               );
        end component REG;

        component MUX is
        port (
               Q0 : in std_logic_vector(7 downto 0);
               Q1 : in std_logic_vector(7 downto 0);
               enable: in std_logic;
               Qmux : out std_logic_vector(7 downto 0)
               );
        end component MUX;

        signal Qout0, Qout1, Qout2, Qout3 : std_logic_vector(7 downto 0);
        begin
        m1: MUX port map (Q(31 downto 24), QK, en0, Qout0);
        m2: MUX port map (Q(23 downto 16), QK, en1, Qout1);
        m3: MUX port map (Q(15 downto 8), QK, en2, Qout2);
        m4: MUX port map (Q(7 downto 0), QK, en3, Qout3);

        r1: REG port map (Qout0, Clock, Reset, Q(31 downto 24));
        r2: REG port map (Qout1, Clock, Reset, Q(23 downto 16));
        r3: REG port map (Qout2, Clock, Reset, Q(15 downto 8));
        r4: REG port map (Qout3, Clock, Reset, Q(7 downto 0));
end BEH_SHIFTREG_ENABLE;

configuration CFG_SHIFTREG_enable_SCHEMATIC of SHIFTREG_ENABLE is
   for BEH_SHIFTREG_ENABLE
   end for;
end CFG_SHIFTREG_enable_SCHEMATIC;
```

Listing 3: SHIFTREG_GATED.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

entity SHIFTREG_GATED is
      Port (    CLK : In     std_logic;
               RESET : In     std_logic;
                  QK : In     std_logic_vector (7 downto 0);
                   Q : Out    std_logic_vector (31 downto 0) );
end SHIFTREG_GATED;

architecture BEH_SHIFTREG_GATED of SHIFTREG_GATED is
   component Counter is
   port(
      clock:     in std_logic;
      clear: in std_logic;
           Qc:  out std_logic_vector(1 downto 0)
   );
   end component Counter;

        component REG is
        port(
                D : in std_logic_vector(7 downto 0);
                Clock, Reset : in std_logic;
                Q : out std_logic_vector(7 downto 0)
                );
        end component REG;

   component DECODER is
      port(      I:       in std_logic_vector(1 downto 0);
                  O:      out std_logic_vector(3 downto 0)
           );
   end component DECODER;

        signal out_counter : std_logic_vector(1 downto 0);
        signal out_decoder : std_logic_vector(3 downto 0);
        begin

        c1: Counter port map (CLK, Reset, out_counter);

   d1: DECODER port map (out_counter, out_decoder);

        r1: REG port map (QK, out_decoder(3), Reset, Q(31 downto 24));
        r2: REG port map (QK, out_decoder(2), Reset, Q(23 downto 16));
        r3: REG port map (QK, out_decoder(1), Reset, Q(15 downto 8));
        r4: REG port map (QK, out_decoder(0), Reset, Q(7 downto 0));

end BEH_SHIFTREG_GATED;

configuration CFG_SHIFTREG_GATED_SCHEMATIC of SHIFTREG_GATED is
   for BEH_SHIFTREG_GATED
   end for;

end CFG_SHIFTREG_GATED_SCHEMATIC;
```

Listing 4: REG.vhd

```vhdl
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

   entity REG is
        port(
               D : in std_logic_vector(7 downto 0);
               Clock, Reset : in std_logic;
               Q : out std_logic_vector(7 downto 0));
end entity REG;

architecture BEH_REG of REG is
   begin
   p0: process (Clock, Reset) is
      begin
      if (Reset = '0') then
         Q <= (others => '0');
      elsif rising_edge(clock) then
         Q <= D;
      end if;
   end process p0;
end architecture BEH_REG;
```

Listing 5: MUX.vhd

```vhdl
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_signed.all;
   use IEEE.std_logic_arith.all;

   entity MUX is
        port (
               Q0 : in std_logic_vector(7 downto 0);
               Q1 : in std_logic_vector(7 downto 0);
               enable: in std_logic;
               Qmux : out std_logic_vector(7 downto 0)
               );
end entity MUX;

architecture BEH_MUX of MUX is
   begin

   process (Q0,Q1,enable) is
      begin
      if (enable = '0') then
        Qmux <= Q0;
      elsif (enable = '1') then
        Qmux <= Q1;
           else
        Qmux <= (others => '0');
      end if;
   end process;
end architecture BEH_MUX;
```

Listing 6: COUNTER.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
port(
      clock:    in std_logic;
      clear: in std_logic;
           Qc:  out std_logic_vector(1 downto 0)
   );
end counter;

architecture beh_counter of counter is

    signal Pre_Q: std_logic_vector(1 downto 0);

begin

    process(clock, clear)
    begin
       if (clear = '0') then
           Pre_Q <= "11";
       elsif (clock='1' and clock'event) then
               Pre_Q <= Pre_Q + "01";
            end if;
    end process;
    Qc <= Pre_Q;
end beh_counter;
```

Listing 7: DECODER.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity DECODER is
port(    I:       in std_logic_vector(1 downto 0);
         O:       out std_logic_vector(3 downto 0)
);
end DECODER;

architecture BEH_DECODER of DECODER is
begin
    process (I)
    begin
    case I is
            when "00" => O <= "1000";
            when "01" => O <= "0100";
            when "10" => O <= "0010";
            when "11" => O <= "0001";
            when others => O <= "1000";
        end case;

   end process;

end BEH_DECODER;
```

Listing 8: Power Report Design A

```
****************************************
Report : power
        -analysis_effort low
Design : SHIFTREG
Version: X-2005.09-SP1
Date   : Fri Nov 16 20:21:52 2007
****************************************



Library(s) Used:

    CORE90GPSVT (File: /cell_libs/cmos090_50a/CORE90GPSVT_SNPS-AVT_2.1/
           SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/CORE90GPSVT_NomLeak.db)



Operating Conditions: NomLeak   Library: CORE90GPSVT
Wire Load Model Mode: enclosed

Design         Wire Load Model          Library
------------------------------------------------
SHIFTREG                 area_0to1K           CORE90GPSVT



Global Operating Voltage = 1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW



  Cell Internal Power  =  52.5652 uW   (95%)
  Net Switching Power  =   2.6312 uW    (5%)
                         ---------
Total Dynamic Power    =  55.1964 uW  (100%)


Cell Leakage Power     = 773.6685 nW
```

Listing 9: Power Report Design B

```
****************************************
Report : power
        -analysis_effort low
Design : SHIFTREG_ENABLE
Version: X-2005.09-SP1
Date   : Fri Nov 16 21:26:47 2007
****************************************




Library(s) Used:

    CORE90GPHVT (File: /cell_libs/cmos090_50a/CORE90GPHVT_SNPS-AVT_2.1.a/
         SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/CORE90GPHVT_NomLeak.db)
    CORE90GPSVT (File: /cell_libs/cmos090_50a/CORE90GPSVT_SNPS-AVT_2.1/
         SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/CORE90GPSVT_NomLeak.db)




Operating Conditions: NomLeak   Library: CORE90GPSVT
Wire Load Model Mode: enclosed

Design          Wire Load Model            Library
------------------------------------------------
SHIFTREG_ENABLE         area_0to1K         CORE90GPSVT
MUX_3                   area_0to1K         CORE90GPSVT
MUX_2                   area_0to1K         CORE90GPSVT
MUX_1                   area_0to1K         CORE90GPSVT
MUX_0                   area_0to1K         CORE90GPSVT
REG_3                   area_0to1K         CORE90GPSVT
REG_2                   area_0to1K         CORE90GPSVT
REG_1                   area_0to1K         CORE90GPSVT
REG_0                   area_0to1K         CORE90GPSVT



Global Operating Voltage = 1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW




  Cell Internal Power  =  42.5321 uW   (91%)
  Net Switching Power  =   4.4528 uW    (9%)
                         ---------
Total Dynamic Power    =  46.9849 uW  (100%)


Cell Leakage Power     = 800.4604 nW
```

Listing 10: Power Report Design C

```
****************************************
Report : power
        -analysis_effort low
Design : SHIFTREG_GATED
Version: X-2005.09-SP1
Date   : Fri Nov 16 23:59:48 2007
****************************************




Library(s) Used:

    CORE90GPSVT (File: /cell_libs/cmos090_50a/CORE90GPSVT_SNPS-AVT_2.1/
            SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/CORE90GPSVT_NomLeak.db)
    CORE90GPHVT (File: /cell_libs/cmos090_50a/CORE90GPHVT_SNPS-AVT_2.1.a/
            SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/CORE90GPHVT_NomLeak.db)




Operating Conditions: NomLeak   Library: CORE90GPSVT
Wire Load Model Mode: enclosed

Design         Wire Load Model          Library
------------------------------------------------
SHIFTREG_GATED         area_0to1K       CORE90GPSVT
counter                area_0to1K       CORE90GPSVT
DECODER                area_0to1K       CORE90GPSVT
REG_3                  area_0to1K       CORE90GPSVT
REG_2                  area_0to1K       CORE90GPSVT
REG_1                  area_0to1K       CORE90GPSVT
REG_0                  area_0to1K       CORE90GPSVT




Global Operating Voltage = 1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW




  Cell Internal Power  =  26.4885 uW   (81%)
  Net Switching Power  =   6.1442 uW   (19%)
                         ---------
Total Dynamic Power    =  32.6327 uW  (100%)


Cell Leakage Power     = 834.9294 nW
```