UCD School of Electrical and Electronic Engineering

EEEN40280
Digital and Embedded Systems

Signal Measurements Report

Name: Jingyi Hu **Student Number: 17200558**

Working with: Anton Shmatov

I certify that ALL of the following are true:

1. I have read the *UCD Plagiarism Policy* and the *College of Engineering and Architecture Plagiarism Protocol*. (These documents are available on Blackboard, under Assignment Submission.)
2. I understand fully the definition of plagiarism and the consequences of plagiarism as discussed in the documents above.
3. I recognise that any work that has been plagiarised (in whole or in part) may be subject to penalties, as outlined in the documents above.
4. I have not previously submitted this work, or any version of it, for assessment in any other module in this University, or any other institution.
5. I have not plagiarised any part of this report. The work described was done by the team, but this report is all my own original work, except where otherwise acknowledged in the report.

Signed: Hu Jingyi **Date:** 12 March 2019

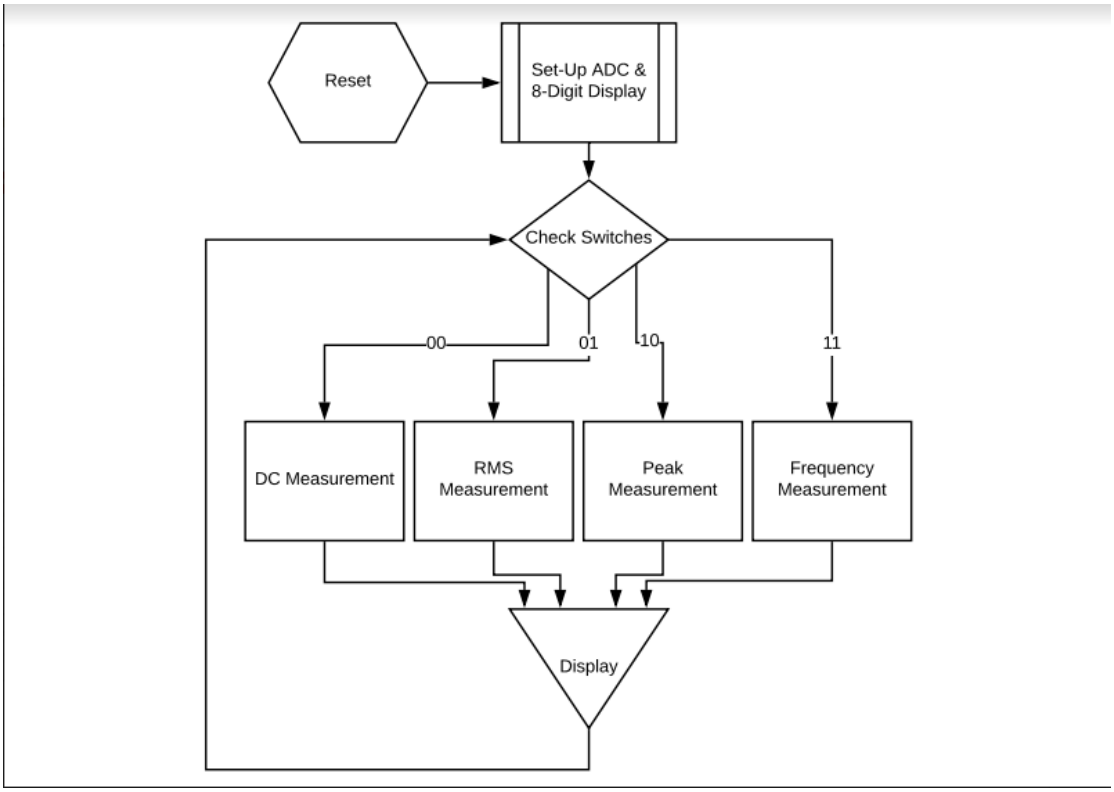
Details of what is expected in the report are included in the assignment instructions.

Your report and program must be uploaded by 23:00 on Monday 11 March.

You should save your report as a pdf document, and upload it through the submission channel in Brightspace. One member of the team should upload the program through a separate channel in Brightspace. This should be the original source file in C or assembly language, as used in your uVision project. It should be well commented and fit to be compiled and run. If there is more than one source file, combine all the source files into one zip file and upload the zip file.

In this lab, we designed a measuring instrument with mode selection switches together, which can be implemented to measure DC Voltage, RMS Voltage, Peak Voltage, and periodic signal frequency and demonstrate the decimal result on the 8-digit LED display in unit either V or Hz.

We firstly started with the 8-digit display, frequency measurement, and the DC measurement. Then we expanded the code for RMS voltage measurement and Peak voltage measurement. And in the last session of the lab, we rebuilt a frequency measurement function as well for more accurate frequency. The workflow diagram as below:



Before switching the mode for measuring, definition work and initialize are needed.

a) For the 8-digit display, we need to define and initialize the display in the beginning.

   We define the P3.7 from which the Load signal comes and the addresses for all the registers we have used (please see: display_8d.h). Also in order to initialize the display (display_8d.c), we set the SPI control register – SPICON to 0011 0011; putted 7 in Scan Limit Register so that all digits will be displayed in decode mode since the Decode Register contains 0xFF; besides, Shut Down Register is set to 1, Display Test register is set to 0, Intensity Register is set to 8 for normal operation.

   To be noticed that the LOAD signal should go high for writing the data into the registers and backs to low once data transmit has finished, but during the transfer period, in this case, 2 bytes wide data will be transmitted where the most significant 8 bits which contain the address of a register will go first, and

then the left 8 bits data will be written to that register. Moreover, a flag called ISPI needs to indicate when a byte has been transmitted. Therefore, by considering the exits Hardware flaw, we built a function 'delay_spi()' for a short delay to wait the ISPI flag turns to 1 which needs at least 8 processor clock cycles.

b) Then we set up the SFRs for ADC for relevant voltage measurements. All interrupt sources (EA) and ADC interrupts (EADC) are enabled in this experiment. And ADCCON1 has been set into 1011 1010

Switches

Once the 8-display and ADC have been set up, the users can use the switches to select the measuring mode any time only if the power has been applied. Because this measuring instrument we made only allows 4 kinds of measurement, 2 bits of Port 2, P2.0 and P2.1, thus have been used. To avoid the error causing by unknown switch value appeared, switch mask 0000 0011 are used to set the most significant 6 bits always be 0. At the same time, switches are active high so that the most useful bits (bit 0 and 1) must be set as 1 in the beginning.

MEASUREMENT EXPLANATIONS

Frequency:

a) Mode 4: **FREQUENCY Measurement** ('frequency.h' & ' frequency.c')
When the users put the switches into xxxx xx11 position, then the hardware will use timer 0 & 1 for measuring the periodic sinusoid frequency. In this case, T0 is running as a 16-bit timer with system clock frequency and T1 is running as a 16-bit counter to count the rising edges of the input signal.

Initialising the timer 0 & 1, TMOD = 0101 0001; EA, ET0, ET1 (in IE SFR) are all enabled before timer 0 & 1 start running while the interrupts from ADC, Timer2, UART, and external are forbidden; the data registers of timer 0 & 1 both contain 0; and the measurement ready signal, which indicates the data is ready for measuring, goes to low; low frequency measurement also set up to 0 since we set high frequency measurement as a default. Once the set-up part finished, 0101 0000 will be written into TCON to let timer 0 & 1 start timing and counting. Then, the system will wait for the next instruction. Once the measurement ready flag turns to high, the result of frequency measurement will be returned and assigned to display functional block 'write_8dsiplay()' for digits displaying.

In this task, we can successfully measure the maximum frequency 1,100,000Hz generated by the signal generator with reasonable error and our lower limit frequency is 9Hz. But theoretically, our upper range is now 11MHz.

Interrupt 1 will occur first as soon as Timer 0 overflow in this experiment since the maximum signal will be measured later on is smaller than the system clock frequency.

Once this interrupt happened, system will determine the measurement is either for high frequency or low frequency. If in **high frequency measurement mode**, both timers will stop running, and then the value in Timer 1 which counts the positive-edge numbers of the input signal. If this number is bigger than 128, a number we picked that how many edges need to count before measuring frequency ideally, then the frequency will be calculated directly; otherwise low frequency measurement will start where the relevant signal equals to 1.

$$\frac{\mathrm{T0_{total}}}{11059200} = \frac{\mathrm{T1}}{\mathrm{f_{input}}}$$

$$\downarrow$$

$$\mathrm{f_{input}} = \mathrm{T1} * \frac{11059200}{2^{16}}$$

However, if it's a **low frequency measurement**, our Timer 0 will at most run for 4 seconds even if the counts number of timer 1 does not reach the minimum limit 128. If there are 128 rising edges of the input signal existing within 4 seconds, interrupt 3 will occurs immediately when the 128th rising edge been detected for the frequency calculation. This algorithm first will calculate the total value that Timer 0 has counted so that the length of measuring can be got by:

$$\mathrm{f_{input}} = \frac{11059200 * 128}{\mathrm{T0} + 65536 * \mathrm{times_{overflow}}} \quad where\ t_{measuring} \leq 4$$

If not, the 'timeout_measurement()' function will be called. Timer 0 & 1 will be disabled first, and then the counting value of timer 1 will be used to calculate how many edges of input signal have been counted, divided by 4 (length of measuring time) to get the value of a frequency of input signal as shown in below:

$$\mathrm{f_{input}} = \frac{128 - 2^{16} + \mathrm{T1}}{4} \quad where\ t_{measuring} \geq 4$$

For either of the frequency measurements above, Once the procedure is done where the frequency value of input signal has been calculated, 'measurement_ready()' signal will to 1, and then the result will be returned and assigned to function 'write_8display()' for displaying.

ADC:

Our ADC is driven by the master clock which is 11.0592MHz. At the initializing, bits in ADCCON1 = 1011 1010, hence ADC divide ratio is 11059200/4 and the total ADC conversion time is 19 ADC clocks:

$$15\ \mathrm{ADC} + 1\ \mathrm{ADC} + 3\ \mathrm{ADC} \quad (\text{selected acquisition time})$$

The reason why these values been chosen is:

1: drifting voltage occurs when we let 2 as clock divisor.

2: the speed of ISR is also a constraint since the computational expenses.

The number of samples for each period we go for 4, thus, the upper limit on the sampling rate of ADC is:

$$f_{\text{sampling upper}} = \frac{11059200}{4 * 19 * 4} = 36379 \text{ Hz}$$

The lower limit should be the timer maximum timeout:

$$f_{\text{sampling lower}} = \frac{169}{4} = 43 \text{ Hz}$$

Timer 2 overflow bit is enabled as what can be seen in ADCCON1. We also set up the Timer 2 for interrupting at a particular frequency. When a frequency value is assigned to Timer 2, if this frequency is outside the range of ADC sampling rate, the Timer 2 will running at the limitation frequency with which the input frequency closest multiply with four, otherwise, the frequency of timer 2 will be 4 times of input signal frequency. Once we got the result that conversion rate, reload value then the reload value can be calculated.

Now, the timer 2 starts running in timer mode. Then the interrupt program will be called after the CPU receives the interrupt request after the ADC conversion is completed, an interrupt request is automatically sent to the CPU

This ADC interrupt is mainly for getting the current number inside the2 SRFs and adding it with the previous sum of samples, and if it's in RMS mode, the sum squared values will also be calculated. In order to know how many samples have been captured so far, a sum pointed is set to increase at the end of the ISR. At the end of this interrupt process, there is a determinate step for checking if enough samples taken. If yes, the sum ready signal will go to high for indicating that the data is transmitted, ADC interrupt then will be disabled and Timer 2 also will stop running.

b) Mode 0: **DC Measurement** (function 'measure_dc()' under adc.c)
If the switches are one the xxxx xx00 position, the system will operate as a DC measuring instrument.

The system will set up the ADC, starting the timer 2 and going to interrupt services routine. In the end, we will get a result after 256 times sampling. Therefore, we average the value by divided 256 and multiply the result with the ADC unit value that 1 LSB represents to get the decimal digital voltage. By calling the function 'write_8display(), the result of measurement will be displayed on the screen.

c) Mode 1: **RMS Measurement** (function 'measure_rms()' under adc.c)
RMS measurement will be targeted when the switches at xxxx xx01 position. Firstly, the frequency of input signal will be measured; Secondly, apply this value to relevant timer 2 function described previously. In our experiment, when we have driven respective equations for RMS voltage, we removed this DC offset.
So rewrite the next equation:

$$x_i = s_i + \text{offset}$$

Into

$$x_i = s_i$$

As a result

$$\text{RMS} = \sqrt{\frac{\sum_{i=1}^{N} s_i^2}{N}}$$

However, the offset could be evaluated by using our board by connecting Vpot to ADC 0 and apply it to the block function for DC measurement.

The algorithm for RMS as same as what has been mentioned in the lecture, as an updated way of Bisection methods. Start from the middle point value 8000H as the root, get the pseudo square and compare with actually squared value of RMS result.
For example,

$$1000\ 0000\ 0000\ 0000^2 > value$$

If yes, the next square root will be 0100 0000 0000 0000; but if it is smaller, square root will be 1100 0000 0000 0000 until the correct result has been found before the loop end or all 16 possible bit positions have been iterated. However, last loop round will not be examined, precision determination therefore is needed to check whether the last bit makes it closer or not (precision determination)

d) Mode 2: **PEAK Measurement** (function 'measure_peak()' under adc.c)
Switches at xxxx xx10 represent the peak voltage of the input signal will be measured. Most of the steps of this mode are basically the same with the RMS voltage measurement mentioned above, except the last step where we scale the measured values by multiple with $\sqrt{2}$ to calculate the results.

$$V_{peak} = V_{rms} * \sqrt{2}$$

DISPLAY OPERATION EXPLANATIONS (results in unit either V or Hz)
All the final results we obtained by the above measurements as well as the decimal digit value will be sent to the function under display_8d.h for displaying.

Display the data in decimal results on the screen, we used the function 'write_8display()'. First, we divided the value to be displayed by the maximum value 99999999 (0x5F5E0FF) which can be displayed in the screen to truncate the value to the number of digits currently being displayed and add the decimal point.

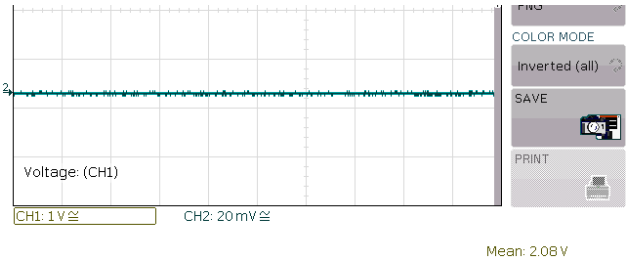
Next, we assigned the value to the Data Register 1 to 8 each corresponds to the digit from right to left by function 'write_dispi()'. And the modulo-10 algorithm has been applied for getting the value of each digit need to be displayed. If the current Data register address smaller than the number of decimal needs to display, the decimal point will stay off and only the

corresponding pattern of this digit will be displayed. Otherwise, if the current Data register address equal to the length of the decimal value, not only the related pattern will be displayed, but also the decimal point will be turned on. However, if the address of the current register does not meet any of the condition above, then we need to see whether there are exits a digit need to display by determine the corresponding is either 1 or 0. If it's 1, the relative pattern will be shown on the screen; otherwise, we left it as a blank. Once the hardware finished the current register operation, it will move to the next data register for another pattern displaying same as before.

Testing

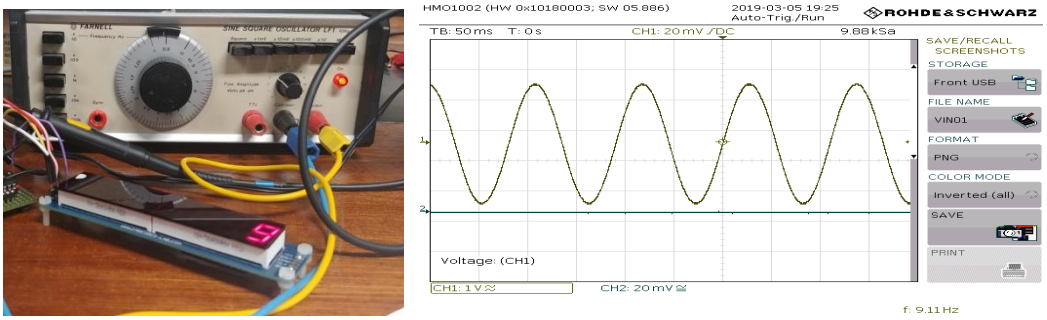
We tested our code every time the function is built, and in the last session, we optimised our frequency measurement methods since frequency fewer than 100 kHz will be not measured precisely.

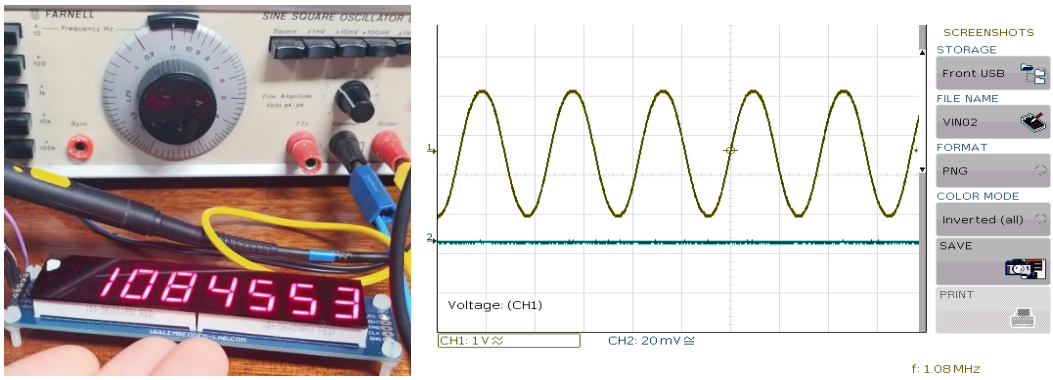
We first tested our DC measurement. In the 2nd session of the lab, the value displayed was always a one digit constant even the reset had been pushed, we used the signal analyser to check which port got wrong and that took several hours to found out the reason which is we assigned a wrong address to the scan limit register. The result got at the last session with switches is below:



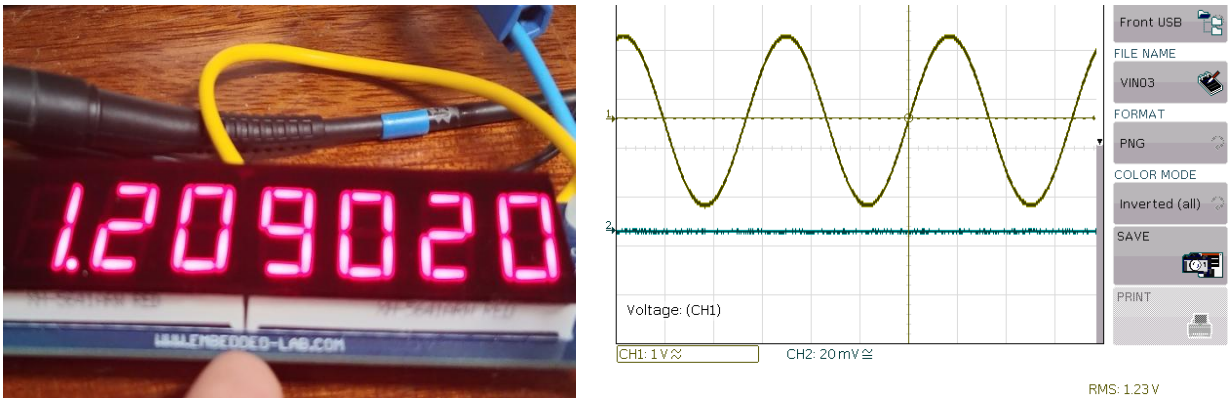
The actual result can be seen is 2.08V and the result of our system is approximate 2.12V, the error is acceptable, 0.04V.

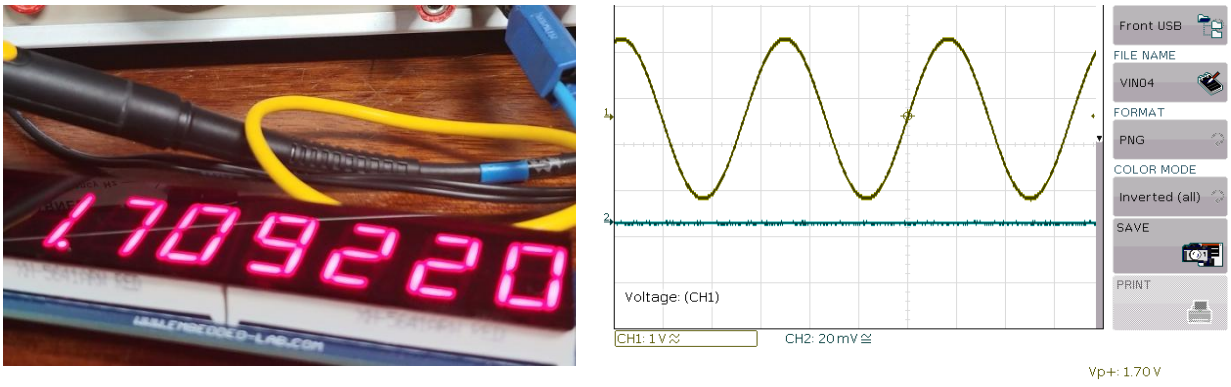
Frequency measurement, initially this function for high frequency measurement was returning very precise results, but for the low frequency what we got was obviously inaccurate. Then we optimised our algorithm for measuring low frequency by extending the sampling time to 4 seconds where timer 0 is allowed to continuously overflow. In the end, the lowest frequency can be measured accurately is 9 Hz, and theoretically there is no upper limit for the frequency any more as mention before. The results of frequency measurement are below:





RMS and Peak voltages testing were also been implemented. In the beginning, we used a wrong pin as V_{in} so that what we observed on the signal analyser was the negative voltage kept going low and we thought it was because the Schmitt trigger was broken which was not. The results are shown below:





As above, the error of RMS measurement is around 0.02V and the one of Peak measurement basically is 0.

Based on the above tests, not only the measurement algorithms are operating properly, but also the performances of switches and the LED display have indicated the successful employment of the relevant function blocks.

Nevertheless this instrument has been implemented; there are still some function needs to be optimized. Like mentioned above, the DC offset might be able to be measured by using the DC voltage measurement. And for the frequency measurement also might be achieved by solely using timer 2.