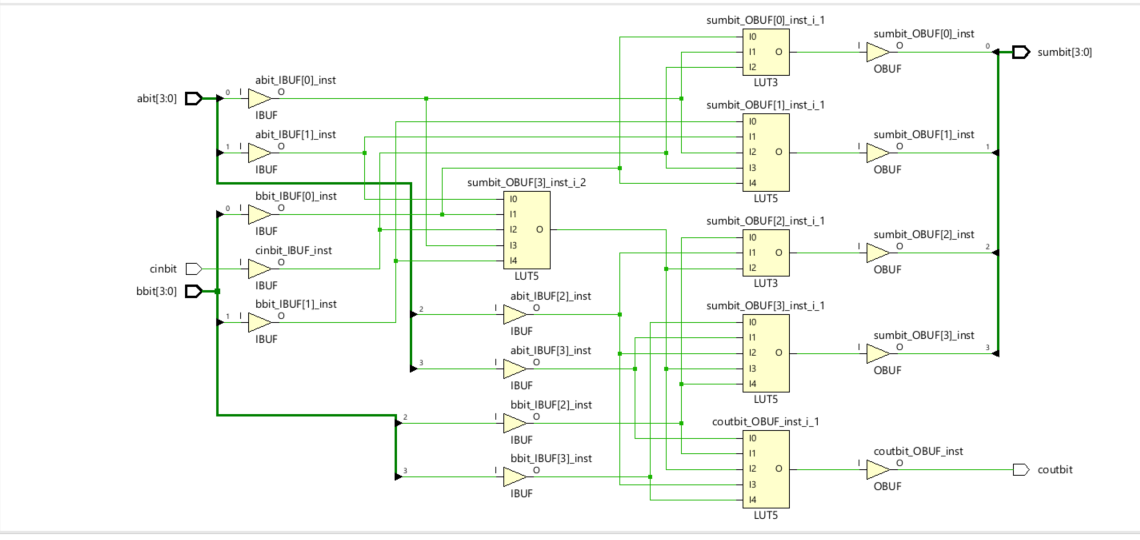
**ECE3300L Lab2 Group H Report (Sherwin Sathish & Mohamed Hamida)**

**SCHEMATIC(4-bitFA):**

****

*This schematic illustrates a 4-bit full adder. There are buffers for the inputs and outputs. There are 14 input and output ports that were addressed in the constraint file (with the inputs being switches and the outputs being LEDs) as shown below:*

***Xdc for 4-bit FA:***

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { abit[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { abit[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { abit[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { abit[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { bbit[0] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { bbit[1] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { bbit[2] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { bbit[3] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { cinbit }]; #IO\_L24N\_T3\_34 Sch=sw[8]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { coutbit }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

**Code Detail:**

`timescale 1ns / 1ps

module MAJORITY(

input x,

input y,

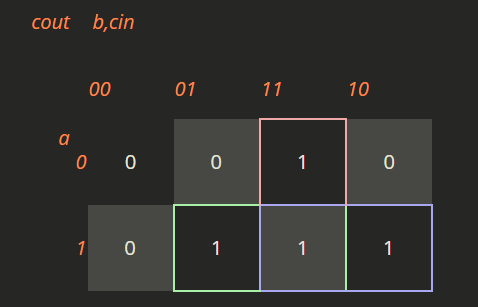
input z,

output p

);

assign p = (y&z)|(x&z)|(x&y);

endmodule



In this module x represents a, y represents b, z represents cin, and p represents cout. Since cout is the output in the majority encoder we have to create a kmap that can help implement the necessary gates and assign a value to p. The majority encoder applies to cout because the cout depends on the majority of the binary input values.

*`timescale 1ns / 1ps*

*module MAJORITY\_TB(*

*);*

*reg x\_tb,y\_tb,z\_tb;*

*wire p\_tb;*

*MAJORITY COMP*

*(*

*.x(x\_tb),*

*.y(y\_tb),*

*.z(z\_tb),*

*.p(p\_tb)*

*);*

*initial*

*begin: TST1*

*x\_tb = 1'b0;*

*y\_tb = 1'b0;*

*z\_tb = 1'b0;*

*#10*

*x\_tb = 1'b0;*

*y\_tb = 1'b1;*

*z\_tb = 1'b1;*

*#10*

*x\_tb = 1'b1;*

*y\_tb = 1'b0;*

*z\_tb = 1'b0;*

*#10*

*x\_tb = 1'b1;*

*y\_tb = 1'b0;*

*z\_tb = 1'b1;*

*#10*

*x\_tb = 1'b1;*

*y\_tb = 1'b1;*

*z\_tb = 1'b1;*

*#1000*

*$finish;*

*end*

*endmodule*

Here for the testbench we instantiated the values for the testbench by calling the values for the MAJORITY class, and tested different inputs to verify that the majority encoder works.

`timescale 1ns / 1ps

module FA(

input a,

input b,

input cin,

output sum,

output cout

);

wire temp;

xor(temp,a,b);

xor(sum,temp,cin);

MAJORITY UNIT

(

.x(a),

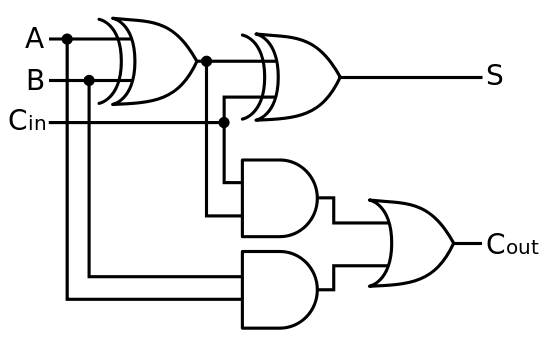
.y(b),

.z(cin),

.p(cout)

);

endmodule



Here for the FA module we call the MAJORITY class and we use gate level implementation to add the 2 xor gates to yield the sum, which completes the full adder. The temp wire is the output of the xor gate with inputs a and b. This temp wire and cin are used to form the xor gate for the sum. This creates the 1-bit FA product.

*`timescale 1ns / 1ps*

*module FA\_TB(*

*);*

*reg a\_tb,b\_tb,cin\_tb;*

*wire sum\_tb,cout\_tb;*

*FA COMP*

*(*

*.a(a\_tb),*

*.b(b\_tb),*

*.cin(cin\_tb),*

*.sum(sum\_tb),*

*.cout(cout\_tb)*

*);*

*initial*

*begin: TST1*

*a\_tb = 1'b1;*

*b\_tb = 1'b0;*

*cin\_tb = 1'b0;*

*#10*

*a\_tb = 1'b0;*

*b\_tb = 1'b1;*

*cin\_tb = 1'b1;*

*#10*

*a\_tb = 1'b1;*

*b\_tb = 1'b0;*

*cin\_tb = 1'b0;*

*#10*

*a\_tb = 1'b1;*

*b\_tb = 1'b0;*

*cin\_tb = 1'b1;*

*#10*

*a\_tb = 1'b1;*

*b\_tb = 1'b1;*

*cin\_tb = 1'b1;*

*#1000*

*$finish;*

*end*

*endmodule*

Here for the testbench we instantiated the values for the testbench by calling the values for the FA class, and tested different inputs to verify that the 1-bit FA works.

*`timescale 1ns / 1ps*

*module FAnbit*

*#(parameter DT = 8)*

*(*

*input cinbit,*

*input [DT-1:0] abit,*

*input [DT-1:0] bbit,*

*output [DT-1:0] sumbit,*

*output coutbit*

*);*

*wire [DT-1:0] bridge;*

*FA UNIT0 (*

*.cin(cinbit),*

*.a(abit[0]),*

*.b(bbit[0]),*

*.sum(sumbit[0]),*

*.cout(bridge[0])*

*);*

*genvar i;*

*generate*

*for(i=1;i<DT;i=i+1)*

*begin*

*FA UNIT(*

*.cin(bridge[i-1]),*

*.a(abit[i]),*

*.b(bbit[i]),*

*.sum(sumbit[i]),*

*.cout(bridge[i])*

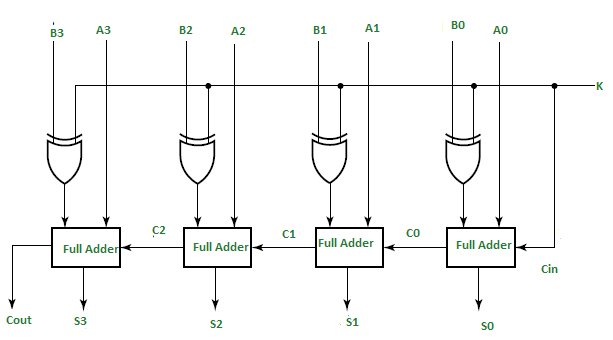
*);*

*end*

*endgenerate*

*assign coutbit=bridge[DT-1];*

*endmodule*



The FAnbits makes the full adder flexible for any amount of inputs. We instantiated the FA for the initial conditions in order to make it flexible for later. The for loop is for calling the FA module to create however many full adders we need. The bridge wire was created to establish the connection between each full adders’ cin and cout. The assign at the end takes care of the final cout wire. The parameter DT allows for flexibility because we can choose how many bit full adders we want.

`timescale 1ns / 1ps

*module FAnbit\_tb*

*#(parameter DT\_TB=4)*

*(*

*);*

*reg [DT\_TB-1:0] abit\_tb, bbit\_tb;*

*reg cinbit\_tb;*

*wire [DT\_TB-1:0] sumbit\_tb;*

*wire coutbit\_tb;*

*FAnbit*

*#(*

*.DT(DT\_TB)*

*)*

*FA\_TB\_NBITS*

*(*

*.cinbit(cinbit\_tb),*

*.abit(abit\_tb),*

*.bbit(bbit\_tb),*

*.sumbit(sumbit\_tb),*

*.coutbit(coutbit\_tb)*

*);*

*initial*

*begin: TST1*

*abit\_tb = 1;*

*bbit\_tb = 3;*

*cinbit\_tb = 0;*

*#10*

*abit\_tb = 4;*

*bbit\_tb = 6;*

*cinbit\_tb = 0;*

*#10*

*abit\_tb = 8;*

*bbit\_tb = 8;*

*cinbit\_tb = 0;*

*#1000*

*$finish;*

*end*

*endmodule*

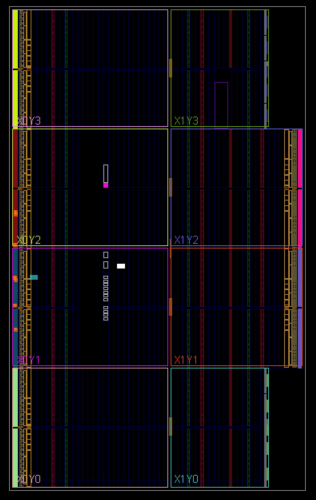
Here for the testbench we instantiated the values for the testbench by calling the values for the FAnits class, and tested different inputs to verify that the n-bit FA works.

For all the testbenches the inputs were reg(store values) and the output was wire.

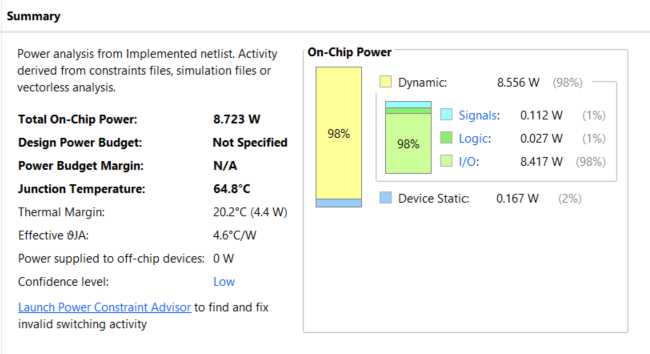
**Corner Cases/Error :**

***One corner case can apply for the n-bit FA testbench if you put an input b of 20 which requires more bits than available, an input of 4 will be shown because the MSB got cut off. However, in order to fix this the only thing that needs to be done is increase the parameter size of the nbits so that 20 will fit in the bit size, and will actually input the correct value of 20.***

**IMPLEMENTED DESIGN:**

****

**POWER SUMMARY:**

****

**UTILIZATION:**

****

**RESOURCE USAGE:**

****