UNIVERSITY OF TEHRAN

School of Electrical and Computer Engineering

Digital Logic Laboratory, ECE 045, Fall 1392

Experiment 3 (Sessions 6, 7, 8)

Binary-Search ADC and Digital Voltmeter

Purpose

In this lab, you will learn about:

- Implementation of ADC using DAC and analog comparator
- Hardware implementation of binary search algorithm
- The conversion of numbers from binary to BCD using a combinational circuit
- Driving common cathode 7-segment displays

Background Information

In this experiment, you are to design a digital voltmeter. Digital voltmeters are a special kind of analog to digital converters (ADCs). They show the measured analog voltage in floating point format using a display like LCD or seven-segment. The most common type of ADC, shown in Figure 1, runs by using a DAC and an analog comparator. An analog comparator simply compares two input voltages (V_+ and V_-) and produces a logic 1 if $V_+ > V_-$ and a logic 0 otherwise. Essentially, the strategy of all these converters is to put a value on the DAC, 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.

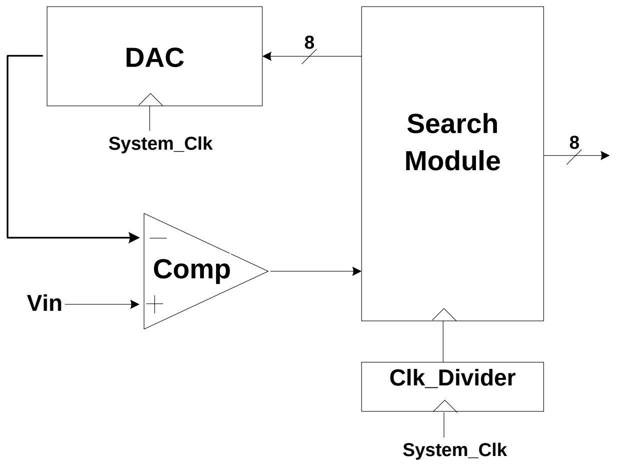


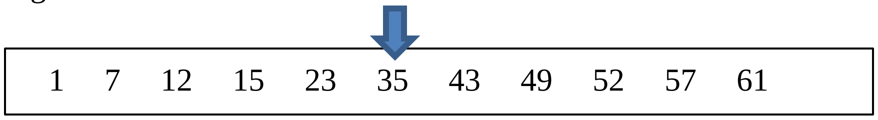
Figure 1 An ADC block diagram using a DAC and a comparator

In your previous experiment, you learnt how to implement a DAC using a PWM module and a low pass RC filter. In the following, the other two parts required to implement the ADC unit are explained in details.

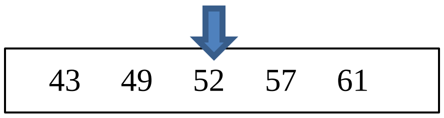
Search Module

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 items 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.

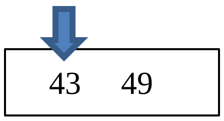
For example, consider the following sequence of integers sorted in ascending order and say we are looking for the number 43:



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 modeled as a search problem. In this problem, voltages 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 voltmeter, 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 cannot be subdivided, because it is only one bit wide.

Analog Comparator

As explained before, we need to compare the input voltage with our generated value in each step. To do so, we use a LM339 Quad Comparator IC. This device consists of four independent op-amp† voltage comparators. In theory, a standard op-amp operating in open-loop configuration can be used as a comparator. When the input V_+ is at a higher voltage than the input V_-, the high gain of the op-amp causes the output to saturate at the highest voltage it can output. When the input V_+ drops below the input V_-, the output saturates at the lowest voltage it can output. The pin connections of this IC are shown in Figure 2. In order to work with this IC, you need to put a 10K pull-up resistor in the output pin of comparator.

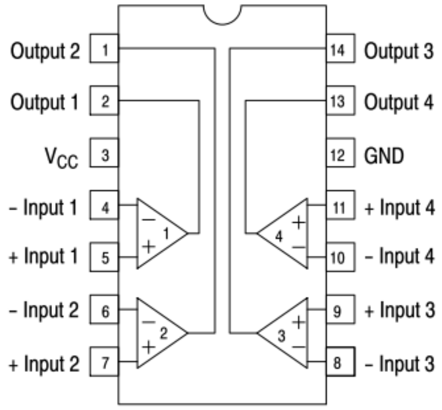


Figure 2. LM339 pin connections

So far, we could obtain the input voltage as an 8-bit binary number in a range between 0 and 255, which is equivalent to the input voltage in range between 0V and 5 V. There are two other steps to show this value on seven-segment display. First, since the input voltage is a value between 0.00V to 5.00V, we should map our measured value to this range. To put it simply, we need to write a module to convert the obtained binary number to another binary number in the range between 0 and 500. (Considering the decimal point comes after most significant digit)

Another important part of this lab is binary to BCD conversion. We want to display the final output on 7-segment displays, but since the output of the voltmeter is in binary, we have to convert it before we can display it. It is possible to build this circuit using an algorithm called "shift-add-3". Here is the concept:

Suppose we have a four-bit binary number that we want to convert to BCD. If it is less than 10, we don't need to change anything. However, if it is larger than 10, we need to do a conversion. We can subtract 10 from the original number to get the ones digit. Then, putting a 1 in the tens digit in BCD can be thought of as adding $0001\ 0000_2=16_{10}$ to the original number. For example,

Start with $12=1100_2$	$0000\ 1100_2$
Subtract $10=1010_2$	$0000\ 0010_2$
Add $16=10000_2$	$0001\ 0010_{BCD}$

† Technically, there is a minor difference between op-amps and analog comparators. For example, some analog comparators cannot source current and that is why we need a 10K pull-up resistor.

By combining these operations, we can reduce the number of operations. Therefore, to convert a four-digit number from binary to BCD, we can simply add 6 or 0110_2 if the number is greater than or equal to 10. This works well for four digit numbers, but what if we have more than that? In this case, we can shift the input, one bit at a time, through sets of four bits, and at each step add 0110_2 to a digit if the number is 10 or greater. Here is $22=10110_2$ as an example:

Table 1 Illustration of converting a 5-bit binary number into a BCD number

Operation	Tens	Ones	Input
Binary Input			10110
Shift Left (and check if ≥10?)		1	0110
Shift Left (and check):		10	110
Shift Left (and check):		101	10
Shift Left ("Ones" is >10:add 6)		**1011**	0
After adding $6=0110_2$	**1**	**0001**	0
Shift Left	10	0010	
BCD Output	2	2	

$$\boxed{\begin{array}{r}1011\\+0110\\\hline 10001\end{array}}$$

After each shift, one needs to check if the number in the "Ones" column is equal or greater than 1010. If it is, one add 6 to it before shifting again. Notice that when we added 0110_2, the carry bit was carried into the next digit. This means we would need five output bits at each step rather than four. Fortunately, we can use a shortcut to avoid this. What does shifting a binary number one step to the left do to the number mathematically? It has the effect of doubling a number. If we switch the order of the operations, we only need four bits of output. Specifically, we can add 3 and double instead of doubling and adding 6, because both give the same result in the end. This is where the algorithm gets its name. The new rule then is to shift the input, one bit at a time, through sets of four bits, and if a digit is five or greater, then we add 3 before shifting again. Here is the same example again, using the "shift-add-3" algorithm:

Table 2 Conversion of a 5-bit number into a BCD number using the shift-add-3 algorithm

Operation	Tens	Ones	Input
Binary Input			10110
Shift Left (and check if ≥ 5?)		1	0110
Shift Left		10	110
Shift Left (Ones >5: add 3)		**101**	10
Add $3=0011_2$		**1000**	10
Shift Left	1	0001	0
Shift Left	10	0010	
BCD Output	2	2	

$$\boxed{\begin{array}{r}101\\+\ \ 11\\\hline 1000\end{array}}$$

The same procedure can be applied for any number of bits. You will keep shifting (and conditionally adding 3) until all input bits have been shifted (e.g. for an 8-bit you will need to shift 8 times).

Now, the question is how to implement this in your circuit. First of all, we need a module which has four-bit input and output and which performs the operation "add 3 if the input is 5 or greater". This is up to you to design however you want. You can use schematics, K-maps, Verilog, or anything else. Then, the converter can be arranged in the following manner:

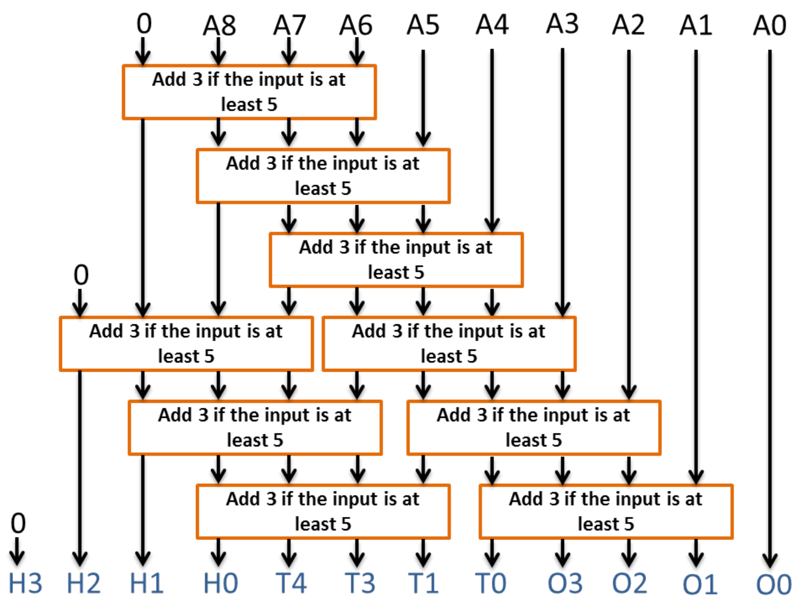


Figure 3. Binary to BCD Converter

Now that we have the individual digits, we can easily convert them into 7-segment display signals. (Don't forget to turn the left-most 7-segment's dot on) However, the 7-segment displays all share the same anodes. Therefore, we will need to come up with a way to show all of the digits. The solution, as you may imagine, is to switch between the digits, displaying one at a time. However, if you do this at just the right speed (just like in the movies), the switching is imperceptible, and the display looks correct. We will need a switcher, a component that takes a number of inputs and passes just one to the output based on a separate 'Sel' signal.

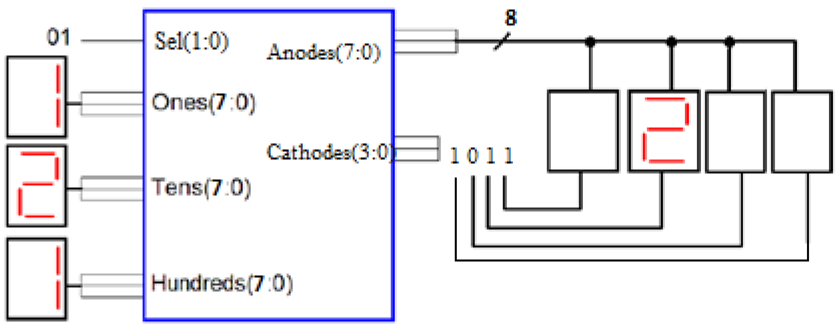


Figure 4. Block diagram of the switching circuit for the 7-segment displays.

Based on the 'Sel' input, it should choose the ones, the tens, or the hundreds digit input to pass to the anodes. The switch circuit also has to alternate between the cathodes of the 7-segment displays in order to make sure only the correct display is on at any time. Unfortunately, there is one more small issue: The 50MHz system clock on the board is too fast for this operation. We need to divide this signal down to be able to use it.

The overall schematic of the digital voltmeter is shown in Figure 5.

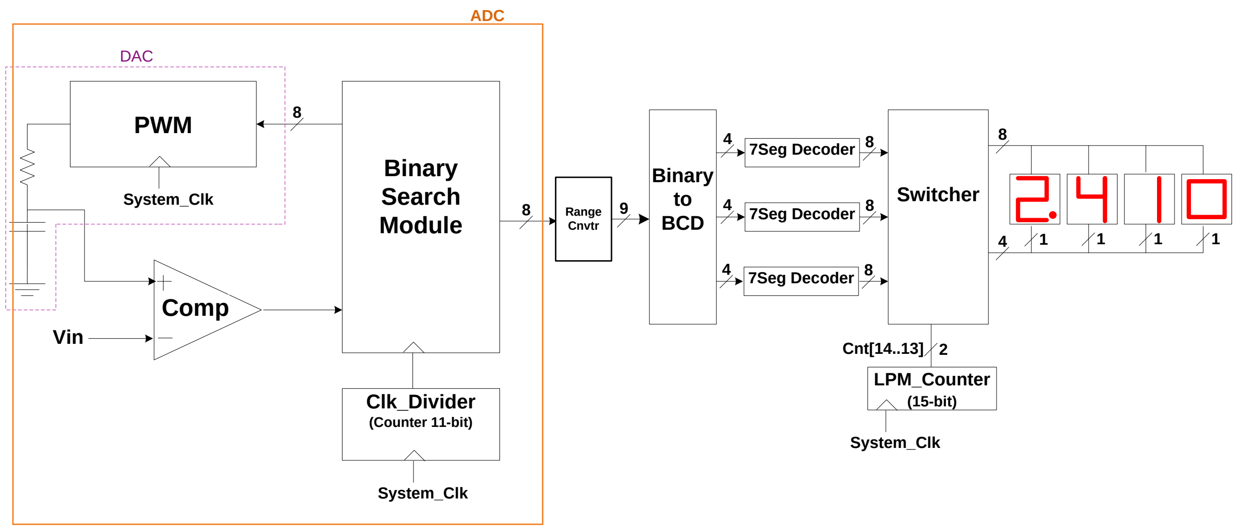


Figure 5. Overall block diagram of digital voltmeter

Pre-Lab assignment

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

1. Suppose that you want to search for 173 using the explained binary search algorithm. How many comparisons you need to make to find this value in the list ranging from 0 to 255?
2. Draw the corresponding Register Transfer Level schematic for linear search module and binary search module. How many clock cycles does it take for each of them to find a desired value in the worst-case scenario? (worst-case scenario is a situation in which the desired value is found passing the longest possible path)
3. In Figure 3, write down the values of the input and output of each block on the schematic for a conversion of 011011110_2. Verify the output by converting 011011110_2 to BCD by hand, and make sure the outputs correspond.

Lab Work

Read the instructions carefully while doing the lab. Your report must include the procedure you followed, as well as any observation and results.

1. Implement the binary search module using Verilog. Perform a simulation to verify its correctness. Do not include the clock divider in this simulation. (Figure 5)
2. Implement DAC circuit. You can use your PWM from previous experiment.
3. Use the LM339 IC to complete the ADC circuit. Download your design on the FPGA board. Verify your ADC by showing its outputs on LEDs.
4. Implement a module to map the output of ADC to a range between 0 and 500.
5. Create the "shift-add-3" module. It should have a four-bit input and a four-bit output. If the input is four or smaller, it should pass through unchanged. If the input is five or larger, the output should be the input plus 3. You can implement this module as a schematic or Verilog code. When you are done, do a simulation and test various values.
6. With the previous module, create the binary to BCD converter, following the schematic in Figure 3. Run a simulation of the converter to make sure it works.

7. Create the 7-segment decoder module. This module gets a BCD number and determines which segments should turn on. Note that cathodes of the 7-segments of the FPGA board are active low and anodes are active high.
8. Build the switching circuit (see Figure 4). It should have as inputs the 7-segment display anode signals for the ones, tens, and hundreds digits, as well as a two-bit ‘Sel’ signal. It should also have two outputs: one 8-bit signal for the anodes and one 4-bit signal for the cathodes. As the ‘Sel’ changes, the switch should alternate which signal is passed to the anodes. At the same time, it should turn on the correct cathode and turn off the rest. Write your switching circuit in a way that the rightmost 7-segment always shows zero. Do a simulation when you are done.
9. Create the top level schematic (Figure 5).
10. Download the circuit to the board and test the digital voltmeter; use the 7-segment displays to show the output.
11. Give a demo to the lab instructor.