National Tsing Hua University
Department of Electrical Engineering
EE4292 IC Design Laboratory
Fall 2018

Homework Assignment #2 **(10%)**

Intensity-weighted average filter

Assigned on Oct 11, 2018

Due by Oct 25, 2018

Assignment Description

Intensity-weighted average filter is an adaptive filter which can remove additive noise while preserving sharp edges as shown in Figure 1.

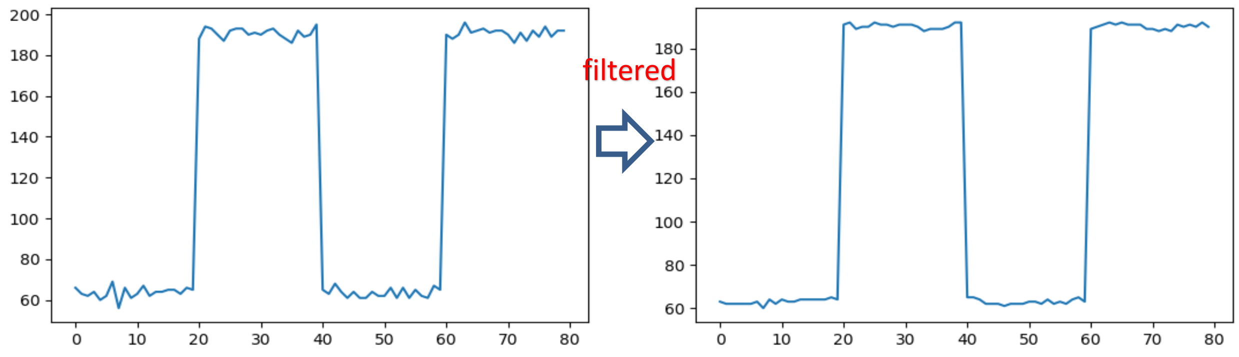


Figure 1. Left: Original signal; Right: Signal processed with filter

In this assignment you are going to implement the Intensity-weighted average filter in digital circuits using Verilog. The assignment consists of two parts: the first part is to implement an Intensity-weighted average filter, and the second part is to construct a filter engine to perform Intensity-weighted average filtering with a given iterations.

Part 1. Intensity-Weighted Average Filter (RTL, 6%)

The Intensity-weighted average filter is defined in

$$Y[n] = \frac{\sum_{k=n-3}^{n+3} w_{k,n} \cdot X[k]}{\sum_{k=n-3}^{n+3} w_{k,n}} \quad \text{and} \quad w_{k,n} = e^{-\frac{(X[k]-X[n])^2}{256}}, \tag{1}$$

where X and Y are the input and output signal respectively. W in (1) represents the filter weight which is adaptively determined by the intensity difference between filtered signal and its neighbor. In this assignment, we consider a *seven-tap* filter. For simplicity, we choose a fixed bandwidth parameter of 256, for the Gaussian kernel.

Considering hardware implementation, we will implement the filter weights with fixed-point approximation for Equation (1). Besides, the odd-symmetric boundary extension is adopted for the case where x[k] is not available. The details are summarized in three implementation guidelines as follows.

Guideline 1 – Fixed-point approximation for $w_{k,n}$:

Use an approximated 4-bit integer $w'_{k,n} = \lfloor 15 \times w_{k,n} + 0.5 \rfloor$ to replace the $w_{k,n}$ in Equation (1). Let s be $(X[k] - X[n])^2$. Then the approximation is given by the following table:

s	0-8	9-26	27-46	47-68	69-91	92-116	117-145	146-177
$w'_{k,n}$	15	14	13	12	11	10	9	8
s	178-214	215-256	257-308	309-372	373-458	459-589	590-870	871-
$w'_{k,n}$	7	6	5	4	3	2	1	0

Guideline 2 – Fixed-point approximation for the division:

Let N = $\sum_{k=n-3}^{n+3} w'_{k,n}$. Then N is the 7-bit integer divisor of Equation (1) after $w'_{k,n}$ substitutes $w_{k,n}$. The division can be approximated by multiplication and bit-shifting:

$$\frac{dividend}{divisor} = dividend \times \frac{2^{div\_shift}}{divisor} \times \frac{1}{2^{div\_shift}} = (dividend \times div\_inverse) \gg \text{div\_shift},$$

where *div_inverse* is an 8-bit integer and *div_shift* is 4-bit. The table lookup for these two signals is provided in the Verilog file *inverse_table.v*. In addition, the mul-and-shift function is given in the Verilog file *mul_and_shift.v*. Please note that the dividing result of mul-and-shift function may greater than 255, just clip the value to 255 before you quantize it to 8 bits.

Guideline 3 – Odd symmetric extension for boundaries:

In Equation (1), signals at positions before and after X[n] are required. However, for a finite input segment some of them will be unavailable when they are out-of-boundary. For example, consider the following input segment x[n] (test pattern in *filter_task_1.dat*):

0x40 0x3e 0x3f | 0x42 0x3f 0x3e 0x40 …. 0xc2 0xbd 0xc0 0xc0 | *0xc0 0xbd 0xc2*
(extended) *(extended)*

The input segment starts from X[0]=0x42 and ends at X[79]=0xc0. Then for signals before X[0] we use the odd symmetric extension X[-k]=X[k], e.g. X[-1]=X[1]=0x3f and X[-2]=X[2]=0x3e. Similarly, at the end we have X[79+k]=X[79-k].
(Note: To implement these extensions, you need to write control circuits carefully.)

The RTL top module name and I/O definition **must** be as follows.

```
    module weighted_avg_filter
    #(parameter DATA_WIDTH=8)
    (
    input clk,
    //==== control signals ====
    input rstn,                    //synchronous reset; 0: reset all your FSMs
    input valid_pixel_in,          //1: pixel_in is valid
    input first_pixel_in,          //1: pixel_in is the first pixel of the current segment
    input last_pixel_in,           //1: pixel_in is the last pixel of the current segment

    output reg valid_pixel_out,    //1: pixel_out is valid
    //==== data signals ====
    input   [DATA_WIDTH-1:0] pixel_in,         //filter input x[n]
    output reg [DATA_WIDTH-1:0] pixel_out      //filter output y[n]
  );
```

Note:
- The outputs should come from flip-flops directly and the data width is fixed as 8-bit.
- The valid signals shall indicate the validness **exactly** for the corresponding input and output.
- The *first_pixel_in* and *last_pixel_in* are pulse signals and occur only once for each input segment (or test pattern).

After finishing the RTL, you will need to pass eight input segments using the given teshbench *test_weighted_avg_filter.v*. Each input segment is stored in a pattern file *filter_task_p.dat* (p: 0-7). The first four patterns, *filter_task_p.dat* (p: 0-3), are short segments for DC, square wave, triangle wave and sine wave respectively. They have consecutive valid inputs as shown in the above timing diagram. On the other hand, the last four patterns (p: 4-7) are long segments for the same corresponding waveforms. Their valid inputs are inconsecutive and assigned randomly for stress testing of your control circuits. Besides, there will be one hidden test pattern for testing your circuits. The timing diagram for the I/O of the *weighted_avg_filter* module is shown as Figure 2.(Note that *test_loop_idx* represents for the pattern number (0-7).)

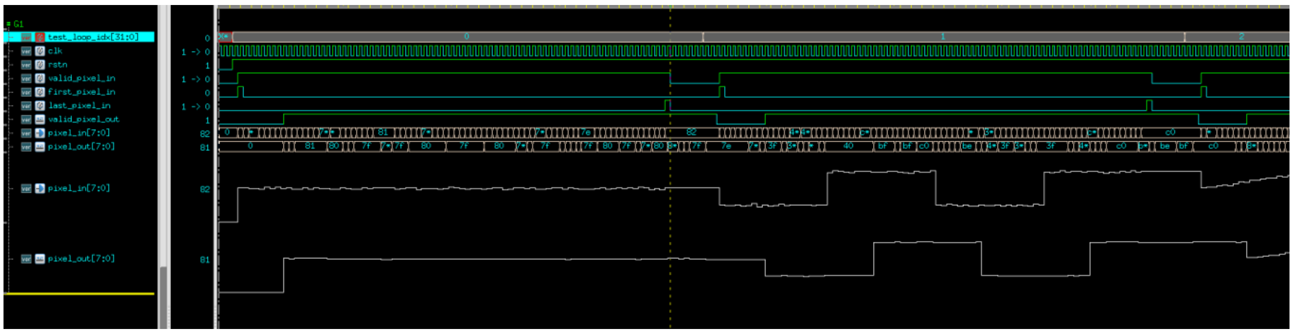
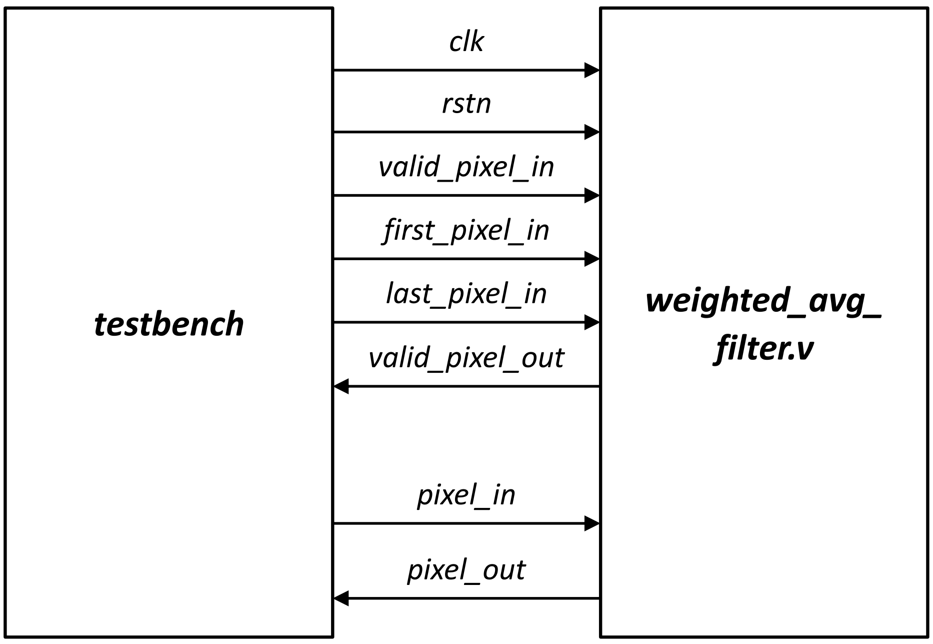


Figure 2. Timing Diagram for I/O of the *weighted_avg_filter* module



Run simulation:

1. Complete *\HW2\PART1\hdl\ weighted_avg_filter.v*
2. Change directory to *\HW2\PART1\sim*
3. Run $ncverilog –f test_weighted_avg_filter.f
4. Make sure you see the pass message in testbench

For this part, you need to complete the following missions:

1. (2%) Pass the first four test patterns *filter_task_p.dat* (p: 0-3) in *test_weighted_avg_filter.v*.
2. (2%) Pass the last four test patterns *filter_task_p.dat* (p: 4-7) in *test_weighted_avg_filter.v*.
3. (1%) Use nLint to show the synthesizability.
4. (1%) Pass the one hidden test pattern. (Run this pattern by TA).

Deliverable (Detailed deliver format is given at the bottom of this document)

1. Synthesizable Verilog functional module *weighted_avg_filter.v* and all other RTL codes for your own defined modules (if any).
2. nLint report file *nLint.rep.*

Part 2. Intensity-weighted Average Filter Engine (RTL, 4%)

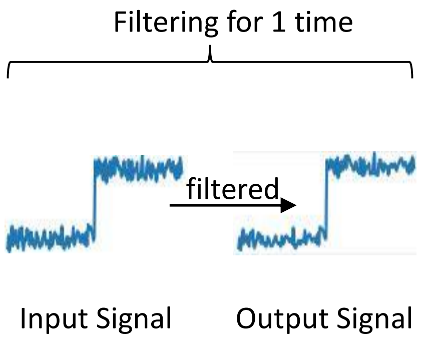
In this part, we will construct a filter engine which is capable of filtering iteratively. You need to design a filter engine which can perform different times of filtering according to a 2-bit input port *filter_mode*. You may reuse the *weighted_avg_filter* module in Part1, and design a FSM to control the circuits. Besides, the filter engine should be able to handle input segment which length is 2001 at most.

Filter Mode:

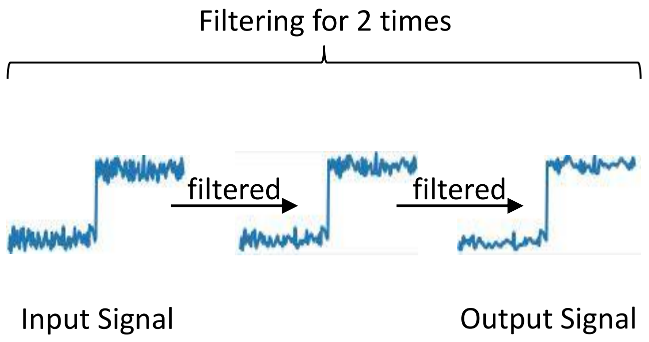
The filter engine should support four different modes. The different modes are described as following:

filter_mode	2'd0	2'd1	2'd2	2'd3
filtering times	1 time	2 times	3 times	4 times

- **Mode 0 (*filter_mode*==2'd0) :**
  Input signal should be filtered by Intensity-Weighted filter for one time.

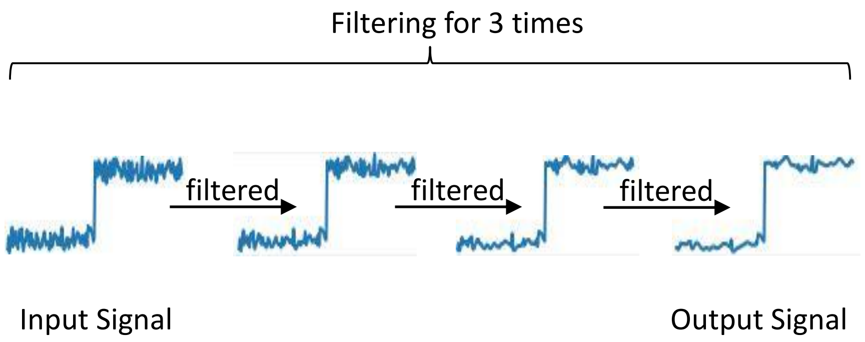


- **Mode 1 (*filter_mode*==2'd1) :**
  Input signal should be filtered by Intensity-Weighted filter for two times.



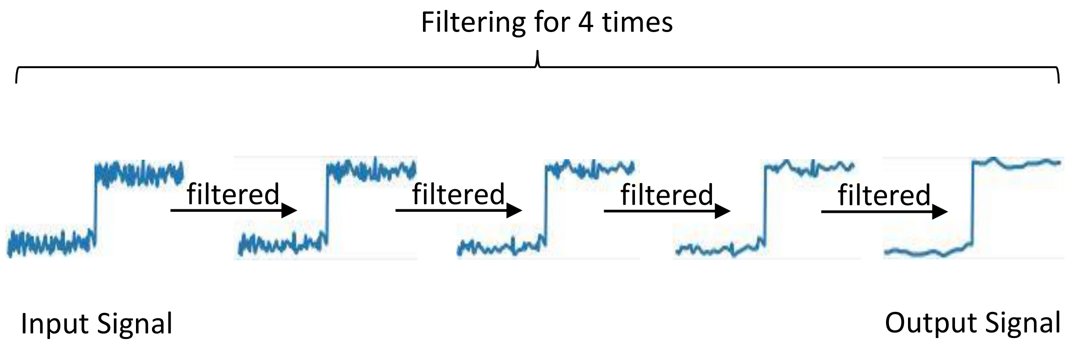
- **Mode 2 (*filter_mode*==2'd2) :**

Input signal should be filtered by Intensity-Weighted filter for three times.



- **Mode 3 (***filter_mode*==2'd3**) :**

  Input signal should be filtered by Intensity-Weighted filter for four times.



The RTL top module name and I/O definition **must** be as follows.

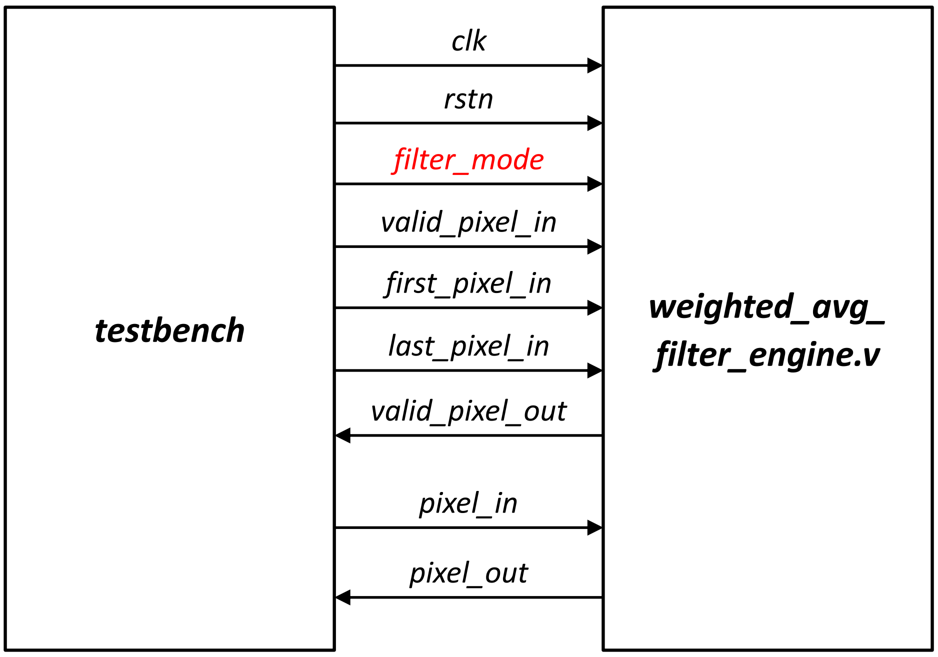
```
    module weighted_avg_filter_engine
    #(parameter DATA_WIDTH=8)
    (
    input clk,
    //==== control signals ====
    input rstn,                   //synchronous reset; 0: reset all your FSMs
    input [1:0] filter_mode,      //the mode of filter
    input valid_pixel_in,         //1: pixel_in is valid
    input first_pixel_in,         //1: pixel_in is the first pixel of the current segment
    input last_pixel_in,          //1: pixel_in is the last pixel of the current segment

    output reg valid_pixel_out,   //1: pixel_out is valid
    //==== data signals ====
    input    [DATA_WIDTH-1:0] pixel_in,      //filter input x[n]
    output reg [DATA_WIDTH-1:0] pixel_out    //filter output y[n]
  );
```

Note:

- The outputs should come from flip-flops directly and the data width is fixed as 8-bit.
- The valid signals shall indicate the validness **exactly** for the corresponding input and output.
- The *first_pixel_in* and *last_pixel_in* are pulse signals and occur only once for each input segment (or test pattern).
- In part2, the longest input sequence will have 2001 samples.

After finishing the RTL, you will need to pass eight input segments using the given testbench *test_weighted_avg_filter_engine.v*. Each input segment will be filtered by one to four times to check the correctness of your circuit.



Run simulation:

1. Complete *\HW2\PART2\hdl\test_weighted_filter_engine.v*
2. Change directory to *\HW2\ PART2\sim*
3. Run $ncverilog -f test_weighted_avg_filter_engine.f
4. Make sure you see the pass message in testbench

For this part, you need to complete the following missions:

1. (1%) Pass the first four test patterns *filter_task_p_iter.dat* (p: 0-3) in *test_weighted_avg_filter_engine.v*.
2. (1%) Pass the last four test patterns *filter_task_p_iter.dat* (p: 4-7) in *test_weighted_avg_filter_engine.v*.
3. (1%) Use nLint to show the synthesizability.
4. (1%) Pass the one hidden test pattern. (Run this pattern by TA).

Deliverable (Detailed deliver format is given at the bottom of this document)

1. Synthesizable Verilog functional module *weighted_avg_filter_engine.v* and all other RTL codes for your own defined modules (if any).
2. nLint report file *nLint.rep*.

File Organization

Directory	Filename	Description
hw2/PART1/hdl/	weighted_avg_filter.v	RTL code for weighted average filter
	inverse_table.v	Given RTL for Guideline 2
	mul_and_shift.v	Given RTL for Guideline 2
hw2/PART1/sim/	test_weighted_avg_filter.v	Testbench for testing weighted average filter
	test_weighted_avg_filter.f	File list for simulation for Part1
hw2/PART1/nLint/	hdl.f	File list for nLint (add your own files) for Part1
hw2/PART2/hdl/	weighted_avg_filter_engine.v	RTL code for weighted average filter engine
hw2/PART2/sim/	test_weighted_avg_filter_engine.v	Testbench for testing weighted average filter engine
	test_weighted_avg_filter_engine.f	File list for simulation for Part2
hw2/PART2/nLint/	hdl.f	File list for nLint (add your own files) for Part2
hw2/PART1/sim/test_pattern/	filter_task_0.dat	Test Pattern for part1 simulation
	filter_task_1.dat	
	filter_task_2.dat	
	filter_task_3.dat	
	filter_task_4.dat	
	filter_task_5.dat	
	filter_task_6.dat	
	filter_task_7.dat	
hw2/PART2/sim/test_pattern/	filter_task_0_iter.dat	Test Pattern for part2 simulation
	filter_task_1_iter.dat	
	filter_task_2_iter.dat	
	filter_task_3_iter.dat	
	filter_task_4_iter.dat	
	filter_task_5_iter.dat	
	filter_task_6_iter.dat	
	filter_task_7_iter.dat	

Delivery Format (one compressed file *HW2_10xxxxxxx.zip*):

First, put all your files in the corresponding directory under *HW2/* as follows:

1. RTL codes for Part1 and Part2 in *PART1/ hdl/* and *PART2/ hdl/* respectively.
2. nLint report files (*nLint.rep*) for Part1 and Part2 in *PART1/ nLint/* and *PART2/* nLint/ respectively.
3. HDL file list (*hdl.f*) for Part1 and Part2 in *PART1/ nLint/* and *PART2/* nLint / respectively.
4. Text file *misc.txt* under *HW2/*

Delivery File Structure:

HW2/
- PART1/
  - hdl/
    - ✓ weighted_avg_filter.v
    - ✓ …
  - nLint/
    - ✓ nLint.rep
    - ✓ hdl.f
- PART2/
  - hdl/
    - ✓ weighted_avg_filter_engine.v
    - ✓ …
  - nLint/
    - ✓ nLint.rep
    - ✓ hdl.f
- misc.txt

Note: Defining this delivery format is to help TAs grade your results more easily. You will **lose** some points if failing to follow this instruction.