Shift And Add (Sequential Multiplier)

Name	ID	Section
Karim Mahmoud Kamal	V23010174	16
Yossef Ibrahim Abbas	V23010442	16
Ayman Ahmed Hanafy	V23010247	16

Supervised by:

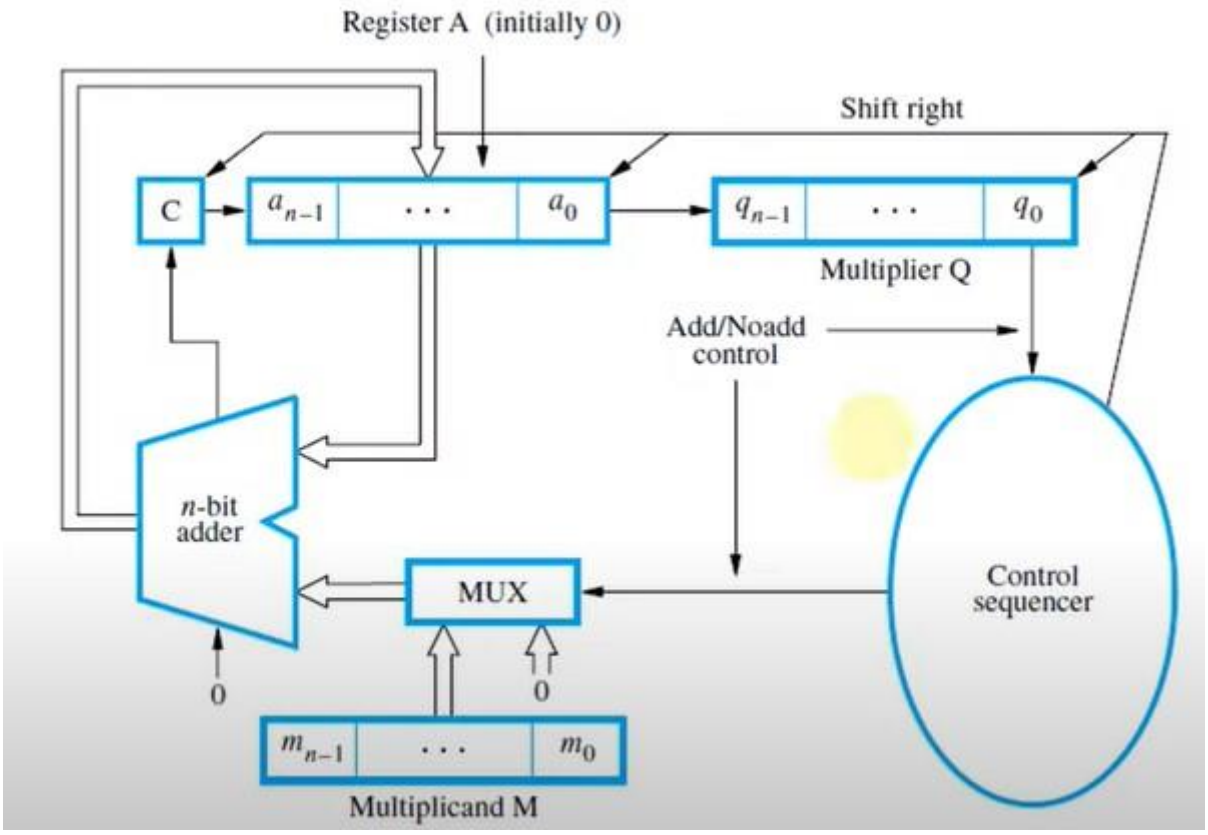
Dr. Eslam Tawfik

Eng. Mohamed El Shafey

Specifications:

- Design and implement using verilog the following 32-bits signed integer multipliers
- The multiplier will have an input register before the multiplier and an output register after the multiplier.
- It has clock, reset, and enable signals.
- Implement a testbench to test the above multipliers (Covering 8 cases):
- Multiplication of positive and negative number
- Multiplication of positive and positive number
- Multiplication of negative and negative number
- Multiplication of negative and positive number
- Multiplication by zero
- Multiplication by 1
- Additional 2 random test cases.
- Your testbench should print “TestCase#1: success” on success and should print the “TestCase#1: failed with input X and Y and Output Z and overflow status N”, elements in blue should be replaced by your values.
- Your testbench should also report the total number of success and failure testcases at the end.

Block Diagram



Logic Design (Verilog Code)

32-bit Register Module

```
module registerNbits #(
    parameter N = 32
) (
    clk,
    reset,
    en,
    inp,
    out
);
  input clk, reset, en;
  output reg [N-1:0] out;
  input [N-1:0] inp;
  always @(posedge clk) begin
    if (reset) out = 0;
    else if (en) out = inp;
  end
endmodule
```

We make a register module to apply RTL concept and to reduce delay issues.

It is a 32-bit Register and we create 2 instances of it to do multiplication.

Controller

```
module controller #(
    parameter N = 32
) (
    input clk,
    input reset,
    input [N-1:0] inputQ_wire,
    input [N-1:0] inputM_wire,
    input Mbit,
    input Qbit,
    input input_plus,
    output reg [2*N-1:0] out
);

  parameter Na = 32;
  parameter idle = 3'b000;
  parameter init = 3'b001;
  parameter test = 3'b010;
  parameter add = 3'b011;
  parameter shift = 3'b100;

  reg lsb_reg;
  reg [2:0] state;
  reg [N-1:0] Acc;
  reg [N-1:0] add_output;
  reg c_output;
  reg [N-1:0] inputQ_reg;
  reg [N-1:0] inputM_reg;
  reg start;
  integer count;
  reg [2*N-1:0] out_inv;
  reg [2*N-1:0] out_inv22;
  reg c_out_inv;

  always @(posedge clk)
    if (reset) begin
      state = 3'b000;
      count = 0;
      start = 1'b1;
      Acc = 0;
      out = 0;
      add_output = 0;
      c_output = 0;
    end else
      case (state)
        idle: begin
          inputQ_reg = inputQ_wire;
          inputM_reg = inputM_wire;
          if (start) begin
            state = init;
          end else begin
            state = idle;
          end
        end
        init: begin
          if (count < Na) begin
            count   = count + 1;
            state   = test;
            lsb_reg = inputQ_reg[0];
          end else begin
            count = 0;
            state = idle;
            start = 1'b0;
            out_inv = ~{Acc, inputQ_reg};
            {c_out_inv, out_inv22} = out_inv + input_plus;
            out = (Mbit ^ Qbit) ? out_inv22 : {Acc, inputQ_reg};
          end
        end
        test: begin
          if (lsb_reg) begin
            state = add;
          end else state = shift;
        end
        add: begin
          state = shift;
          {c_output, add_output} = inputM_wire + Acc;
        end
        shift: begin
          inputQ_reg = (lsb_reg) ? {add_output[0], inputQ_reg[N-1:1]} : {Acc[0],
inputQ_reg[N-1:1]};
          Acc = (lsb_reg) ? {c_output, add_output[N-1:1]} : {1'b0, Acc[N-1:1]};
          state = init;
        end
      endcase
endmodule
```

Top Module

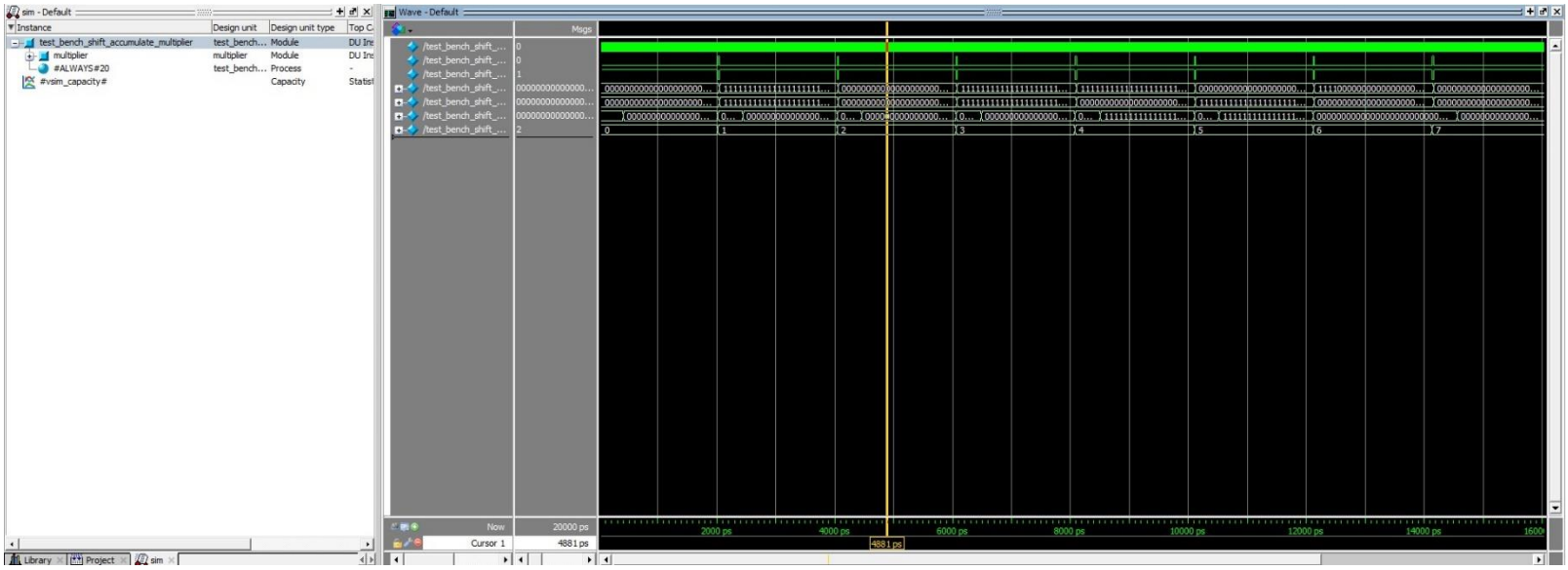
```
module multiplier #(
    parameter N = 32
) (
    input clk,
    input reset,
    input en,
    input [N-1:0] inputM,
    input [N-1:0] inputQ,
    input input_plus,
    output [2*N-1:0] out
);

  wire [N-1:0] inputM_wire;
  wire [N-1:0] inputQ_wire;
  wire [N-1:0] inputM_wire_out;
  wire [N-1:0] inputQ_wire_out;
  wire [N-1:0] inputM_inv;
  wire [N-1:0] inputQ_inv;
  wire [N-1:0] inputM_inv22;
  wire c_out_Minv;
  wire c_out_Qinv;
  wire [N-1:0] inputQ_inv22;
  wire [2*N-1:0] preout;

  assign inputM_inv = ~inputM;
  assign inputQ_inv = ~inputQ;
  assign inputM_inv22 = inputM_inv + input_plus;
  assign inputQ_inv22 = inputQ_inv + input_plus;


  assign {c_out_Minv, inputM_wire} = (inputM[N-1]) ? inputM_inv22 : inputM;
  assign {c_out_Qinv, inputQ_wire} = (inputQ[N-1]) ? inputQ_inv22 : inputQ;

  registerNbits #(32) reg1 (
      .clk(clk),
      .reset(reset),
      .en(en),
      .inp(inputM_wire),
      .out(inputM_wire_out)
  );

registerNbits #(32) reg2 (
      .clk(clk),
      .reset(reset),
      .en(en),
      .inp(inputQ_wire),
      .out(inputQ_wire_out)
  );

  controller controller (
      .clk(clk),
      .reset(reset),
      .inputQ_wire(inputQ_wire),
      .inputM_wire(inputM_wire),
      .Mbit(inputM[N-1]),
      .Qbit(inputQ[N-1]),
      .input_plus(input_plus),
      .out(preout)
  );

  registerNbits #(64) reg3 (
      .clk(clk),
      .reset(reset),
      .en(en),
      .inp(preout),
      .out(out)
  );

endmodule
```

TestBench

```
module test_bench_shift_accumulate_multiplier;

  reg clk, reset,en;
  reg  [31:0] inputM;
  reg  [31:0] inputQ;
  wire [63:0] out;
  integer counter;

  multiplier multiplier (
      .clk(clk),
      .reset(reset),
      .en(en),
      .inputM(inputM),
      .inputQ(inputQ),
      .input_plus(1'b1),
      .out(out)
  );

  always #2 clk = ~clk;

  initial begin
    clk   = 1'b0;
    reset = 1'b1;
    en =1'b0;
    counter=0;

  /*-------------------------------------First Test Case-------------------------------------*/
    #20;
    reset = 0;
    en =1'b1;
    //2 posatives ==> 14
    inputM = 32'b0000_0000_0000_0000_0000_0000_0000_0111;
    inputQ = 32'b0000_0000_0000_0000_0000_0000_0000_0010;

  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 14 ) begin
    $display( "TestCase# 1 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 1 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    //two negatives ==> 10
    inputM = 32'b1111_1111_1111_1111_1111_1111_1111_1110;
    inputQ = 32'b1111_1111_1111_1111_1111_1111_1111_1011;

  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 10 ) begin
    $display( "TestCase# 2 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 2 : FAILED  \n" ) ;
    end

    reset = 1'b1;
    en =1'b0;

    #20;
    reset = 0;
    en =1'b1;
    //2 posatives ==> 9
    inputM = 32'b0000_0000_0000_0000_0000_0000_0000_0011;
    inputQ = 32'b0000_0000_0000_0000_0000_0000_0000_0011;


  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 9 ) begin
    $display( "TestCase# 3 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 3 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    //two negatives ==> -6
    inputM = 32'b1111_1111_1111_1111_1111_1111_1111_1110;
    inputQ = 32'b1111_1111_1111_1111_1111_1111_1111_1101;
```

```
  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 6 ) begin
    $display( "TestCase# 4 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 4 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    // M -ve & Q +ve ==> -10
    inputM = 32'b1111_1111_1111_1111_1111_1111_1111_1011;
    inputQ = 32'b0000_0000_0000_0000_0000_0000_0000_0010;


  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == -10 ) begin
    $display( "TestCase# 5 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 5 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    // M +ve & Q -ve ==> -10
    inputM = 32'b0000_0000_0000_0000_0000_0000_0000_0010;
    inputQ = 32'b1111_1111_1111_1111_1111_1111_1111_1011;




  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ($signed(out) == -10 ) begin
    $display( "TestCase# 6 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 6 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    // multiply by 0 ==> 0
    inputM = 32'b1111_0000_0000_0000_0000_0000_1111_0101;
    inputQ = 32'b0000_0000_0000_0000_0000_0000_0000_0000;



  /*-------------------------------------First Test Case-------------------------------------*/
    #2000;

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 0 ) begin
    $display( "TestCase# 7 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 7 : FAILED  \n" ) ;
    end

    reset = 1;
    en=1'b0;

    #20;
    reset = 0;
    en=1'b1;
    // multiply by 1
    inputM = 32'b0000_0000_0000_0000_0000_0000_0000_0001;
    inputQ = 32'b0000_0000_0000_0000_0000_0000_0000_1111;

    #2000

    $display( "inputA:%d , inputB:%d , output:%d  \n",$signed(inputM) , $signed(inputQ)
, $signed(out) ) ;

    if ( $signed(out) == 15 ) begin
    $display( "TestCase# 8 : SUCCESS  \n" ) ;
    counter=counter+1;
    end
    else begin
    $display( "TestCase# 8 : FAILED  \n" ) ;
    end

    #100
    $display( "The number of success test cases:%d  \n", counter ) ;

  end

endmodule
```

Waveform



Log Results

```
# inputA:          7 , inputB:          2 , output:                  14
#
# TestCase# 1 : SUCCESS
#
# inputA:         -2 , inputB:         -5 , output:                  10
#
# TestCase# 2 : SUCCESS
#
# inputA:          3 , inputB:          3 , output:                   9
#
# TestCase# 3 : SUCCESS
#
# inputA:         -2 , inputB:         -3 , output:                   6
#
# TestCase# 4 : SUCCESS
#
VSIM 7> run
# inputA:         -5 , inputB:          2 , output:                 -10
#
# TestCase# 5 : SUCCESS
#
# inputA:          2 , inputB:         -5 , output:                 -10
#
# TestCase# 6 : SUCCESS
#
# inputA: -268435211 , inputB:          0 , output:                   0
#
# TestCase# 7 : SUCCESS
#
# inputA:          1 , inputB:         15 , output:                  15
#
# TestCase# 8 : SUCCESS
#
# The number of success test cases:          8
#

VSIM 7>
```

Description

A 32-bit signed sequential multiplier designed using a switch-case structure described as follows:

1- This sequential multiplier is a digital circuit that performs the multiplication of two 32-bit signed numbers using a controlled sequence of steps. It employs a switch-case structure to manage and control its operation, consisting of four main states: Initialize, Idle, Add, and Shift.
2- Initialize State: In this state, the multiplier initializes the operation. It sets up the necessary registers and variables to prepare for the multiplication. This step is essential for ensuring that the multiplier begins with a clean slate for each new multiplication operation.
3- Idle State: After initialization, the multiplier enters the idle state. In this state, it waits for the input values, which are the two 32-bit signed numbers that need to be multiplied. These numbers are usually stored in registers or memory locations.
4- Add State: Once the input values are available, the multiplier enters the add state. In this state, the multiplier performs the addition operation. It accumulates the results of partial products obtained during the multiplication. This addition typically involves adding two numbers, and the result is stored in a temporary accumulator.
5- Shift State: After the addition, the multiplier enters the shift state. In this state, it shifts the contents of the temporary accumulator to the left, mimicking the shifting of digits in a traditional long multiplication process. This left shift is usually accompanied by a shift of the other operand (the multiplicand) to the right.
6- The cycle of Add and Shift states continues until all bits of the multiplier have been processed, resulting in the completion of the multiplication operation. The final result is stored in the accumulator or in the designated output register.

This 32-bit signed sequential multiplier is a fundamental component in digital signal processing and arithmetic operations, enabling the efficient multiplication of two signed numbers. Its switch-case structure allows for precise control over the various stages of the multiplication process, ensuring accurate results for a wide range of applications.