EEE4120F YODA

Milestone 3: Status Update- Smoothing Filter

Topic Code: P01- SF: Smoothing Filter

Sindiso Mkhatshwa (MKHSIN035) | Maneno Mgwami (MGWMAN006) | Emmanuel Francis (FRNEMM004)

Role Allocation

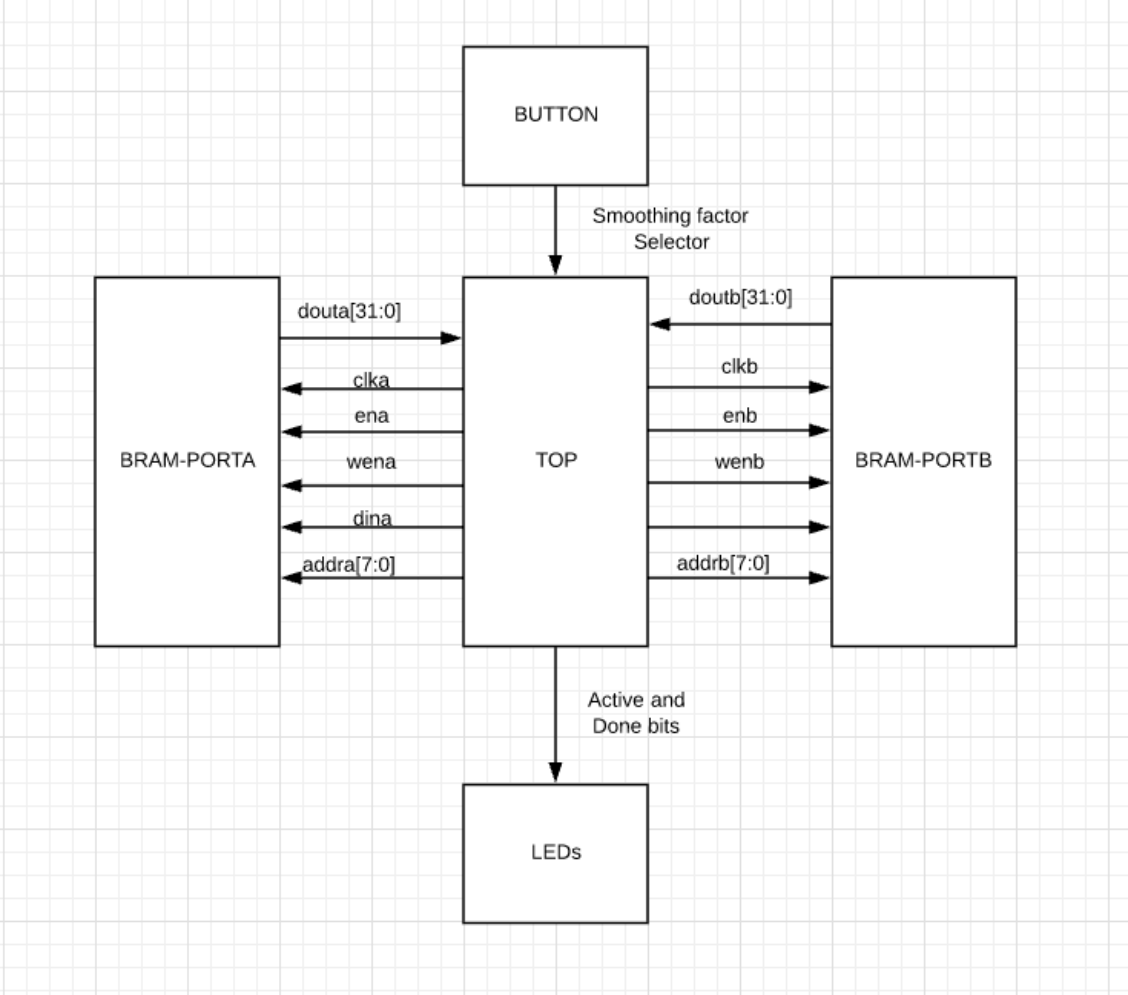
CEO: Sindiso Mkhatshwa

Chief Researcher: Maneno Mgwami

Director of Operations: Emmanuel Francis

Director of Support Services: Chris Hill

- Design document (block diagrams, schematics)



- Snippets of code

(1) Top Level Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Sindiso Mkhatshwa
//
// Create Date: 04.06.2020 05:11:12
// Design Name:
// Module Name: SmoothingFilter
// Project Name:
// Target Devices:
```

```
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module SmoothingFilter(
   input CLK100MHZ,    //Clock signal
   input BTNC,       //Reset signal
   input BTNU,        //Increase window size
   input BTND,        //Decrease window size
   input src,         //Source address
   input length,      //Length of data to be processed
   input dest,        //Destination address
   input active,     //Set this bit to indicate that device is busy working
   output done        //Set this bit to high once processing is complete
   );
   //Internal wires
   wire Reset;        //Reset signal
   wire btn_up;       //increase smoothing factor signal
   wire btn_down;     //decrease smoothing factor signal
   wire Done;
   //Internal registers
   reg [7:0] window_size;         //Smoothing factor
   //Reset
   Delay_Reset d(CLK100MHZ, BTNC, Reset);
   //Instantiate debounce module
   Debounce debounce_up(CLK100MHZ, BTNU, btn_up);        //Debounce BTNU
   Debounce debounce_dn(CLK100MHZ, BTND, btn_down);      //Debounce BTNU
   //Instantiate filter module
   filter(CLK100MHZ, Reset, length, src, dest, window_size, Done);


   always @(posedge CLK100MHZ)begin
      if(Reset)begin     //Handle reset
         window_size = 4;    //Default value of 4
      end else
      if(btn_up)begin     //Handle smoothing factor increment
         if(window_size < length) window_size = window_size + 1;
      end else
      if(btn_down)begin   //Handle smoothing factor decrease
         if(window_size > 4) window_size = window_size - 1;
```

```
      end

   end

   assign done = Done;
   assign active = ~Done;

endmodule
```

Top level module incorporating all other sub modules.

(2) LPF module and LPF Test Bench

```
timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Emmanuel Francis
// Create Date: 28.05.2020 18:48:38
// Module Name: lpf
// Project Name: YODA
// Description: Low Pass Filter
//////////////////////////////////////////////////////////////////////////////////


module lpf(
   input clk,
   input reset,
   input loadmem,
   input readmem,
   input dolpf,
   input [7:0] i, //8 bit value
   input [7:0] xin, //8 bit x input
   output reg [7:0] xout //8 bit x ouput
   );

   //reg n; //count value
   reg next; //count value
   //reg j; //count value
   reg[7:0] n; // amount of memory used
   reg[7:0] j; // (i-1) % n
   reg[7:0] k; // (i+2) % n
   reg[7:0] l; // (i+1) % n
   reg ru; // true if need to round up
   reg[9:0] sum; // sum - note need a few extra bits for carry

   // set up parameters
    parameter MEM_N = 100; // number of words of memory
```

```
// internal memory
reg[7:0] mem_raw [0:MEM_N-1];
reg[7:0] mem_lpf [0:MEM_N-1];
// Memory I/O
reg ena = 1;
reg wea = 0;
reg [7:0] addra=0;
reg [10:0] dina;
wire [10:0] douta;
//reg [7:0] mem_raw;
//reg [7:0] mem_lpf;

//Instantiate block memory
//blk_mem raw (clk,ena,wea,addra,dina,mem_raw);
//blk_mem lpf (clk,ena,wea,addra,dina,mem_lpf);


always @ (posedge clk)begin
   if(reset)begin
      n <= 0; j <= 0;
      l <= 0; k <=0;
   end//reset
   else begin
      if(loadmem)begin
         //next<=i+1;
         mem_raw[i]<= xin;
         if((i+1)>n) n<=i+1;
      end//if loadmem
      else if(readmem) xout <= mem_lpf[i];
      ///// Handle processing
      if(dolpf)begin
         /*check for backwards wrap
         if(i>0) j<=i-1; // j set to next sample
         else j<=n-1; // j set to last sample
         //check for forwards wrap
         if((i+1)>n) j<=0;// if larger than sample
         if((i+2)>n) j<=1; //if larger than sample*/

         // calculate j = (i-1)%n
         j = (i>0) ? (i-1) : (n - 1);

         // calculate l = (i+1)%n
         l = i+1;
         if (l == n) l = 0;
         else if (l == (n+1)) l = 1;

         // calculate k = (i+2)%n
         k = i+2;
```

```
            if (k == n) k = 0;
            else if (k == (n+1)) k = 1;

            // calculate the average: (a+b+c+d)/4
            sum = mem_raw[j] + mem_raw[i] + mem_raw[l] + mem_raw[k];
              ru = (sum[1]==1)? 1 : 0;
            sum = sum >> 2; // divide by 4
              if (ru) sum = sum + 1;
            mem_lpf[i] = sum;

          end //if dolpf

        end//else reset

    end// always @ clk or reset
    //raw (clk,ena,wea,addra,dina,mem_raw);
    //lpf (clk,ena,wea,addra,dina,mem_lpf);

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Emmanuel Francis
// Create Date: 14.06.2020 20:35:50
// Module Name: lpf_tb
// Project Name: YODA- Smoothing Filter
//////////////////////////////////////////////////////////////////////////////////


module lpf_tb();

     // set up regsiters
    reg  clk; // need a clock for activating the module
    reg  reset, loadmem, readmem, dolpf;
    reg  [7:0] i;
    reg  [7:0] xin;
    wire [7:0] xout;
    reg  [7:0] y_test [0:24]; // using 25 samples!

    // simulation variables
    integer j;

    // instantiate the module under test
    lpf uut (clk,reset,loadmem,readmem,dolpf,i,xin,xout);

    // perform simulation operations
    initial
    begin
```

```
$display("clk reset loadmem readmem dolpf\t   i   xin  xout");
$monitor("%b   %b\t    %b      %b      %b    %03d  %03d\t %03d",
     clk, reset, loadmem, readmem, dolpf, i, xin, xout);
// initialize control registers
clk   <= 0; reset <= 0; loadmem <= 0; readmem <= 0;
dolpf <= 0;    i <= 0;    xin <= 0;

// set up the memory
y_test[0]<=100;  y_test[1]<=125;  y_test[2]<=148;
y_test[3]<=168;  y_test[4]<=184;  y_test[5]<=185;
y_test[6]<=200;  y_test[7]<=198;  y_test[8]<=190;
y_test[9]<=177;  y_test[10]<=159; y_test[11]<=148;
y_test[12]<=113; y_test[13]<=87;  y_test[14]<=63;
y_test[15]<=41;  y_test[16]<=23;  y_test[17]<=17;
y_test[18]<=2;   y_test[19]<=0;   y_test[20]<=5;
y_test[21]<=16;  y_test[22]<=32;  y_test[23]<=52;
y_test[24]<=75;

// do a reset of the lpf
#5 reset<=1;
// do a clock so that the lpf sees the reset
#5 clk = ~clk; #5 clk = ~clk;
reset<=0;

// Load in y_test to raw data
loadmem = 1;  // change to loadmem mode
for (j = 0; j < 25; j = j + 1)
begin
  i<=j;
  xin<=y_test[j];
  // do a clk pulse to make the module process the new input
  #5 clk = ~clk; #5 clk = ~clk;
end

// change to apply lpf process
loadmem<=0; xin<=0; dolpf <= 1;
for (j = 0; j < 25; j = j + 1)
begin
  i<=j; // need to set reg i, fed to lpf, to the simulated var
  // do a clk pulse to make the module process the new input
  #5 clk = ~clk; #5 clk = ~clk;
end
// now read the memory out
dolpf<=0; xin<=0; readmem<=1;
// the results should be displayed due to the monitor
for (j = 0; j < 25; j = j + 1)
begin
  i<=j;
```

```
      #5 clk = ~clk; #5 clk = ~clk;
    end
  end //end initial begin
endmodule
```

This module loads the noisy data from memory. Once all the data has been loaded the data is processed(filtered) at once and then written to a destination address (supplied by user) in memory.

(3) Delay Module

```
//////////////////////////////////////////////////////////////////////////////////
//////////////
// Company: UCT
// Engineer: Sindiso
//
// Create Date: 06.06.2020 00:17:54
// Design Name:
// Module Name: Delay_Reset
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////////
module Delay_Reset(
    input Clk, // Input clock signal
    input BTNS, // Input reset signal (external button)
    output reg Reset // Output reset signal (delayed)
);

//----------------------------------------------------------------------
-----------
reg LocalReset;
reg [22:0] Count;
// Assume Count is null on FPGA configuration

//----------------------------------------------------------------------
-----------
always @(posedge Clk)
begin
    // Activates every clock edge
```

```
    LocalReset <= BTNS;
    // Localise the reset signal
    if(LocalReset)
        begin
        // Reset block
        Count <= 0;
        // Reset Count to 0
        Reset <= 1'b1;
        // Reset the output to 1
    end
    else if(&Count)
    begin
    // When Count is all ones...
        Reset <= 1'b0;
        // Release the output signal
        // Count remains unchanged
        // And do nothing else
    end else
    begin
        // Otherwise...
        Reset <= 1'b1;
        // Make sure the output signal is high
        Count <= Count + 1'b1;
        // And increment Count
    end
end
endmodule
```

Delay reset to reset modules on start up.

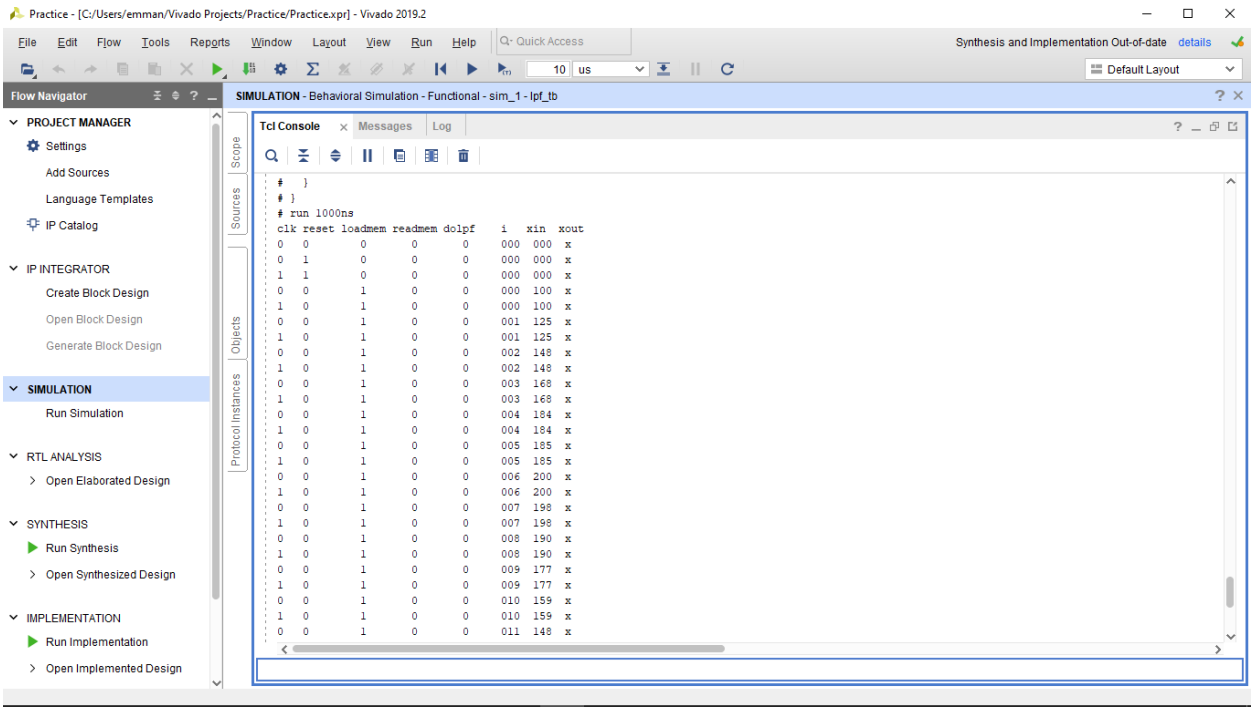
(4) Debounce Module

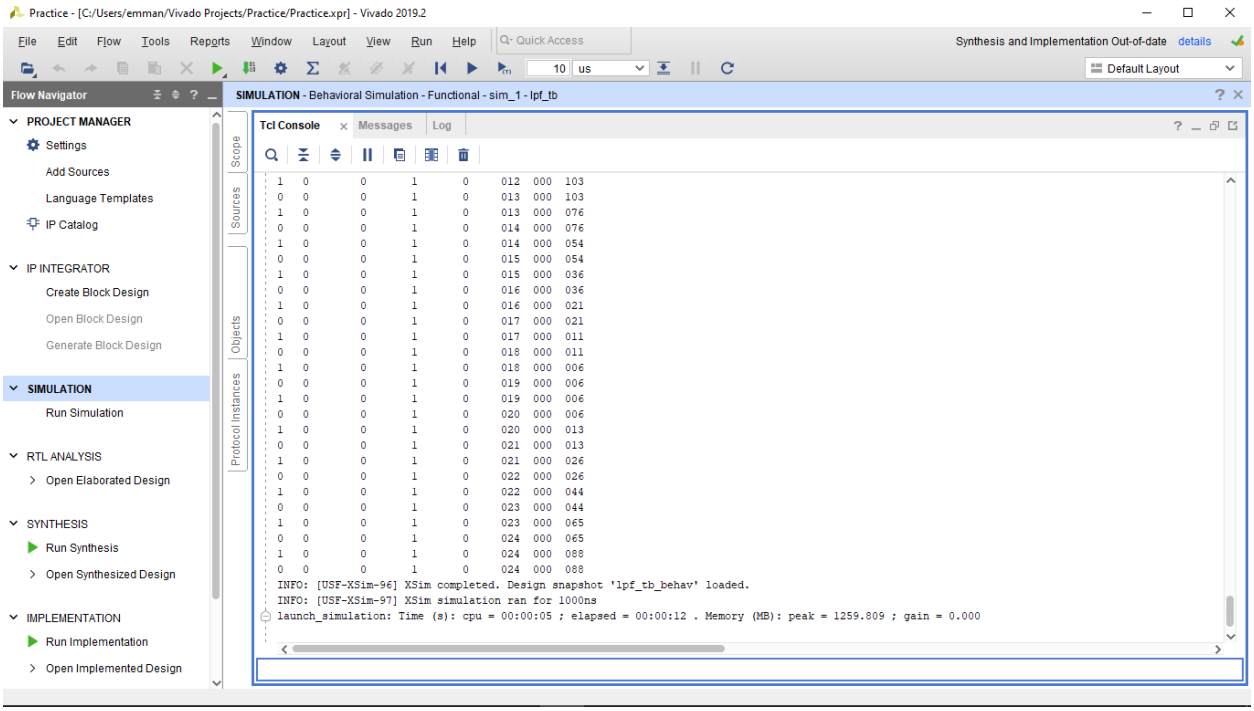
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UCT
// Engineer: Sindiso Mkhatshwa
//
// Create Date: 04.06.2020 05:03:18
// Design Name:
// Module Name: Debounce
// Project Name: YODA
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
//////////////////////////////////////////////////////////////////////
//////////////
module Debounce(
input clk,         //input clock
input Button,  //input reset signal (external button)
output reg Flag //output reset signal (delayed)
    );
//----------------------------------------------
reg previous_state;
reg [21:0]Count; //assume count is null on FPGA configuration

//----------------------------------------------
always @(posedge clk) begin      //activates every clock edge
//previous_state <= Button;         // localise the reset signal
   if (Button && Button != previous_state && &Count) begin        //
reset block
    Flag <= 1'b1;                     // reset the output to 1
     Count <= 0;
     previous_state <= 1;
  end
  else if (Button && Button != previous_state) begin
     Flag <= 1'b0;
     Count <= Count + 1'b1;
  end
  else begin
     Flag <= 1'b0;
     previous_state <= Button;
  end

end //always
//----------------------------------------------
endmodule
//----------------------------------------------
```

Debounce module to handle button presses.

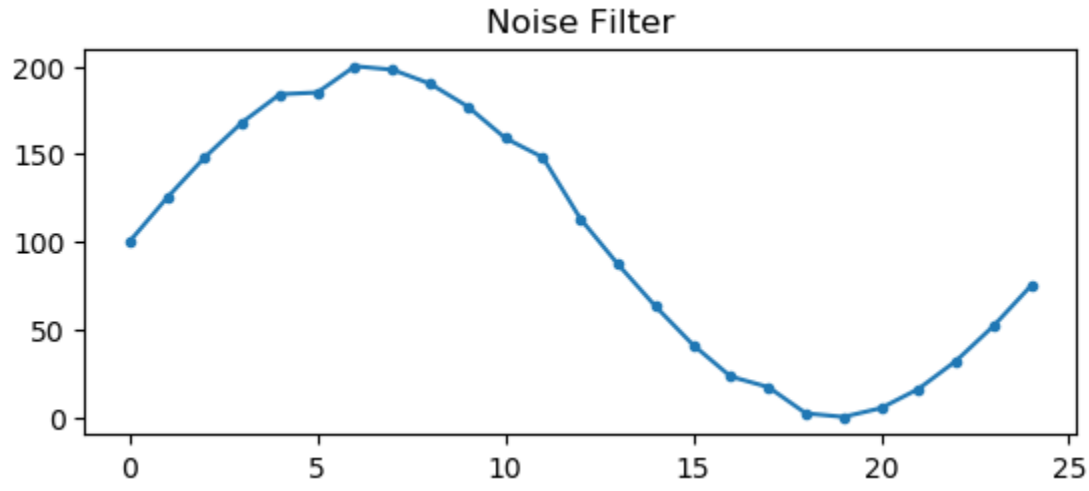
• Some initial simulations of the C code and Verilog Code, Producing the Same Output:

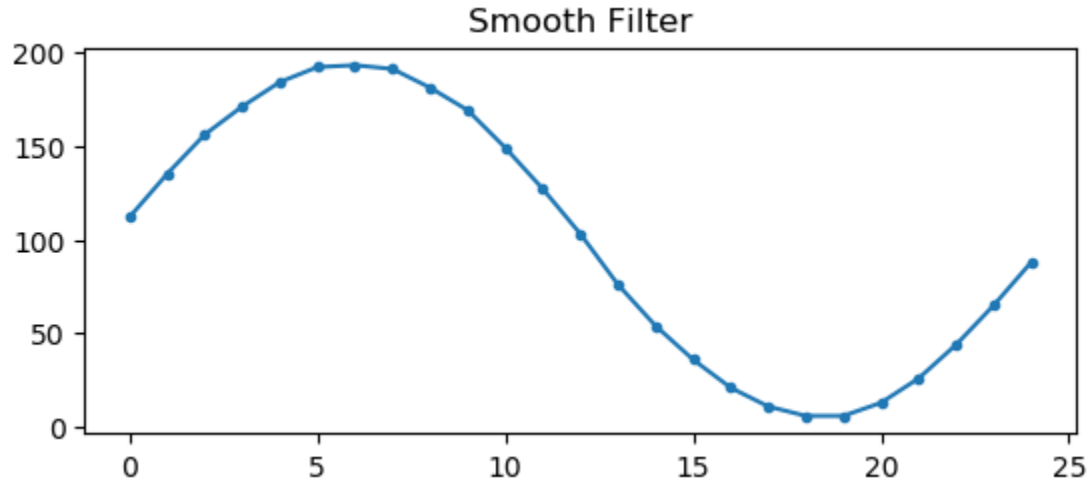


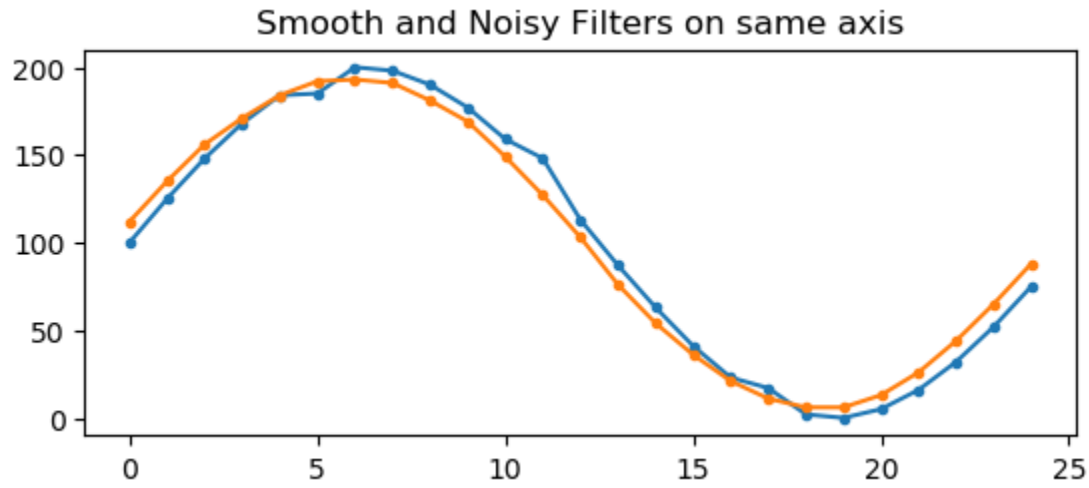



```
using PyPlotinput= [100,125, 148, 168, 184, 185, 200, 198, 190, 177, 159, 148, 113, 87, 63, 41, 23, 17, 2, 0, 5, 16, 32, 52, 75]figure(figsize=(6.4,2.5));    # default is seems to be figsize=(6.4,5) plot(input,".-");title("Noise Filter");figure(figsize=(6.4,2.5));    # default is seems to be figsize=(6.4,5)
output = [112,135,156,171,184,192,193,191,181,169,149,127,103,76,54,36,21,11,6,6,13,26,44,65,88]plot(output,".-");title("Smooth Filter");# Both filters on same
```

```
axisfigure(figsize=(6.4,2.5)); plot(input,".-");plot(output,".-");title("Smooth
and Noisy Filters on same axis");
```



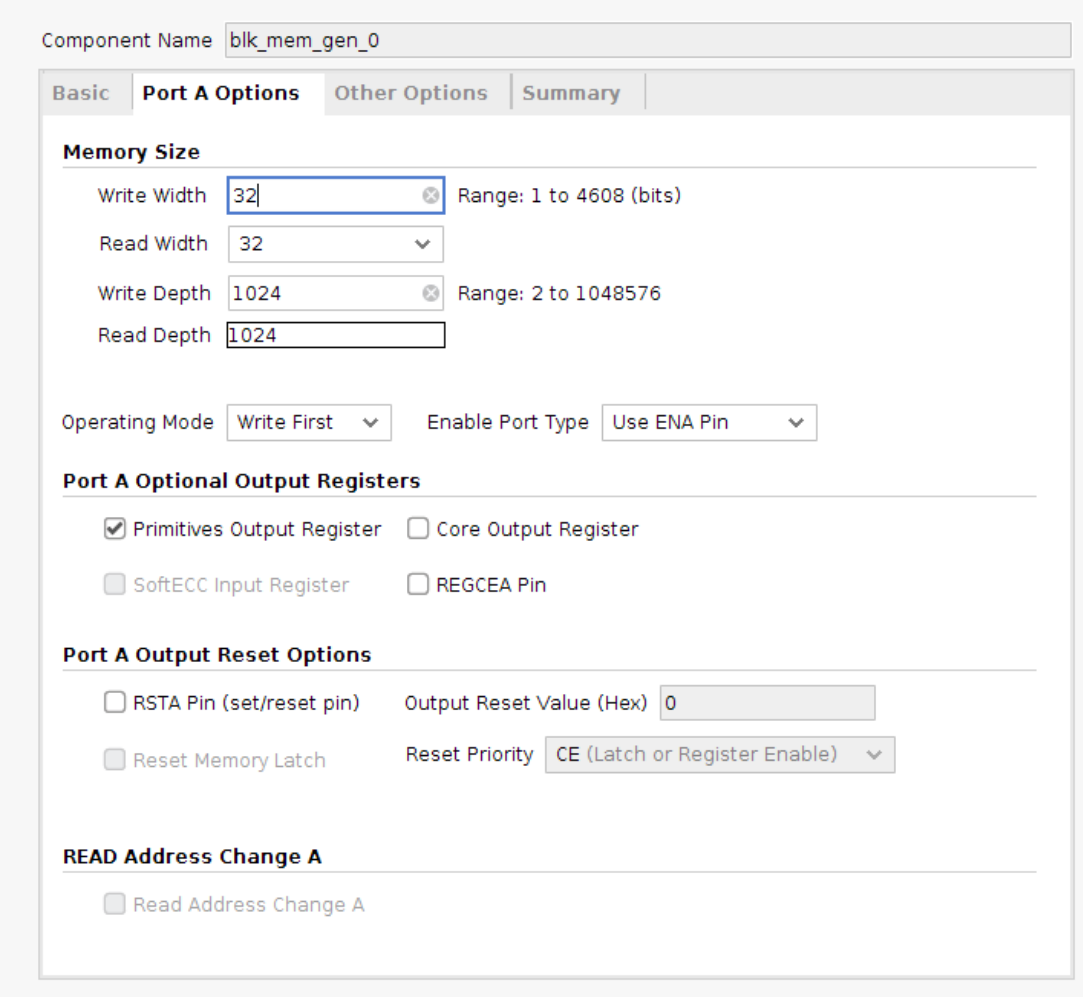




Smooth Filter = Orange

Noise Filter = Blue

Specification of BRAM



Things to be done:

- Add functionality to change smoothing- factor
- Simulate Verilog implementation and compare to golden measure
- Time allowing, implement Plan B

Things done:

- Filter algorithm has been implemented to work on 1D data
- Block RAM has been created

- Code has been implemented to read/write from/to BRAM