Sukhbir Singh

Greg Schmit

Philip Horwitz

Project 1 part 2

Work Load report:

part1: SQM

This part of the project took a negligible amount of time. We learned that HDL has the power to reduce time needed to implement arithmetic operations.

part2: CA2

This part took between 0.5 to 1 hour to complete. Very simple function, we implemented using muxes.

part3: output analysis / serial1s / s1s / consecutive ones counter

This part several hours (4-6) of brainstorming on one group member's part. However, another group member was familiar with this problem and already had a basic idea of the implementation. Estimated time of writing code and testing is less than 1.5 hours.

part4: Controlled Fault Injector

This part took less than 0.5 hour of brainstorming to figure out. Coding and testing took near 1.5 hours.

part5: Not complete yet

**Modules:**

**------------------------------------------**

`include "ca2.sv"

`include "serial1s.sv"

/\*

main s84 module

test vector lines will look like A\_B\_op\_fLoc\_fType\_Yexpected\_Zexpected

\*/

module s84(input wire [7:0] A,

input wire [3:0] B,

input wire op,

input wire [2:0] f\_loc,

input wire [1:0] f\_type,

output wire [7:0] Cout,

output wire [7:0] Y,

output wire [3:0] Z);

wire [7:0] ca2Out;

wire [7:0] sqmOut;

wire [3:0] c1sOut;

ca2 ca2Device(A,B,ca2Out);

sqm sqmDevice(A, B, f\_loc, f\_type, Cout, sqmOut);

serial1s c1sDevice(Y,c1sOut);

assign Y = op ? ca2Out : sqmOut;

assign Z = c1sOut;

endmodule

/\*

sqm module which can compute 'normal'sqm or contain specified faults

\*/

module sqm(input wire [7:0] A,

input wire [3:0] B,

input wire [2:0] f\_loc,

input wire [1:0] f\_type,

output wire [7:0] Cout,

output wire [7:0] out);

//f\_loc goes into decoder that produces 8bit string with only one 1

//each bit of C(=Bsqrd) goes into 4to1 mux e.g. (C,0,1,C\_not) controlled by f\_type

//this 4to1 mux output goes into 2to1 mux containing (C,4to1muxout) controlled by the f\_loc\_decoder output

wire [7:0] loc;//output of a decoder, used to determine which bit to change.

wire [7:0] m41out; //output of 8 separate 4to1 muxes

wire [7:0] C; // intermediate, equal to B\*B

wire [7:0] m21out;// output of 8 separate 2to1 muxes

square multB(B,C);

decoder dec\_loc(f\_loc,loc);

mux4to1\_1bit m41\_0(C[0],1'b0,1'b1,~C[0],f\_type,m41out[0]);

mux4to1\_1bit m41\_1(C[1],1'b0,1'b1,~C[1],f\_type,m41out[1]);

mux4to1\_1bit m41\_2(C[2],1'b0,1'b1,~C[2],f\_type,m41out[2]);

mux4to1\_1bit m41\_3(C[3],1'b0,1'b1,~C[3],f\_type,m41out[3]);

mux4to1\_1bit m41\_4(C[4],1'b0,1'b1,~C[4],f\_type,m41out[4]);

mux4to1\_1bit m41\_5(C[5],1'b0,1'b1,~C[5],f\_type,m41out[5]);

mux4to1\_1bit m41\_6(C[6],1'b0,1'b1,~C[6],f\_type,m41out[6]);

mux4to1\_1bit m41\_7(C[7],1'b0,1'b1,~C[7],f\_type,m41out[7]);

mux2to1\_1bit m21\_0(C[0],m41out[0],loc[0],m21out[0]);

mux2to1\_1bit m21\_1(C[1],m41out[1],loc[1],m21out[1]);

mux2to1\_1bit m21\_2(C[2],m41out[2],loc[2],m21out[2]);

mux2to1\_1bit m21\_3(C[3],m41out[3],loc[3],m21out[3]);

mux2to1\_1bit m21\_4(C[4],m41out[4],loc[4],m21out[4]);

mux2to1\_1bit m21\_5(C[5],m41out[5],loc[5],m21out[5]);

mux2to1\_1bit m21\_6(C[6],m41out[6],loc[6],m21out[6]);

mux2to1\_1bit m21\_7(C[7],m41out[7],loc[7],m21out[7]);

assign Cout = m21out;

assign out = m21out % A;

endmodule

module square(input wire [3:0] B, output wire [7:0] C);

assign C = B\*B;

endmodule

module decoder(input wire [2:0] loc, output wire [7:0] out);

assign out[0] = ~loc[0] & ~loc[1] & ~loc[2];

assign out[1] = loc[0] & ~loc[1] & ~loc[2];

assign out[2] = ~loc[0] & loc[1] & ~loc[2];

assign out[3] = loc[0] & loc[1] & ~loc[2];

assign out[4] = ~loc[0] & ~loc[1] & loc[2];

assign out[5] = loc[0] & ~loc[1] & loc[2];

assign out[6] = ~loc[0] & loc[1] & loc[2];

assign out[7] = loc[0] & loc[1] & loc[2];

endmodule

**------------------------------------------------------**

module serial1s(input logic [7:0] a,

output logic [3:0] y);

int count;

int found;

logic [7:0] im[7];

always\_comb begin

im[0] = a & (a << 1);

im[1] = im[0] & (im[0] << 1);

im[2] = im[1] & (im[1] << 1);

im[3] = im[2] & (im[2] << 1);

im[4] = im[3] & (im[3] << 1);

im[5] = im[4] & (im[4] << 1);

im[6] = im[5] & (im[5] << 1);

if (a == 0)

y = 0;

else if (im[0] == 0)

y = 1;

else if (im[1] == 0)

y = 2;

else if (im[2] == 0)

y = 3;

else if (im[3] == 0)

y = 4;

else if (im[4] == 0)

y = 5;

else if (im[5] == 0)

y = 6;

else if (im[6] == 0)

y = 7;

else

y = 8;

end

endmodule

**------------------------------------------**

/\*

This is the module for a 2to1 mux

Used for 1-bit data paths

\*/

module mux2to1\_1bit(input wire i0,i1, op , output wire out);

assign out = op ? i1 : i0;

endmodule

/\*

This is the module for a 4to1 mux

Used for 1-bit data paths, built with three 2to1 muxes

\*/

module mux4to1\_1bit(input wire i0,i1,i2,i3,input wire [1:0] op , output wire out);

//intermediate wires for mux2to1 outputs

wire mux1out,mux2out;

//instantiate three 2by1 muxes

mux2to1\_1bit mux1(i0,i1,op[0],mux1out);

mux2to1\_1bit mux2(i2,i3,op[0],mux2out);

mux2to1\_1bit mux3(mux1out,mux2out,op[1],out);//out is final output

endmodule

/\*

module for CA2, uses eight 4to1 muxes

\*/

module ca2(input wire [7:0] A, input wire [3:0] B, output wire [7:0] Y);

//for the op input into 4to1 mux, we give {A[(i+1) % 8],A[i]} to find Y[i]

mux4to1\_1bit m0(B[0],B[1],B[2],B[3],A[1:0],Y[0]); // i = 0 --> (i+1) % 8 = 1

mux4to1\_1bit m1(B[0],B[1],B[2],B[3],A[2:1],Y[1]); // i = 1 --> (i+1) % 8 = 2

mux4to1\_1bit m2(B[0],B[1],B[2],B[3],A[3:2],Y[2]); // i = 2 --> (i+1) % 8 = 3

mux4to1\_1bit m3(B[0],B[1],B[2],B[3],A[4:3],Y[3]); // i = 3 --> (i+1) % 8 = 4

mux4to1\_1bit m4(B[0],B[1],B[2],B[3],A[5:4],Y[4]); // i = 4 --> (i+1) % 8 = 5

mux4to1\_1bit m5(B[0],B[1],B[2],B[3],A[6:5],Y[5]); // i = 5 --> (i+1) % 8 = 6

mux4to1\_1bit m6(B[0],B[1],B[2],B[3],A[7:6],Y[6]); // i = 6 --> (i+1) % 8 = 7

mux4to1\_1bit m7(B[0],B[1],B[2],B[3],{A[0],A[7]},Y[7]); // i = 7 --> (i+1) % 8 = 0

endmodule

**------------------------------------------------------**

`include "s84.sv"

module s84\_tb();

logic [7:0] A;

logic [3:0] B;

logic op;

logic [2:0] f\_loc;

logic [1:0] f\_type;

logic [7:0] C,C\_expected;

logic [7:0] Y, Y\_expected;

logic [3:0] Z, Z\_expected;

logic [37:0] testvector [29:0];

s84 s84alu(A, B, op, f\_loc, f\_type, C, Y, Z);

integer num;

initial

begin

$readmemb("./s84.tv",testvector);

for (num = 0; num<30; num = num + 1)

begin

{A, B, op, f\_loc, f\_type, C\_expected, Y\_expected, Z\_expected} = testvector[num]; #10;

//$display("num=%d",num+1);

//$display("A=%b B=%b op=%b f\_loc=%b f\_type=%b C\_exp=%b Y\_exp=%b Z\_exp=%d", A, B, op, f\_loc, f\_type, C\_expected, Y\_expected, Z\_expected);

//$display("C=%b Y=%d Z=%d",C,Y,Z);

//csv layout

$display("%b,%b,%b,%b,%b,%b,%b,%b", A, B, op, f\_loc, f\_type, C\_expected, Y\_expected, Z\_expected);

if (C != C\_expected)

begin

$display("C does not equal C\_exp line num=%d",num+1);

end

if (Y != Y\_expected)

begin

$display("Y does not equal Y\_exp line num=%d",num+1);

end

if (Z != Z\_expected)

begin

$display("Z does not equal Z\_exp line num=%d",num+1);

end

end//end for

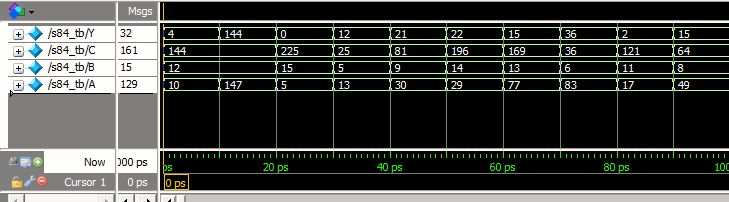
end

endmodule

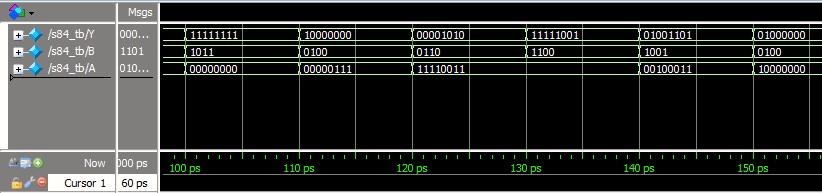
**---------------------------------------------------**

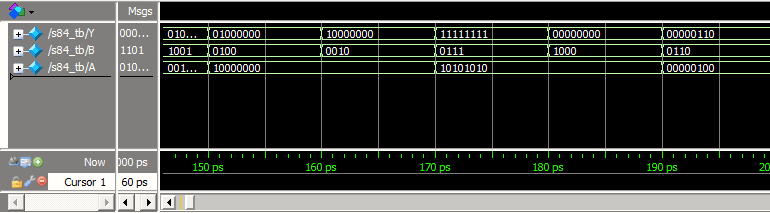
Waveforms:

SQM

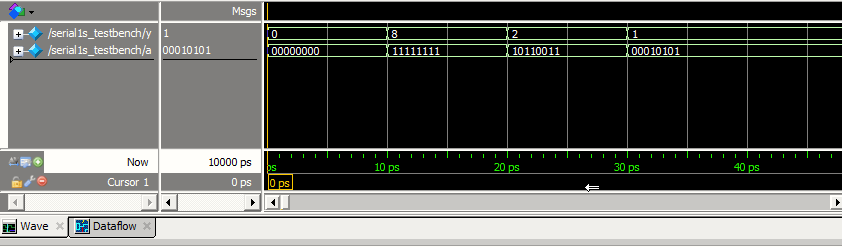
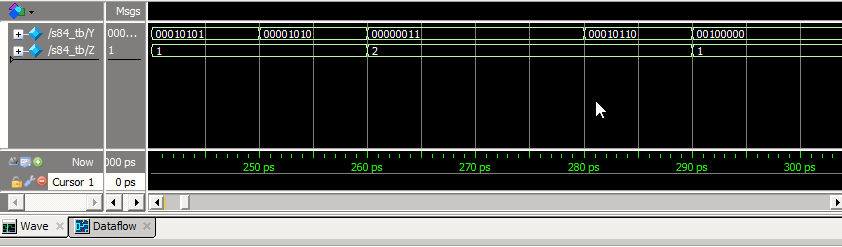
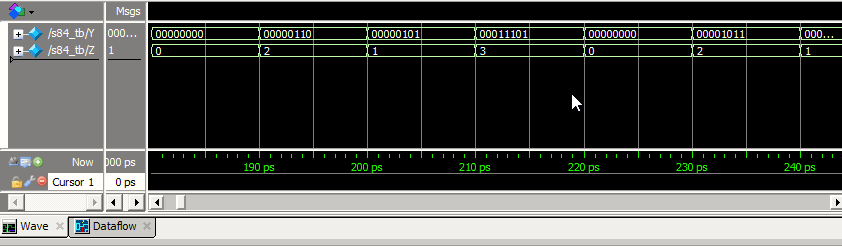
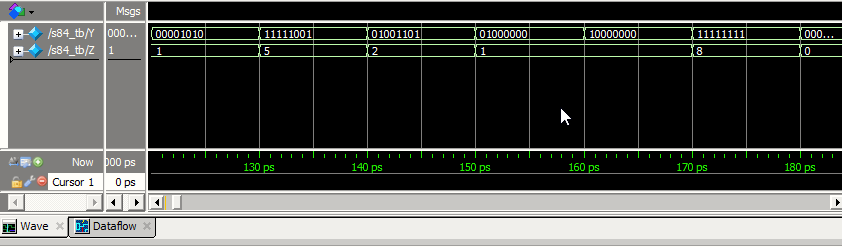
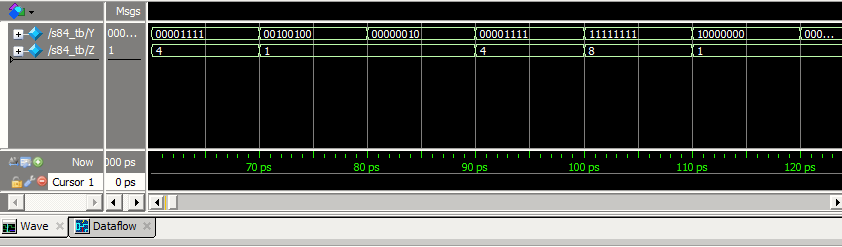
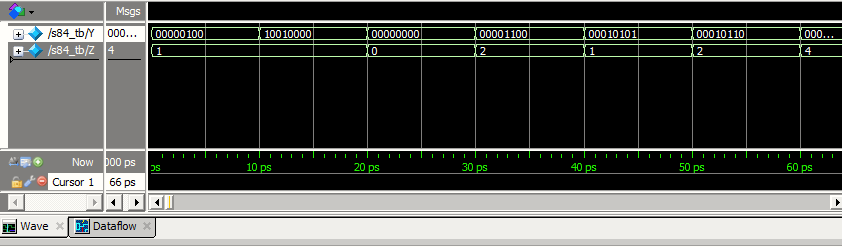


CA2



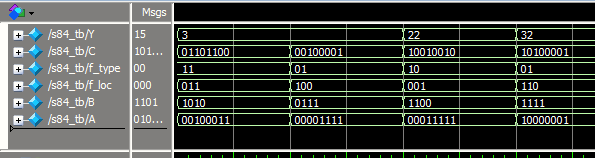
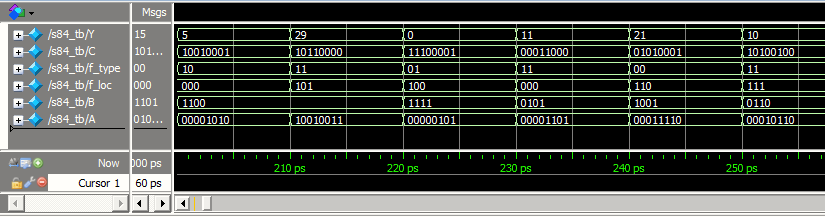


Serial1s



In the last image (above), ‘a’ is ‘Y’ and ‘y’ is Z

CFI



Testvector

A\_B\_op\_fLoc\_fType\_C\_Y\_Z

00001010\_1100\_0\_000\_00\_10010000\_00000100\_0001

10010011\_1100\_0\_000\_00\_10010000\_10010000\_0001

00000101\_1111\_0\_000\_00\_11100001\_00000000\_0000

00001101\_0101\_0\_000\_00\_00011001\_00001100\_0010

00011110\_1001\_0\_000\_00\_01010001\_00010101\_0001

00011101\_1110\_0\_000\_00\_11000100\_00010110\_0010

01001101\_1101\_0\_000\_00\_10101001\_00001111\_0100

01010011\_0110\_0\_000\_00\_00100100\_00100100\_0001

00010001\_1011\_0\_000\_00\_01111001\_00000010\_0001

00110001\_1000\_0\_000\_00\_01000000\_00001111\_0100

00000000\_1011\_1\_000\_00\_01111001\_11111111\_1000

00000111\_0100\_1\_000\_00\_00010000\_10000000\_0001

11110011\_0110\_1\_000\_00\_00100100\_00001010\_0001

11110011\_1100\_1\_000\_00\_10010000\_11111001\_0101

00100011\_1001\_1\_000\_00\_01010001\_01001101\_0010

10000000\_0100\_1\_000\_00\_00010000\_01000000\_0001

10000000\_0010\_1\_000\_00\_00000100\_10000000\_0001

10101010\_0111\_1\_000\_00\_00110001\_11111111\_1000

10101010\_1000\_1\_000\_00\_01000000\_00000000\_0000

00000100\_0110\_1\_000\_00\_00100100\_00000110\_0010

00001010\_1100\_0\_000\_10\_10010001\_00000101\_0001

10010011\_1100\_0\_101\_11\_10110000\_00011101\_0011

00000101\_1111\_0\_100\_01\_11100001\_00000000\_0000

00001101\_0101\_0\_000\_11\_00011000\_00001011\_0010

00011110\_1001\_0\_110\_00\_01010001\_00010101\_0001

00010110\_0110\_0\_111\_11\_10100100\_00001010\_0001

00100011\_1010\_0\_011\_11\_01101100\_00000011\_0010

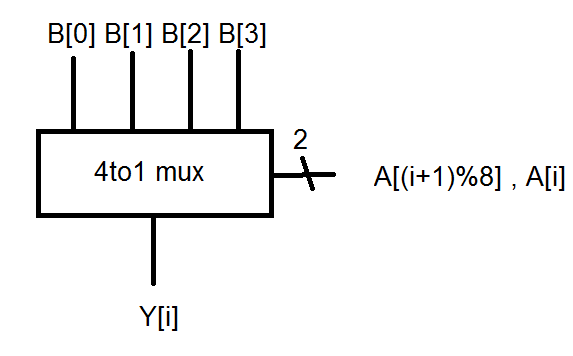
00001111\_0111\_0\_100\_01\_00100001\_00000011\_0010

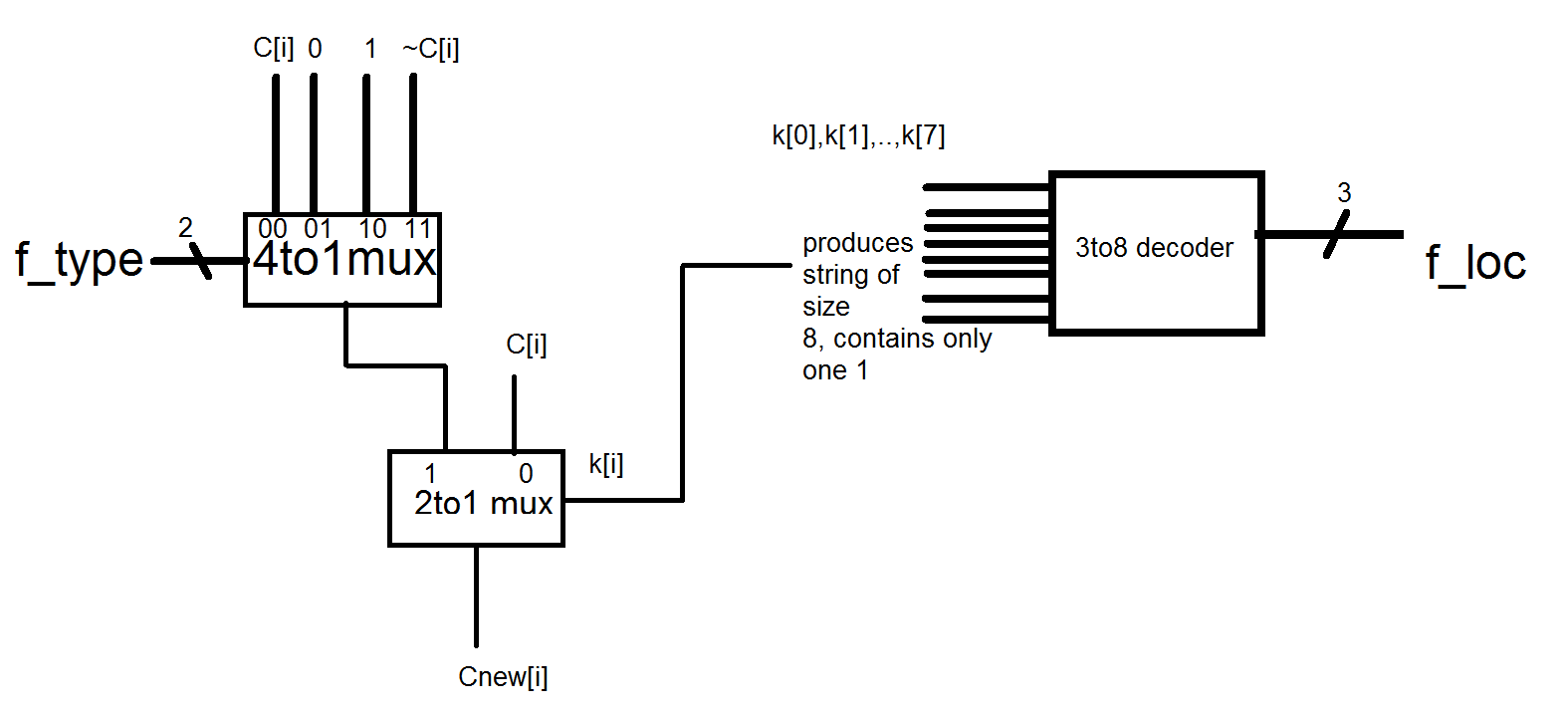
00011111\_1100\_0\_001\_10\_10010010\_00010110\_0010

10000001\_1111\_0\_110\_01\_10100001\_00100000\_0001

**Designs:**

CA2 below



CFI below