Name: Surbhika Rastogi

Roll No: 213070056

Date: 21-10-2021

Friday Oct. 15, 2021	EE 671: VLSI Design Assignment 4	Due on Friday Oct. 22, 2021

Q–1 Logarithmic adders use a tree structure to reduce the time taken for addition to a logarithmic function of the number of bits being added. Brent Kung adder is a simple example of this approach. Describe a 32 bit Brent Kung adder in VHDL and simulate it using a test bench.

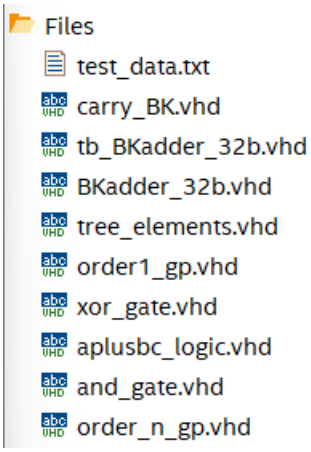
a) The adder needs logic functions AND, XOR, A + B.C to compute different orders of G, P and final sum and carry outputs. Write VHDL code in data flow style (using logic equations in assignments) to implement these functions. Each logic block should insert a delay of 100 ps in the assignments.

b) Using the above entities as components, write structural descriptions of each level of the tree for generating various orders of G and P values.
The right most blocks of all levels should use the available value of C0 to compute the output carry directly and term these as the G values for for computation of P and G for the next level.

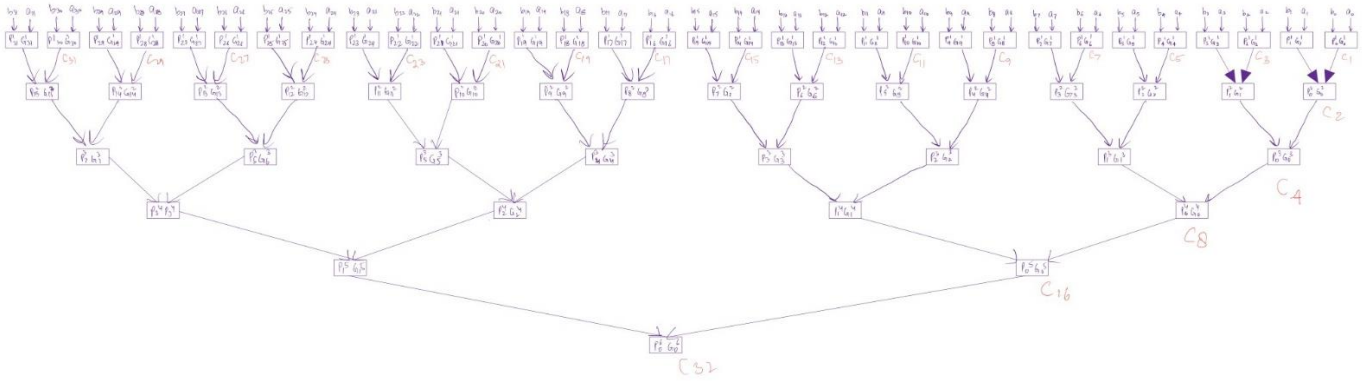
c) Using the outputs of the tree above, write structural VHDL code for generating the bit wise sum and carry values. Test the final adder with a test bench which reads pairs of 32 bit words and a single bit input carry from a file, adds them and compares the result with the expected 32 bit sum and 1 bit carry values stored in the same file.
It should use assert statements to flag errors if there is a mismatch between the computed sum/carry and the stored sum/carry.

Test the design with 64 randomly chosen pairs of numbers and input carry to be added.

Folder structure of the submitted files:

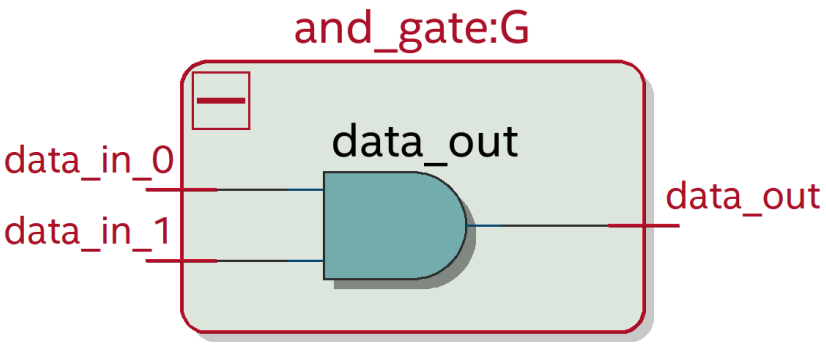


Brent Kung Adder Tree Diagram:

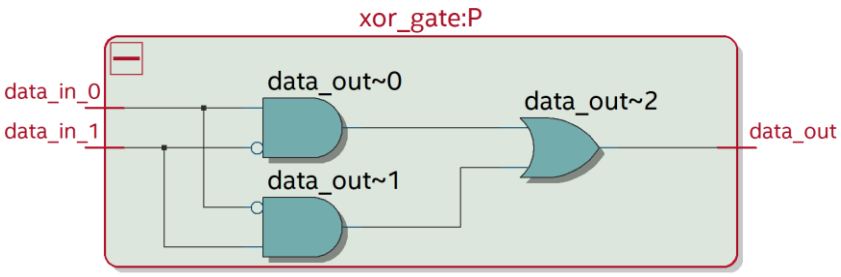


a) The adder needs logic functions AND, XOR, A + B.C to compute different orders of G, P and final sum and carry outputs.

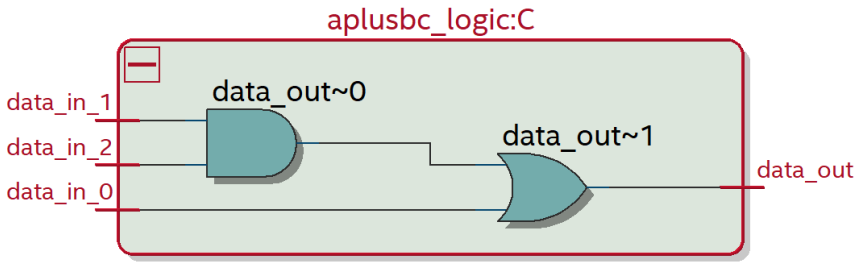
Refer to **and_gate.vhd** file for vhdl code of AND gate in dataflow style.



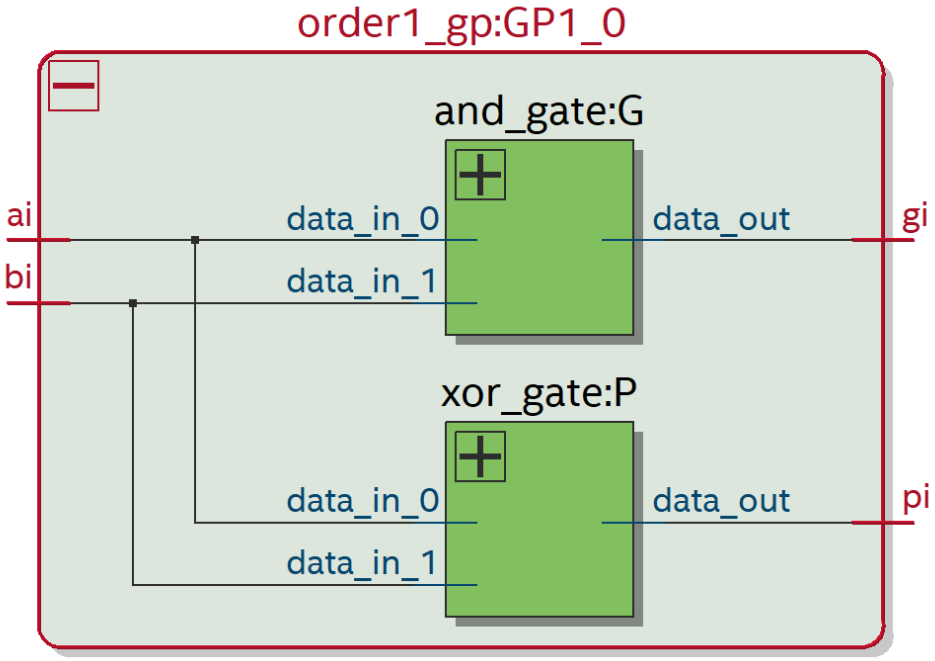
Refer to **xor_gate.vhd** file for vhdl code of XOR gate in dataflow style.



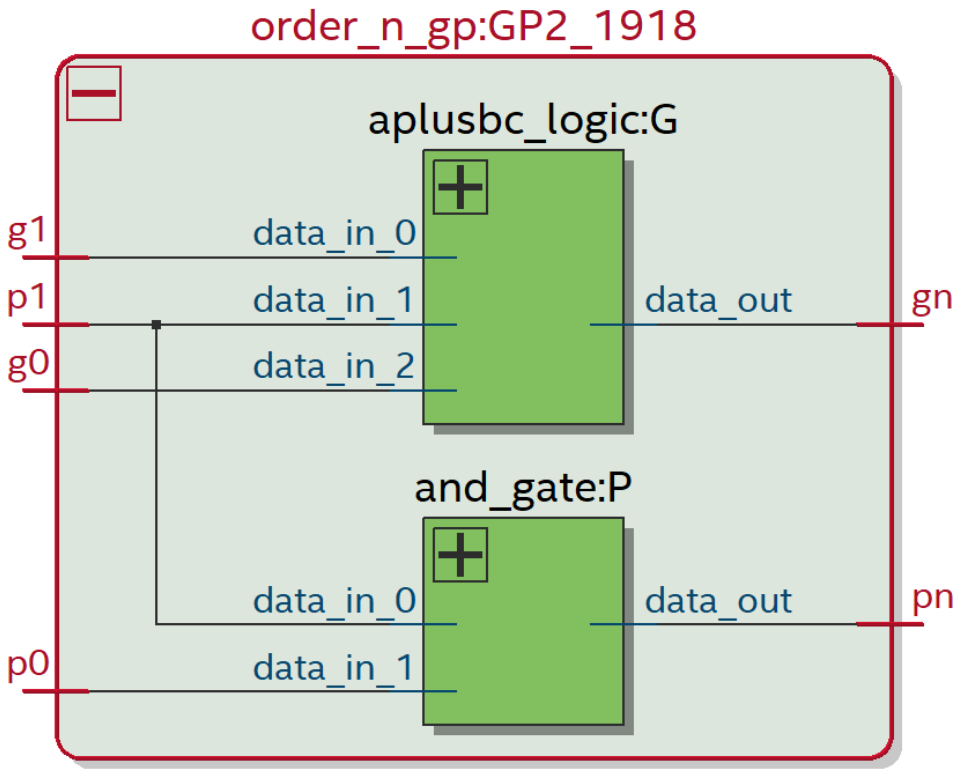
Refer to **aplusbc_logic.vhd** file for vhdl code of A+BC logic in dataflow style.

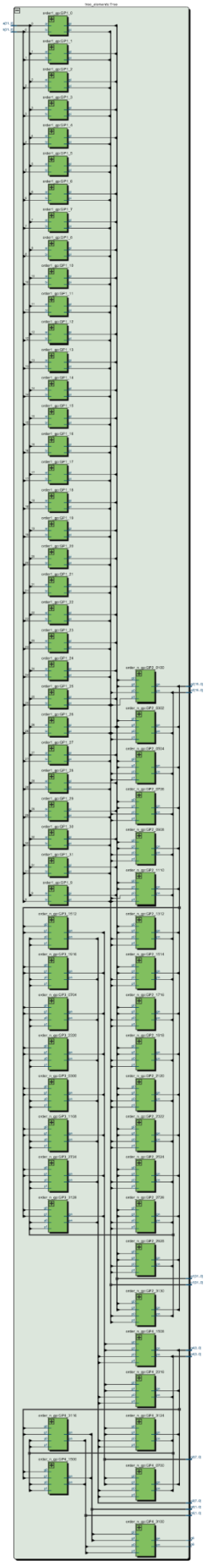


b) structural descriptions of level of the tree for generating **first orders** of G and P values:
For vhdl code, refer to **order1_gp.vhd**



structural descriptions of levels of the tree for generating **higher orders** of G and P values:
For vhdl code, refer to **order_n_gp.vhd**





Complete tree logic for calculation of all g and p values for 32-bit Brent Kung Adder:

It takes two 32-bit input vectors to determine the g and p values for the tree of depth 6 as shown in below RTL diagram: →

Refer to **tree_elements.vhd** for code of calculation of g and p values in structural style for all orders.

Equations used:

For 1st order:

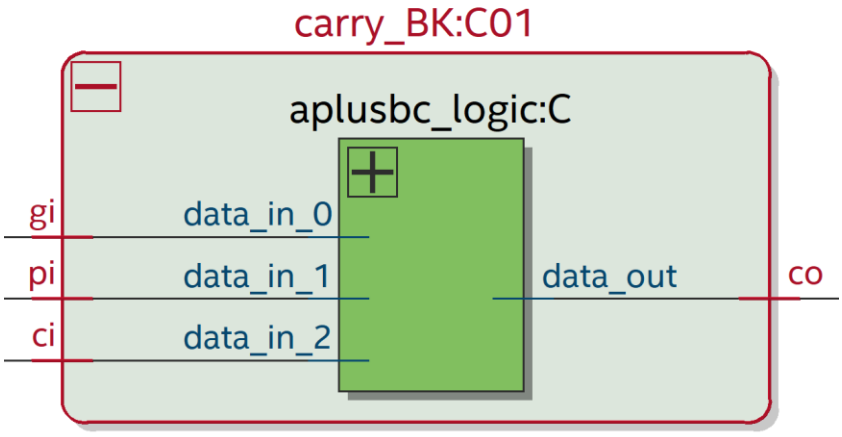
$$G_i = A_i \cdot B_i, \quad P_i = A_i \oplus B_i$$

For higher orders (>1) :

$$G^2_{2i+1,2i} = G^1_{2i+1} + P^1_{2i+1} \cdot G^1_{2i}, \qquad P^2_{2i+1,2i} = P^1_{2i+1} \cdot P^1_{2i}$$

Refer to **carry_BK.vhd** for code of calculating carry using g and p values.

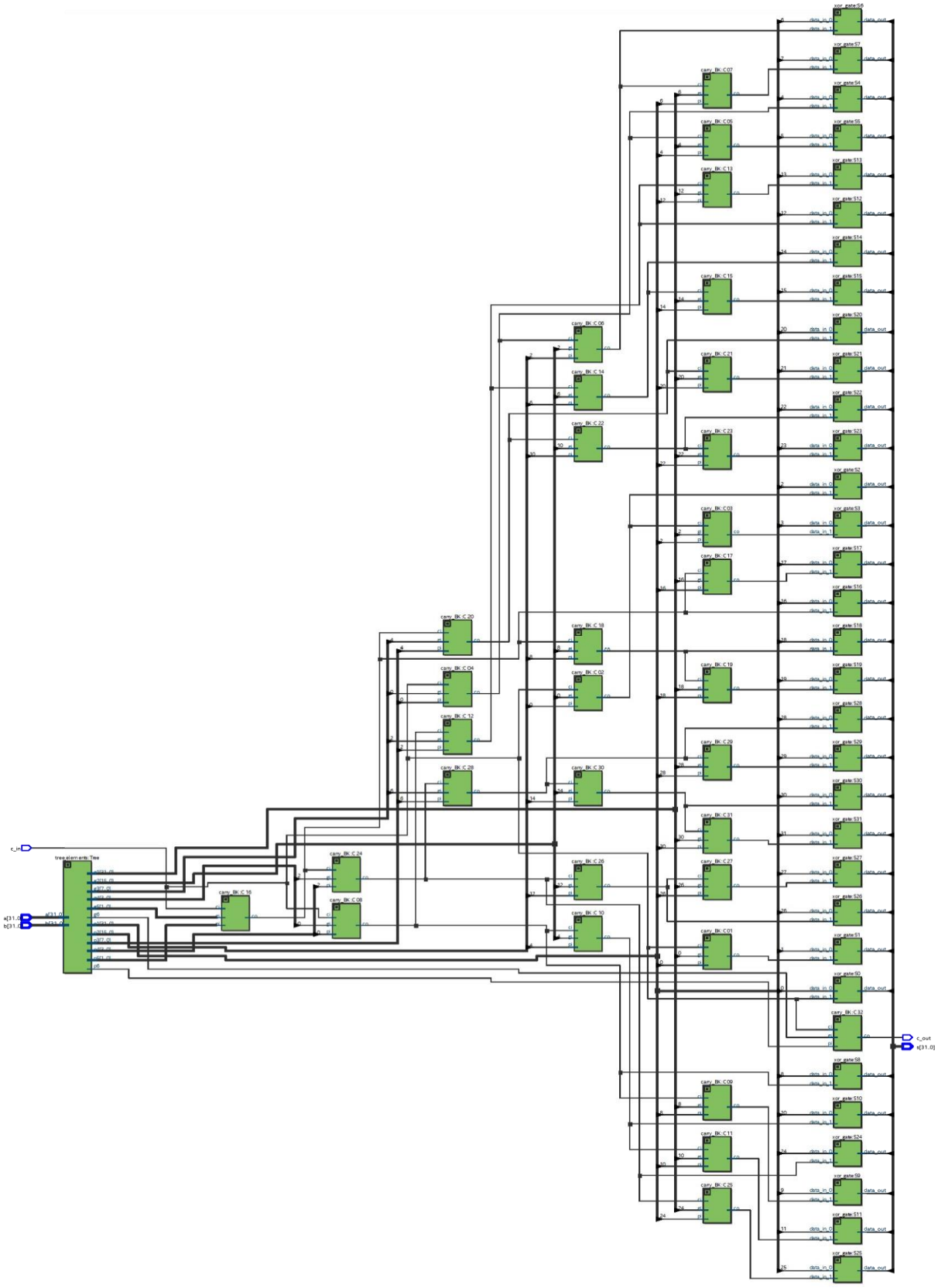
The rtl diagram for the same is as below:



c) Refer to **BKadder_32b.vhd** for Brent kung adder carry and sum values calculated using the structural style:

$$\text{Sum}_i = P_{i1} \oplus C_i.$$

The RTL logic diagram for the same is on next page:



Refer to **tb_BKadder_32b.vhd** for test bench code to check the Brent Kung Adder.

Simulation and testing:

64 randomly generated test combinations are saved in file **test_data.txt**

Format of the test_data file is:

1-bit carry out + 32-bit sum output + 1-bit carry in + 32-bit input1 + 32-bit input2

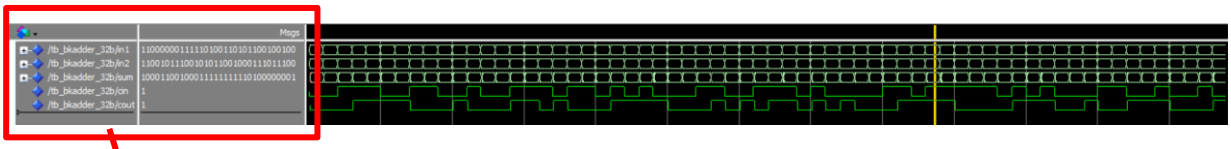
e.g., first test combination is stored as:

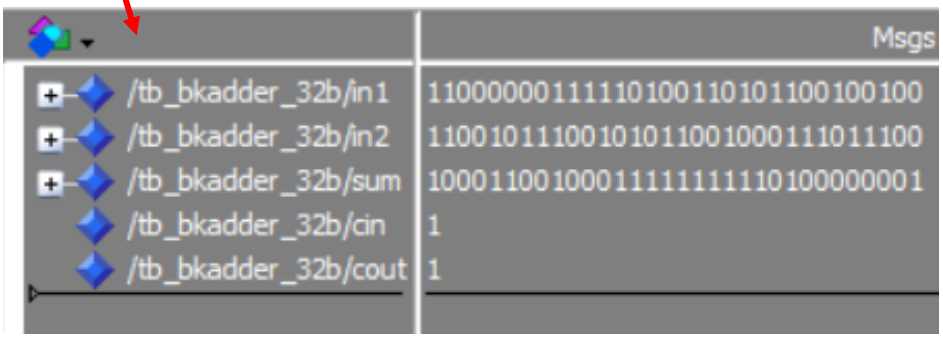
0111011010110100011100011000000110101100010011101101111010011000110011110000100110101101000010100000

Result of successful simulation using model-sim altera:

```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: End of file. All combinations check and success
#    Time: 640 ns  Iteration: 0  Instance: /tb_bkadder_32b
```

All 64 combinations checked:





Given each logic gate provides delay of 100ps.

We observe that the outputs (sum and carry out) settle after **700ps** of inputs applied.