ES203 Digital Systems Project

Verilog Code Generator

Submitted by-

Akshat Mangal | 17110010

Gaurav Sonkusle | 17110055

Mohamed Shamir | 17110084

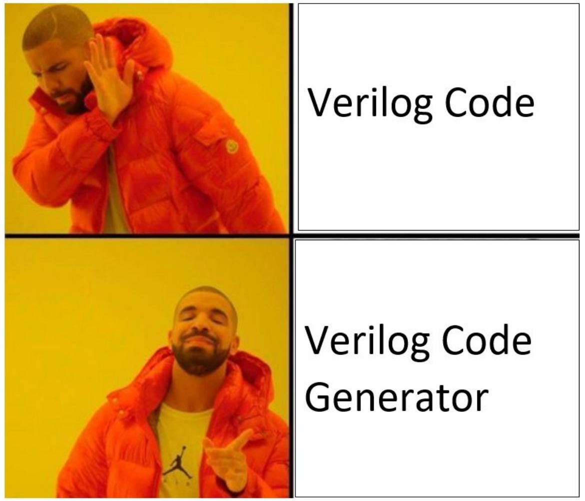
Mohmmad Aslam | 17110086

Submitted to:-

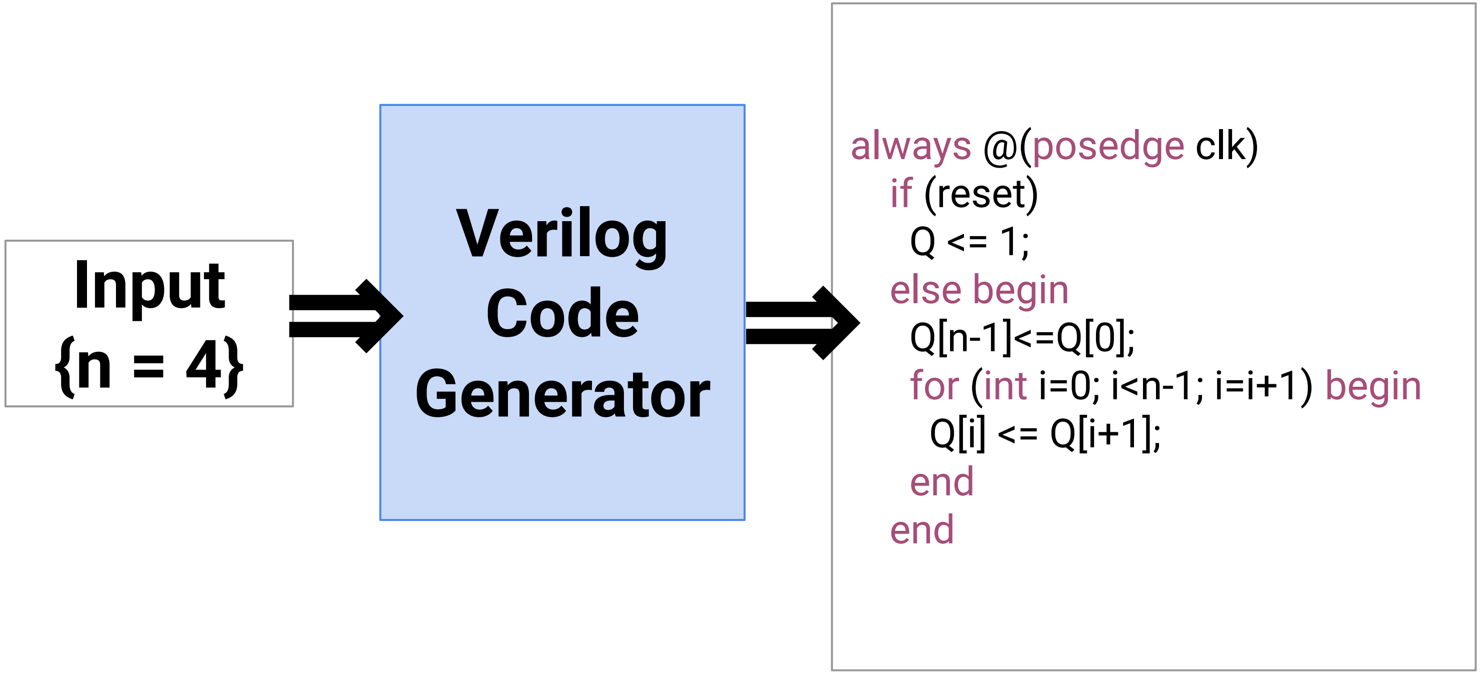
Prof. Joycee Mekie

Motivation:

- Writing hardware code is time-consuming.
- Verilog Code generated for various modules can be used effectively to fasten the programming.

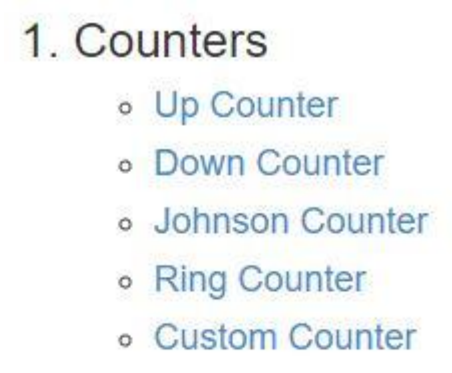


Our Project:

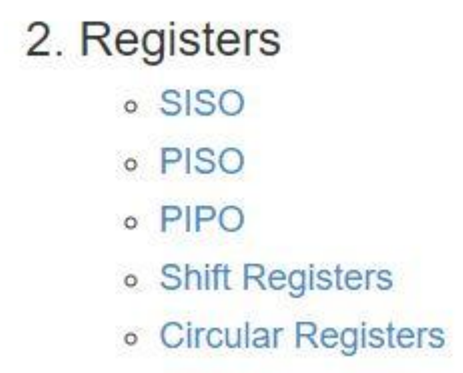


```
always @(posedge clk)
    if (reset)
      Q <= 1;
    else begin
      Q[n-1]<=Q[0];
      for (int i=0; i<n-1; i=i+1) begin
        Q[i] <= Q[i+1];
      end
    end
```

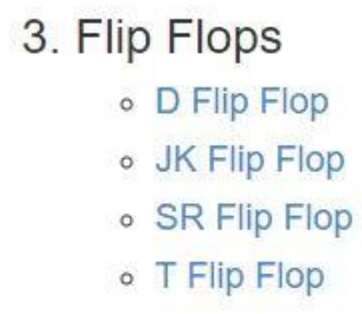
Modules Implemented:



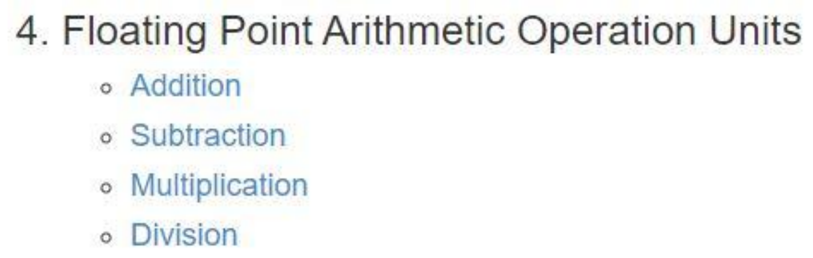
1. Counters
   - Up Counter
   - Down Counter
   - Johnson Counter
   - Ring Counter
   - Custom Counter



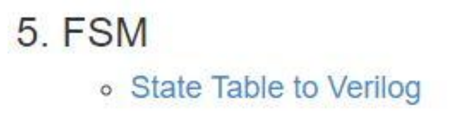
2. Registers
   - SISO
   - PISO
   - PIPO
   - Shift Registers
   - Circular Registers



3. Flip Flops
   - D Flip Flop
   - JK Flip Flop
   - SR Flip Flop
   - T Flip Flop



4. Floating Point Arithmetic Operation Units
   - Addition
   - Subtraction
   - Multiplication
   - Division



5. FSM
   - State Table to Verilog

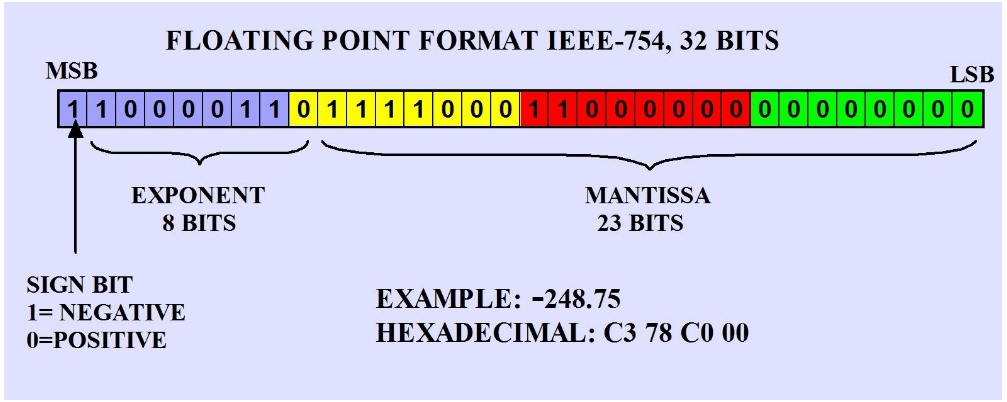
Project Timeline:

W1	W2	W3	W4
Counters			
	Registers and Flip Flops		
		Floating Point Arithmetic Operations	
Flask Web App			

Project Website URL

"https://python-verilog-gen.herokuapp.com/"

Addition:



Single Precision - 32 bit input

Exponent
1.2345 = 12345×10^-4
Mantissa

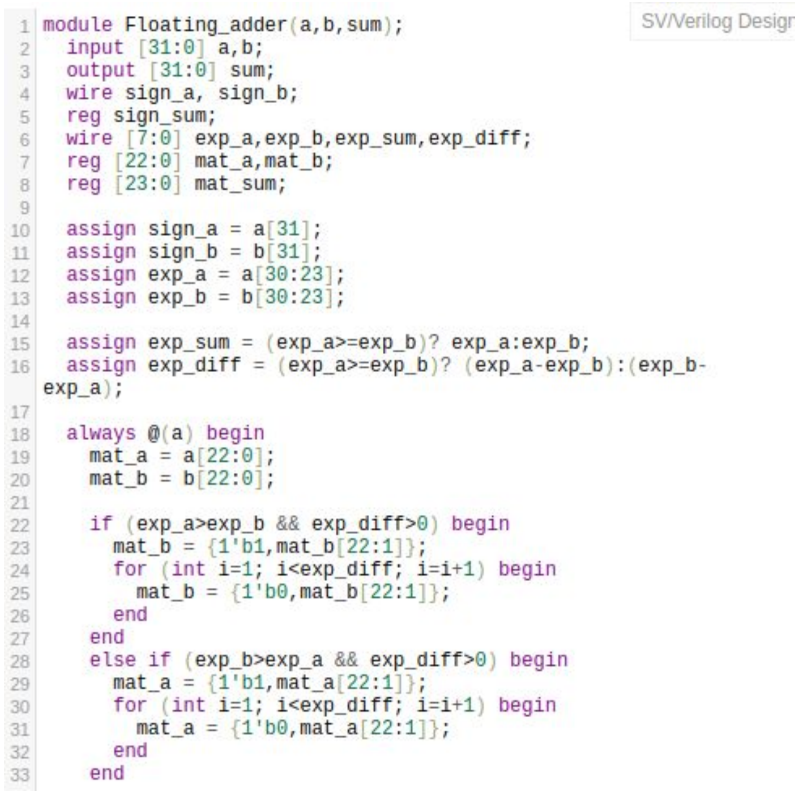
Example for 10-bit addition:
In which 1st bit represent sign, next 5 bit for Mantissa and last 4 bits for exponent

010000 0011 -> first no. (2^3 * 0.10000 = 100.00 = 4)
010010 0010 -> second no. (2^2 * 0.10010 = 2^3 * 0.01001 = 10.01 = 2.25)

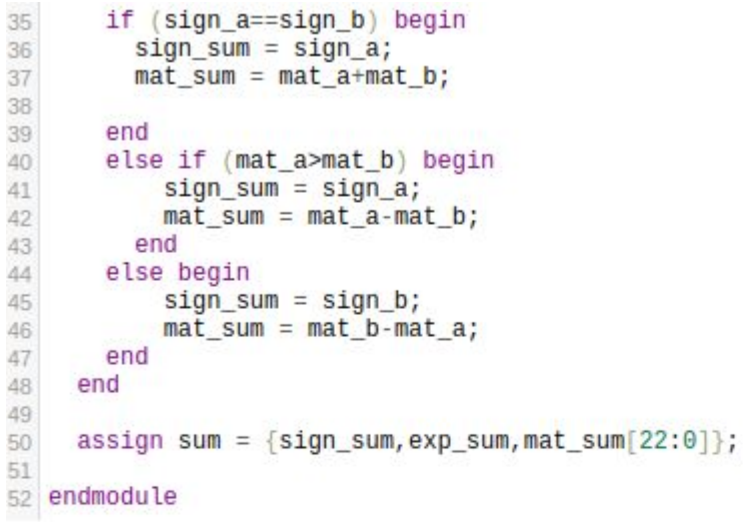
We need to make sure that the exponent of the numbers should be same.
Then we perform addition on the numbers

0.10000 + 0.01001 = 0.11001 * 2^3 = 110.01 = 6.25
Sum = 011001 0011

Verilog Code:

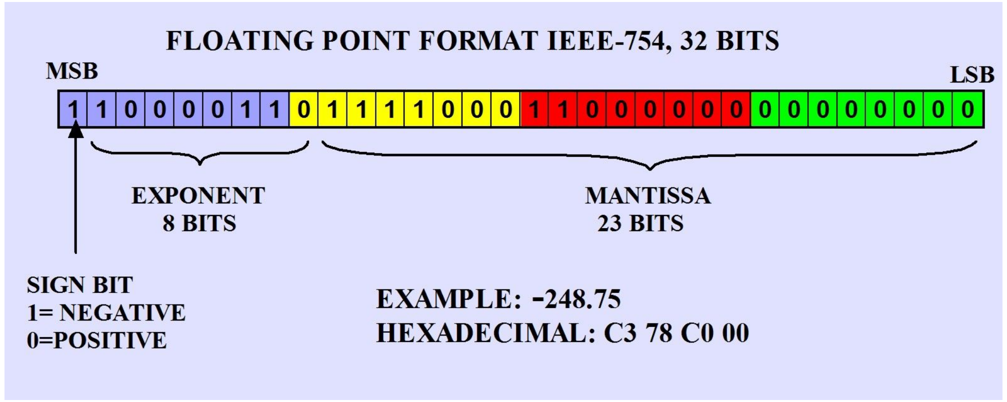


```verilog
module Floating_adder(a,b,sum);
  input [31:0] a,b;
  output [31:0] sum;
  wire sign_a, sign_b;
  reg sign_sum;
  wire [7:0] exp_a,exp_b,exp_sum,exp_diff;
  reg [22:0] mat_a,mat_b;
  reg [23:0] mat_sum;

  assign sign_a = a[31];
  assign sign_b = b[31];
  assign exp_a = a[30:23];
  assign exp_b = b[30:23];

  assign exp_sum = (exp_a>=exp_b)? exp_a:exp_b;
  assign exp_diff = (exp_a>=exp_b)? (exp_a-exp_b):(exp_b-exp_a);

  always @(a) begin
    mat_a = a[22:0];
    mat_b = b[22:0];

    if (exp_a>exp_b && exp_diff>0) begin
      mat_b = {1'b1,mat_b[22:1]};
      for (int i=1; i<exp_diff; i=i+1) begin
        mat_b = {1'b0,mat_b[22:1]};
      end
    end
    else if (exp_b>exp_a && exp_diff>0) begin
      mat_a = {1'b1,mat_a[22:1]};
      for (int i=1; i<exp_diff; i=i+1) begin
        mat_a = {1'b0,mat_a[22:1]};
      end
    end
```

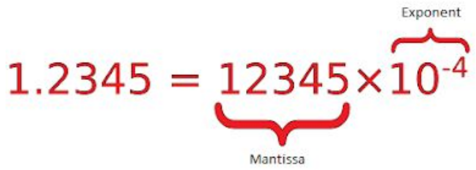


```verilog
    if (sign_a==sign_b) begin
      sign_sum = sign_a;
      mat_sum = mat_a+mat_b;

    end
    else if (mat_a>mat_b) begin
        sign_sum = sign_a;
        mat_sum = mat_a-mat_b;
      end
    else begin
        sign_sum = sign_b;
        mat_sum = mat_b-mat_a;
    end
  end

  assign sum = {sign_sum,exp_sum,mat_sum[22:0]};

endmodule
```

Multiplication:



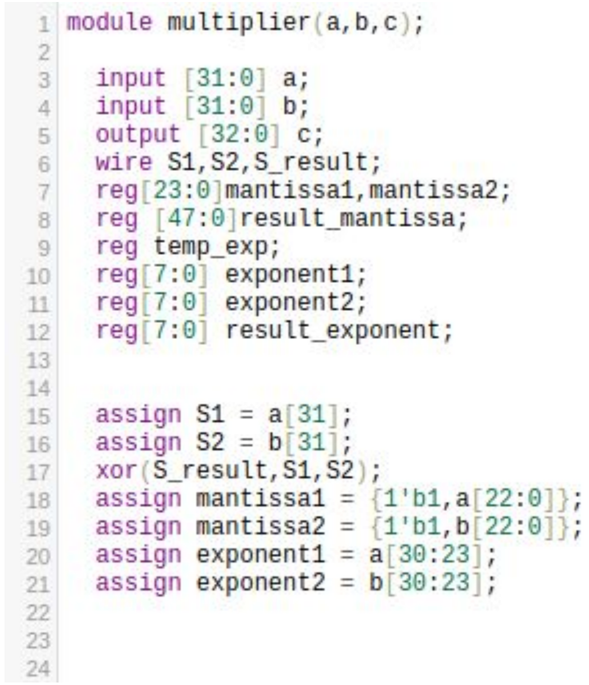
Single Precision - 32 bit input



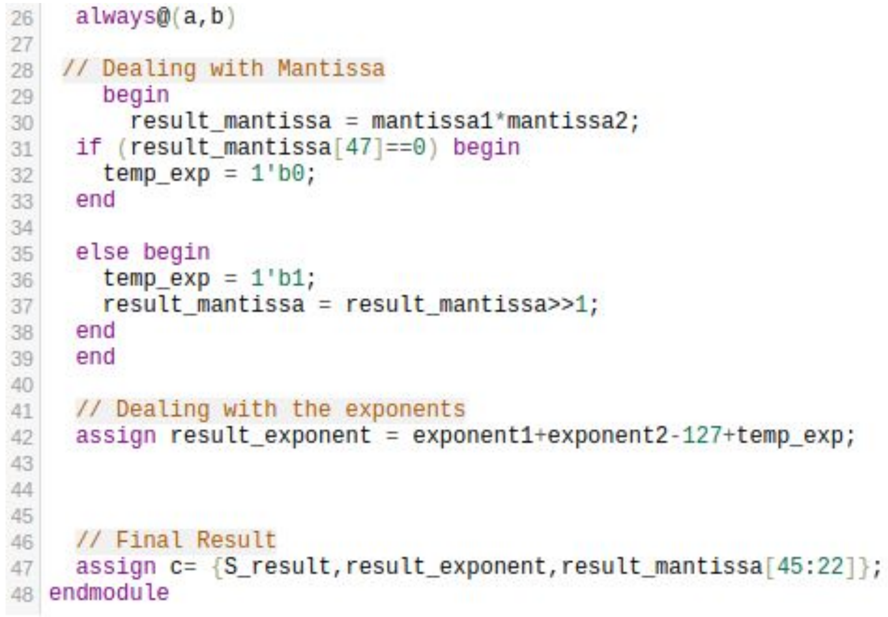
Steps Involved:

1. Sign Bit of result = XOR (S1,S2)
2. Multiply Mantissa
   - We need to add “1” in the left end of the mantissa to each of the inputs.
   - Multiply the two mantissa.
   - Round off the result: One digit at the ones place. By increasing the power of exponents.
3. Exponents: exponent 1 + exponent 2 + the extra exponent - bias
4. Final result:{signbit,exponent,mantissa}

Verilog Code:

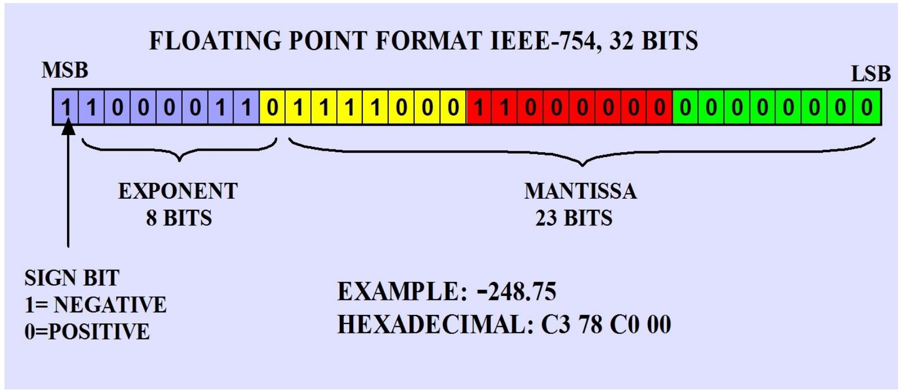


```verilog
module multiplier(a,b,c);

  input [31:0] a;
  input [31:0] b;
  output [32:0] c;
  wire S1,S2,S_result;
  reg[23:0]mantissa1,mantissa2;
  reg [47:0]result_mantissa;
  reg temp_exp;
  reg[7:0] exponent1;
  reg[7:0] exponent2;
  reg[7:0] result_exponent;


  assign S1 = a[31];
  assign S2 = b[31];
  xor(S_result,S1,S2);
  assign mantissa1 = {1'b1,a[22:0]};
  assign mantissa2 = {1'b1,b[22:0]};
  assign exponent1 = a[30:23];
  assign exponent2 = b[30:23];



```

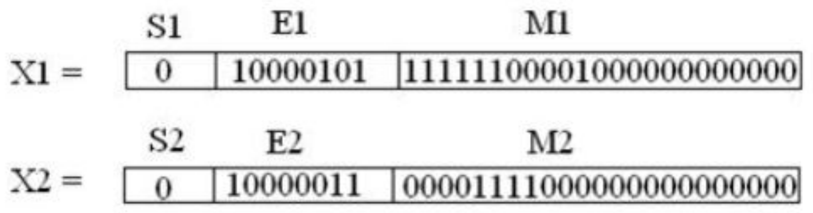


```verilog
  always@(a,b)

 // Dealing with Mantissa
   begin
     result_mantissa = mantissa1*mantissa2;
  if (result_mantissa[47]==0) begin
    temp_exp = 1'b0;
  end

  else begin
    temp_exp = 1'b1;
    result_mantissa = result_mantissa>>1;
  end
  end

  // Dealing with the exponents
  assign result_exponent = exponent1+exponent2-127+temp_exp;



  // Final Result
  assign c= {S_result,result_exponent,result_mantissa[45:22]};
endmodule
```

Division:



X1 = 127.03125
X2 = 16.937500
X3 = X1/X2 = 7.5 (in decimal form)

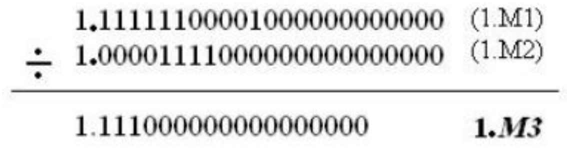


	S1	E1	M1
X1 =	0	10000101	11111100001000000000000

	S2	E2	M2
X2 =	0	10000011	00001111000000000000000

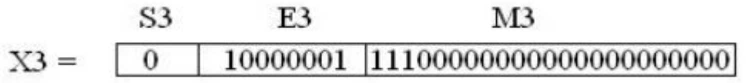
Steps Involved:
1. Sign Bit of result = XOR (S1,S2)
2. Exponent bit of result = Exponent1-Exponent2+bias (bias = 127)
3. Divide Mantissa
   - We need to add "1" in the left end of the mantissa to each of the inputs.
   - Divide the two mantissa.
   - Round off the result: One digit at the ones place. By increasing the power of exponents.
4. Final result:{signbit,exponent,mantissa}

Sign_result = XOR(S1, S2) = 0
Exponent_result = E1-E2+bias = 133-131+127 = 129
= 10000001
Mantissa result = 1.M1/1.M2 = 1.M3



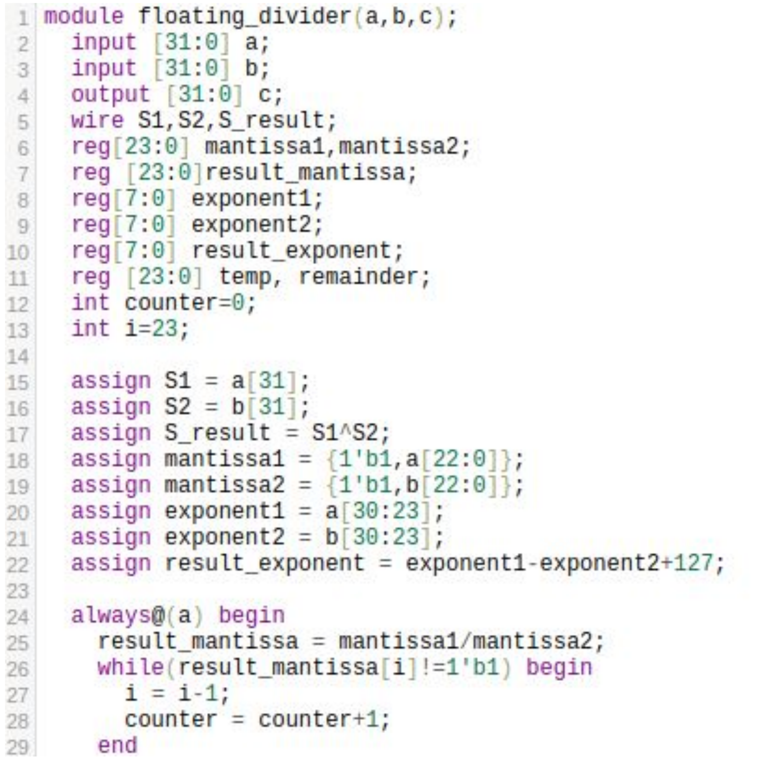
$$
\begin{array}{rl}
1.11111100001000000000000 & (1.M1) \\
\div\ 1.00001111000000000000000 & (1.M2) \\
\hline
1.111000000000000000 & \mathbf{1.}\boldsymbol{M3}
\end{array}
$$

X3 (result) = (Sign_result,Exponent_result,Mantissa_result)

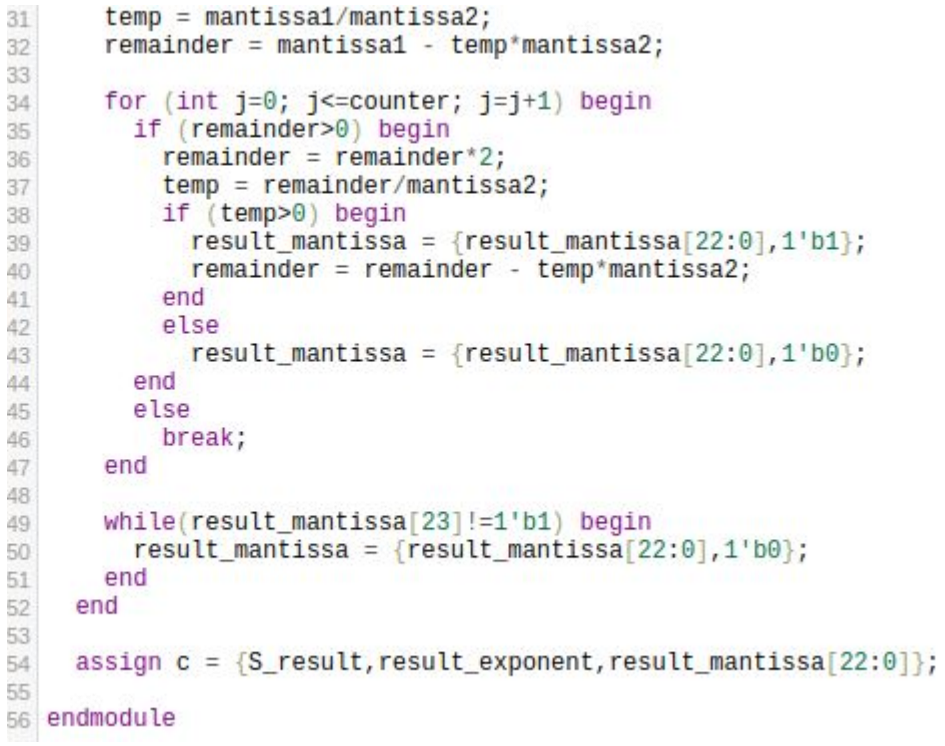


	S3	E3	M3
X3 =	0	10000001	11100000000000000000000

Verilog Code:



```verilog
module floating_divider(a,b,c);
  input [31:0] a;
  input [31:0] b;
  output [31:0] c;
  wire S1,S2,S_result;
  reg[23:0] mantissa1,mantissa2;
  reg [23:0]result_mantissa;
  reg[7:0] exponent1;
  reg[7:0] exponent2;
  reg[7:0] result_exponent;
  reg [23:0] temp, remainder;
  int counter=0;
  int i=23;

  assign S1 = a[31];
  assign S2 = b[31];
  assign S_result = S1^S2;
  assign mantissa1 = {1'b1,a[22:0]};
  assign mantissa2 = {1'b1,b[22:0]};
  assign exponent1 = a[30:23];
  assign exponent2 = b[30:23];
  assign result_exponent = exponent1-exponent2+127;

  always@(a) begin
    result_mantissa = mantissa1/mantissa2;
    while(result_mantissa[i]!=1'b1) begin
      i = i-1;
      counter = counter+1;
    end
```



```verilog
    temp = mantissa1/mantissa2;
    remainder = mantissa1 - temp*mantissa2;

    for (int j=0; j<=counter; j=j+1) begin
      if (remainder>0) begin
        remainder = remainder*2;
        temp = remainder/mantissa2;
        if (temp>0) begin
          result_mantissa = {result_mantissa[22:0],1'b1};
          remainder = remainder - temp*mantissa2;
        end
        else
          result_mantissa = {result_mantissa[22:0],1'b0};
      end
      else
        break;
    end

    while(result_mantissa[23]!=1'b1) begin
      result_mantissa = {result_mantissa[22:0],1'b0};
    end
  end

  assign c = {S_result,result_exponent,result_mantissa[22:0]};

endmodule
```

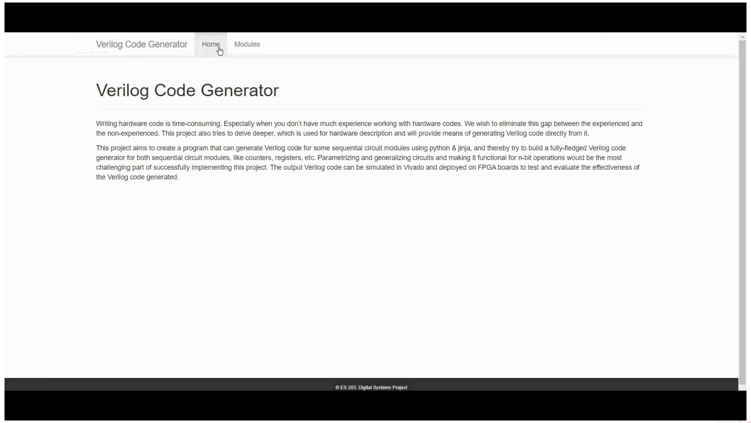
Live Demo:

Live Demo

Any Questions!

Implementation:

1. Up Counter:

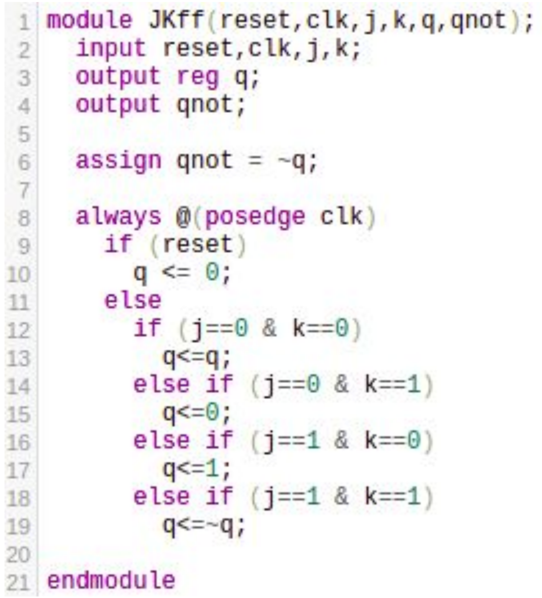


Verilog Code Generator

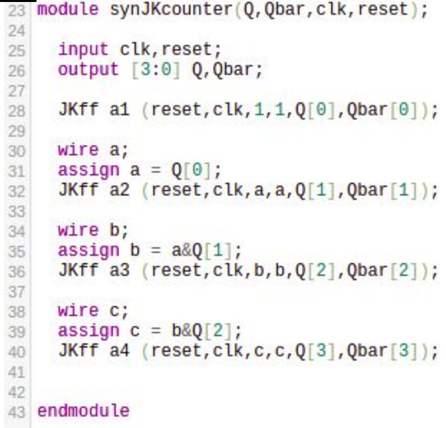
Writing hardware code is time-consuming. Especially when you don't have much experience working with hardware codes. We wish to eliminate this gap between the experienced and the non-experienced. This project also tries to delve deeper, which is used for hardware description and will provide means of generating Verilog code directly from it.

This project aims to create a program that can generate Verilog code for some sequential circuit modules using python & jinja, and thereby try to build a fully-fledged Verilog code generator for both sequential circuit modules, like counters, registers, etc. Parametrizing and generalizing circuits and making it functional for n-bit operations would be the most challenging part of successfully implementing this project. The output Verilog code can be simulated in Vivado and deployed on FPGA boards to test and evaluate the effectiveness of the Verilog code generated.

© ES 203: Digital Systems Project



```verilog
module JKff(reset,clk,j,k,q,qnot);
  input reset,clk,j,k;
  output reg q;
  output qnot;

  assign qnot = ~q;

  always @(posedge clk)
    if (reset)
      q <= 0;
    else
      if (j==0 & k==0)
        q<=q;
      else if (j==0 & k==1)
        q<=0;
      else if (j==1 & k==0)
        q<=1;
      else if (j==1 & k==1)
        q<=~q;

endmodule
```

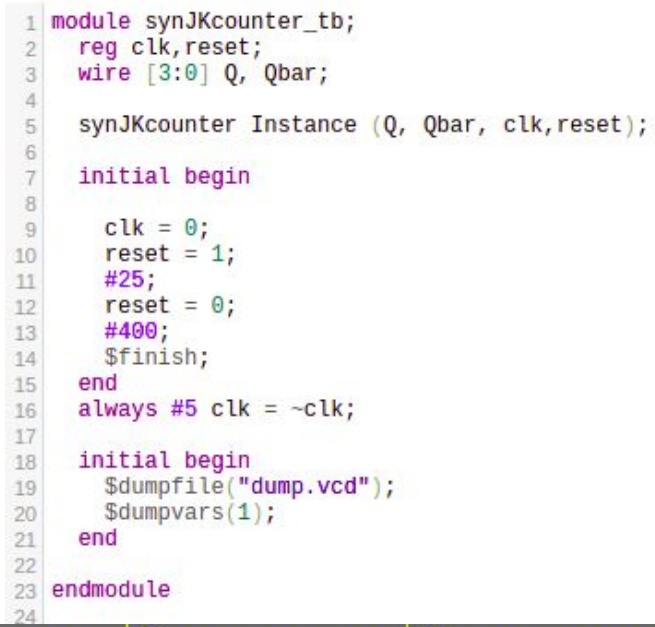


```verilog
module synJKcounter(Q,Qbar,clk,reset);

  input clk,reset;
  output [3:0] Q,Qbar;

  JKff a1 (reset,clk,1,1,Q[0],Qbar[0]);

  wire a;
  assign a = Q[0];
  JKff a2 (reset,clk,a,a,Q[1],Qbar[1]);

  wire b;
  assign b = a&Q[1];
  JKff a3 (reset,clk,b,b,Q[2],Qbar[2]);

  wire c;
  assign c = b&Q[2];
  JKff a4 (reset,clk,c,c,Q[3],Qbar[3]);


endmodule
```

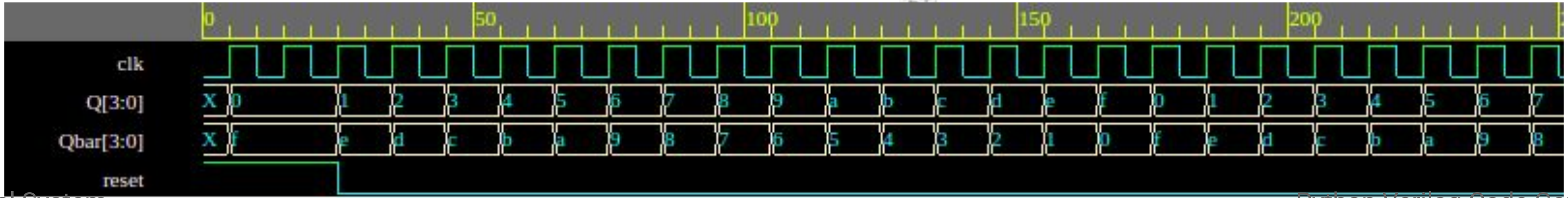
Implementation:

1. Up Counter:

- **Validating the Code:**

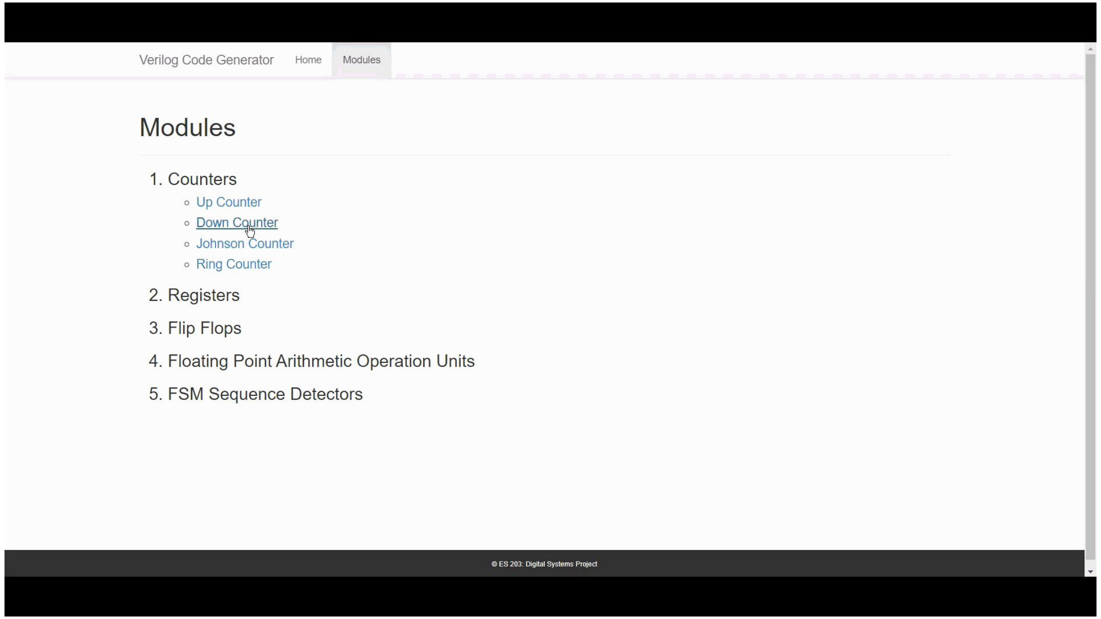


```
module synJKcounter_tb;
  reg clk,reset;
  wire [3:0] Q, Qbar;

  synJKcounter Instance (Q, Qbar, clk,reset);

  initial begin

    clk = 0;
    reset = 1;
    #25;
    reset = 0;
    #400;
    $finish;
  end
  always #5 clk = ~clk;

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```



Implementation:

2. Down Counter:



Verilog Code Generator Home Modules

Modules

1. Counters
   - Up Counter
   - Down Counter
   - Johnson Counter
   - Ring Counter
2. Registers
3. Flip Flops
4. Floating Point Arithmetic Operation Units
5. FSM Sequence Detectors

© ES 203: Digital Systems Project

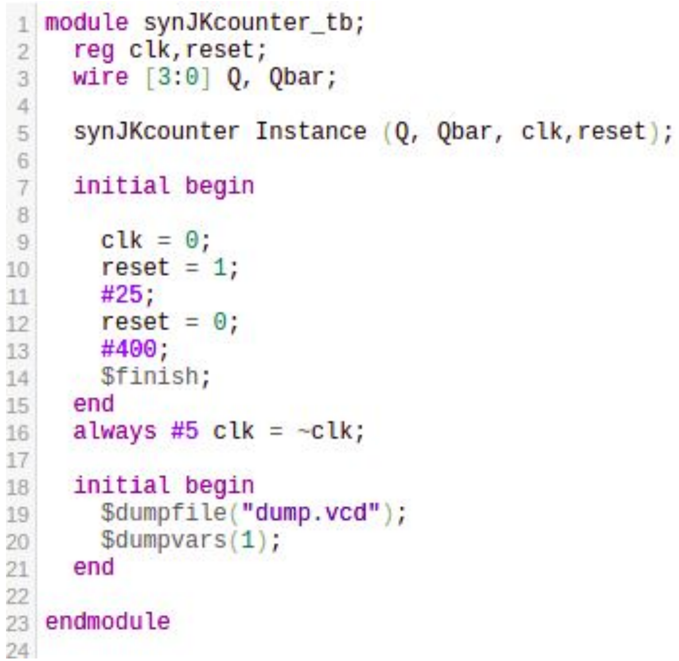
```
module JKff(reset,clk,j,k,q,qnot);
  input reset,clk,j,k;
  output reg q;
  output qnot;

  assign qnot = ~q;

  always @(posedge clk)
    if (reset)
      q <= 0;
    else
      if (j==0 & k==0)
        q<=q;
      else if (j==0 & k==1)
        q<=0;
      else if (j==1 & k==0)
        q<=1;
      else if (j==1 & k==1)
        q<=~q;

endmodule
```

```
module synJKcounter(Q,Qbar,clk,reset);

  parameter n = 4;
  input clk,reset;
  output [n-1:0] Q,Qbar;

  JKff a1 (reset,clk,1,1,Qbar[0],Q[0]);

  wire w1;
  assign w1 = Qbar[0];
  JKff a2 (reset,clk,w1,w1,Qbar[1],Q[1]);

  wire w2;
  assign w2 = w1&Qbar[1];
  JKff a3 (reset,clk,w2,w2,Qbar[2],Q[2]);

  wire w3;
  assign w3 = w2&Qbar[2];
  JKff a4 (reset,clk,w3,w3,Qbar[3],Q[3]);
endmodule
```

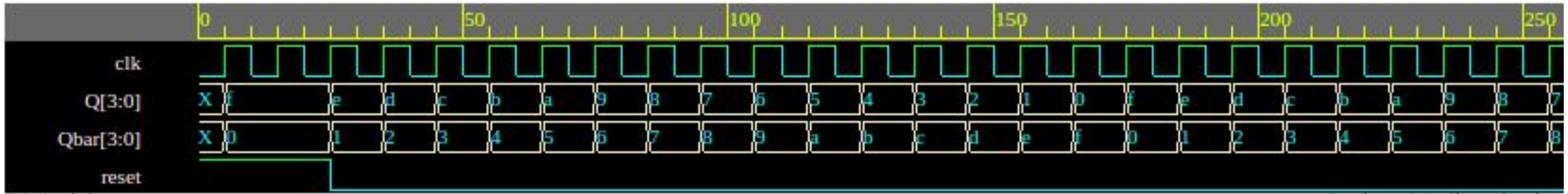
Implementation:

2. Down Counter:

- **Validating the Code:**

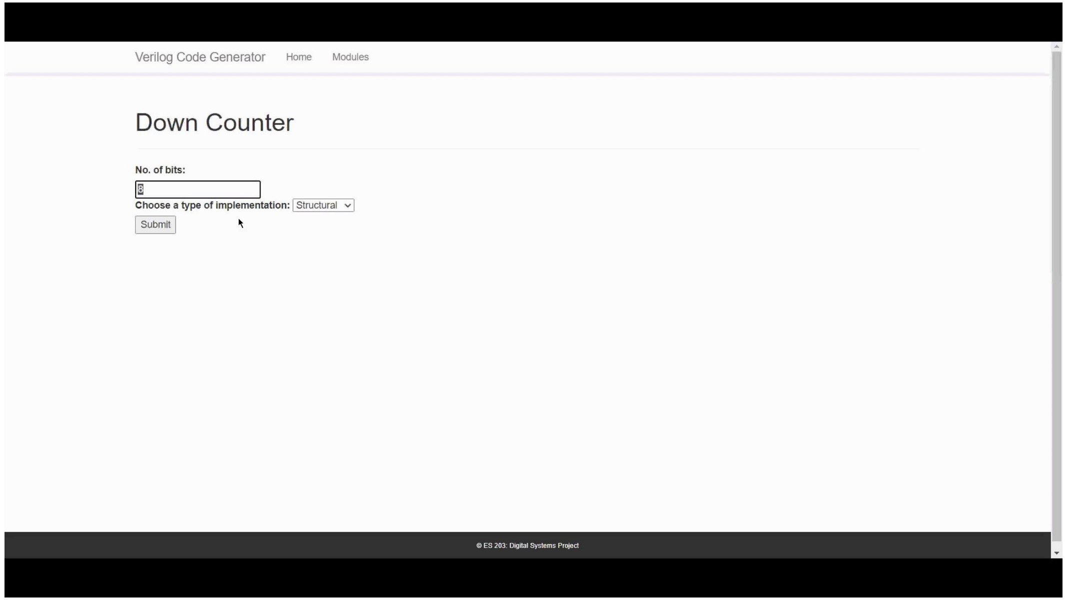


```
module synJKcounter_tb;
  reg clk,reset;
  wire [3:0] Q, Qbar;

  synJKcounter Instance (Q, Qbar, clk,reset);

  initial begin

    clk = 0;
    reset = 1;
    #25;
    reset = 0;
    #400;
    $finish;
  end
  always #5 clk = ~clk;

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```



Implementation:

3. Johnson Counter:

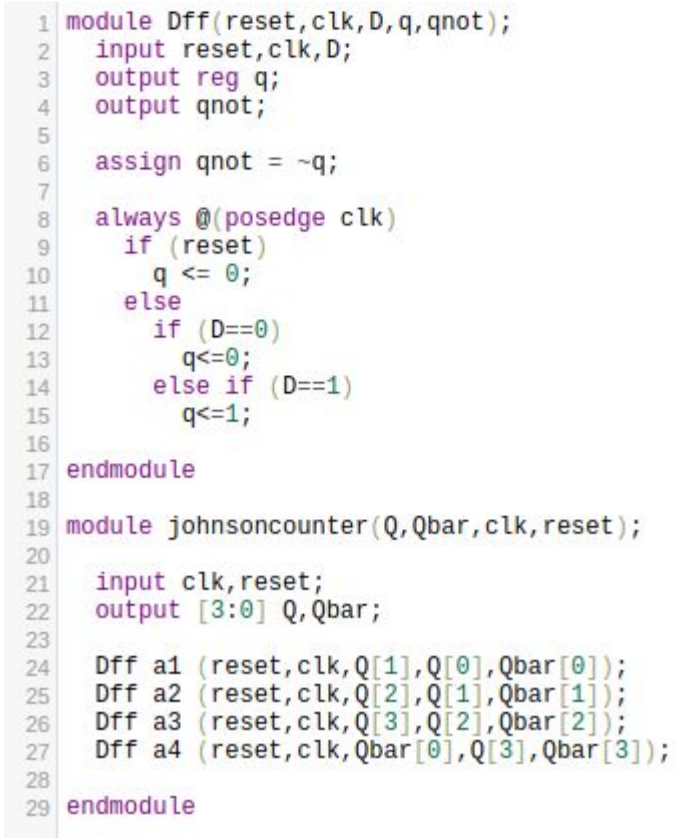


Down Counter

No. of bits:

Choose a type of implementation: Structural

© ES 203: Digital Systems Project

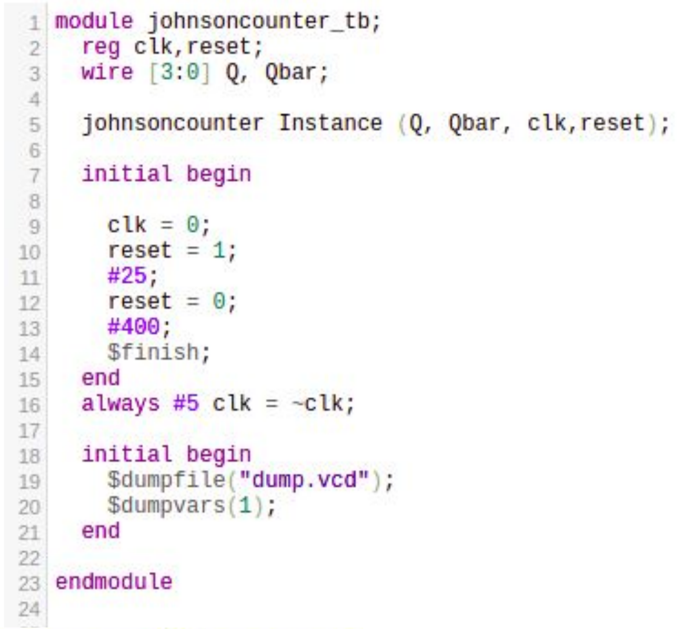


```
module Dff(reset,clk,D,q,qnot);
  input reset,clk,D;
  output reg q;
  output qnot;

  assign qnot = ~q;

  always @(posedge clk)
    if (reset)
      q <= 0;
    else
      if (D==0)
        q<=0;
      else if (D==1)
        q<=1;

endmodule

module johnsoncounter(Q,Qbar,clk,reset);

  input clk,reset;
  output [3:0] Q,Qbar;

  Dff a1 (reset,clk,Q[1],Q[0],Qbar[0]);
  Dff a2 (reset,clk,Q[2],Q[1],Qbar[1]);
  Dff a3 (reset,clk,Q[3],Q[2],Qbar[2]);
  Dff a4 (reset,clk,Qbar[0],Q[3],Qbar[3]);

endmodule
```

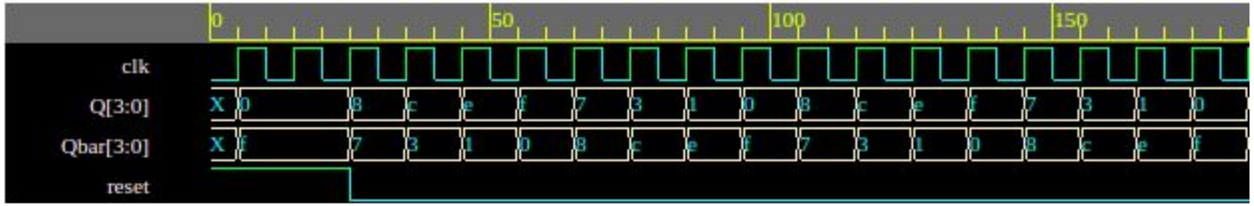
Implementation:

3. Johnson Counter:

- **Validating the Code:**

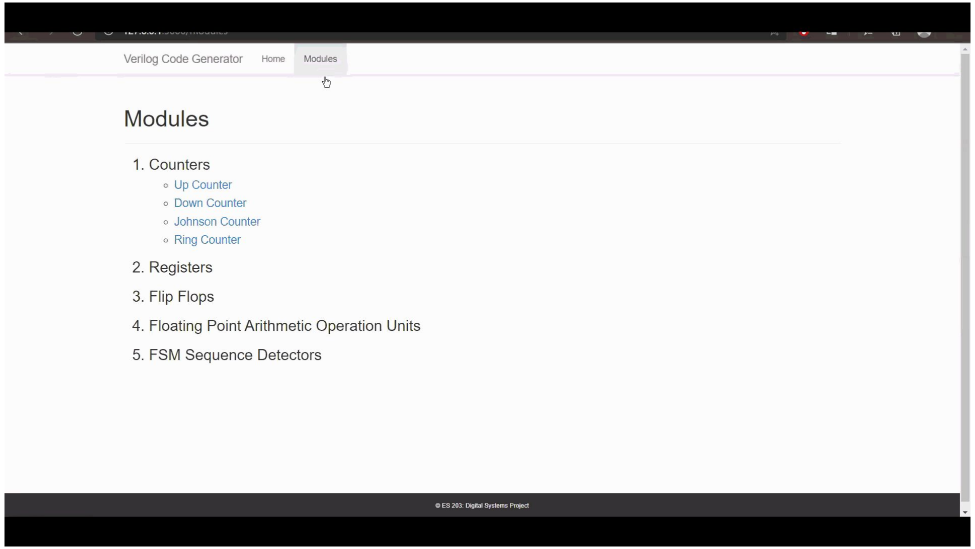


```
module johnsoncounter_tb;
  reg clk,reset;
  wire [3:0] Q, Qbar;

  johnsoncounter Instance (Q, Qbar, clk,reset);

  initial begin

    clk = 0;
    reset = 1;
    #25;
    reset = 0;
    #400;
    $finish;
  end
  always #5 clk = ~clk;

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```



Implementation:

4. Ring Counter:


Verilog Code Generator Home Modules

Modules

1. Counters
   - Up Counter
   - Down Counter
   - Johnson Counter
   - Ring Counter
2. Registers
3. Flip Flops
4. Floating Point Arithmetic Operation Units
5. FSM Sequence Detectors

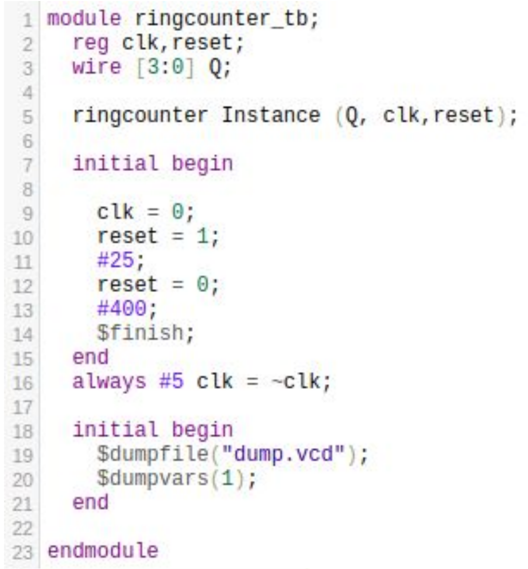
© ES 203: Digital Systems Project

```
module ringcounter(Q,clk,reset);
  parameter n=4;
  input clk,reset;
  output reg [n-1:0] Q;

  always @(posedge clk)
    if (reset)
      Q <= 1;
    else begin
      Q[n-1]<=Q[0];
      for (int i=0; i<n-1; i=i+1) begin
        Q[i] <= Q[i+1];
      end
    end
endmodule
```

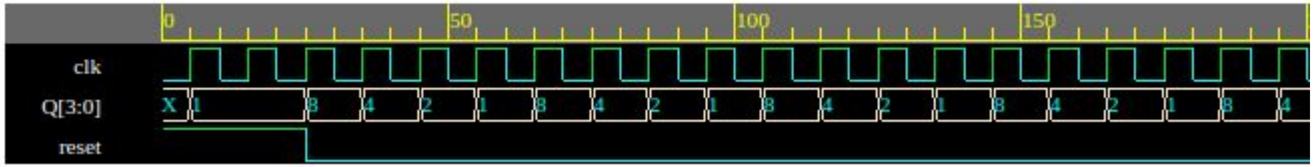
Implementation:

4. Ring Counter:

- **Validating the Code:**



```
module ringcounter_tb;
  reg clk,reset;
  wire [3:0] Q;

  ringcounter Instance (Q, clk,reset);

  initial begin

    clk = 0;
    reset = 1;
    #25;
    reset = 0;
    #400;
    $finish;
  end
  always #5 clk = ~clk;

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```



Implementation:

Flip Flops

1. JK FF
2. D Flip Flop
3. SR Flip Flop
4. T Flip Flop

Registers

1. SISO
2. PISO
3. PIPO
4. Shift Registers
5. Circular Registers

Week 3

Floating Point Arithmetic Unit

Operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division