

# Assignment 2

## VLSI Design

Amit Kumar,16D070034

October 13, 2019

### Q-1

Assume that the delay of some common gates (inclusive of parasitic delay) is as given below:

- **Inverter** 100 ps
- **NAND gate** 150 ps
- **NOR gate** 150 ps
- **A+B.C** 200 ps
- **Tiny XOR** 200 ps

## 32-bit logarithmic adder using Brent-Kung architecture

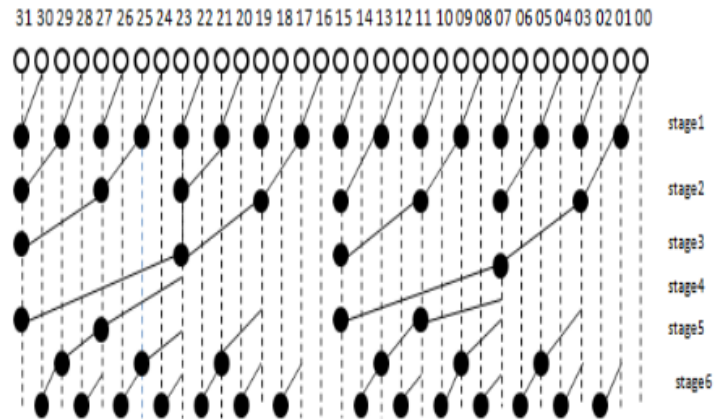


Figure 1: 32-bit Brent-Kung adder

## Common Gates implementation in VHDL

```

1  -----
2  library std;
3  use std.standard.all;
4  -----
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  entity Inverter is
9      port (a: in std_logic;
10           b: out std_logic);
11 end entity Inverter;
12 architecture Behave of Inverter is
13 begin
14     b <= not a after 100ps;  -----Delay of Inverter gate
15 end Behave;
16
17 -----
18
19 library ieee;
20 use ieee.std_logic_1164.all;
21 entity NOR_2 is
22     port (a, b: in std_logic;

```

```

23         c: out std_logic);
24     end entity NOR_2;
25     architecture Behave of NOR_2 is
26     begin
27         c <= not(a or b) after 150ps; -----Delay of NOR gate
28     end Behave;
29
30     -----
31
32     library ieee;
33     use ieee.std_logic_1164.all;
34     entity NAND_2 is
35     port (a, b: in std_logic;
36          c: out std_logic);
37     end entity NAND_2;
38     architecture Behave of NAND_2 is
39     begin
40         c <= not (a and b) after 150ps; -----Delay of NAND gate
41     end Behave;
42
43     -----
44
45     library ieee;
46     use ieee.std_logic_1164.all;
47     entity tinyXOR is
48     port (a, b: in std_logic;
49          c: out std_logic);
50     end entity tinyXOR;
51     architecture Behave of tinyXOR is
52     begin
53         c <= (a xor b) after 200ps; -----Delay of Tiny_XOR gate
54     end Behave;
55
56     -----
57
58     library ieee;
59     use ieee.std_logic_1164.all;
60     entity Custom is
61     port (a, b, c : in std_logic;
62          d: out std_logic);
63     end entity Custom;
64     architecture Behave of Custom is
65     begin
66         d <= not(a or (b and c)) after 200ps;
67     end Behave;

```

68  
69

---

## Pre-Processing Stage

In this stage we compute, the generate and propagate signals which are then used to generate carry input of each adder.

A and B are inputs.

These signals are given by the equation 1 & 2.

$$P_i = A_i \oplus B_i \quad (1)$$

$$G_i = A_i \cdot B_i \quad (2)$$

---

```
1  -----
2  library std;
3  use std.standard.all;
4
5  -----
6  --////////////////////////////////////Carry Propagation////////////////////////////////////
7  -----Delay 200ps-----
8  library ieee;
9  use ieee.std_logic_1164.all;
10 -----
11 entity P_gen is
12     port( x,y: in std_logic ;
13           p : out std_logic) ;
14 end entity P_gen;
15
16 architecture behave of P_gen is
17     component tinyXOR is
18     port (a, b: in std_logic;
19           c: out std_logic);
20     end component tinyXOR;
21
22 -----
23 begin
24 propagation_generator: tinyXOR port map(a => x, b => y , c => p );
25
26 end behave;
27 -----
28 --////////////////////////////////////Carry Generation////////////////////////////////////
29 -----Delay 250ps-----
```

```

30 library ieee;
31 use ieee.std_logic_1164.all;
32 -----
33 entity G_gen is
34     port( x,y: in std_logic ;
35           g : out std_logic) ;
36 end entity G_gen;
37
38 architecture behave of G_gen is
39     signal gbar : std_logic;
40
41     component NAND_2 is
42     port (a, b: in std_logic;
43           c: out std_logic);
44     end component NAND_2;
45
46     component Inverter is
47     port (a: in std_logic;
48           b: out std_logic);
49     end component Inverter;
50
51     -----
52 begin
53
54     Carry_Generation: Nand_2 port map(a => x, b => y , c=> gbar);
55     inv: Inverter port map(a => gbar, b => g);
56
57 end behave;

```

---

## Carry generation network

In this stage carries corresponding to each bit is calculated. Execution is done in parallelly and after the computation of carries they are divided into smaller pieces.

Carry operator (**GroupPG**) contain two AND gates, one OR gate. It uses propagate and generate as intermediate signals which are given by the equations 3 & 4.

$$P_{i:k} = P_{i:k} \oplus P_{j-1:k} \quad (3)$$

$$G_{i:k} = G_{i:k} + G_{j-1:k} \cdot P_{i:j} \quad (4)$$

```

1  -----
2  library std;
3  use std.standard.all;
4
5  -----
6  --////////////////////////////////////Carry Propagation////////////////////////////////////
7  -----Delay 200ps-----
8  library ieee;
9  use ieee.std_logic_1164.all;
10 -----
11 entity P_gen is
12     port( x,y: in std_logic ;
13           p : out std_logic) ;
14 end entity P_gen;
15
16 architecture behave of P_gen is
17     component tinyXOR is
18     port (a, b: in std_logic;
19           c: out std_logic);
20     end component tinyXOR;
21
22 -----
23 begin
24 propagation_generator: tinyXOR port map(a => x, b => y , c => p );
25
26 end behave;
27
28 --////////////////////////////////////Carry Generation////////////////////////////////////
29 -----Delay 250ps-----
30 library ieee;
31 use ieee.std_logic_1164.all;
32 -----
33 entity G_gen is
34     port( x,y: in std_logic ;
35           g : out std_logic) ;
36 end entity G_gen;
37
38 architecture behave of G_gen is
39 signal gbar : std_logic;
40
41     component NAND_2 is
42     port (a, b: in std_logic;
43           c: out std_logic);
44     end component NAND_2;

```

```

45
46         component Inverter is
47     port (a: in std_logic;
48         b: out std_logic);
49     end component Inverter;
50
51     -----
52 begin
53
54 Carry_Generation: Nand_2 port map(a => x, b => y , c=> gbar);
55 inv: Inverter port map(a => gbar, b => g);
56
57 end behave;
58
59 -----Basic Cell for Bent-Kung-----
60 library ieee;
61 use ieee.std_logic_1164.all;
62
63 entity groupPG is
64     port( g1,p1,g2,p2: in std_logic ;
65         gout,pout : out std_logic) ;
66 end entity groupPG;
67
68 architecture behave of groupPG is
69     signal gbar,pbar : std_logic;
70
71     component NAND_2 is
72     port (a, b: in std_logic;
73         c: out std_logic);
74     end component NAND_2;
75
76     component Inverter is
77     port (a: in std_logic;
78         b: out std_logic);
79     end component Inverter;
80
81     component Custom is
82     port (a, b, c : in std_logic;
83         d: out std_logic);
84     end component Custom;
85
86     -----
87 begin
88 g_new: Custom port map(a => g2, b =>g1, c => p2, d => gbar ); -----g2+ g1*p2
89 inv1: Inverter port map(a => gbar, b => gout); --Delay 300ps

```

```

90
91 p_new: Nand_2 port map(a => p1, b => p2 , c=> pbar);
92 inv2: Inverter port map(a => pbar, b => pout);      --Delay 250ps
93
94 end behave;

```

---

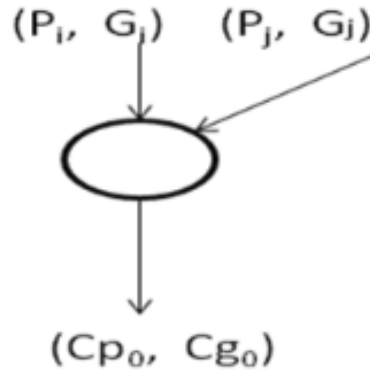


Figure 2: GroupPG block

## 16-Bit Brent-Kung group PG generation

---

```

1  -----
2  library std;
3  use std.standard.all;
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  entity BK16Bit is
9      port (a,b : in std_logic_vector(15 downto 0);
10           g_sig,p_sig: out std_logic_vector(15 downto 0 );
11           p_vec:  out std_logic_vector(15 downto 0 )
12           );
13 end entity BK16Bit;
14
15 architecture Behave of BK16Bit is
16
17 signal g0,p0 ,gout,pout  : std_logic_vector(15 downto 0) ;
18 signal gs1,ps1 : std_logic_vector(7 downto 0) ;

```



```

19 signal gs2,ps2 : std_logic_vector(3 downto 0) ;
20 signal gs3,ps3 : std_logic_vector(1 downto 0) ;
21 signal gs4,ps4 : std_logic_vector(1 downto 0) ;
22 signal gs5,ps5 : std_logic_vector(2 downto 0) ;
23
24 component P_gen is
25     port( x,y: in std_logic ;
26           p : out std_logic) ;
27 end component P_gen;
28
29 component G_gen is
30     port( x,y: in std_logic ;
31           g : out std_logic) ;
32 end component G_gen;
33
34 component groupPG is
35     port( g1,p1,g2,p2: in std_logic ;
36           gout,pout : out std_logic) ;
37 end component groupPG;
38
39 begin
40
41 ----- P and G generation -----
42 loop1: for i in 0 to 15 generate
43 begin
44     propagation1: P_gen port map ( x => a(i) , y => b(i) , p => p0(i) ) ;
45     generation1 : G_gen port map ( x => a(i) , y => b(i) , g => g0(i) ) ;
46 end generate loop1;
47 -----
48
49 ----- Groop PG Generation using Parallel Prefix for 16-bit Brent-Kung -----
50 -----Stage 1
51 loop2: for i in 0 to 7 generate
52 begin
53     propagation_after_stage1: groupPG port map ( g1 => g0(2*i) , p1 => p0(2*i) , g2 => g0(2*i+1),
54     end generate loop2;
55
56 -----Stage 2
57 loop3: for i in 0 to 3 generate
58 begin
59     propagation_after_stage2: groupPG port map ( g1 => gs1(2*i) , p1 => ps1(2*i) , g2 => gs1(2*i+1),
60     end generate loop3;
61
62 -----Stage 3
63 loop4: for i in 0 to 1 generate

```

```

64  begin
65    propagation_after_stage3: groupPG port map ( g1 => gs2(2*i) , p1 => ps2(2*i) , g2 => gs2(2*i+1)
66  end generate loop4;
67
68  -----Stage 4
69
70  propagation_after_stage4_1: groupPG port map ( g1 => gs3(0) , p1 => ps3(0) , g2 => gs3(1), p2 => ps3(1)
71  propagation_after_stage4_0: groupPG port map ( g1 => gs3(0) , p1 => ps3(0) , g2 => gs2(2), p2 => ps2(2)
72
73  ----Stage 5
74  propagation_after_stage5_2: groupPG port map ( g2 => gs1(6) , p2 => ps1(6) , g1 => gs4(0), p1 => ps4(0)
75  propagation_after_stage5_1: groupPG port map ( g2 => gs1(4) , p2 => ps1(4) , g1 => gs3(0), p1 => ps3(0)
76  propagation_after_stage5_0: groupPG port map ( g2 => gs1(2) , p2 => ps1(2) , g1 => gs2(0), p1 => ps2(0)
77
78  ----Allocating odd carries for using in even carries
79  gout(15) <= gs4(1) ;
80  gout(13) <= gs5(2) ;
81  gout(11) <= gs4(0) ;
82  gout(9)  <= gs5(1) ;
83  gout(7)  <= gs3(0) ;
84  gout(5)  <= gs5(0) ;
85  gout(3)  <= gs2(0) ;
86  gout(1)  <= gs1(0) ;
87
88  gout(0) <= g0(0) ;  --first carry
89  pout(0) <= p0(0) ;  --first carry
90
91  pout(15) <= ps4(1) ;
92  pout(13) <= ps5(2) ;
93  pout(11) <= ps4(0) ;
94  pout(9)  <= ps5(1) ;
95  pout(7)  <= ps3(0) ;
96  pout(5)  <= ps5(0) ;
97  pout(3)  <= ps2(0) ;
98  pout(1)  <= ps1(0) ;
99
100 -----Stage 6
101 loop5: for i in 1 to 7 generate
102  begin
103    propagation_after_stage6: groupPG port map ( g1 => gout(2*i-1) , p1 => pout(2*i-i) , g2 => g0(0)
104  end generate loop5;
105
106  g_sig <= gout ;
107  p_sig <= pout ;
108  p_vec <= p0 ;

```

109    end Behave;

---

## 32-Bit Brent-Kung group PG generation

---

```
1  -----
2  library std;
3  use std.standard.all;
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  entity BK32Bit is
9      port (a,b : in std_logic_vector(31 downto 0);
10           cout: out std_logic_vector(31 downto 0 );
11           pi: out std_logic_vector(31 downto 0 )      --just simple p not group p
12           );
13  end entity BK32Bit;
14
15  architecture Behave of BK32Bit is
16  signal g,p : std_logic_vector(31 downto 0) ;
17  --signal temp1,temp2,temp3,temp4,temp5,temp6 : std_logic ;
18
19  component BK16Bit is
20      port (a,b : in std_logic_vector(15 downto 0);
21           g_sig,p_sig: out std_logic_vector(15 downto 0 );
22           p_vec: out std_logic_vector(15 downto 0 )
23           );
24  end component BK16Bit;
25
26  component MSB16Bit is
27      port (a,b : in std_logic_vector(15 downto 0);
28           g15,p15: in std_logic ;
29           g_sig,p_sig: out std_logic_vector(15 downto 0 );
30           p_vec: out std_logic_vector(15 downto 0 )
31           );
32  end component MSB16Bit;
33
34  component groupPG is
35      port( g1,p1,g2,p2: in std_logic ;
36           gout,pout : out std_logic) ;
37  end component groupPG;
38
39  begin
40      LSB16_carry: BK16Bit port map ( a => a(15 downto 0) , b => b(15 downto 0), g_sig => g(15 downto 0), p_sig => p(15 downto 0),

```

```

41     cout(15 downto 0) <= g(15 downto 0) ;
42
43
44     MSB16_carry: MSB16Bit port map ( a => a(31 downto 16), b => b(31 downto 16), g15 => g(15) , p15
45     cout(31 downto 16) <= g(31 downto 16) ;
46
47
48     end Behave;

```

---

## 32-Bit Final Brent-Kung adder (post-processing)

---

```

1  -----
2  library std;
3  use std.standard.all;
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  entity BKadder is
9      port (a,b : in std_logic_vector(31 downto 0);
10           sum: out std_logic_vector(31 downto 0 )
11           -- cout: out std_logic
12           );
13  end entity BKadder;
14
15  architecture Behave of BKadder is
16
17      signal ci,pi : std_logic_vector(31 downto 0) ;
18
19          component tinyXOR is
20              port (a, b: in std_logic;
21                  c: out std_logic);
22          end component tinyXOR;
23
24      component BK32Bit is
25          port (a,b : in std_logic_vector(31 downto 0);
26              cout: out std_logic_vector(31 downto 0 );
27              pi: out std_logic_vector(31 downto 0 )
28              );
29      end component BK32Bit;
30
31      begin
32
33      carry_generatetion_stage: BK32Bit port map ( a => a , b => b , cout => ci, pi => pi ) ;

```

```

34
35 loop1: for i in 1 to 31 generate
36   begin
37     sum_bits: tinyXOR port map ( a => pi(i), b => ci(i-1) , c => sum(i) ) ;
38   end generate loop1;
39
40 sum0: tinyXOR port map ( a => a(0), b => b(0) , c => sum(0) ) ;
41
42 end Behave;

```

---

## Tracefile generation for testbench and testing

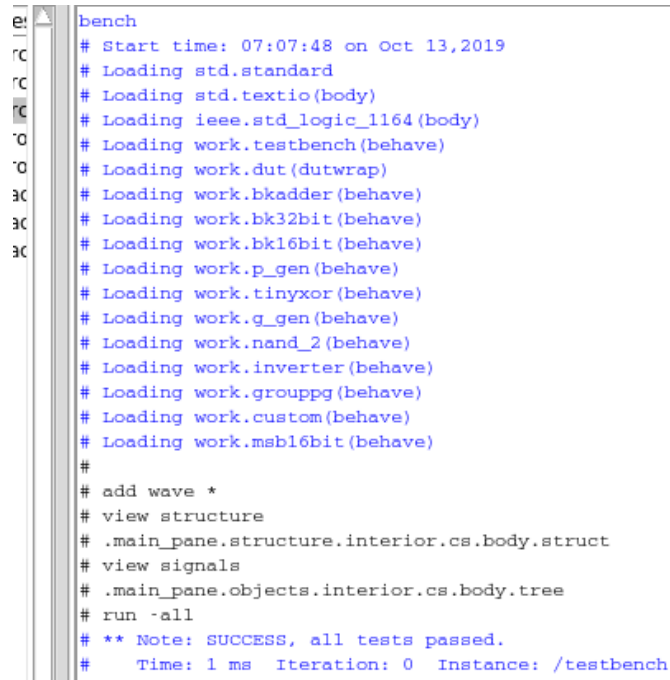
---

```

1 import random
2 f=open("tracefile.txt", "a+")
3 for i in range(50000):
4   a = random.randint(1,2147483648)
5   b = random.randint(1,2147483648)
6   f.write("{:032b}".format(a)+"{:032b}".format(b)+" "+"{:032b}".format(a+b)+" "+"{:032b}".format(4294967295)+'\n')

```

---



```

bench
# Start time: 07:07:48 on Oct 13,2019
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.testbench(behave)
# Loading work.dut(dutwrap)
# Loading work.bkadder(behave)
# Loading work.bk32bit(behave)
# Loading work.bk16bit(behave)
# Loading work.p_gen(behave)
# Loading work.tinyxor(behave)
# Loading work.g_gen(behave)
# Loading work.nand_2(behave)
# Loading work.inverter(behave)
# Loading work.grouppg(behave)
# Loading work.custom(behave)
# Loading work.msb16bit(behave)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#   Time: 1 ms  Iteration: 0  Instance: /testbench

```

Figure 3: Simulation result with 50000 test cases

## Critical Path and timing analysis

Critical path is the longest delay path and in our case it is the path by the carry of **30th bit** ( $G_{30}$ ) is generated and the figure is shown below.

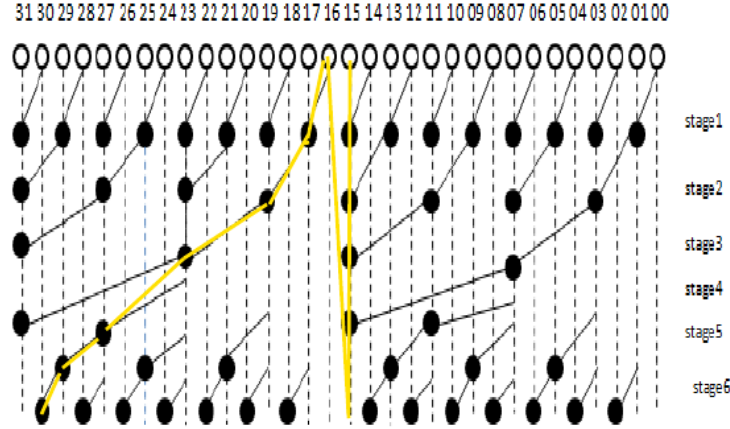


Figure 4: Critical Path

Here Black dots are my Brent-kung cell for P,G generation in carry generation stage which take **300ps** (Custom + Inverter).

So total time taken in **carry generation** is  $11 \times 300 = \mathbf{3300ps}$

Time taken in pre-processing stage in **intial p,g generation** (nand + Inverter) is  $150 + 100 = \mathbf{250 ps}$

Finally, time taken in **sum bits generation** (post-processing stage) is one xor which is **200 ps**.

Hence the time which we guarantee that the addition will be complete =  $3300 + 250 + 200 = \mathbf{3750 ps}$

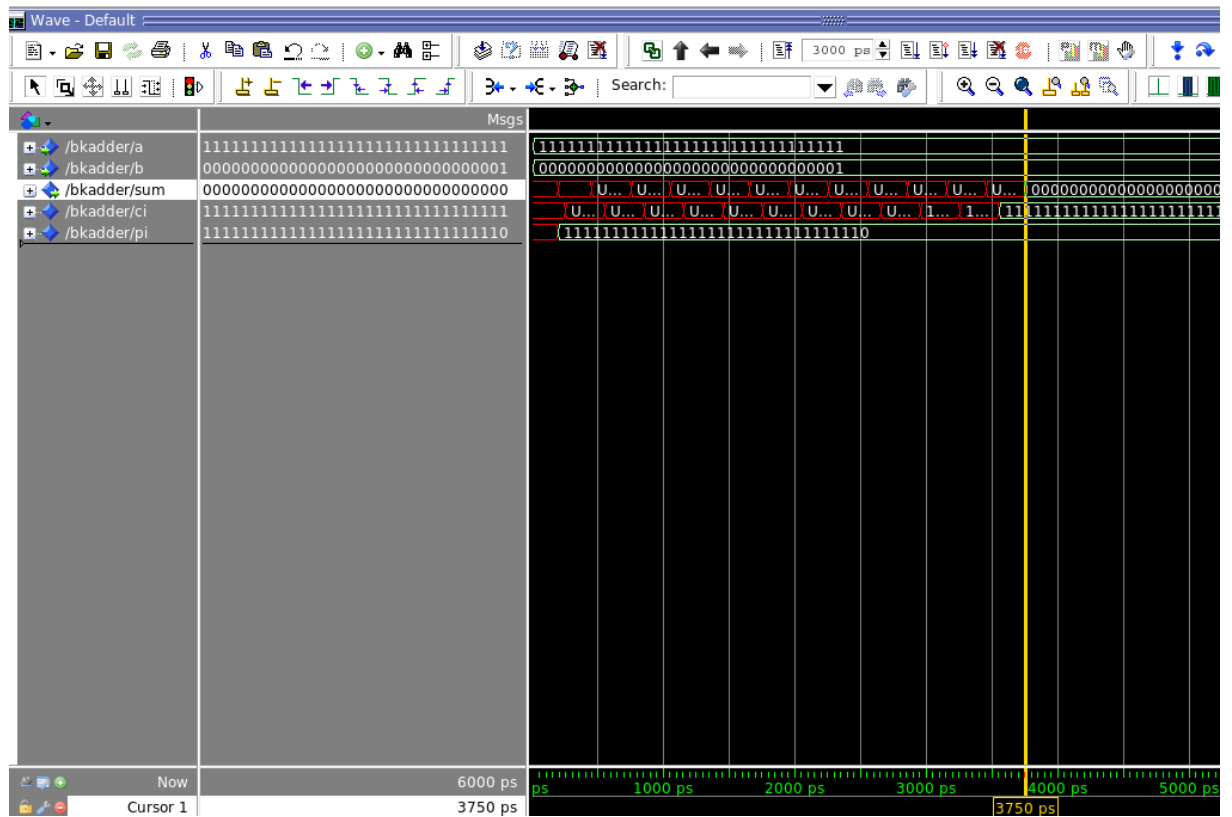


Figure 5: Delay of Critical Path