EE2003: Computer Organisation

Assignment # 2

By :

Anchal Debnath

(EE21B017)

Question 22

Design a 16-bit Kogge-Stone prefix adder module in Verilog. The module should take two 16-bit inputs (A and B) and produce a 16-bit output Sum

Date:
1st October 2023

Contents

0.1 About Kogge Stone Adder

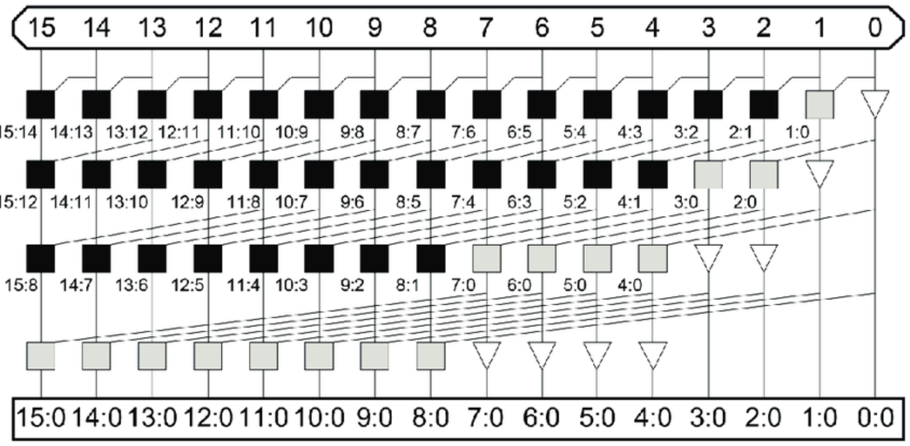


Figure 1: 16 bit Kogge stone Adder Schematic 1

A 16-bit Kogge-Stone adder is a high-performance digital circuit used for adding two 16-bit binary numbers efficiently. It relies on a parallel prefix tree structure to compute the addition result with minimal delay. Here's how the mechanism of a 16-bit Kogge-Stone adder works:

1. Input Signals: - A[15:0] and B[15:0]: The two 16-bit binary numbers to be added. - Cin: The input carry bit, which is typically set to zero for the first stage of the adder.

2. Generation of Generate (g) and Propagate (p) Signals: - For each bit position from the least significant bit (LSB) to the most significant bit (MSB): - g[i] (Generate Signal): Indicates whether a carry will be generated at bit position i if there is a carry-in at this position. It is calculated as g[i] = A[i] and B[i].
- p[i] (Propagate Signal): Indicates whether a carry from the previous bit position will propagate to bit position i. It is calculated as p[i] = A[i] xor B[i].

$$P_{i:j} = P_{i:k} P_{k-1:j}$$
$$G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$$
$$G_{-1} = C_{in}$$

3. Parallel Prefix Tree Structure: - The Kogge-Stone adder employs a tree-like structure to efficiently compute the carry signals and the sum outputs in parallel. - The tree structure consists of multiple levels, with each level processing a different set of bits. Level 0 handles the LSB, and each subsequent level processes additional bits.

4. Carry Computation in Each Level: - At each level of the tree, the generate (g) and propagate (p) signals are used to calculate the carry-out (Cout) and carry-in (Cin)

for the next level. - The carry-out (Cout) for each bit position is determined by the OR operation on the generate signal of the current bit and the AND operation between the propagate signal of the current bit and the carry-in (Cin) from the previous bit. This can be expressed as: - Cout[i] = g[i] or (p[i] and Cin)

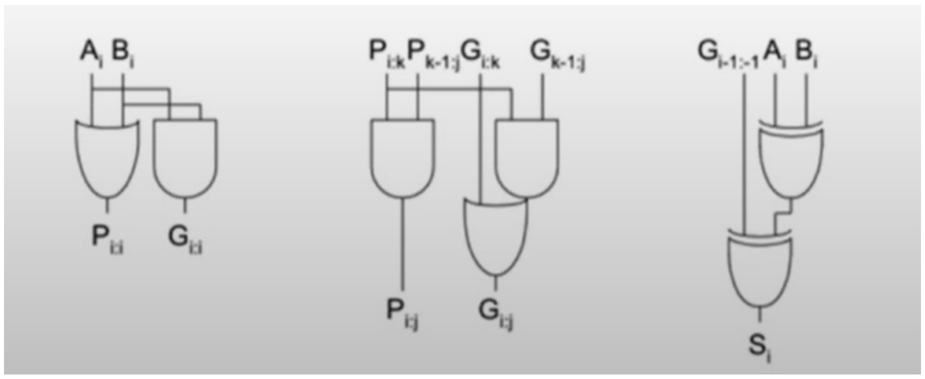
5. Sum Calculation: - The sum bit for each position is calculated by XORing the propagate signal of the current bit with the carry-in (Cin). This can be expressed as: - Sum[i] = p[i] ^{C}in

6. Output Carry (Cout): - The carry-out (Cout) of the most significant bit (MSB) represents the overall carry-out from the addition. It is directly taken as the output carry (Cout) of the adder.

7. Output Sum: - The sum bits (Sum[15:0]) represent the result of the addition of the two 16-bit binary numbers.

8. Final Output: - The final output consists of the 16-bit sum (Sum[15:0]) and the output carry (Cout), which can be used to detect overflow.

By employing this parallel prefix tree structure, the 16-bit Kogge-Stone adder can efficiently add two 16-bit binary numbers in a single clock cycle while minimizing the critical path delay, making it suitable for high-speed arithmetic operations in digital circuits.



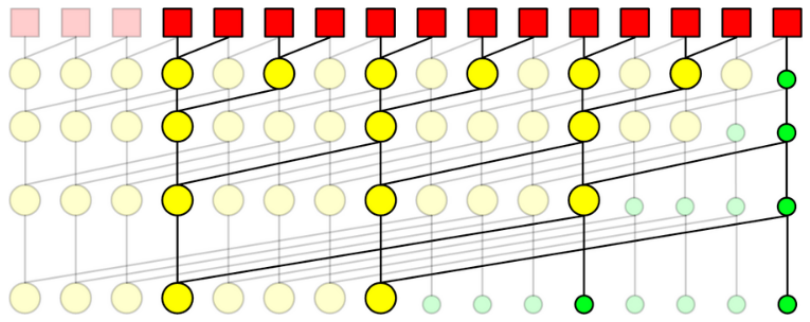


Figure 2: 16 bit Kogge stone Adder Schematic 1

1 Verilog Code

I used EDA Playground (an online simulator) for the code complilation and simulation.

1.0.1 Verilog Design Code

```
// 16 bit Kogge Stone Adder
module and2(input wire i0,i1, output wire o);
  assign o = i0 & i1;
endmodule

module or2(input wire i0, i1, output wire o);
  assign o = i0 | i1;
endmodule

module xor2 (input wire i0,i1, output wire o);
   assign o = i0 ^ i1;
endmodule

module pro_gen1(input wire x,y, output wire g,p);
  xor2 o(x,y,p);
  and2 a(x,y,g);
endmodule

module gen(input wire p10,g00,g10,output wire g11);
  wire temp;
  and2 a(g00,p10,temp);
  or2 o(g10,temp,g11);
endmodule

module pro_gen2(input wire p00,p10,g00,g10, output wire p11,g11);
  wire tem;
  and2 a1(p00,p10,p11);
  and2 a2(g00,p10,tem);
  or2 o1(tem,g10,g11);
endmodule

module sum(input wire x,y, output wire s);
  xor2 xor1(x,y,s);
endmodule

module koggestone16(input wire [15:0] A,B,input wire Cin,output
    wire [15:0] Sum,cout);
  wire p0[15:0],g0[15:0];
  wire p1[15:2],g1[15:1];
  wire p2[15:4],g2[15:2];
  wire p3[15:8],g3[15:4];
  wire g4[15:8];

  //0 level
  pro_gen1 Prop0(A[0],B[0],g0[0],p0[0]);
  pro_gen1 Prop1(A[1],B[1],g0[1],p0[1]);
```

```
pro_gen1 Prop2(A[2],B[2],g0[2],p0[2]);
pro_gen1 Prop3(A[3],B[3],g0[3],p0[3]);
pro_gen1 Prop4(A[4],B[4],g0[4],p0[4]);
pro_gen1 Prop5(A[5],B[5],g0[5],p0[5]);
pro_gen1 Prop6(A[6],B[6],g0[6],p0[6]);
pro_gen1 Prop7(A[7],B[7],g0[7],p0[7]);
pro_gen1 Prop8(A[8],B[8],g0[8],p0[8]);
pro_gen1 Prop9(A[9],B[9],g0[9],p0[9]);
pro_gen1 Prop10(A[10],B[10],g0[10],p0[10]);
pro_gen1 Prop11(A[11],B[11],g0[11],p0[11]);
pro_gen1 Prop12(A[12],B[12],g0[12],p0[12]);
pro_gen1 Prop13(A[13],B[13],g0[13],p0[13]);
pro_gen1 Prop14(A[14],B[14],g0[14],p0[14]);
pro_gen1 Prop15(A[15],B[15],g0[15],p0[15]);


// 1 level
gen pro1(p0[1],g0[0],g0[1],g1[1]);
pro_gen2 pro2(p0[1],p0[2],g0[1],g0[2],p1[2],g1[2]);
pro_gen2 pro3(p0[2],p0[3],g0[2],g0[3],p1[3],g1[3]);
pro_gen2 pro4(p0[3],p0[4],g0[3],g0[4],p1[4],g1[4]);
pro_gen2 pro5(p0[4],p0[5],g0[4],g0[5],p1[5],g1[5]);
pro_gen2 pro6(p0[5],p0[6],g0[5],g0[6],p1[6],g1[6]);
pro_gen2 pro7(p0[6],p0[7],g0[6],g0[7],p1[7],g1[7]);
pro_gen2 pro8(p0[7],p0[8],g0[7],g0[8],p1[8],g1[8]);
pro_gen2 pro9(p0[8],p0[9],g0[8],g0[9],p1[9],g1[9]);
pro_gen2 pro10(p0[9],p0[10],g0[9],g0[10],p1[10],g1[10]);
pro_gen2 pro11(p0[10],p0[11],g0[10],g0[11],p1[11],g1[11]);
pro_gen2 pro12(p0[11],p0[12],g0[11],g0[12],p1[12],g1[12]);
pro_gen2 pro13(p0[12],p0[13],g0[12],g0[13],p1[13],g1[13]);
pro_gen2 pro14(p0[13],p0[14],g0[13],g0[14],p1[14],g1[14]);
pro_gen2 pro15(p0[14],p0[15],g0[14],g0[15],p1[15],g1[15]);

// 2 level
gen prop1(p1[2],g0[0],g1[2],g2[2]);
gen prop2(p1[3],g1[1],g1[3],g2[3]);
pro_gen2 prop3(p1[2],p1[4],g1[2],g1[4],p2[4],g2[4]);
pro_gen2 prop4(p1[3],p1[5],g1[3],g1[5],p2[5],g2[5]);
pro_gen2 prop5(p1[4],p1[6],g1[4],g1[6],p2[6],g2[6]);
pro_gen2 prop6(p1[5],p1[7],g1[5],g1[7],p2[7],g2[7]);
pro_gen2 prop7(p1[6],p1[8],g1[6],g1[8],p2[8],g2[8]);
pro_gen2 prop8(p1[7],p1[9],g1[7],g1[9],p2[9],g2[9]);
pro_gen2 prop9(p1[8],p1[10],g1[8],g1[10],p2[10],g2[10]);
pro_gen2 prop10(p1[9],p1[11],g1[9],g1[11],p2[11],g2[11]);
pro_gen2 prop11(p1[10],p1[12],g1[10],g1[12],p2[12],g2[12]);
pro_gen2 prop12(p1[11],p1[13],g1[11],g1[13],p2[13],g2[13]);
pro_gen2 prop13(p1[12],p1[14],g1[12],g1[14],p2[14],g2[14]);
pro_gen2 prop14(p1[13],p1[15],g1[13],g1[15],p2[15],g2[15]);

// 3 level
gen pr1(p2[4],g0[0],g2[4],g3[4]);
gen pr2(p2[5],g1[1],g2[5],g3[5]);
gen pr3(p2[6],g2[2],g2[6],g3[6]);
gen pr4(p2[7],g2[3],g2[7],g3[7]);
pro_gen2 pr5(p2[4],p2[8],g2[4],g2[8],p3[8],g3[8]);
pro_gen2 pr6(p2[5],p2[9],g2[5],g2[9],p3[9],g3[9]);
pro_gen2 pr7(p2[6],p2[10],g2[6],g2[10],p3[10],g3[10]);
```

```
  pro_gen2 pr8(p2[7],p2[11],g2[7],g2[11],p3[11],g3[11]);
  pro_gen2 pr9(p2[8],p2[12],g2[8],g2[12],p3[12],g3[12]);
  pro_gen2 pr10(p2[9],p2[13],g2[9],g2[13],p3[13],g3[13]);
  pro_gen2 pr11(p2[10],p2[14],g2[10],g2[14],p3[14],g3[14]);
  pro_gen2 pr12(p2[11],p2[15],g2[11],g2[15],p3[15],g3[15]);

  //4 level
  gen po1(p3[8],g0[0],g3[8],g4[8]);
  gen po2(p3[9],g1[1],g3[9],g4[9]);
  gen po3(p3[10],g2[2],g3[10],g4[10]);
  gen po4(p3[11],g2[3],g3[11],g4[11]);
  gen po5(p3[12],g3[4],g3[12],g4[12]);
  gen po6(p3[13],g3[5],g3[13],g4[13]);
  gen po7(p3[14],g3[6],g3[14],g4[14]);
  gen po8(p3[15],g3[7],g3[15],g4[15]);

  //sum
  assign Sum[0] = p0[0];
  sum s1(p0[1],g0[0],Sum[1]);
  sum s2(p0[2],g1[1],Sum[2]);
  sum s3(p0[3],g2[2],Sum[3]);
  sum s4(p0[4],g2[3],Sum[4]);
  sum s5(p0[5],g3[4],Sum[5]);
  sum s6(p0[6],g3[5],Sum[6]);
  sum s7(p0[7],g3[6],Sum[7]);
  sum s8(p0[8],g3[7],Sum[8]);
  sum s9(p0[9],g4[8],Sum[9]);
  sum s10(p0[10],g4[9],Sum[10]);
  sum s11(p0[11],g4[10],Sum[11]);
  sum s12(p0[12],g4[11],Sum[12]);
  sum s13(p0[13],g4[12],Sum[13]);
  sum s14(p0[14],g4[13],Sum[14]);
  sum s15(p0[15],g4[14],Sum[15]);
  assign cout= g4[15];
endmodule


```

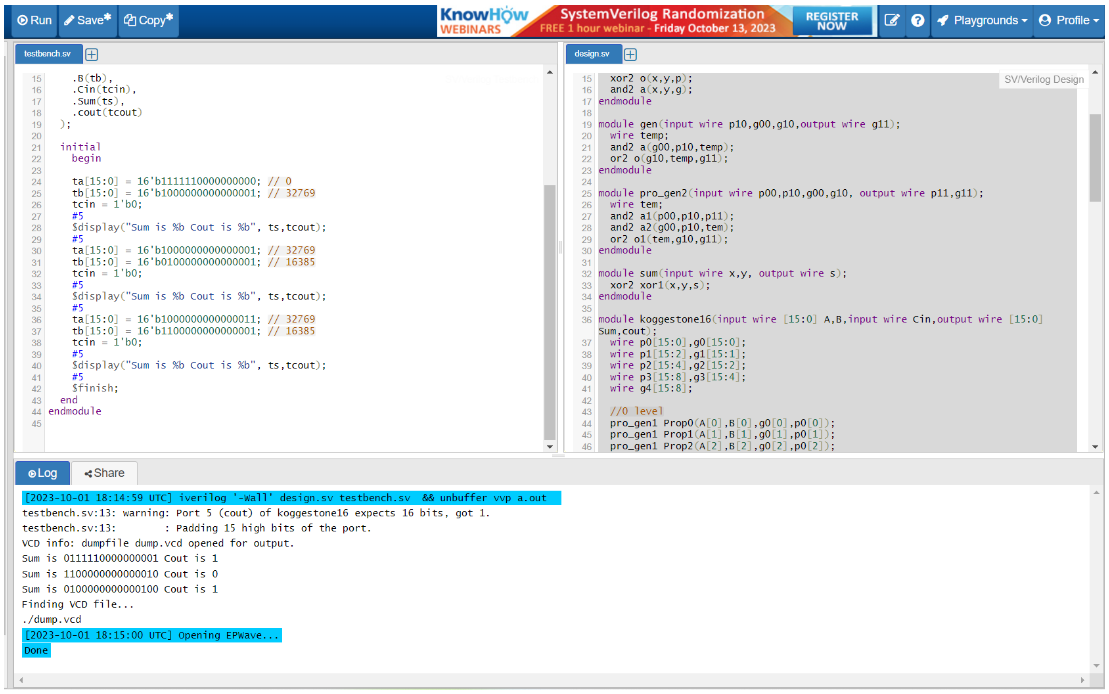
Listing 1: 1st problem code

1.0.2 Verilog Testbench Code

```
// 16 bit kogge stone adder testbench

module tb_kogge;
  reg [15:0] ta, tb;
  reg tcin;
  wire [15:0] ts;
  wire tcout;
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, tb_kogge);
  end

  koggestone16 final (
    .A(ta),
    .B(tb),
    .Cin(tcin),
    .Sum(ts),
    .cout(tcout)
  );

  initial
    begin

    ta[15:0] = 16'b1111110000000000; // 0
    tb[15:0] = 16'b1000000000000001; // 32769
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    ta[15:0] = 16'b1000000000000001; // 32769
    tb[15:0] = 16'b0100000000000001; // 16385
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    ta[15:0] = 16'b1000000000000011; // 32769
    tb[15:0] = 16'b1100000000000001; // 16385
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    $finish;
  end
endmodule



```

Listing 2: 1st problem code

2 Outputs



```
    .B(tb),
    .Cin(tcin),
    .Sum(ts),
    .cout(tcout)
  );

  initial
    begin

    ta[15:0] = 16'b1111110000000000; // 0
    tb[15:0] = 16'b1000000000000001; // 32769
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    ta[15:0] = 16'b1000000000000001; // 32769
    tb[15:0] = 16'b0100000000000001; // 16385
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    ta[15:0] = 16'b1000000000000011; // 32769
    tb[15:0] = 16'b1100000000000001; // 16385
    tcin = 1'b0;
    #5
    $display("Sum is %b Cout is %b", ts,tcout);
    #5
    $finish;
  end
endmodule
```

```
  xor2 o(x,y,p);
  and2 a(x,y,g);
endmodule

module gen(input wire p10,g00,g10,output wire g11);
  wire temp;
  and2 a(g00,p10,temp);
  or2 o(g10,temp,g11);
endmodule

module pro_gen2(input wire p00,p10,g00,g10, output wire p11,g11);
  wire tem;
  and2 a1(p00,p10,p11);
  and2 a2(g00,p10,tem);
  or2 o1(tem,g10,g11);
endmodule

module sum(input wire x,y, output wire s);
  xor2 xor1(x,y,s);
endmodule

module koggestone16(input wire [15:0] A,B,input wire Cin,output wire [15:0]
Sum,cout);
  wire p0[15:0],g0[15:0];
  wire p1[15:2],g1[15:1];
  wire p2[15:4],g2[15:2];
  wire p3[15:8],g3[15:4];
  wire g4[15:8];

  //0 level
  pro_gen1 Prop0(A[0],B[0],g0[0],p0[0]);
  pro_gen1 Prop1(A[1],B[1],g0[1],p0[1]);
  pro_gen1 Prop2(A[2],B[2],g0[2],p0[2]);
```

```
[2023-10-01 18:14:59 UTC] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:13: warning: Port 5 (cout) of koggestone16 expects 16 bits, got 1.
testbench.sv:13:        : Padding 15 high bits of the port.
VCD info: dumpfile dump.vcd opened for output.
Sum is 0111110000000001 Cout is 1
Sum is 1100000000000010 Cout is 0
Sum is 0100000000000100 Cout is 1
Finding VCD file...
./dump.vcd
[2023-10-01 18:15:00 UTC] Opening EPWave...
Done
```

Figure 3: Output in the terminal

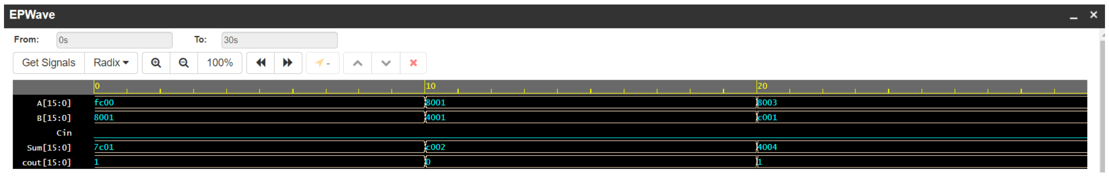


Figure 4: Output Waveform

3 References

Wikipedia
Youtube