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

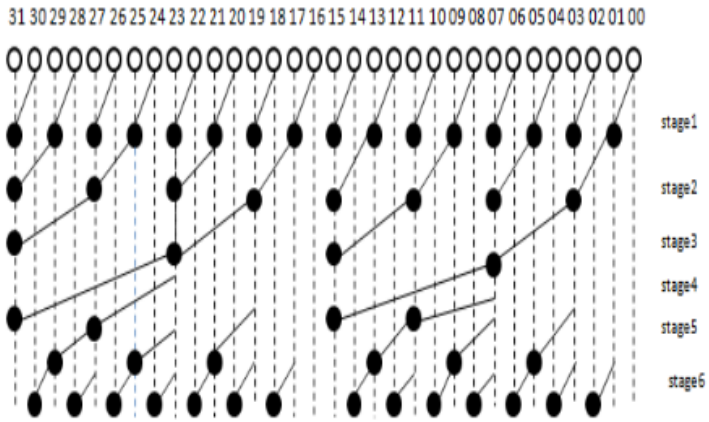


Figure 1: 32-bit Brent-Kung adder

Common Gates implementation in VHDL

```
-----------------------------------------------
library std;
use std.standard.all;
-------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
entity Inverter is
  port (a: in std_logic;
         b: out std_logic);
end entity Inverter;
architecture Behave of Inverter is
begin
  b <= not a after 100ps;   -----Delay of Iinverter gate
end Behave;

-------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
entity NOR_2 is
  port (a, b: in std_logic;
```

```
          c: out std_logic);
end entity NOR_2;
architecture Behave of NOR_2 is
begin
  c <= not(a or b) after 150ps; -----Delay of NOR gate
end Behave;

-------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
entity NAND_2 is
  port (a, b: in std_logic;
          c: out std_logic);
end entity NAND_2;
architecture Behave of NAND_2 is
begin
  c <= not (a and b) after 150ps;   -----Delay of NAND gate
end Behave;

--------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
entity tinyXOR is
  port (a, b: in std_logic;
          c: out std_logic);
end entity tinyXOR;
architecture Behave of tinyXOR is
begin
  c <= (a xor b) after 200ps;   -----Delay of Tiny_XOR gate
end Behave;

---------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
entity Custom is
  port (a, b, c : in std_logic;
          d: out std_logic);
end entity Custom;
architecture Behave of Custom is
begin
  d <= not(a or (b and c)) after 200ps;
end Behave;
```

```
-------------------------------------------------------
```

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.

$$P_i = A_i \oplus B_i \tag{1}$$

$$G_i = A_i \cdot B_i \tag{2}$$

```
-------------------------------------------------------------------
library std;
use std.standard.all;

-----------------------------------------------------------------------------------------------
--////////////////////////////////////////Carry Propagation////////////////////////////////////////
-----------------------------------------Delay 200ps-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity P_gen is
        port( x,y: in std_logic ;
           p : out std_logic) ;
end entity P_gen;

architecture behave of P_gen is
        component tinyXOR is
   port (a, b: in std_logic;
         c: out std_logic);
   end component tinyXOR;

-----------------------------------------------
begin
propagation_generator: tinyXOR port map(a => x, b => y , c => p );

end behave;
-----------------------------------------------------------------------------------------------
--//////////////////////////////////////////////Carry Generation////////////////////////////////////////
-----------------------------------------------Delay 250ps-----------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------
entity G_gen is
        port( x,y: in std_logic ;
           g : out std_logic) ;
end entity G_gen;

architecture behave of G_gen is
signal gbar : std_logic;

        component NAND_2 is
   port (a, b: in std_logic;
         c: out std_logic);
   end component NAND_2;

        component Inverter is
   port (a: in std_logic;
         b: out std_logic);
   end component Inverter;

----------------------------------------------
begin

Carry_Generation: Nand_2 port map(a => x, b => y , c=> gbar);
inv: Inverter port map(a => gbar, b => g);

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.

$$P_{i:k} = P_{i:k} \oplus P_{j-1:k} \tag{3}$$

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

```
-----------------------------------------------------------------------------
library std;
use std.standard.all;

-----------------------------------------------------------------------------------------------------
--////////////////////////////////////Carry Propagation//////////////////////////////////////////
-----------------------------------------Delay 200ps-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity P_gen is
        port( x,y: in std_logic ;
           p : out std_logic) ;
end entity P_gen;

architecture behave of P_gen is
        component tinyXOR is
   port (a, b: in std_logic;
         c: out std_logic);
   end component tinyXOR;

-----------------------------------------------
begin
propagation_generator: tinyXOR port map(a => x, b => y , c => p );

end behave;
-----------------------------------------------------------------------------------------------------
--//////////////////////////////////////////Carry Generation//////////////////////////////////////////
---------------------------------------------Delay 250ps-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity G_gen is
        port( x,y: in std_logic ;
           g : out std_logic) ;
end entity G_gen;

architecture behave of G_gen is
signal gbar : std_logic;

        component NAND_2 is
   port (a, b: in std_logic;
         c: out std_logic);
   end component NAND_2;
```

```
        component Inverter is
   port (a: in std_logic;
         b: out std_logic);
   end component Inverter;

-----------------------------------------------
begin

Carry_Generation: Nand_2 port map(a => x, b => y , c=> gbar);
inv: Inverter port map(a => gbar, b => g);

end behave;
-------------------------------------------------------------------------
----------------------------Basic Cell for Bent-Kung--------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------
entity groupPG is
        port( g1,p1,g2,p2: in std_logic ;
           gout,pout : out std_logic) ;
end entity groupPG;

architecture behave of groupPG is
        signal gbar,pbar : std_logic;

        component NAND_2 is
   port (a, b: in std_logic;
         c: out std_logic);
   end component NAND_2;

        component Inverter is
   port (a: in std_logic;
         b: out std_logic);
   end component Inverter;

        component Custom is
   port (a, b, c : in std_logic;
         d: out std_logic);
   end component Custom;

------------------------------
begin
g_new: Custom port map(a => g2, b =>g1, c => p2, d => gbar  );  --------g2+ g1*p2
inv1: Inverter port map(a => gbar, b => gout);        --Delay 300ps
```

```
p_new: Nand_2 port map(a => p1, b => p2 , c=> pbar);
inv2: Inverter port map(a => pbar, b => pout);        --Delay 250ps

end behave;
```

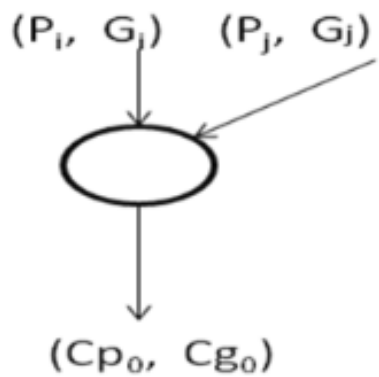


Figure 2: GroupPG block

16-Bit Brent-Kung group PG generation

```
----------------------------------------------
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity BK16Bit is
  port (a,b : in std_logic_vector(15 downto 0);
        g_sig,p_sig: out std_logic_vector(15 downto 0 );
                  p_vec:  out std_logic_vector(15 downto 0 )
                  );
end entity BK16Bit;

architecture Behave of BK16Bit is

signal g0,p0 ,gout,pout  : std_logic_vector(15 downto 0) ;
signal gs1,ps1 : std_logic_vector(7 downto 0) ;
```

```
signal gs2,ps2 : std_logic_vector(3 downto 0) ;
signal gs3,ps3 : std_logic_vector(1 downto 0) ;
signal gs4,ps4 : std_logic_vector(1 downto 0) ;
signal gs5,ps5 : std_logic_vector(2 downto 0) ;

component P_gen is
        port( x,y: in std_logic ;
           p : out std_logic) ;
end component P_gen;

component G_gen is
        port( x,y: in std_logic ;
           g : out std_logic) ;
end component G_gen;

component groupPG is
        port( g1,p1,g2,p2: in std_logic ;
           gout,pout : out std_logic) ;
end component groupPG;

begin

-------------- P and G generation -------------------------------
 loop1: for i in 0 to 15 generate
 begin
  propagation1: P_gen port map ( x => a(i) , y => b(i) , p => p0(i) ) ;
  generation1 : G_gen port map ( x => a(i) , y => b(i) , g => g0(i) ) ;
 end generate loop1;
 ---------------------------------------------------------------------------

 ------------------ Groop PG Generation using Parallel Prefix for 16-bit Brent-Kung -------
 -----Stage 1
 loop2: for i in 0 to 7 generate
 begin
  propagation_after_stage1: groupPG port map ( g1 => g0(2*i) , p1 => p0(2*i) , g2 => g0(2*i+1),
 end generate loop2;

 -----Stage 2
 loop3: for i in 0 to 3 generate
 begin
  propagation_after_stage2: groupPG port map ( g1 => gs1(2*i) , p1 => ps1(2*i) , g2 => gs1(2*i+1
 end generate loop3;

 -----Stage 3
 loop4: for i in 0 to 1 generate
```

```
 begin
  propagation_after_stage3: groupPG port map ( g1 => gs2(2*i) , p1 => ps2(2*i) , g2 => gs2(2*i+1
 end generate loop4;

-----Stage 4

 propagation_after_stage4_1: groupPG port map ( g1 => gs3(0) , p1 => ps3(0) , g2 => gs3(1), p2 =
 propagation_after_stage4_0: groupPG port map ( g1 => gs3(0) , p1 => ps3(0) , g2 => gs2(2), p2 =

 ----Stage 5
 propagation_after_stage5_2: groupPG port map ( g2 => gs1(6) , p2 => ps1(6) , g1 => gs4(0), p1 =
 propagation_after_stage5_1: groupPG port map ( g2 => gs1(4) , p2 => ps1(4) , g1 => gs3(0), p1 =
 propagation_after_stage5_0: groupPG port map ( g2 => gs1(2) , p2 => ps1(2) , g1 => gs2(0), p1 =

 ----Allocating odd carries for using in even carries
gout(15) <= gs4(1) ;
gout(13) <= gs5(2) ;
gout(11) <= gs4(0) ;
gout(9)  <= gs5(1) ;
gout(7)  <= gs3(0) ;
gout(5)  <= gs5(0) ;
gout(3)  <= gs2(0) ;
gout(1)  <= gs1(0) ;

gout(0) <= g0(0) ;  --first carry
pout(0) <= p0(0) ;  --first carry

pout(15) <= ps4(1) ;
pout(13) <= ps5(2) ;
pout(11) <= ps4(0) ;
pout(9)  <= ps5(1) ;
pout(7)  <= ps3(0) ;
pout(5)  <= ps5(0) ;
pout(3)  <= ps2(0) ;
pout(1)  <= ps1(0) ;

----Stage 6
 loop5: for i in 1 to 7 generate
 begin
  propagation_after_stage6: groupPG port map ( g1 => gout(2*i-1) , p1 => pout(2*i-i) , g2 => g0(
 end generate loop5;

 g_sig <= gout ;
 p_sig <= pout ;
 p_vec <= p0 ;
```

```
 end Behave;
```

32-Bit Brent-Kung group PG generation

```
---------------------------------------------
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity BK32Bit is
  port (a,b : in std_logic_vector(31 downto 0);
        cout: out std_logic_vector(31 downto 0 );
                  pi: out std_logic_vector(31 downto 0 )        --just simple p not group p
                 );
end entity BK32Bit;

architecture Behave of BK32Bit is
signal g,p  : std_logic_vector(31 downto 0) ;
--signal temp1,temp2,temp3,temp4,temp5,temp6 : std_logic  ;

component BK16Bit is
  port (a,b : in std_logic_vector(15 downto 0);
        g_sig,p_sig: out std_logic_vector(15 downto 0 );
                  p_vec:  out std_logic_vector(15 downto 0 )
                 );
end component BK16Bit;

component MSB16Bit is
  port (a,b : in std_logic_vector(15 downto 0);
        g15,p15: in std_logic ;
        g_sig,p_sig: out std_logic_vector(15 downto 0 );
                  p_vec:  out std_logic_vector(15 downto 0 )
                  );
end component MSB16Bit;

component groupPG is
        port( g1,p1,g2,p2: in std_logic ;
           gout,pout : out std_logic) ;
end component groupPG;

begin
  LSB16_carry: BK16Bit port map ( a => a(15 downto 0) , b => b(15 downto 0), g_sig => g(15 downt
```

```
  cout(15 downto 0)  <= g(15 downto 0)  ;


  MSB16_carry: MSB16Bit port map ( a => a(31 downto 16), b => b(31 downto 16),g15 => g(15) , p15
  cout(31 downto 16) <= g(31 downto 16) ;


 end Behave;
```

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

```
----------------------------------------------
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity BKadder is
  port (a,b : in std_logic_vector(31 downto 0);
        sum: out std_logic_vector(31 downto 0 )
                --  cout: out std_logic
                 );
end entity BKadder;

architecture Behave of BKadder is

signal ci,pi : std_logic_vector(31 downto 0) ;

        component tinyXOR is
   port (a, b: in std_logic;
         c: out std_logic);
   end component tinyXOR;

component BK32Bit is
  port (a,b : in std_logic_vector(31 downto 0);
        cout: out std_logic_vector(31 downto 0 );
                 pi: out std_logic_vector(31 downto 0 )
                );
end component BK32Bit;

begin

carry_generatetion_stage: BK32Bit port map ( a => a , b => b , cout => ci, pi => pi ) ;
```

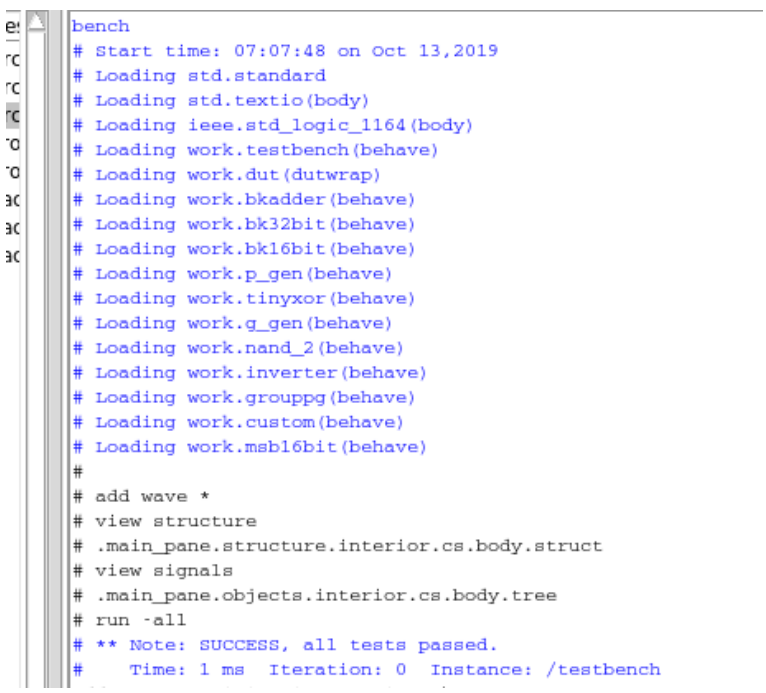
```
loop1: for i in 1 to 31 generate
 begin
  sum_bits: tinyXOR port map ( a => pi(i), b => ci(i-1) , c => sum(i) ) ;
 end generate loop1;

 sum0: tinyXOR port map ( a => a(0), b => b(0) , c => sum(0) ) ;

 end Behave;
```

Tracefile generation for testbench and testing

```
import random
f=open("tracefile.txt", "a+")
for i in range(50000):
  a = random.randint(1,2147483648)
  b = random.randint(1,2147483648)
  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 the path by the carry of **30th bit**(G_{30})is generated and the figure is shown below.

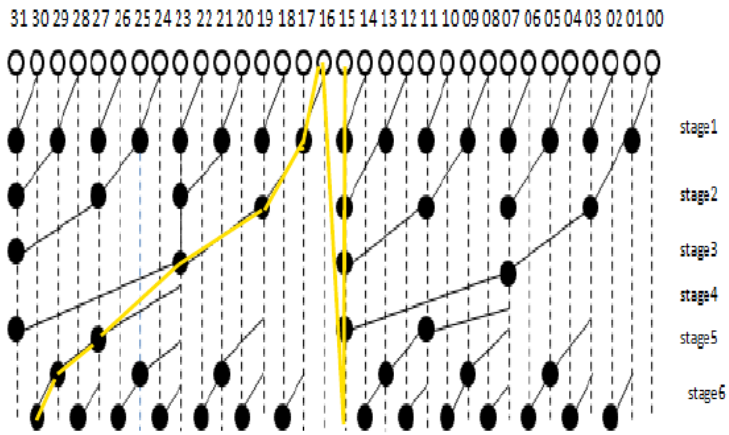


Figure 4: Critical Path

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 $11 \times 300 =$ **3300ps**
Time taken in pre-processing stage in **intial p,g generation** (nand + Inverter) is $150 + 100 =$ **250 ps**s
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 = **3750 ps**

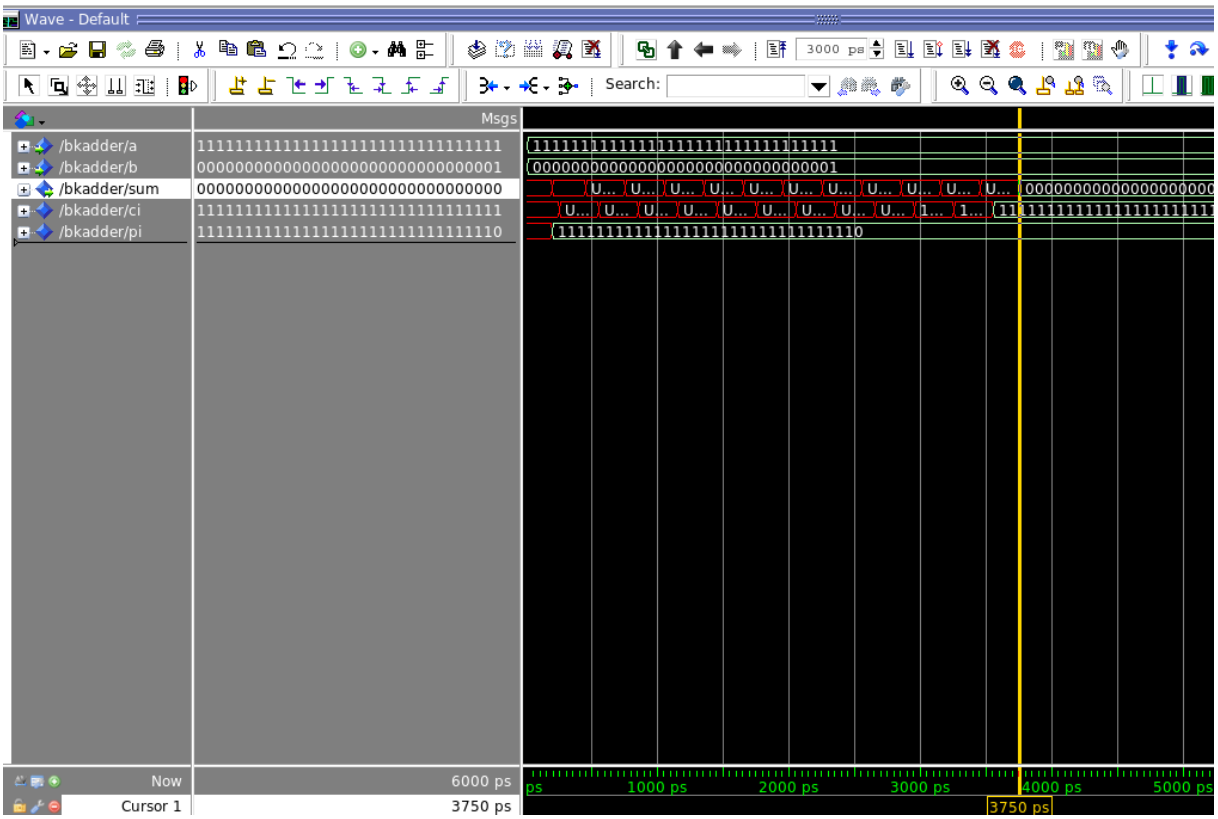


Figure 5: Delay of Critical Path