LABORATORY MANUAL

Experiment 4
Audio Processing

UNIVERSITY OF TEHRAN

School of Electrical and Computer Engineering

Digital Logic Laboratory
ECE 045

Fall 1396

EXPERIMENT 4: sessions (9, 10)

Audio processing

INTRODUCTION

In this experiment, you will be familiar with the audio processing concepts using FPGAs. Two converting interfaces ADC and DAC converters, as shown in Figure 1, are necessary to allow digital electronic equipments to process analog audio signals. Hence, you will learn how to design and use them along with your processor.

Different effects can be applied to an audio signal such as echo, down mixing and so on. In order to create a sound effect like echo you need delay elements. Such delay can be provided by a memory structure named FIFO that is introduced in this experiment. By the end of this experiment, you will be able to process an audio signal and apply an echo to it with a variable delay. Below is the topics that are explained in the following of this experiment:

- ADC and DAC converter implementation
- Playing a speech/ music file and listening to the DAC output
- Echo synthesizing principles
- FIFO structures
- Echo synthesizer

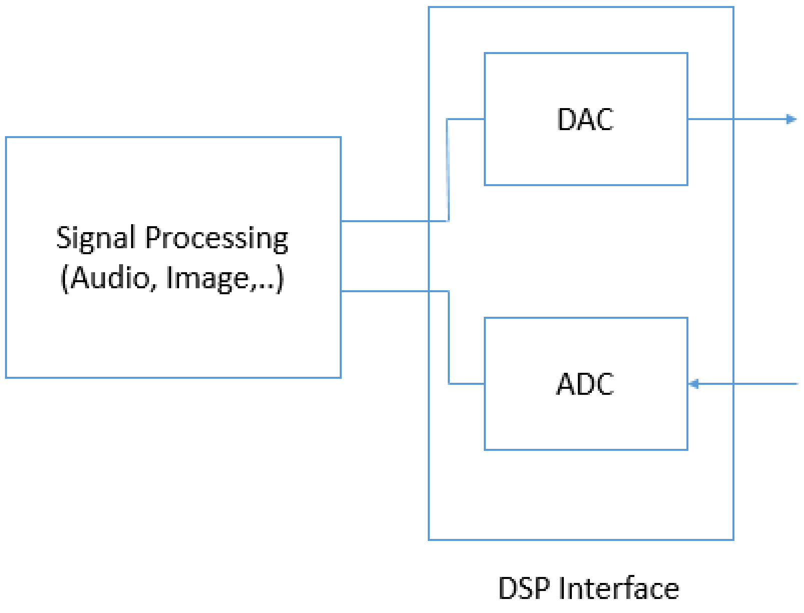


Figure 1: DSP interface

PART I
ADC Converter

An analog to digital converter is a system that converts an analog signal, such as sound picked up by a microphone, into a digital signal. There are several architectures of the ADC converter that can be implemented. Three different implementation method are explained briefly here. However just one of them is going to be used.

- A VCO- based ADC
- Binary search algorithm
- Using an external ADC chip like MPC3002

I- VCO- based ADC

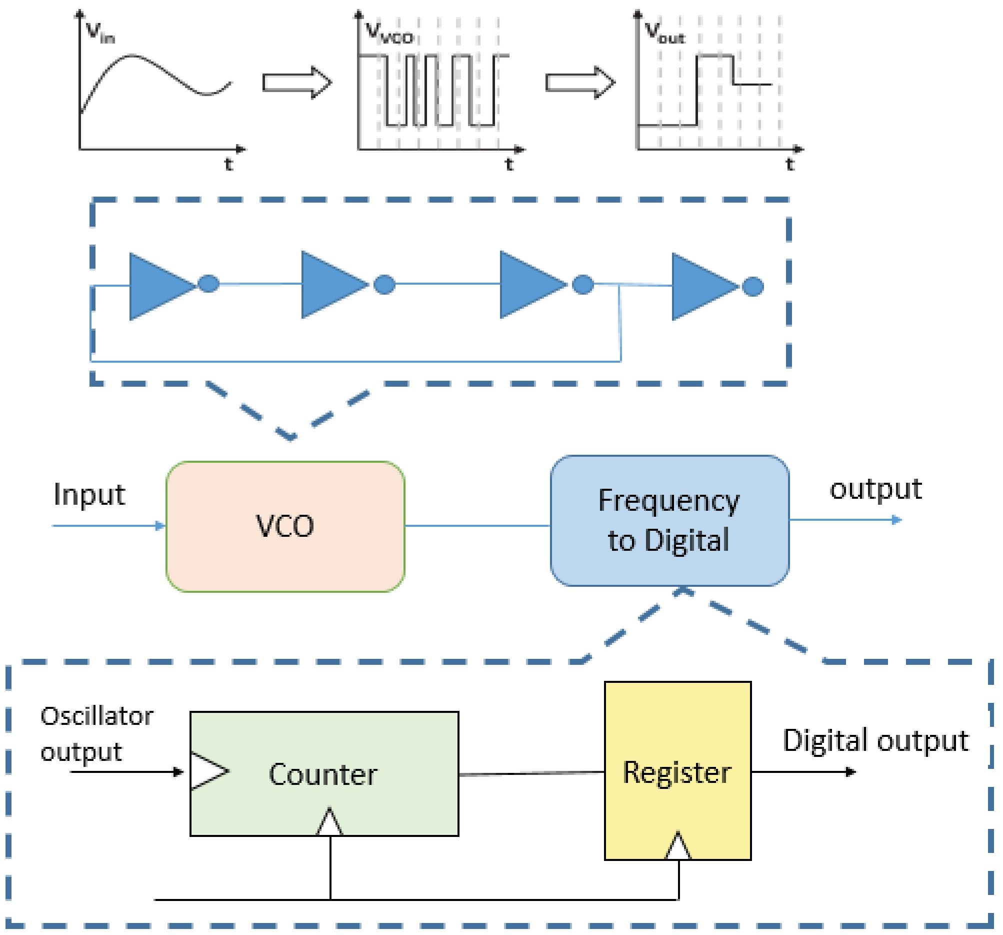


Figure 2: A VCO based ADC

The VCO-based ADC comprises a VCO and a frequency-to-digital converter (FDC) as shown in Fig. 2. The VCO converts the input voltage into frequency information and the frequency-to-digital converter translates this frequency information into a binary code by integrating the VCO output over a time interval Ts. A simple frequency-to digital converter is also shown in Fig. 3, consists of a counter and a register. The counter counts the edges in the VCO output within the given time interval Ts and stores the value in the register.

II- Binary Search Algorithm

By using a digital comparator and comparing the two inputs and producing 1 or 0 for binary search module you will find the input value. Essentially, the strategy of all these converters is to put a value on the input of digital comparator, compare the value with the input voltage and then update the DAC for another guessed value. This is repeated until the sought value is obtained. The differences between the converters essentially lie in the guess-strategy or search algorithm.

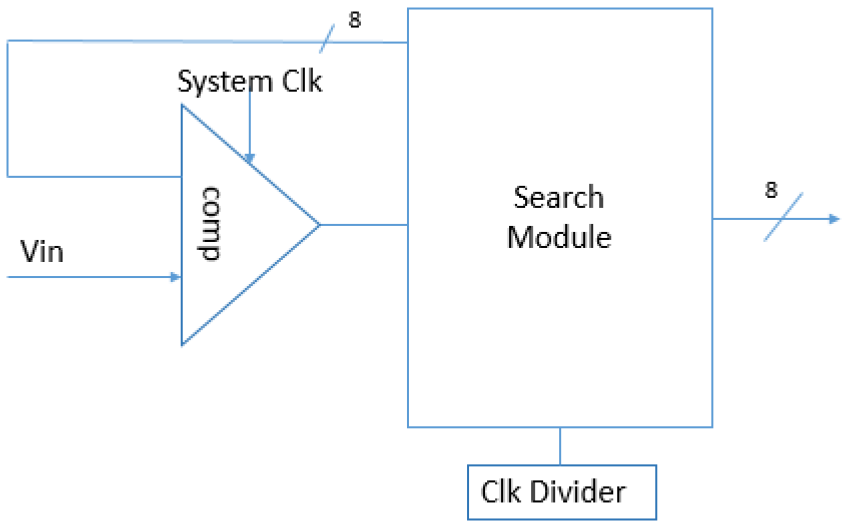
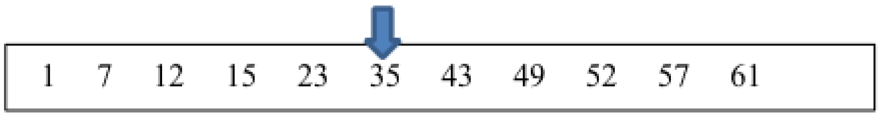


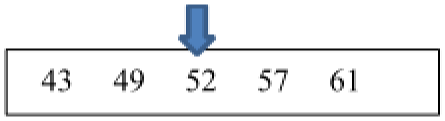
Figure3: Binary search module with a digital converter

Search Module

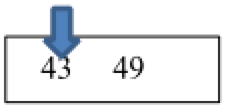
A search algorithm is a method of locating a specific item of information in a larger collection of data. In computer science, there are many methods to find the position of a particular value within an array. The simplest search algorithm that comes to mind is linear search. This method uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for and stops when the desired value is found or the end of the array is reached. Linear search is very simple to implement, and is practical when the array has only a few elements, or when the search is performed in an unordered list. However, this algorithm is slow. It takes time proportional to the number of item in the list to find the desired value. In other words, it is an O(N) "order n" algorithm and is not cost-effective when we have large amount of data. The more efficient algorithm to find an item when we have a sorted array is binary search. A binary search requires fewer comparisons than a linear search. In fact, it uses a “divide and conquer” approach. This algorithm compares the median element in the sorted array to the desired input value in each step. If the input value is equal to the median value, then search is done. Otherwise, based on the comparison, it can eliminate half of the search space. If the input value is less than the middle element's value, then the algorithm repeats its action on the sub-array to the left of the middle element, or if the input value is greater, on the sub-array to the right. By doing this repeatedly, it will eventually be left with a search space consisting of a single element, i.e. the desired input value. For example, consider the following sequence of integers sorted in ascending order and say we are looking for the number 43:



First, we compare 43 with the median value, which is 35 in this example. Since 43 is greater than 35, our search space is reduced to:



Again, we compare 43 with median value (52). This brings the search space reduce to:



Depending on how we choose the median of an even number of elements, we will either find 43 in the next step or chop off 49 to get a search space of only one element. Either way, we will reach to the desired value in the sorted array. The operation of ADC can be modelled as a search problem. In this problem, sensors’ values that DAC is able to produce form the sorted array and input voltage is the element we search for. So, in case of our Digital Plant Words system, first, the total possible range of values is divided into two and the input voltage value is checked for presence in the upper half-space or the lower using the comparator (>128 or <128 for an 8-bit converter). If the comparator output indicates that the result is over the middle value, then the half space is sub-divided into two equal spaces and the value is compared to that. Depending on the result, the upper or lower quarter is divided and so on. The process terminates when the remaining space can not be subdivided, because it is only one bit wide.

The Verilog code for the Binary search algorithm is provided to you in a file named BS_ADC.v. Test the ADC converter by running this code on the FPGA. Verify the design, you can even use the LEDs on the board to show the output. You can use the other methods as an alternative.

Design Verification:
- ✓ **Demonstrate the ADC results to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part**

PART II
A simple audio loop test

In this part, you are going to make an audio in-out loop inside the FPGA. You will play an audio signal (any speech or music file) from your PC and hear it with your earphones. To do this as described before ADC and DAC converters are required. You have constructed a DAC circuit in the previous experiment. Use a 10-bit DAC based on lab3 and also the ADC of part I as the interfaces of your signal processor.

- Instantiate the DAC and ADC modules inside a top-level Verilog design as Figure4
- Inside the processor use a Hexto7segment code to display the value of converted digital values

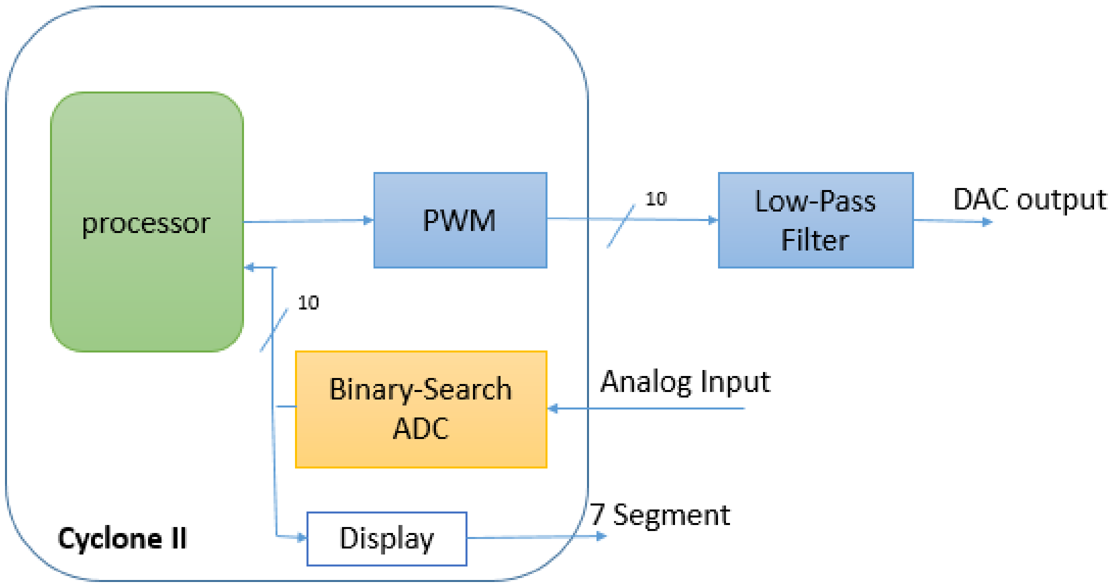


Figure 4: An audio in-out loop

Design Verification:

- ✓ **Demonstrate the results of part II to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part**

PART III
FIFO

In this part, you are going to learn about a memory structure called FIFO that stands for First Input First, a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. You can see the performance of a FIFO in Figure 5. When a ne data item arrives and the FIFO is not full, it is ritten to the FIFO. As a stored data item is retrieved, it is removed from the FIFO. If the buffer is full, it should not receive any more data (otherwise, existing store data would be corrupted). A “Full” status signal is asseted to tell the sender not send any more data. Similary, if the buffer is empty, it can not provide any data for retrieval. An “empty” status signal is used to indicate that the FIFO has no more data to provide.

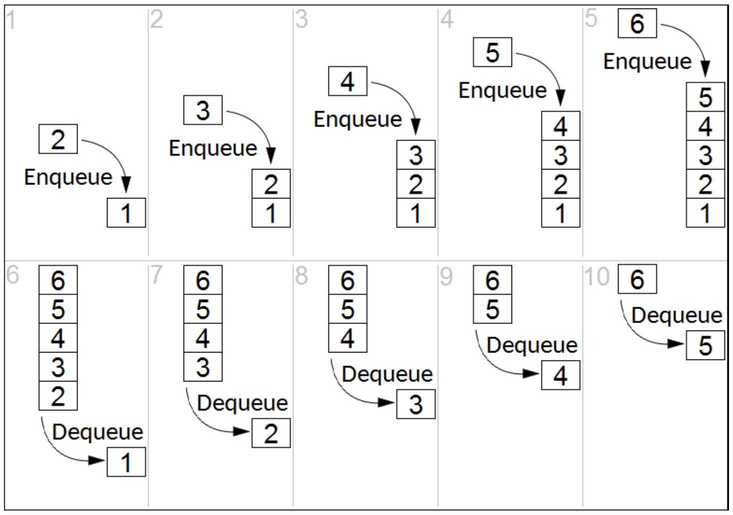


Figure 5: Representation of a FIFO

Figure 6 shows the RTL view of the FIFO. This memory structure has read and write pointers that keep track of data in FIFO. When the pointer gets the maximum ram depth, the signal “Ful” gets “1” and when it reaches “0” the “empty” signal gets “1”. According to this block diagram, the flow can be designed with three always blocks as Figure 7. The WRITE POINTER always block is for incrementing the pointer for writing data using WR control signal. A similar procedure is true for the READ POINTER and READ DATA. The las

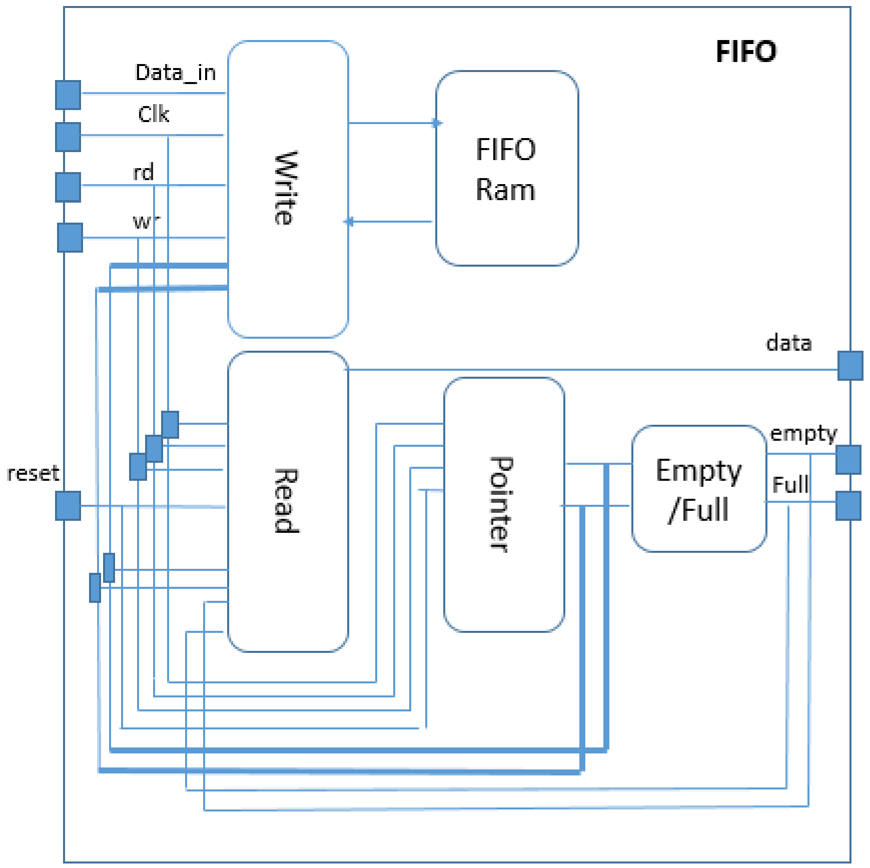


Figure 6: FIFO RTL view

```
always @ (posedge clk or posedge rst)
begin : WRITE_POINTER
  if (rst) begin
    wr_pointer <= ...;
  end else if (//a control signal// )
  begin
    wr_pointer <= ...;
  end
end
```

Figure 7: a) write pointer always block

```
always @ (posedge clk or posedge rst)
begin : READ_POINTER
  if (rst) begin
    rd_pointer <= ...;
  end else if (//a control signal// )
  begin
    rd_pointer <= rd_pointer + 1;
  end
end
```

b) read pointer always block

```
always  @ (posedge clk or posedge rst)
begin : READ_DATA
  if (rst) begin
    data_out <= ...;
  end else if (... ) begin
    data_out <= ...;
  end
end
```

c) read pointer always block

```
always @ (posedge clk or posedge rst)
begin : STATUS_COUNTER
  if (rst) begin
    status_cnt <= 0;
  // Read but no write.
  end else if (...) begin
    status_cnt <= ...;
  // Write but no read.
  end else if (...) begin
    status_cnt <= ...;
  end
end
```

d) Status counter always block

The last block is used to control the Full and empty signals. When the FIFO is reading and the reading has not finished yet the counter must be decreased until it reaches zero. The same is true for writing and incrementing the counter hen it has not reached the maximum depth of RAM.

Complete the always block based on the control signals and include them in a Verilog code for the FIFO. For the ease of you, the RAM block code named RAM.v has been provided. Instantiate the RAM block and put all other required parts inside your code and make a project of it. Test your design with a testbench and observe the results in Modelsim.

Design Verification:

- ✓ **Demonstrate the result of part III to your TA .**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

PART IV
Producing echo and multiple echo

In this part of the experiment, you will design and test a circuit that simulates the effect of echo. The diagram belo shos the components of a sound source reaching its listener: the direct path signal x(t) and the echo signal βx(t-T) which is weaker version of x(t) attenuated by factor β, bounced off the floor. The echo signal is also delayed by T relative to the direct-path signal x(t).

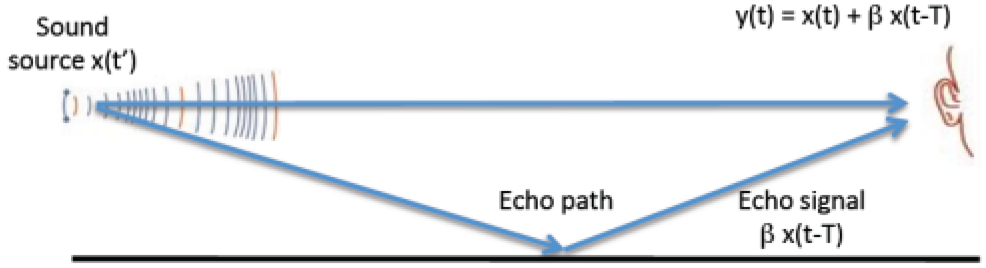


Figure 8: the concept of an audio echo

A) Simple Echo
Based on part III and the definition of FIFO, it can simply apply such delay to an audio signal. This is illustrated in Figure 9.

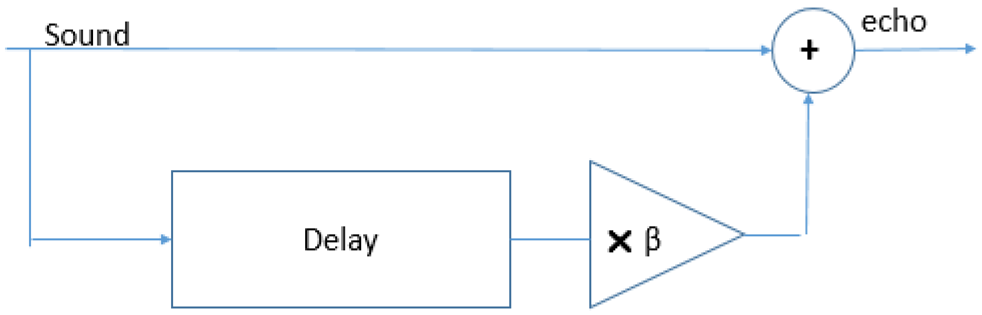


Figure 9: The block diagram of a simple echo generation

Use the FIFO of part III as a delay and a simple binary shift for the attenuation part. Design this echo generation system.

B) Multiple Echo

In order to produce a multiple echo on an audio signal the instead of the input, the output must be delayed and feedbacked. Note that in this case the output is subtracted from the input.Figure 10 shows the block diagram of a multiple echo processor. A pulse generation block sends request for writing the sampled data in FIFO. The FIFO writes the input data and once it is full it starts reading from the FIFO. So one Write_req port will be enough for both operation. Based on this concept, guess the block with question mark. Use the FIFO of part II and design the multiple echo system of Figure 10.

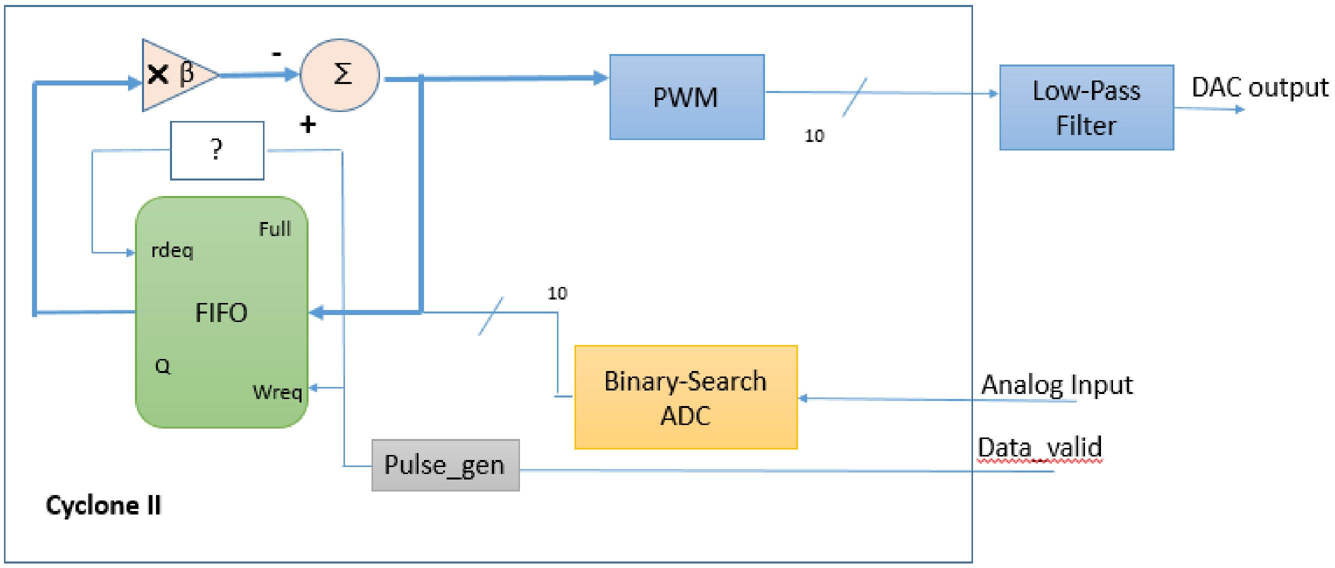


Figure 10: Block diagram of a multiple echo processor

Design Verification:

- ✓ **Demonstrate the result of part IV to your TA .**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

PRELAB ASSIGNMENT

Before coming to the lab, answer these questions. The pre-lab needs to be handed in at the start of the lab.

1- Complete the always blocks; write_pointer, Read_pointer, Read_data and status_counter.

2- Write a testbench for the FIFO. (Try to make a preliminary design at least).