Fall 2019

California State University, Northridge

Department of Electrical and Computer Engineering

ECE 526L

Experiment # 6
Sum of Products

October 24, 2019
SUBMITTED: November 14, 2019

Authors: Ridge Tejuco
Professor: Ronald Mehler Ph.D

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) _____

Name (signed) _____

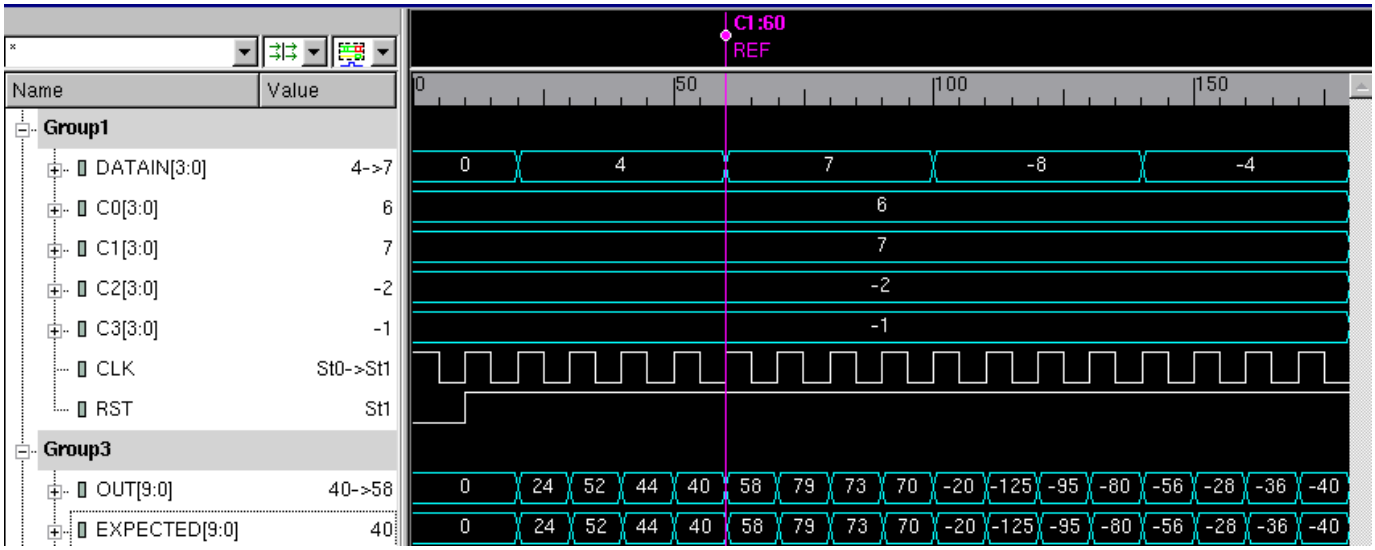
Date _____

Analysis

Before an exhaustive Test Bench was done, the vectors 0, 4, 7, 8, and C were input consecutively every 4 clock cycles. And multiplied against the constant values of the experiment which were 6, 7, E, and F. Looking closely at Figure 1, The reset sets all Registers to 4'b0. Once the input is incremented at 60 ns, OUT starts to show the expected values.

The chosen vectors are robust because 0111 test the value 7 which when multiplied by 7 gives us the largest signed value possible through multiplication. Likewise, the value 1000 tests -8*7 which gives us the lowest value through multiplication.

Figure 1. Non-exhaustive Test



For example the following equations test the output.
When the registers all equal 7.
(7*6) + (7*7) + (7*-2) + (7*-1) = 70

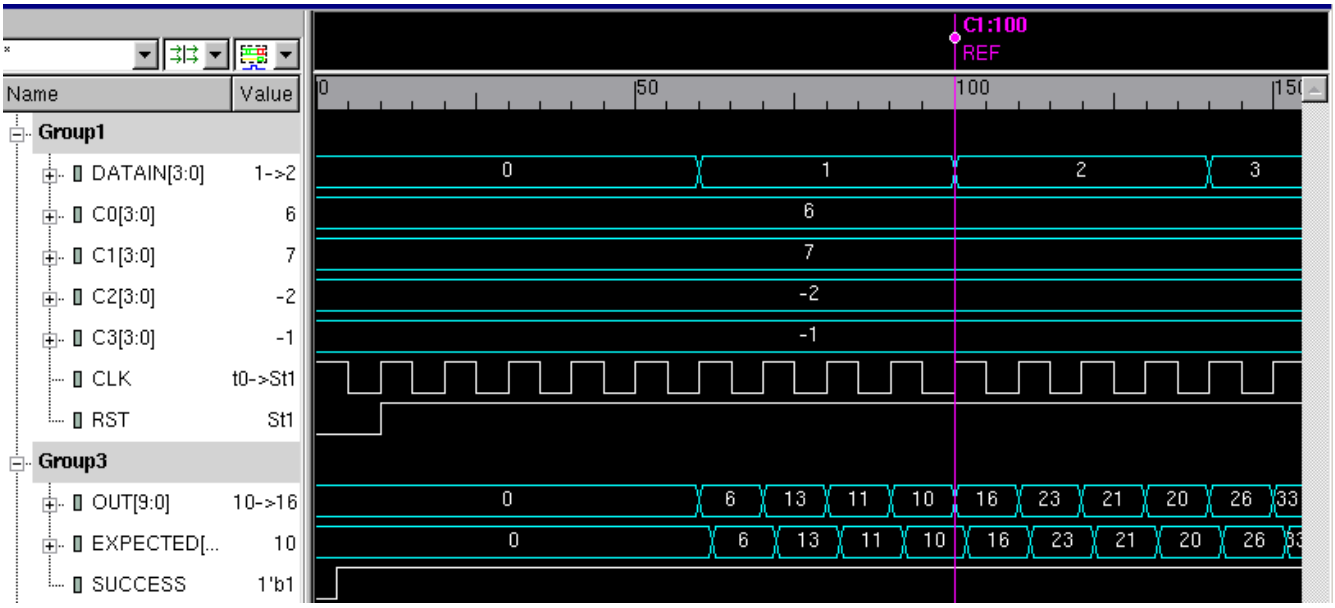
(7*7) tests if data is lost through multiplication.
(7*6) + (7*7) tests if there is overflow in addition.

When the registers all equal 8
(-8*6) + (-8*7) + (-8*-2) + (-8*-1) = -80

(-8*7) tests if data is lost in multiplication
(-8*6) tests overflow when adding two negative numbers.

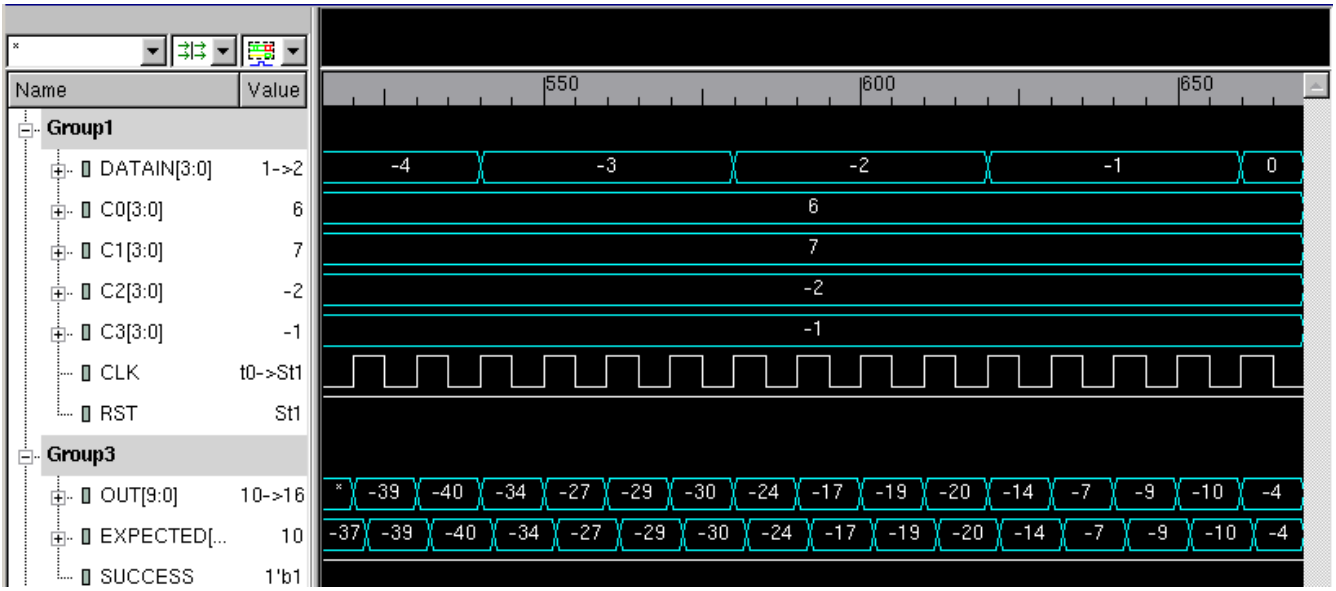
An exhaustive test bench was done by incrementing the input every 4 clock cycles. This way all constants are multiplied against the values of -8 to 7. A non-hierarchical sum of products was assigned to EXPECTED and then compared with the output OUT. If the two regs were equivalent the logic SUCCESS was assigned 1'b1. The success of this test can be seen in Figure 2a and 2b. SUCCESS is high for the initial inputs and is logic high until the end.

Figure 2a. Initial inputs of Exhaustive Test bench and log




```
Time:      0, SUCCESS: 0, OUT: 0000000000, EXPECTED: 0000000000
Time:      0, SUCCESS: 1, OUT: 0000000000, EXPECTED: 0000000000
Time:     60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000000
Time:     60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000110
```

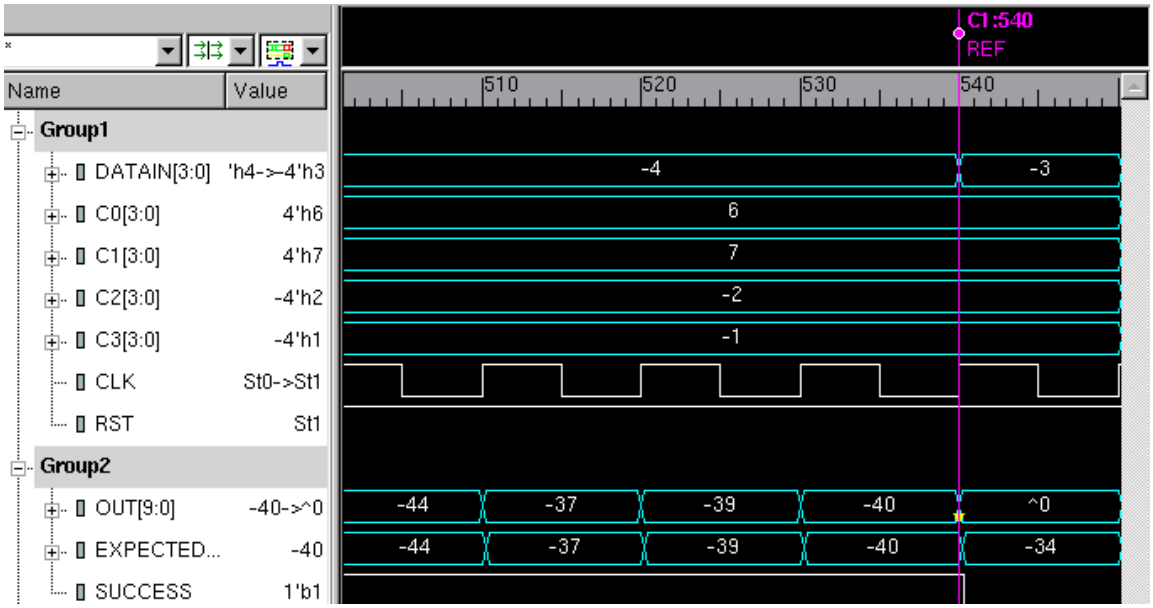
Figure 2b. Final inputs of exhaustive test bench




```
Time:  660, SUCCESS: 1, OUT: 1111111100, EXPECTED: 1111110110
Time:  660, SUCCESS: 1, OUT: 1111111100, EXPECTED: 1111111100
```

Next, the error checking was verified by forcing error after 75% of all inputs were done. Looking closer at Figure 3, the SUCCESS line drops low after the OUTPUT is forced to 4’b0 at 540 ns.

Figure 3. Final inputs of force error




```
Time:  540, SUCCESS: 1, OUT: 0000000000, EXPECTED: 1111011110
Time:  540, SUCCESS: 0, OUT: 0000000000, EXPECTED: 1111011110
```

```
//exhaustive_lab6.log
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-2_Full64; Runtime version
N-2017.12-SP2-2_Full64;  Nov 13 20:17 2019
VCD+ Writer N-2017.12-SP2-2_Full64 Copyright (c) 1991-2017 by Synopsys
Inc.
Time:     0, SUCCESS: 0, OUT: 0000000000, EXPECTED: 0000000000
Time:     0, SUCCESS: 1, OUT: 0000000000, EXPECTED: 0000000000
Time:    60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000000
Time:    60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000110
Time:    70, SUCCESS: 1, OUT: 0000001101, EXPECTED: 0000000110
Time:    70, SUCCESS: 1, OUT: 0000001101, EXPECTED: 0000001101
Time:    80, SUCCESS: 1, OUT: 0000001011, EXPECTED: 0000001101
Time:    80, SUCCESS: 1, OUT: 0000001011, EXPECTED: 0000001011
Time:    90, SUCCESS: 1, OUT: 0000001010, EXPECTED: 0000001011
Time:    90, SUCCESS: 1, OUT: 0000001010, EXPECTED: 0000001010
Time:   100, SUCCESS: 1, OUT: 0000010000, EXPECTED: 0000001010
Time:   100, SUCCESS: 1, OUT: 0000010000, EXPECTED: 0000010000
Time:   620, SUCCESS: 1, OUT: 1111110010, EXPECTED: 1111110010
Time:   630, SUCCESS: 1, OUT: 1111111001, EXPECTED: 1111110010
Time:   630, SUCCESS: 1, OUT: 1111111001, EXPECTED: 1111111001
Time:   640, SUCCESS: 1, OUT: 1111110111, EXPECTED: 1111111001
Time:   640, SUCCESS: 1, OUT: 1111110111, EXPECTED: 1111110111
Time:   650, SUCCESS: 1, OUT: 1111110110, EXPECTED: 1111110111
Time:   650, SUCCESS: 1, OUT: 1111110110, EXPECTED: 1111110110
Time:   660, SUCCESS: 1, OUT: 1111111100, EXPECTED: 1111110110
Time:   660, SUCCESS: 1, OUT: 1111111100, EXPECTED: 1111111100
$finish called from file "TB_TOP_SOP.sv", line 63.
$finish at simulation time              6700
          V C S   S i m u l a t i o n   R e p o r t
Time: 670000 ps
CPU Time:      0.200 seconds;      Data structure size:   0.0Mb
Wed Nov 13 20:17:21 2019
```

```
//force_lab6.log

Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-2_Full64; Runtime version
N-2017.12-SP2-2_Full64;  Nov 13 20:13 2019
VCD+ Writer N-2017.12-SP2-2_Full64 Copyright (c) 1991-2017 by Synopsys
Inc.
Time:     0, SUCCESS: 0, OUT: 0000000000, EXPECTED: 0000000000
Time:     0, SUCCESS: 1, OUT: 0000000000, EXPECTED: 0000000000
Time:    60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000000
Time:    60, SUCCESS: 1, OUT: 0000000110, EXPECTED: 0000000110
Time:    70, SUCCESS: 1, OUT: 0000001101, EXPECTED: 0000000110
Time:    70, SUCCESS: 1, OUT: 0000001101, EXPECTED: 0000001101
Time:    80, SUCCESS: 1, OUT: 0000001011, EXPECTED: 0000001101
Time:    80, SUCCESS: 1, OUT: 0000001011, EXPECTED: 0000001011
Time:    90, SUCCESS: 1, OUT: 0000001010, EXPECTED: 0000001011
Time:    90, SUCCESS: 1, OUT: 0000001010, EXPECTED: 0000001010
Time:   100, SUCCESS: 1, OUT: 0000010000, EXPECTED: 0000001010
Time:   100, SUCCESS: 1, OUT: 0000010000, EXPECTED: 0000010000
Time:   110, SUCCESS: 1, OUT: 0000010111, EXPECTED: 0000010000
Time:   110, SUCCESS: 1, OUT: 0000010111, EXPECTED: 0000010111
Time:   120, SUCCESS: 1, OUT: 0000010101, EXPECTED: 0000010111
Time:   120, SUCCESS: 1, OUT: 0000010101, EXPECTED: 0000010101
Time:   130, SUCCESS: 1, OUT: 0000010100, EXPECTED: 0000010101
Time:   130, SUCCESS: 1, OUT: 0000010100, EXPECTED: 0000010100
Time:   140, SUCCESS: 1, OUT: 0000011010, EXPECTED: 0000010100

Time:   540, SUCCESS: 1, OUT: 0000000000, EXPECTED: 1111011000
Time:   540, SUCCESS: 1, OUT: 0000000000, EXPECTED: 1111011110
Time:   540, SUCCESS: 0, OUT: 0000000000, EXPECTED: 1111011110
$finish called from file "TB_TOP_SOP.sv", line 43.
$finish at simulation time              5503
          V C S   S i m u l a t i o n   R e p o r t
Time: 550300 ps
CPU Time:      0.210 seconds;      Data structure size:   0.0Mb
Wed Nov 13 20:13:37 2019
```

```
/*=================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
=================================================
TB_TOP_SOP.v
GOALS
- non-exhaustive TB
- Exhaustive TB
- Force Errors, stop if error found
- log file $monitoroff and $monitoron
=================================================*/

`timescale 1ns/ 100ps
`define FORCE_ERROR 0//comment out to test all inputs
module TB_TOP_SOP();
    parameter SIZE = 4, PERIOD = 10;
    reg signed [(2*SIZE)+1:0] OUT,EXPECTED;
    reg signed [SIZE - 1: 0] DATAIN,C0,C1,C2,C3;
  reg signed [SIZE - 1:0] F0,F1,F2,F3;
  logic SUCCESS;// For error checking
    reg CLK,RST;
    //TOP_ADD(OUT,DATAIN,C0,C1,C2,C3,CLK,RST);
    TOP_SOP #(SIZE) UUT(OUT,DATAIN,C0,C1,C2,C3,CLK,RST);
    //SETUP CLK
    initial begin
      CLK <= 1'b1;
      forever #(5) CLK <= ~CLK;
    end
  // Begin Error checking simulation at CLK
  always@(posedge CLK) begin
     #0.1 F0 <= UUT.MID0.REG0.DOUT;
     F1 <= UUT.MID0.REG1.DOUT;
     F2 <= UUT.MID1.REG0.DOUT;
     F3 <= UUT.MID1.REG1.DOUT;
     #0.1 EXPECTED <= (C0*F0)
     + (C1*F1)
     + (C2*F2)
     + (C3*F3);
     #0.1 if(OUT == EXPECTED) SUCCESS <= 1;
     else begin
```

```
      SUCCESS <= 0;
      #PERIOD $finish;// If fail stop simulation
      end
  end
  // Begin simulation
    initial begin
      $vcdpluson;
       C0 = 4'h6; C1 = 4'h7; C2 = 4'hE; C3 = 4'hF;
      RST = 0; DATAIN = 4'b0;SUCCESS <= 0;EXPECTED = 4'b0;
      $monitor("Time:%5d, SUCCESS: %b, OUT: %10b, EXPECTED: %10b
",$time,SUCCESS,OUT,EXPECTED);
      #PERIOD RST = 1;
      #PERIOD repeat(16) begin #(PERIOD*4)
      DATAIN <= DATAIN + 1;
      if(DATAIN > 4'b1011) $monitoron; // log first 25%
      else if(DATAIN > 4'b0011) $monitoroff; //log last 25%
      else;
      `ifdef FORCE_ERROR
      if(DATAIN > 4'b1011) begin // last 25% of inputs
           force OUT = {(SIZE-1){1'b0}};
      end
      `endif
      end
       #PERIOD $finish;
    end
endmodule
```

```
/*=================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
=================================================
TOP_SOP.sv
GOALS
- TWO instances of MID_MULT_ACC.v
- MID0_OUT => MID_1IN;
- C0, C1, C2, C3 => MULTIPLIERS
- ADD_OUT0 => ADDER(X); ADD_OUT1 => ADDER(Y);
=================================================*/
`timescale 1ns/ 100ps
module TOP_SOP(OUT,DATAIN,C0,C1,C2,C3,CLK,RST);
    parameter SIZE = 4;
    output wire [(2*SIZE) + 1: 0] OUT;// 2 adders 1 Mult
    input wire [SIZE - 1: 0] DATAIN,C0,C1,C2,C3;
    input wire CLK,RST;
    wire [SIZE-1:0] MID0_DOUT,NOUT;
    wire [2*SIZE:0] ADD_OUT0, ADD_OUT1;
    //MID_MULT_ACC(MID_OUT,DATAOUT,DATAIN,C0,C1,CLK,RST);
    MID_MULT_ACC #(SIZE) MID0(ADD_OUT0,MID0_DOUT,DATAIN,C0,C1,CLK,RST);
    MID_MULT_ACC #(SIZE) MID1(ADD_OUT1,NOUT,MID0_DOUT,C2,C3,CLK,RST);
    //ADDER(X,Y,OUT);
    ADDER #((2*SIZE)+1) ADD0(ADD_OUT0,ADD_OUT1,OUT);
endmodule
```

```
/*=====================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
REGISTER.sv
=====================================================
GOALS
- behavioral register
=====================================================*/
`timescale 1ns/ 100ps
module REGISTER (DIN,DOUT,CLK,RST);
     parameter SIZE = 4;
     input wire [SIZE - 1: 0] DIN;
     output reg [SIZE - 1: 0] DOUT;
     input wire CLK, RST;

     always@ (posedge CLK, negedge RST) begin
          if(RST)
               DOUT <= DIN;
          else
               DOUT <= {(SIZE-1){1'b0}};
     end
endmodule
```

```
/*=================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
MULTIPLIER.sv
=================================================
GOALS
- scalable multiplier
- scaled input x and y; signed values
- scaled output; signed values
- OUT = X*Y behavioral multiplication?
=================================================*/
`timescale 1ns/ 100ps
module MULTIPLIER(X,Y,OUT);
    parameter SIZE = 4;
    input wire signed [SIZE - 1: 0] X,Y;
    output reg signed[(2*SIZE) - 1: 0] OUT; //OUT 2 times X/Y
    always@(X,Y) begin
      OUT <= X*Y;
    end
endmodule
```

```
/*=================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
ADDER.sv
=================================================
GOALS
- scalable ADDER
- OUT = X + Y; signed addition
=================================================*/
`timescale 1ns/ 100ps
module ADDER(X,Y,OUT);
    parameter SIZE = 8;
    input wire signed [SIZE - 1: 0] X,Y;
    output reg signed [SIZE: 0] OUT; //+1 bit to avoid overflow
    always@(X,Y) begin
      OUT <= X+Y;
    end
endmodule
```

```
/*=================================================
CSUN - ECE 526 Lab - Experiment 6
Ridge Tejuco
MID_MULT_ACC.sv
=================================================
GOALS (Two registers, Two multipliers, then 1 adder)
    Registers
- DATAIN => REG_IN0
- REG_OUT0 => REG_IN1
- REG_OUT1 => DATAOUT
- CLK => CLK of REG0/REG1

    MULTIPLIERS to ADDER
- REG_OUT0 => MULT0(X);
- C0 => MULT0(Y);
- MULT0_OUT => ADD0(X);
=================================================*/
`timescale 1ns/ 100ps
module MID_MULT_ACC(MID_OUT,DATAOUT,DATAIN,C0,C1,CLK,RST);
    parameter SIZE = 1;
    output reg [(2*SIZE):0] MID_OUT;//MID_OUT is 2*X+1 greater
    output reg [SIZE - 1:0] DATAOUT;
    input wire [SIZE - 1:0] DATAIN, C0,C1;
    input wire CLK,RST;
    wire [SIZE-1:0] REG_OUT0;
    wire [(2*SIZE) - 1: 0] MULT0_OUT,MULT1_OUT;

    //instantiations
    //REGISTER (DIN,DOUT,CLK,RST);
    REGISTER #(SIZE) REG0(DATAIN,REG_OUT0,CLK,RST);
    REGISTER #(SIZE) REG1(REG_OUT0,DATAOUT,CLK,RST);
    //MULTIPLIER(X,Y,OUT);
    MULTIPLIER #(SIZE) MULT0(REG_OUT0,C0,MULT0_OUT);
    MULTIPLIER #(SIZE) MULT1(DATAOUT,C1,MULT1_OUT);
    //ADDER(X,Y,OUT);
    ADDER #(2*SIZE) ADDER0(MULT0_OUT,MULT1_OUT,MID_OUT);
    //ADDER uses SIZE of multiplier out
endmodule;
```