CSIE DIC Lab 2023

Lab06. Multi-Bank Filter

Description

LTI systems stand for Linear Time-Invariant systems. This implies that if the same input signal **x(n)** inputs into an LTI system multiple times, the output signal **y(n)** generated each time will always be identical. The behavior of the system does not change over time.

The relationship between the system's input **x(n)** and output **y(n)** can be described by the system's impulse response **h(n)**. The impulse response is the system's response to the Dirac impulse function **δ[n]**, which is a pulse unit function indicating a pulse input at **n = 0**. If the impulse response of an LTI system is known, the output signal for any input signal can be calculated. For example, the output **y(n)** of the system input **x(n)** can be calculated using the following Eq. 1:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{1}$$

Eq. 1 is the convolution sum and calculates the relationship between the input signal **x(n)** and the impulse response **h(n)**. The * symbol in the formula represents the convolution operator, which means multiplying the two signals and summing them. Therefore, if the input signal **x(n)** and the system's impulse response **h(n)** are known, the system's output signal **y(n)** can be calculated by computing their convolution sum. It should be noted that if the input signal is infinitely long, the resulting output signal will also be infinitely long. In practical applications, only a finite length of the input signal is usually considered, so the output signal is also finite.

For instance, suppose the input signal **x(n)** has **N** values, and the system's impulse response **h(n)** has **M** values; the number of values in the output signal **y(n)** would be **(N+M-1)**. Suppose the input signal is **x(n) = [x(0), x(1), x(2), x(3), x(4), x(5)]**, and the impulse response is **h(n) = [h(0), h(1), h(2), h(3)]**. The system is causal, meaning it only responds to current and future input signals, not past ones. In this case, the input signal **x(n)** has 6 values, the system's impulse response **h(n)** has 4 values, and the system's output signal **y(n)** will have **9 (=6+4-1)** values.

y(0) = x(0)h(0)
y(1) = x(0)h(1) + x(1)h(0)
y(2) = x(0)h(2) + x(1)h(1) + x(2)h(0)
y(3) = x(0)h(3) + x(1)h(2) + x(2)h(1) + x(3)h(0)
y(4) = x(1)h(3) + x(2)h(2) + x(3)h(1) + x(4)h(0)

y(5) = x(2)h(3) + x(3)h(2) + x(4)h(1) + x(5)h(0)
y(6) = x(3)h(3) + x(4)h(2) + x(5)h(1)
y(7) = x(4)h(3) + x(5)h(2)
y(8) = x(5)h(3)

In this example, the system is causal. Therefore, when n is less than 0, the impulse response h(n) equals 0.

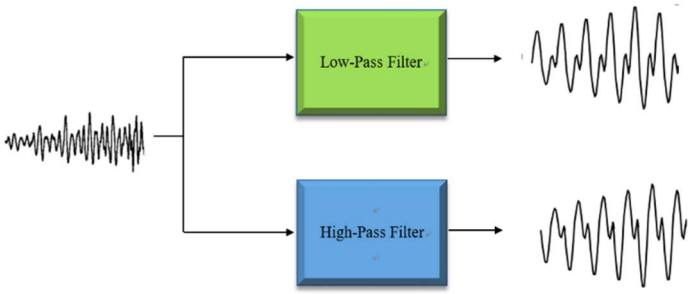

Figure 1: Multi-Bank Filter (MBF)

In this lab, you need to design a Multi-Bank Filter (MBF) circuit, as illustrated in Figure 1. This system includes a High Pass Filter (HPF) and a Low Pass Filter (LPF) that can process the input signal differently. Table I shows the impulse response coefficients for the high-pass and low-pass filters. The input signal passes through both filters separately and outputs the respective signals after processing.

Table I: filter coefficients

High-pass filter coefficient				Low-pass filter coefficient			
H(0)	0.43750	H(6)	0.87500	L(0)	0.84375	L(6)	0.50000
H(1)	0.12500	H(7)	0.50000	L(1)	0.59375	L(7)	0.59375
H(2)	0.09375	H(8)	0.96875	L(2)	0.15625	L(8)	0.34375
H(3)	0.03125	H(9)	0.87500	L(3)	0.28125	L(9)	0.37500
H(4)	0.50000	H(10)	0.96875	L(4)	0.65625	L(10)	0.50000
H(5)	0.43750	H(11)	0.53125	L(5)	0.53125	L(11)	0.96875

The general architecture of a single filter can be referred to in Figure 2, consisting of shifter registers, multipliers, and adders. In this lab, the input signal is represented using fixed-point notation, where the most significant 9 bits are the integer part, and the least significant 4 bits are the decimal part. In addition, the filter coefficients can be represented using fixed-point notation. Finally, the output signal is obtained by rounding the decimal part to the nearest integer.

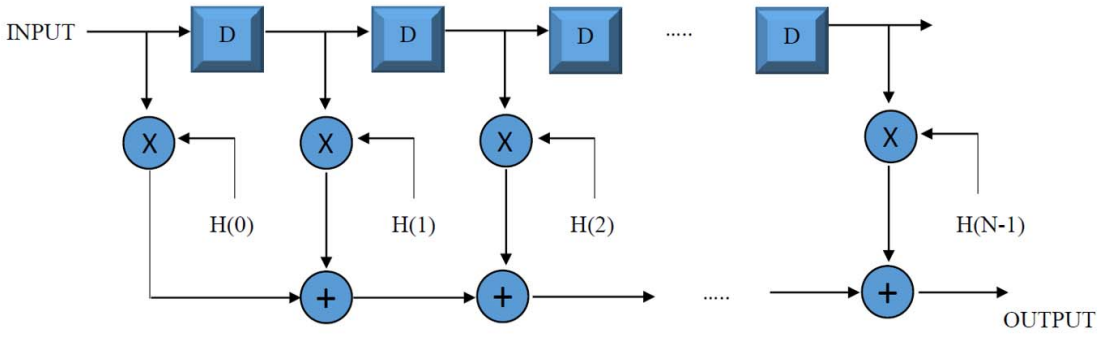

Figure 2: The general architecture of the filter

Specification

1. Top module name: MBF
2. Top module filename: MBF.v
3. Input/output definition

Signal Name	Direction	Bit Width	Description
CLK	Input	1	Clock signal
RESET	Input	1	Asynchronous reset signal (active high)
IN_VALID	Input	1	Asserted when IN_DATA is valid
IN_DATA	Input	13	Input data (unsigned number, with the most significant 9 bits as the integer and the least significant 4 bits as the decimal)
OUT_VALID	Output	1	Asserted when OUT_DATA is valid
X_DATA	Output	13	HPF Output data (unsigned integer numbers)
Y_DATA	Output	13	LPF Output data (unsigned integer numbers)

4. All input signals are **synchronized at the clock rising edge**.
5. The reset scheme is an **active-high asynchronous reset**.

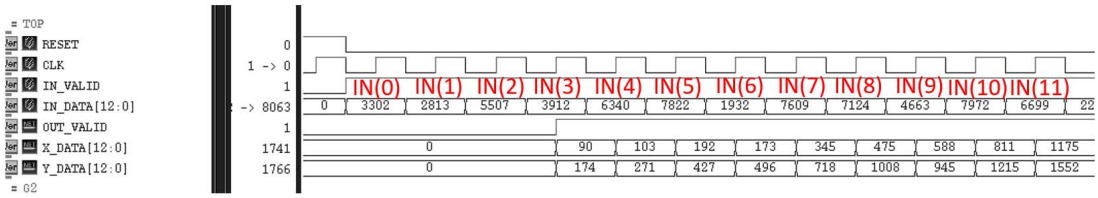

Figure 3: Input timing diagram

6. When IN_DATA is valid, the IN_VALID signal is pulled high. The input timing diagram for the data input (IN_DATA) is shown in Figure 3.

7. When the circuit is ready to start outputting the filtered outputs, the OUT_VALID signal should be pulled high until all filter output signals are output. Then the OUT_VALID signal should be pulled low again. The high-pass filter (HPF) and low-pass filter (LPF) output signals are X_DATA and Y_DATA, respectively. The detailed output timing diagram is shown in Figure 4.

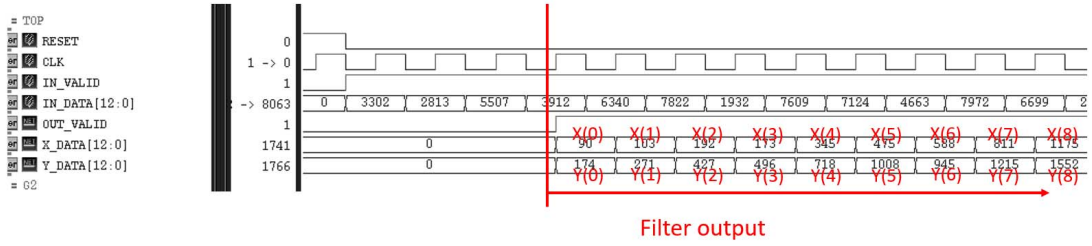

Figure 4: Output timing diagram

8. After the OUT_VALID signal is pulled low, the system will reset and continue to input the next filter input test data, as shown in Figure 5.

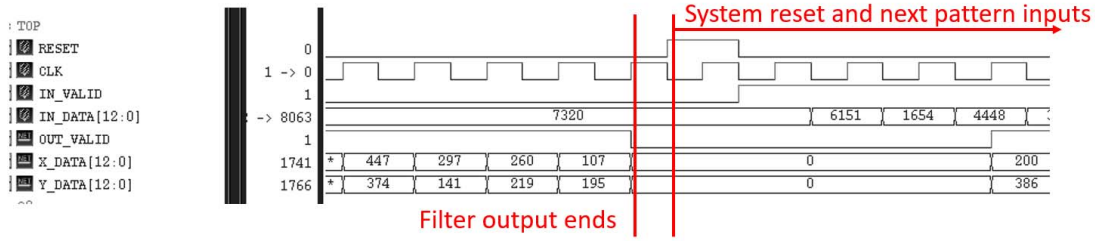

Figure 5: System resets and next pattern inputs

9. **Ensure that the latency of your design is less than 5 clock cycles**. Latency definition: the number of cycles between the first input and the first output, as illustrated in Figure 6.

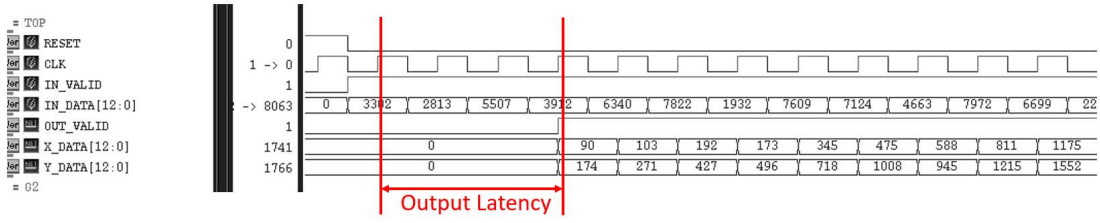

Figure 6: Output latency timing diagram

10. In logic synthesis, the timing constraint for the clock period **MUST** be **5ns**. Thus your design should operate correctly at this clock speed.

Lab Instructions

1. Extract LAB data from TA's directory:
   % tar xzvf ~dicta/lab06.tar.gz

2. The extracted LAB directory (**lab06**) contains the following:
   a. **Democase/** :Design example, including RTL code and synthesis script
   b. **Exercise/** :Test pattern for this lab

Democase:

3. Change to the directory: **Democase/**
4. This directory contains:
   a. **RTL/** :RTL source code of design example
   b. **Synthesis/** :Synthesis script
   c. **Netlist/** :Synthesized netlist and test pattern for gate-level simulation
5. Change to the directory: **RTL/**
6. Run RTL simulation
   % ncverilog –f run.f
7. Change to the directory: **Synthesis/**
8. Open and read the synthesis script and synthesis constraint file.
   % gedit SYN018_RC.tcl &
   % gedit SYN_RTL.sdc &
9. Start logic synthesis,
   % ./01_run_synthesis
10. Open and read the synthesis reports.
    % gedit report.area &
    % gedit report.timing &
    % gedit report.datapath &
    % gedit report.summary &
11. Change to directory: **Netlist/**
12. Check the difference between **(TEST.v & run.f)** in **RTL/** and **(TEST_gate.v & run.f)** in **Netlist/**.
13. Run gate-level simulation:
    % ncverilog –f run.f

Exercise:

14. Change to the directory: **Exercise**
15. This directory contains:
    a. **TEST.v** : Design test bench for RTL simulation
    b. **TEST_gate.v** : Design test bench for Gate-level simulation
    c. **MBF.v** : Empty design module
    d. **lab06.rc** : nWave saved signal file
    e. **SYN_RTL.sdc** : Design constraint file for logic synthesis
    f. **data.dat** : Input data file
    g. **high_answer.dat** : HPF Answer data file

h. **low_answer.dat** : LPF Answer data file

16. Write your RTL code in a synthesizable coding style:
    % gedit MBF.v &
    or % gvim MBF.v &
    or % joe MBF.v
    or % vim MBF.v
17. Run RTL simulation:
    % ncverilog –f run.f
18. Open waveform:
    % nWave &
    Choose **File → Open,** then select **" MBF.fsdb"** and press **OK**
19. Restore signal recorded for debugging:
    Choose **File → Restore signal**, then select **" lab06.rc"** and press **OK**
20. Meaning of signals in DEBUG signal group:
    data_number : number of input test pattern
    cnt_error_x : number of HPF errors
    cnt_error_y : number of LPF errors
    YOU_X_DATA : HPF output data of your design
    YOU_Y_DATA : LPF output data of your design
    high_ans_correct : HPF output data answer
    low_ans_correct : LPF output data answer

Logic Synthesis:

21. Repeat steps 7 to 13 described in the Demo case to synthesize your design. Also, you must prepare all your files required for logic synthesis. We only provide the design constraint file (**SYN_RTL.sdc**) for logic synthesis and the test bench (**TEST_gate.v**) for gate-level simulation. Make sure your design can pass the gate-level simulation without timing violations.