M2 Master E3A Integration Circuits Systems

TD9: Synthesis of a frequency divider based on a gray code counter

1. **IFTAKHER Mohammed Akib – No. 22211204**
2. **KIBIWOTT Albert Kiplagat - No. 22211612**

Instructor: Professor Hervé MATHIAS

In this practical work, the focus has been on the synthesis of the gray code counter. First, we assumed that the factor input is assumed to be constant/not changing during the normal operation mode. The counter should be operated at a frequency close to 1GHz.

1. Timing1 Results

```
            Setup:-     330
     Required Time:=    670
      Launch Clock:-      0
         Data Path:-   1888
             Slack:=  -1218
```

From the result above, it can be seen that the data path requires more time than the available (Required) time. The goal for this practical session is to operate the gray code counter in 1 GHz frequency or in another word the gray code counter should operate in 1000 ps. Here, the set up time requires 330 ps and the left time is dedicated for the required time session to transmit through the datapath. But it is already mentioned that datapath requires more time. As a result, we have a slack of 1218 ps. So, the next steps require us to reduce the slack time.

2. Here, the syn_global_effort and syn_opt_effort parameters have been changed to ‘high’ in the synthesis tool settings. As a result, the following timing results were obtained:

Timing1 Results

```
            Setup:-     330
     Required Time:=    670
      Launch Clock:-      0
         Data Path:-   1789
             Slack:=  -1119
```

It has been observed from the result above that with the changes of the status of syn_global_effort and syn_opt_effort parameter, the slack has been reduced in a low margin. So, the effort has been continued to reduce the slack time.

3. In this part, the library of the logic gates has been changed in order to replace the 1.8 V gates with 3.3 V gates. This was done in the mmmc.tcl file by replacing the following libraries: $LIBSET_SLOWHV, $LIBSET_FASTHV, and $LIBSET_TYPHV. The following timing results were obtained:

Timing1 Results

```
            Setup:-     169
     Required Time:=    831
      Launch Clock:-      0
         Data Path:-   1889
             Slack:=  -1058
```

Similar to steps 2, the libraries mentioned above has been modified in order to conduct higher voltage. From the observation, it can be stated that the slack has been slightly reduced. Moreover, the setup time has been reduced almost by half which assist to increase the available (Required) time in order to reach near to datapath time.

4. Architecture a5

Here, we divided the working clock by 2 as shown in the architecture a5 below. The slack obtained is a shown below:

```
1   ------------------------------------------------
2   -- architecture a4 du compteur de gray
3   -- on génère l'horloge divisée en sortie
4   ------------------------------------------------
5
6   architecture a5 of compteur_gray is
7
8   signal count : unsigned(7 downto 0) := (others => '0');
9   signal countc : unsigned(6 downto 0) := (others => '0');
10  signal clkdiv : std_logic := '0';
11
12  begin
13  clkdiv : process(clk)
14  begin
15     if (clk'event and clk='1') then
16        clkdiv <= not clkdiv;
17     end if;
18  end process;
19
20
21  P1 : process(clkdiv,reset)
22  begin
23    if(reset='0') then
24     count<= (0 => '1', others => '0');
25     clock_out <= '0';
26
27     elsif(clkdiv'event and clkdiv='1') then
28           if (enable='1') then
29        if (count(count'left downto 1)=bin2gray(factor)) then
30            count<= (0 => '1', others => '0');
31            clock_out <= '0' ;        -- quand le compteur est remis à 0, clock_out ↵
    est aussi remis à 0
32        else
33            count(0) <= not count(0);
34            count(1) <= count(1) xor count(0) ;
35            for i in 2 to count'left-1 loop
36           if (count(i-1)='1') and (count(i-2 downto 0)=0 ) then
37               count(i) <= not count(i) ;
38           end if ;
39            end loop;
40            if (count(count'left-2 downto 0)=0 ) then
41           count(count'left) <= not count(count'left) ;
42            end if ;
43        end if ;
44        if (count(count'left downto 1)=bin2gray(factor/2)) then  -- quand on atteint ↵
    la moitié du modulo, clock_out est mis à 1
45          clock_out <= '1' ;
46        end if ;
47           end if ;
48    end if;
49  end process;
50
51  P2 : process(clk2)
52  begin
53    if(clk2'event and clk2='1') then
54     countc <= count(count'left downto 1) ;
55    end if ;
56  end process ;
57
58  P3 : count_out <= gray2bin(countc) ;
59
60
61  end a5 ;
```

Timing1 Results

```
            Setup:-     191
   Required Time:=     809
    Launch Clock:-       0
       Data Path:-     522
           Slack:=     287
```

Timing2 Results

```
            Setup:-     201
   Required Time:=    1799
    Launch Clock:-       0
       Data Path:-    1944
           Slack:=    -145
```

From the results obtained from the architecture a5, it can be stated that with the division of the working clock by 2, the frequency will reduce which means it takes longer time to process in the datapath, as shown by the timing 2 results. Here the system is driven by clk2.

5. Architecture a6

To further optimize our code, we introduced in the loop registers cp1 and cp2 which detects 1 clock cycle in advance the comparison with the factor input as depicted in architecture a6 below:

```
1   ------------------------------------------------
2   -- architecture a4 du compteur de gray
3   -- on génère l'horloge divisée en sortie
4   ------------------------------------------------
5
6   architecture a6 of compteur_gray is
7
8   signal count : unsigned(7 downto 0) := (others => '0');
9   signal countc : unsigned(6 downto 0) := (others => '0');
10  signal clkdiv : std_logic := '0';
11  signal cp1, cp2 : std_logic;
12
13
14  begin
15  P4 : process(clk)
16  begin
17     if (clk'event and clk='1') then
18        clkdiv <= not clkdiv;
19     end if;
20  end process;
21
22
23
24
25  P1 : process(clkdiv,reset)
26  begin
27    if(reset='0') then
28     count<= (0 => '1', others => '0');
29     clock_out <= '0';
30
31     elsif(clkdiv'event and clkdiv='1') then
32           if (enable='1') then
33        cp1 <= '0';
34        cp2 <= '0';
35        if (count(count'left downto 1)=bin2gray(factor-1)) then
36           cp1 <= '1';
37        end if;
38        if (count(count'left downto 1)=bin2gray(factor/2-1)) then  -- quand on atteint ↵
    la moitié du modulo, clock_out est mis à 1
39          cp2<= '1' ;
40        end if ;
41
42
43        if (cp1='1') then
44            count<= (0 => '1', others => '0');
45            clock_out <= '0' ;        -- quand le compteur est remis à 0, clock_out ↵
    est aussi remis à 0
46        else
47            count(0) <= not count(0);
48            count(1) <= count(1) xor count(0) ;
49            for i in 2 to count'left-1 loop
50           if (count(i-1)='1') and (count(i-2 downto 0)=0 ) then
51               count(i) <= not count(i) ;
52           end if ;
53            end loop;
54            if (count(count'left-2 downto 0)=0 ) then
55           count(count'left) <= not count(count'left) ;
56            end if ;
57        end if ;
58        if (cp2 ='1') then  -- quand on atteint la moitié du modulo, clock_out est mis ↵
    à 1
59          clock_out <= '1' ;
60        end if ;
61           end if ;
62    end if;
63  end process;
64
65  P2 : process(clk2)
66  begin
67    if(clk2'event and clk2='1') then
68     countc <= count(count'left downto 1) ;
```

```
69      end if ;
70    end process ;
71
72    P3 : count_out <= gray2bin(countc) ;
73
74
75    end a6 ;
76
77
```

Timing1 Results

```
              Setup:-     191
      Required Time:=     809
       Launch Clock:-       0
          Data Path:-     522
              Slack:=     287
```

Timing2 Results

```
              Setup:-     205
      Required Time:=    1795
       Launch Clock:-       0
          Data Path:-    1795
              Slack:=       1
```

From the above results, it can be deduced that with the use of registers to further optimize the code, the required time matches exactly with the datapath time which ultimately reduces the slack time. This is due to the fact that the registers provide quick and efficient access to storage.

6. Conclusion

The maximum frequency at which our frequency divider counter will be able to operate can be calculated as:

Maximum time = set up time + required time = (205 + 1795) ps = 2000 ps

So, the maximum frequency = $\frac{1}{2000 \times 10^{-12}} = 500\ MHz.$