ECE 469 HDL
Project 1, Part 2
Tamanna Ravi Rupani
UIN: 665679988
S84 (Special 8/4 – bit ALU)

Components finished:

1. Function SQM
2. Function CA2
3. Output Analysis
4. Controlled fault injection

Number of hours spent on each component:

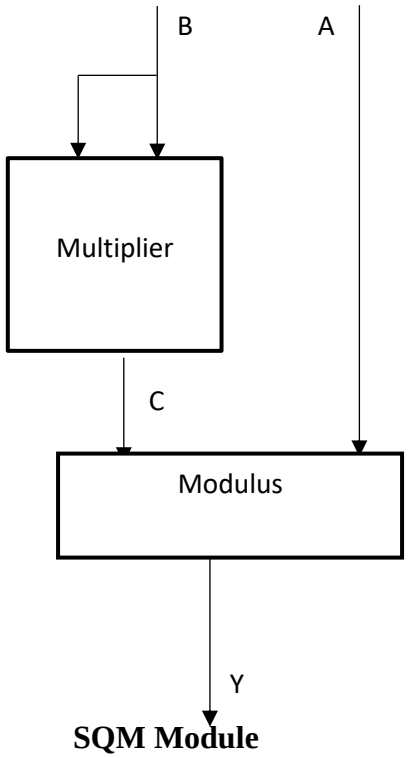
1. Function SQM
   30 minutes – Understanding the concept and ModelSim code. Creating the test vector table took the most amount of time.
2. Function CA2
   60 minutes – Understanding the concept and code work. Writing the code took the most amount of time as I applied various logics to implement it before deciding on one.
3. Output Analysis
   90-120 minutes – Understanding the concept and code work. Writing the code took the most amount of time as I encountered many bugs and errors while simulating the code.
4. Controlled fault injection
   120 minutes approximately – Understanding the concept and code work. Writing the code took the most amount of time as I applied various logics to implement it before deciding on one and each idea came up with errors, solving which took a lot of time.
5. Report making : 2 hours

In total creating the test vector table and solving errors takes the most significant amount of time.

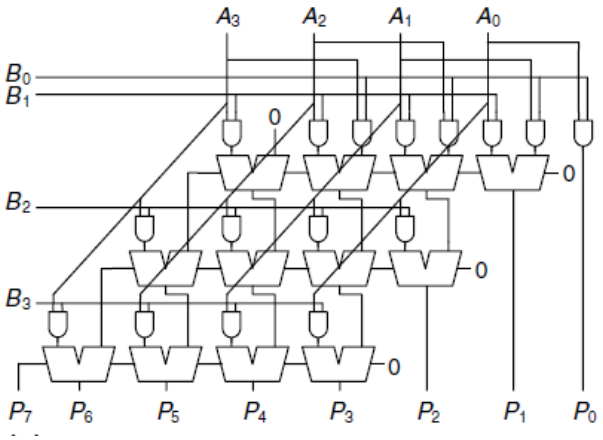
1. Function SQM

The square modulus function takes in two inputs. A 4-bit input B which is to be squared and an 8-bit input A which is made use of in the modulus function.

For squaring B, we can use a 4*4-bit multiplier. C = B*B, where C is divided by A using an 8-bit divider which gives us the modulus value. The sketch of the module roughly looks like this,



SQM Module



Multiplier logic

Test vectors	A (input)	B (input)	C = B² (intermediate)	Y = C % A (output)
1	10	12	144	4
2	147	12	144	144
3	5	15	225	0
4	13	5	25	12
5	30	9	81	21
below are your (well thought-out) choices				
6	15	13	169	4
7	20	10	100	0
8	150	15	225	75
9	10	9	81	1
10	5	10	100	0

SystemVerilog SQM Code

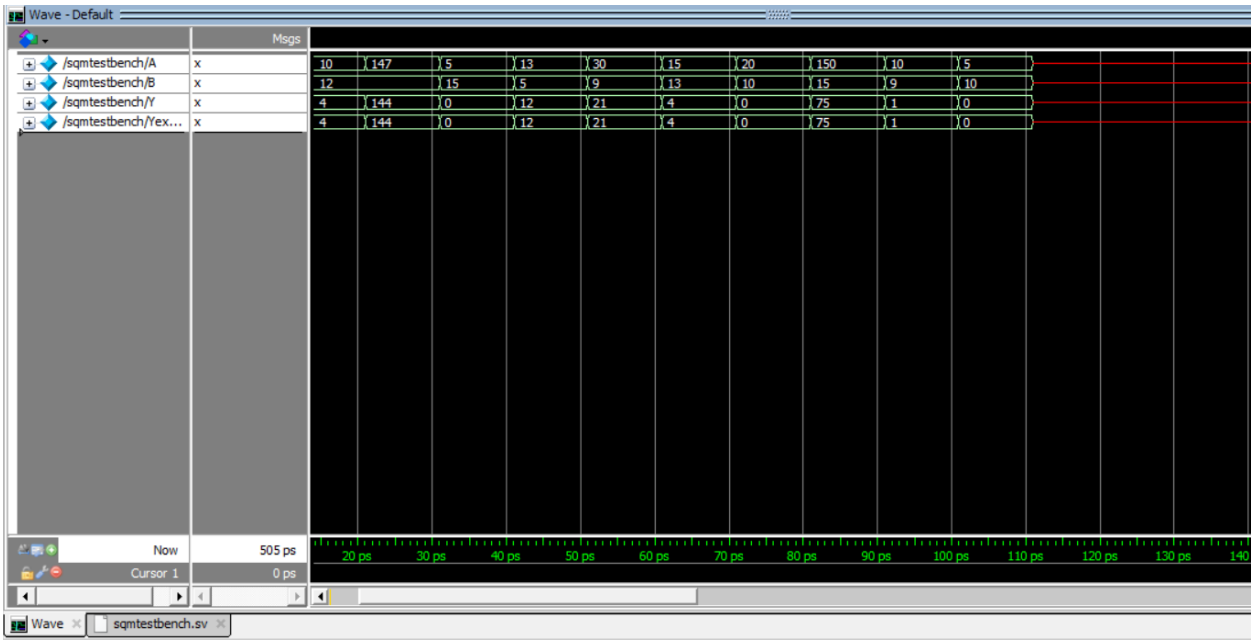
```
module sqmcode (input logic [7:0] A, input logic [3:0] B, output logic [7:0] Y);
logic [7:0]C;
assign C = B * B;
assign Y = C % A;
endmodule
```

SystemVerilog SQM Testbench

```
module sqmtestbench ();

logic clk, reset; // Clock and Reset are internal
logic [7:0] A; // Input 1
logic [3:0] B; // Input 2
logic [7:0]Yexpected; // Expected value of the output
logic [7:0] Y; // Output of circuit
reg [7:0] vectornum, errors; // Bookkeeping variables
reg [19:0] testvectors [0:100] ;// Array of testvectors

sqmcode dut (.A(A), .B(B), .Y(Y));  // Instantiate device under test

// Generate clock
always // No sensitivity list, so it always executes
begin
clk = 1; #5; clk = 0; #5; // 10ns period
end
// At start of test, load vectors
// And pulse reset
initial // Will execute at the beginning once
begin
$readmemb("D:/MS Sem 2/ECE 469 HDL/Project 1/Part2/sqmtv.txt", testvectors); // Read vectors
vectornum = 0; errors = 0; // Initialize
reset = 1; #10; reset = 0; // Apply reset wait
end
// Apply test vectors on rising edge of clk
always @(posedge clk)
begin
#1; {A, B, Yexpected} = testvectors[vectornum];
end
// Check results on falling edge of clk
always @(negedge clk)
if (~reset) // Skip during reset
begin
if (Y !== Yexpected)
```

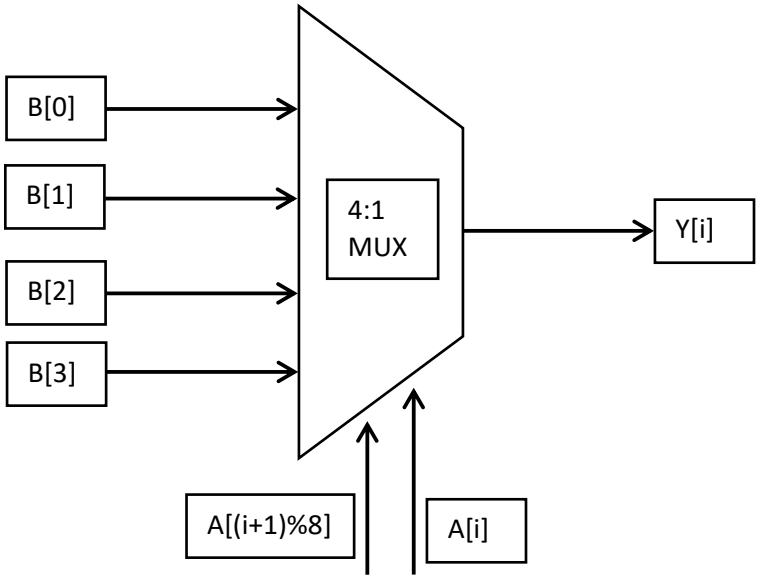
```
begin
$display("Error: inputs = %b", {A, B});
$display(" outputs = %b (%b expected)",Y,Yexpected);
errors = errors + 1;
end
// Increment array index and read next testvector
vectornum = vectornum + 1;
if ( vectornum == 50)
begin
$display("%d tests completed with %d errors",
vectornum, errors);
$finish; // End simulation
end
end
endmodule
```

Testvectors
00001010_1100_00000100
10010011_1100_10010000
00000101_1111_00000000
00001101_0101_00001100
00011110_1001_00010101
00001111_1101_00000100
00010100_1010_00000000
10010110_1111_01001011
00001010_1001_00000001
00000101_1010_00000000



2. Function CA2

The module has two inputs, 4-bit input B and 8-bit input A. Based on the value of i and (i+1), which is the location of a bits in A, value of B will be substituted based on the truth table value. The sketch of the module is as follows,



A[(i+1) % 8]	A[i]	Y[i]
0	0	B[0]
0	1	B[1]
1	0	B[2]
1	1	B[3]

Test vectors	A (input)	B (input)	Y =CA2 (A, B) (output)
1	00000000	1011	11111111
2	00000111	0100	10000000
3	11110011	0110	00001010
4	11110011	1100	11111001
5	00100011	1001	01001101
below are your (well thought-out) choices			
6	11111111	0001	00000000
7	11110000	1110	11111000
8	10101010	0010	10101010
9	00000001	1011	01111111
10	11001100	1010	11001100

SystemVerilog CA2 Code

```
module catwocode(input logic [3:0]B, input logic [7:0]A, output logic [7:0]Y);
always_comb
```

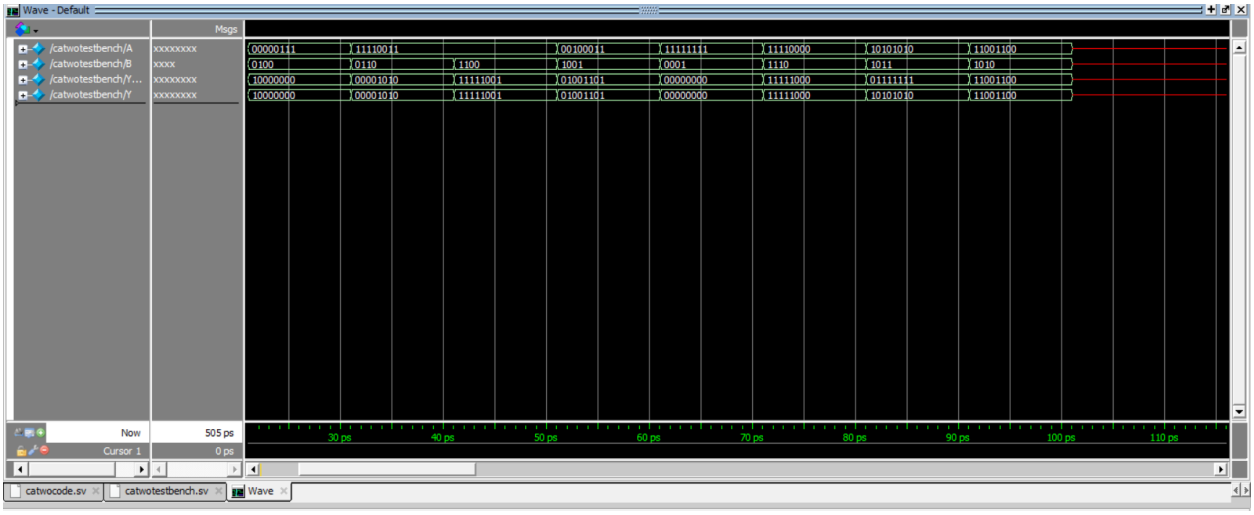
```
begin
Y[0]=A[1]?(A[0]?B[3]:B[2])
        :(A[0]?B[1]:B[0]);
Y[1]=A[2]?(A[1]?B[3]:B[2])
        :(A[1]?B[1]:B[0]);
Y[2]=A[3]?(A[2]?B[3]:B[2])
        :(A[2]?B[1]:B[0]);
Y[3]=A[4]?(A[3]?B[3]:B[2]):
        (A[3]?B[1]:B[0]);
Y[4]=A[5]?(A[4]?B[3]:B[2])
        :(A[4]?B[1]:B[0]);
Y[5]=A[6]?(A[5]?B[3]:B[2])
        :(A[5]?B[1]:B[0]);
Y[6]=A[7]?(A[6]?B[3]:B[2])
        :(A[6]?B[1]:B[0]);
Y[7]=A[0]?(A[7]?B[3]:B[2])
        :(A[7]?B[1]:B[0]);
end
endmodule
```

SystemVerilog CA2 Testbench

```
module catwotestbench();
logic clk, reset; // Clock and Reset are internal
logic [7:0]A; // Input 1
logic [3:0]B; // Input 2
logic [7:0]Yexpected; // Expected value of the output
logic [7:0]Y; // Output of circuit
reg [7:0]vectornum, errors; // Bookkeeping variables
reg [19:0]testvectors[0:100];// Array of testvectors

catwocode dut (.A(A), .B(B), .Y(Y));  // Instantiate device under test
// Generate clock
always // No sensitivity list, so it always executes
begin
clk = 1; #5; clk = 0; #5; // 10ns period
end
// At start of test, load vectors
// And pulse reset
initial // Will execute at the beginning once
begin
$readmemb("D:/MS Sem 2/ECE 469 HDL/Project 1/Part2/catwotv.txt", testvectors); // Read vectors
vectornum = 0; errors = 0; // Initialize
reset = 1; #10; reset = 0; // Apply reset wait
```

```
end

// Apply test vectors on rising edge of clk
always @(posedge clk)
begin
#1; {A, B, Yexpected} = testvectors[vectornum];
end

// Check results on falling edge of clk
always @(negedge clk)
if (~reset) // Skip during reset
begin
if (Y !== Yexpected)
begin
$display("Error: inputs = %b", {A, B});
$display(" outputs = %b (%b expected)",Y,Yexpected);
errors = errors + 1;
end

// Increment array index and read next testvector
vectornum = vectornum + 1;
if ( vectornum == 50)
begin
$display("%d tests completed with %d errors",
vectornum, errors);
$finish; // End simulation
end
end
endmodule
```

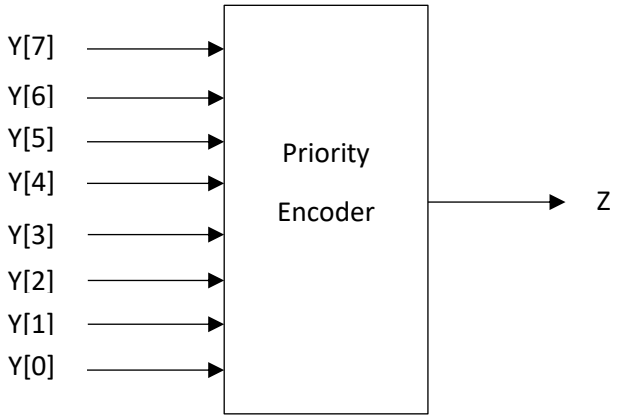
Testvector
00000000_1011_11111111
00000111_0100_10000000
11110011_0110_00001010
11110011_1100_11111001
00100011_1001_01001101
11111111_0001_00000000
11110000_1110_11111000
10101010_1011_01111111
11001100_1010_11001100



3. Output Analysis

This module has an 8 bit input Y. The function of the module is count the longest string of 1's in the given input and output the largest string value.

The sketch is as follows,



Test vectors	Y (in binary) (output)	Z (in dec) (output)
1	10110001	2
2	01101111	4
3	01010101	1
4	11111011	5
5	01111110	6
below are your (well thought-out) choices		
6	10101010	1
7	11111000	5
8	11011100	3
9	11100110	3

10	11111101	6

SystemVerilog Output Analysis Code

```
module onesstringcode( input logic [7:0]Y, output logic [3:0]Z);
always_comb
begin
casez (Y)
8'b00000000 : Z = 4'b0000 ; // Maximum length on one's string = 0

8'b0?0?0?01 : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b?0?0?010 : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b0?0?010? : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b?0?010?0 : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b0?010?0? : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b?010?0?0 : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b010?0?0? : Z = 4'b0001 ; // Maximum length on one's string = 1
8'b10?0?0?0 : Z = 4'b0001 ; // Maximum length on one's string = 1

8'b??0??011 : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b?0??0110 : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b0??0110? : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b??0110?0 : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b?0110??? : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b0110??0? : Z = 4'b0010 ; // Maximum length on one's string = 2
8'b110??0?? : Z = 4'b0010 ; // Maximum length on one's string = 2

8'b0???0111 : Z = 4'b0011 ; // Maximum length on one's string = 3
8'b???01110 : Z = 4'b0011 ; // Maximum length on one's string = 3
8'b??01110? : Z = 4'b0011 ; // Maximum length on one's string = 3
8'b?01110?0 : Z = 4'b0011 ; // Maximum length on one's string = 3
8'b01110??? : Z = 4'b0011 ; // Maximum length on one's string = 3
8'b1110???0 : Z = 4'b0011 ; // Maximum length on one's string = 3

8'b???01111 : Z = 4'b0100 ; // Maximum length on one's string = 4
8'b??011110 : Z = 4'b0100 ; // Maximum length on one's string = 4
8'b?011110? : Z = 4'b0100 ; // Maximum length on one's string = 4
8'b011110?0 : Z = 4'b0100 ; // Maximum length on one's string = 4
8'b11110??? : Z = 4'b0100 ; // Maximum length on one's string = 4

8'b??011111 : Z = 4'b0101 ; // Maximum length on one's string = 5
8'b?0111110 : Z = 4'b0101 ; // Maximum length on one's string = 5
8'b0111110? : Z = 4'b0101 ; // Maximum length on one's string = 5
8'b111110?? : Z = 4'b0101 ; // Maximum length on one's string = 5
```

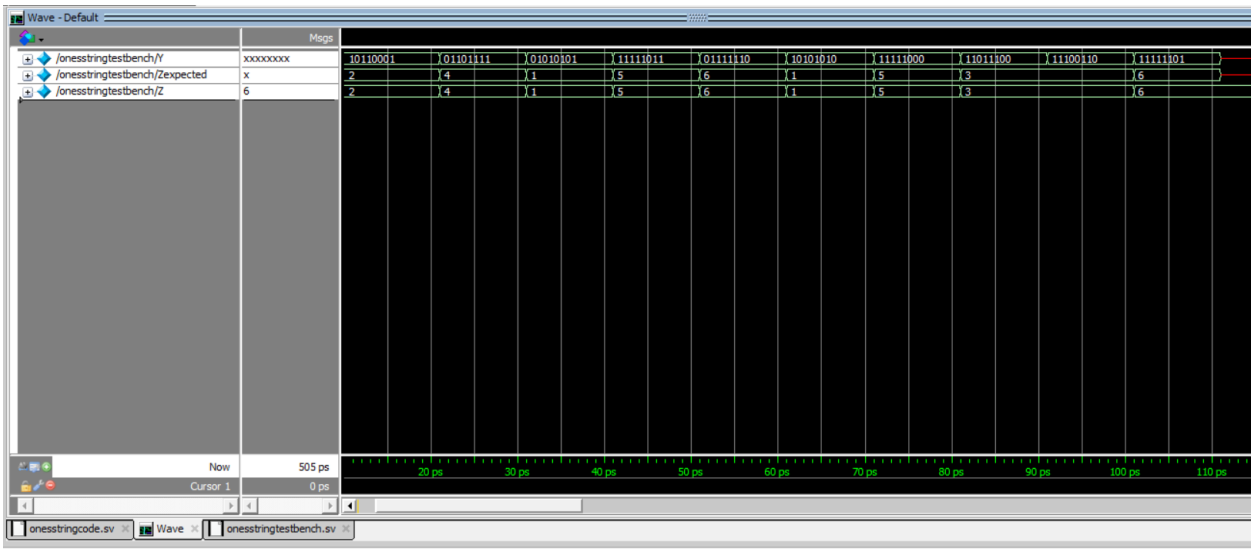
```
8'b?0111111 : Z = 4'b0110 ; // Maximum length on one's string = 6
8'b01111110 : Z = 4'b0110 ; // Maximum length on one's string = 6
8'b1111110? : Z = 4'b0110 ; // Maximum length on one's string = 6

8'b01111111 : Z = 4'b0111 ; // Maximum length on one's string = 7
8'b11111110 : Z = 4'b0111 ; // Maximum length on one's string = 7

8'b11111111 : Z = 4'b1000 ; // Maximum length on one's string = 8
endcase
end
endmodule
```

SystemVerilog Output Analysis testbench

```
module onesstringtestbench();
logic clk, reset; // Clock and Reset are internal
logic [7:0]Y; // Input 1
logic [3:0]Z; // Output 2
logic [3:0]Zexpected; // Expected value of the output
reg [7:0]vectornum, errors; // Bookkeeping variables
reg [11:0]testvectors[0:100];// Array of testvectors

onesstringcode dut (.Y(Y), .Z(Z));  // Instantiate device under test
// Generate clock
always // No sensitivity list, so it always executes
begin
clk = 1; #5; clk = 0; #5; // 10ns period
end
// At start of test, load vectors
// And pulse reset
initial // Will execute at the beginning once
begin
$readmemb("D:/MS Sem 2/ECE 469 HDL/Project 1/Part2/onesstringtv.txt", testvectors); // Read
vectors
vectornum = 0; errors = 0; // Initialize
reset = 1; #10; reset = 0; // Apply reset wait
end
// Apply test vectors on rising edge of clk
always @(posedge clk)
begin
#1; {Y, Zexpected} = testvectors[vectornum];
end
// Check results on falling edge of clk
always @(negedge clk)
if (~reset) // Skip during reset
```

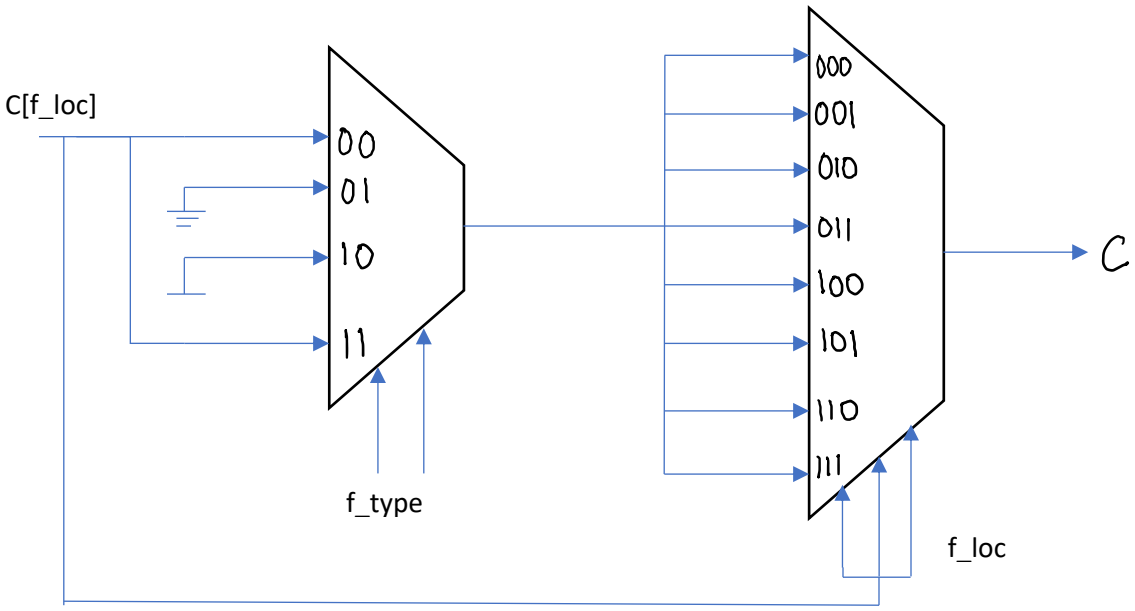
```
begin
if (Z !== Zexpected)
begin
$display("Error: inputs = %b", {Y});
$display(" outputs = %b (%b expected)",Z,Zexpected);
errors = errors + 1;
end
// Increment array index and read next testvector
vectornum = vectornum + 1;
if ( vectornum == 50)
begin
$display("%d tests completed with %d errors",
vectornum, errors);
$finish; // End simulation
end
end
endmodule
```

Testvectors
10110001_0010
01101111_0100
01010101_0001
11111011_0101
01111110_0110
10101010_0001
11111000_0101
11011100_0011
11100110_0011
11111101_0110



4. Controlled fault injection

This module works based on square modulus function, addition to which, a fault is injected into the output based on the location of the bit. Hence based on the value of fault type and fault location the intermediate value of C is injected with a fault and modulus is calculated.
The sketch of the module is like as below




```
00:   no fault          no change to C[i]
01:   stuck-at-0:       C[i] = 0
10:   stuck-at-1:       C[i] = 1
11:   flipping:         C[i] is inverted
```

Test vectors	A	B	f_loc	f_type	C	Y (dec)
1	00001010	1100	000	10	1001000***1***	5
2	10010011	1100	101	11	10***1***10000	29
3	00000101	1111	100	01	11100001	0
4	00001101	0101	000	11	0001100***0***	11
5	00011110	1001	110	00	01010001	21
below are your (well thought-out) choices						
6	00001111	1101	001	00	10101001	4
7	00010100	1010	010	10	01100100	0
8	10010110	1111	011	11	1110***1***001	8
9	00001010	1001	100	01	010***0***0001	2
10	00000101	10	111	01	01100100	0

SystemVerilog Fault injection Code

```
module faultinjectioncode (input logic [7:0]A, input logic [3:0]B, input logic [2:0]f_loc,
                           input logic [1:0]f_type, output logic [7:0]Y,output logic [7:0]C, output logic
[7:0]Cin);
assign Cin = B * B;
always_comb
begin
C = B * B;
case (f_loc[2:0])
3'b000 :
if (f_type == 2'b00)
C[0] = C[0] ;
else if (f_type == 2'b01)
C[0] = 0;
else if (f_type == 2'b10)
C[0] = 1;
else if (f_type == 2'b11)
C[0] = ~C[0] ;
3'b001 :
if (f_type == 2'b00)
C[1] = C[1] ;
else if (f_type == 2'b01)
C[1] = 0;
else if (f_type == 2'b10)
C[1] = 1;
else if (f_type == 2'b11)
C[1] = ~C[1] ;
3'b010 :
if (f_type == 2'b00)
C[2] = C[2] ;
else if (f_type == 2'b01)
C[2] = 0;
else if (f_type == 2'b10)
C[2] = 1;
else if (f_type == 2'b11)
C[2] = ~C[2] ;
3'b011 :
if (f_type == 2'b00)
C[3] = C[3] ;
else if (f_type == 2'b01)
C[3] = 0;
else if (f_type == 2'b10)
C[3] = 1;
else if (f_type == 2'b11)
```

```
C[3] = ~C[3] ;
3'b100 :
if (f_type == 2'b00)
C[4] = C[4] ;
else if (f_type == 2'b01)
C[4] = 0;
else if (f_type == 2'b10)
C[4] = 1;
else if (f_type == 2'b11)
C[4] = ~C[4] ;
3'b101 :
if (f_type == 2'b00)
C[5] = C[5] ;
else if (f_type == 2'b01)
C[5] = 0;
else if (f_type == 2'b10)
C[5] = 1;
else if (f_type == 2'b11)
C[5] = ~C[5] ;
3'b110 :
if (f_type == 2'b00)
C[6] = C[6] ;
else if (f_type == 2'b01)
C[6] = 0;
else if (f_type == 2'b10)
C[6] = 1;
else if (f_type == 2'b11)
C[6] = ~C[6] ;
3'b111 :
if (f_type == 2'b00)
C[7] = C[7] ;
else if (f_type == 2'b01)
C[7] = 0;
else if (f_type == 2'b10)
C[7] = 1;
else if (f_type == 2'b11)
C[7] = ~C[7] ;
endcase
if(A!=0)
        Y = C % A;
else
        Y = 8'bXXXXXXXX;
end
endmodule
```

SystemVerilog Fault injection testbench

```
module faultinjectiontestbench();

logic clk, reset; // Clock and Reset are internal
logic [7:0]A; // Input A
logic [3:0]B; // Input B
logic [2:0]f_loc; // Input for location of fault
logic [1:0]f_type; // Input for tyoe of fault
logic [7:0]Y; // Output of the circuit
logic [7:0]Yexpected; // Expected output
logic [7:0]C; // Intermediate value for fault injection
reg [7:0]vectornum, errors; // Bookkeeping variables
reg [32:0]testvectors[0:100];// Array of testvectors

faultinjectioncode dut (.A(A), .B(B), .f_loc(f_loc), .f_type(f_type), .C(C), .Y(Y));  // Instantiate device under test

// Generate clock
always // No sensitivity list, so it always executes
begin
clk = 1; #5; clk = 0; #5; // 10ns period
end

// At start of test, load vectors
// And pulse reset
initial // Will execute at the beginning once
begin
$readmemb("D:/MS Sem 2/ECE 469 HDL/Project 1/Part2/faultinjectiontv.txt", testvectors); // Read vectors
vectornum = 0; errors = 0; // Initialize
reset = 1; #10; reset = 0; // Apply reset wait
end

// Apply test vectors on rising edge of clk
always @(posedge clk)
begin
#1; {A, B, f_loc, f_type, C, Yexpected} = testvectors[vectornum];
end

// Check results on falling edge of clk
always @(negedge clk)
if (~reset) // Skip during reset
begin
```

```
if (Y !== Yexpected)
begin
$display("Error: inputs = %b", {A, B});
$display(" outputs = %b (%b expected)",Y,Yexpected);
errors = errors + 1;
end

// Increment array index and read next testvector
vectornum = vectornum + 1;
if ( vectornum == 50)
begin
$display("%d tests completed with %d errors",
vectornum, errors);
$finish; // End simulation
end
end
endmodule
```

Testvectors
00001010_1100_000_10_10010001_00000101
10010011_1100_101_11_10110000_00011101
00000101_1111_100_01_11100001_00000000
00001101_0101_000_11_00011000_00001011
00011110_1001_110_00_01010001_00010101
00001111_1101_001_00_10101001_00000100
00010100_1010_010_10_01100100_00000000
10010110_1111_011_11_11101001_01010011
00001010_1001_100_01_01000001_00000101
00000101_1010_111_01_01100100_00000000