**Lab 3 Report**

**Objectives:**

Requirements Document-

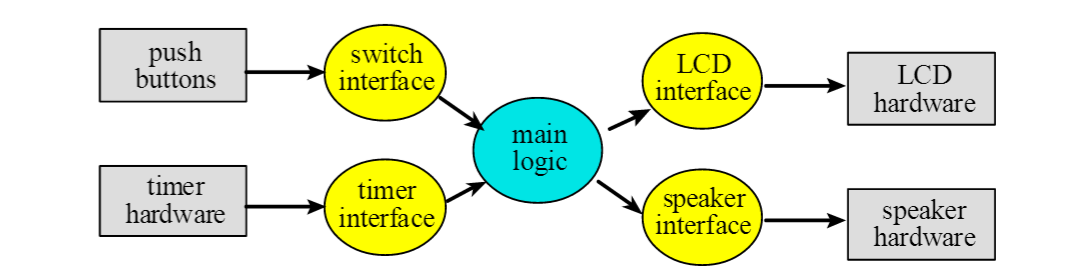
**Hardware Design:**

On Board-

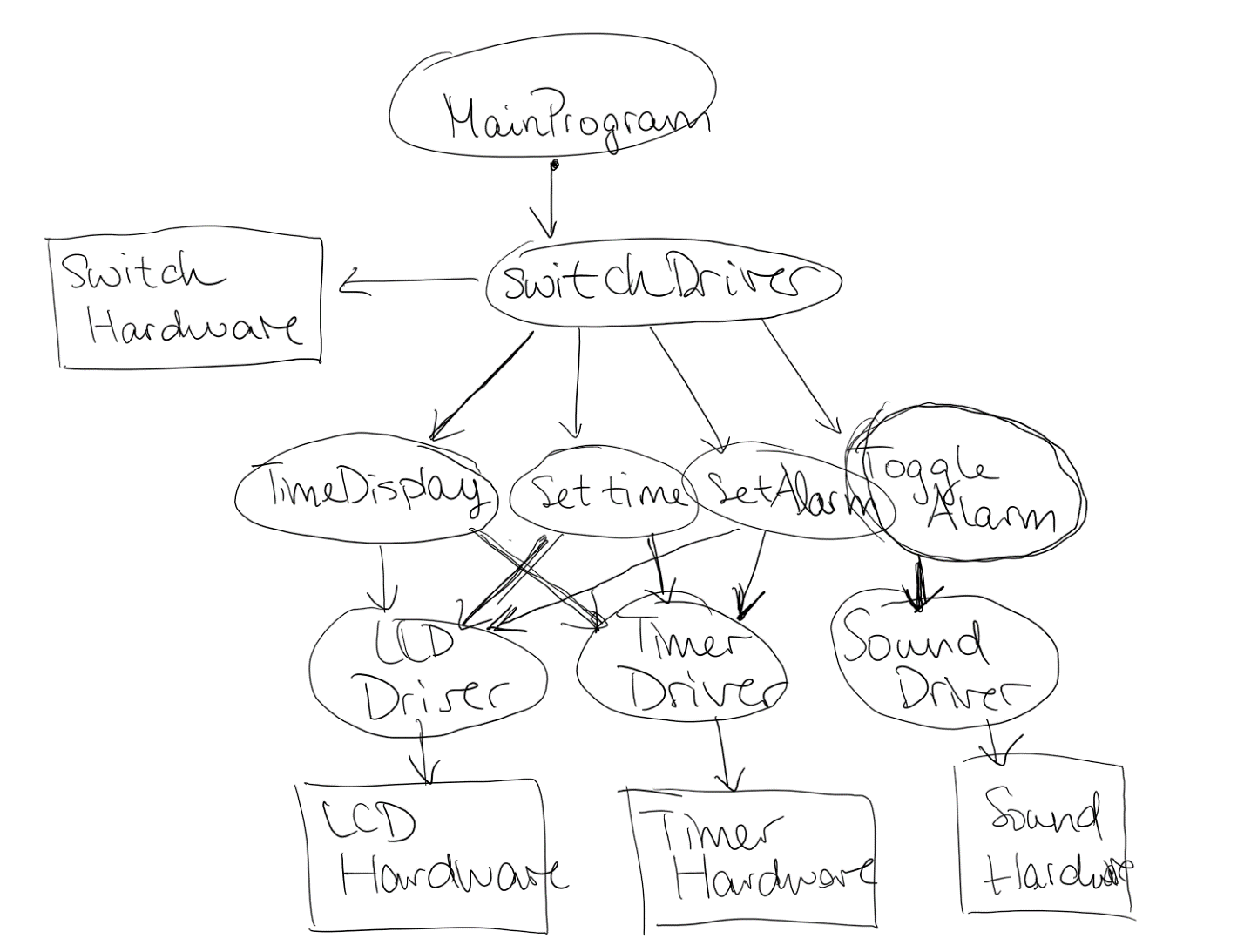
**Software Design:**

Code-at end:

Data Flow

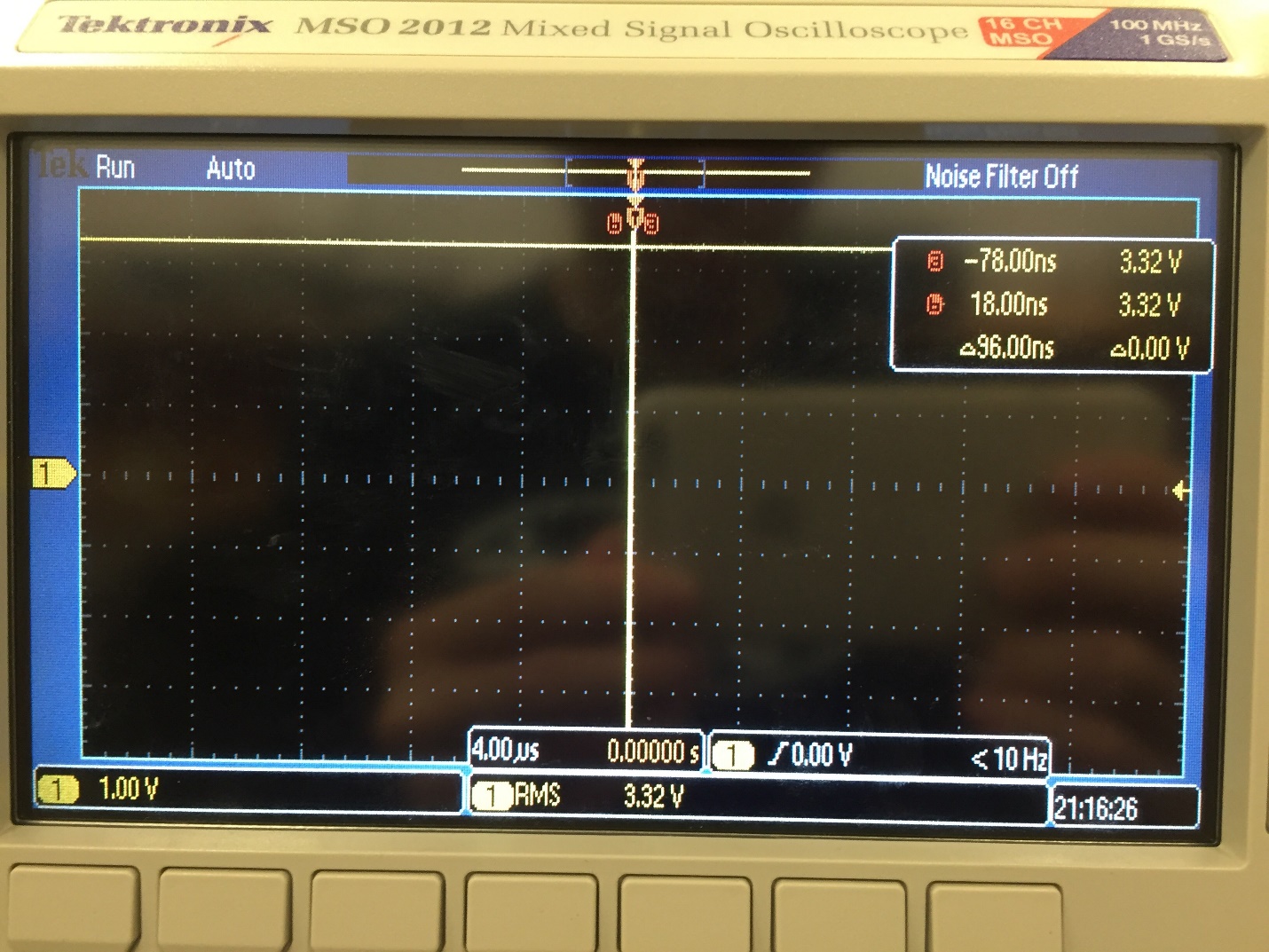


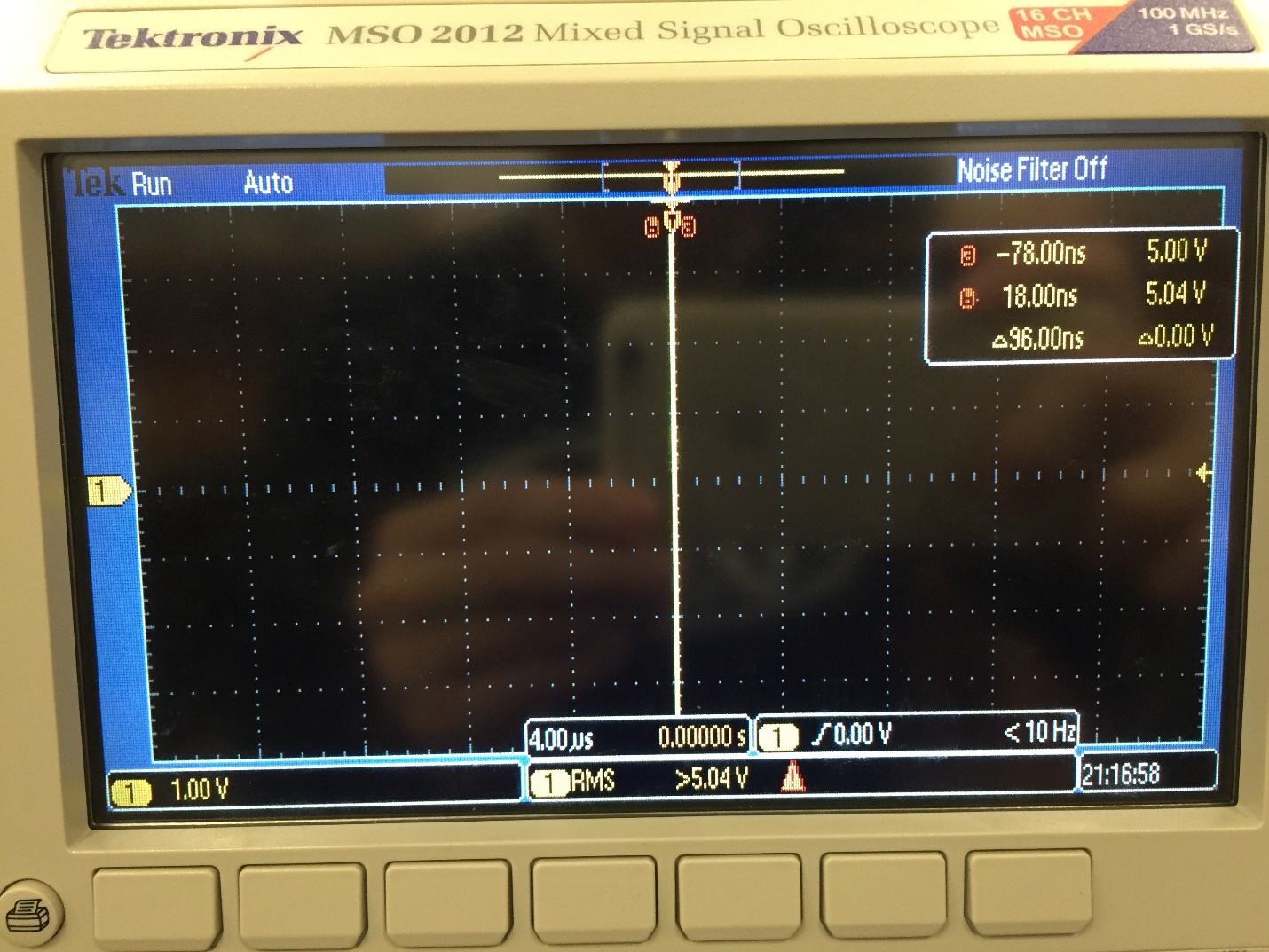
Call Graph

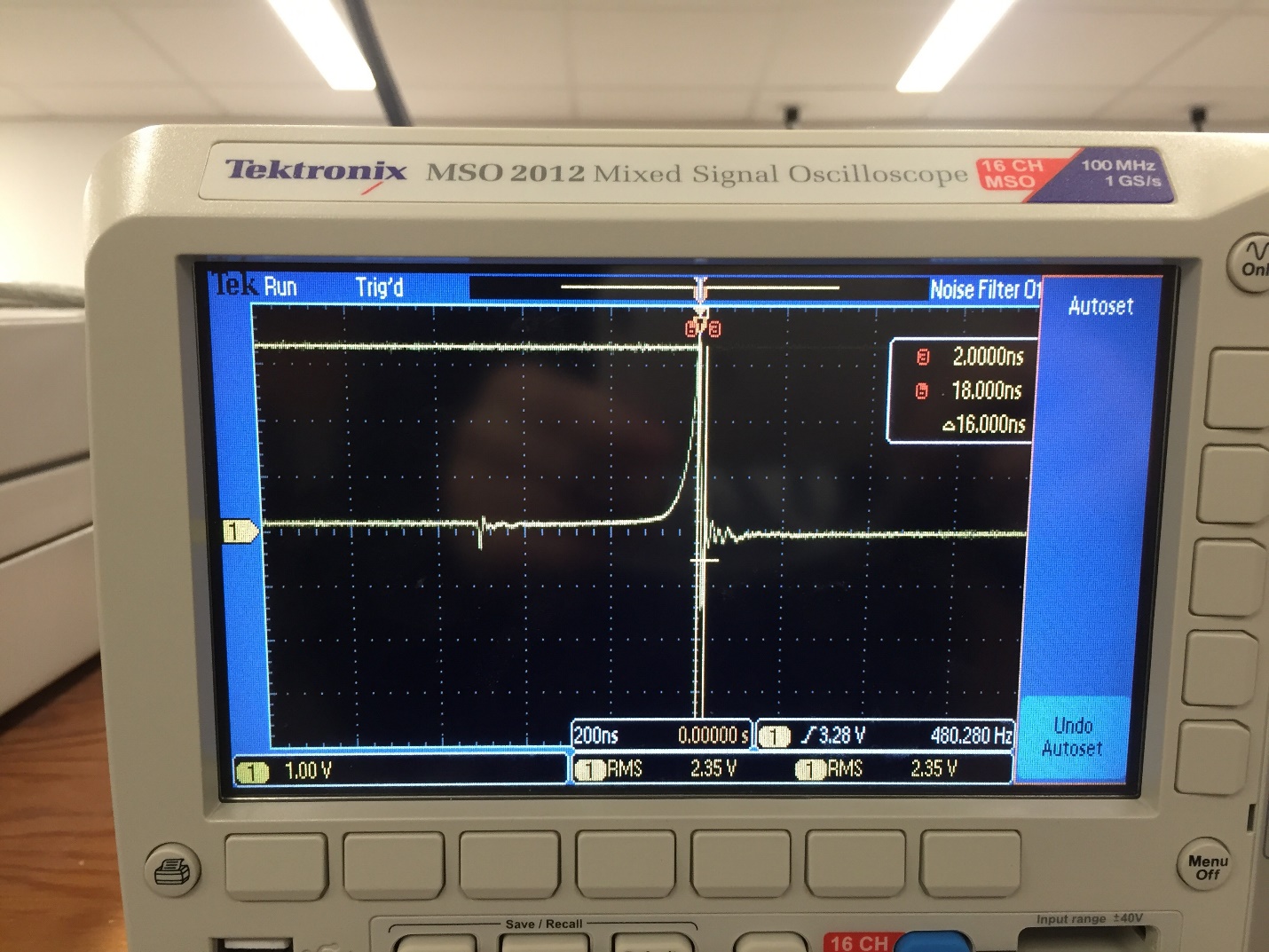


**Measurement Data:**

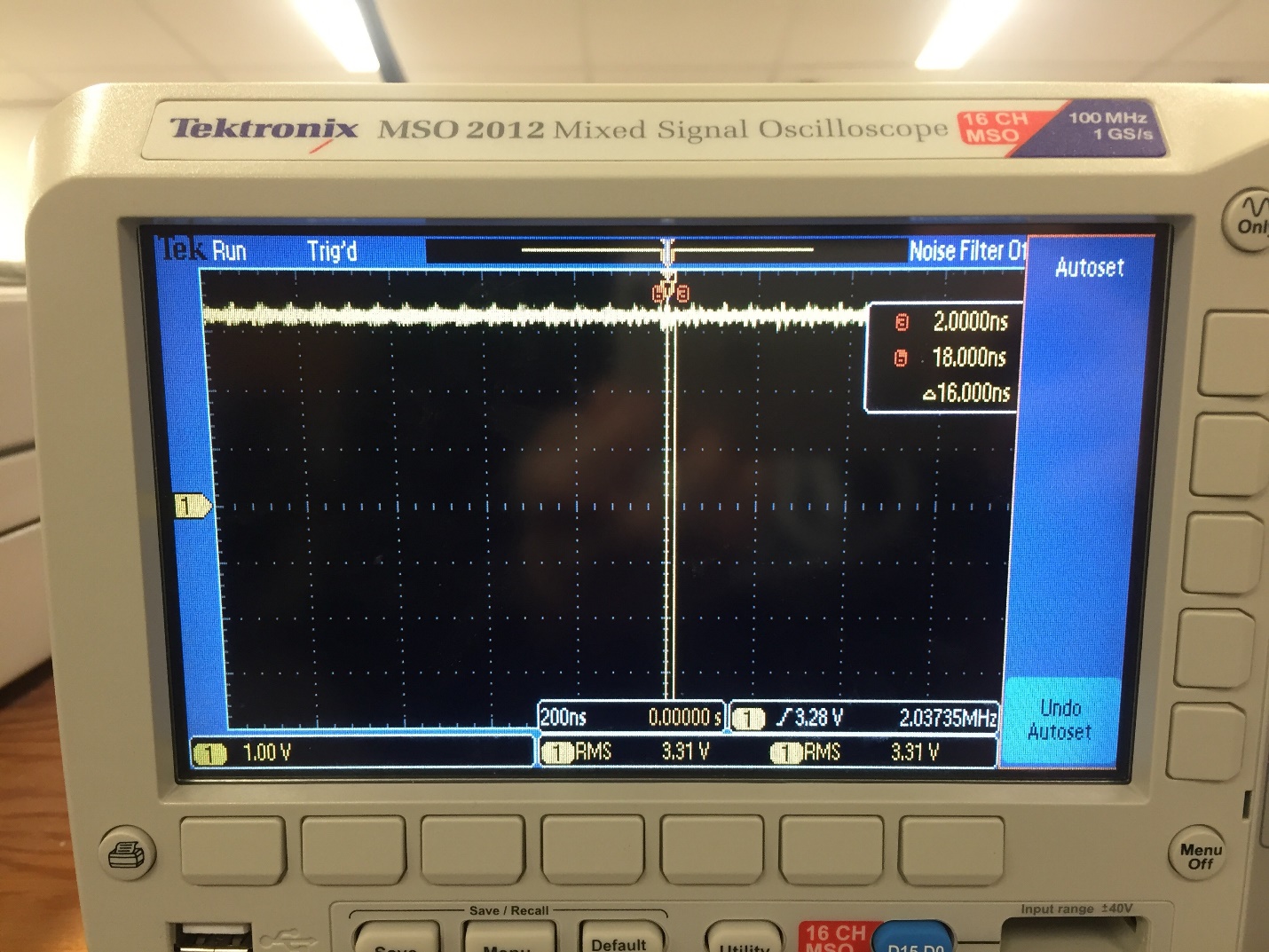
**+3.3 V vs time RMS magnitude : 3.32V**



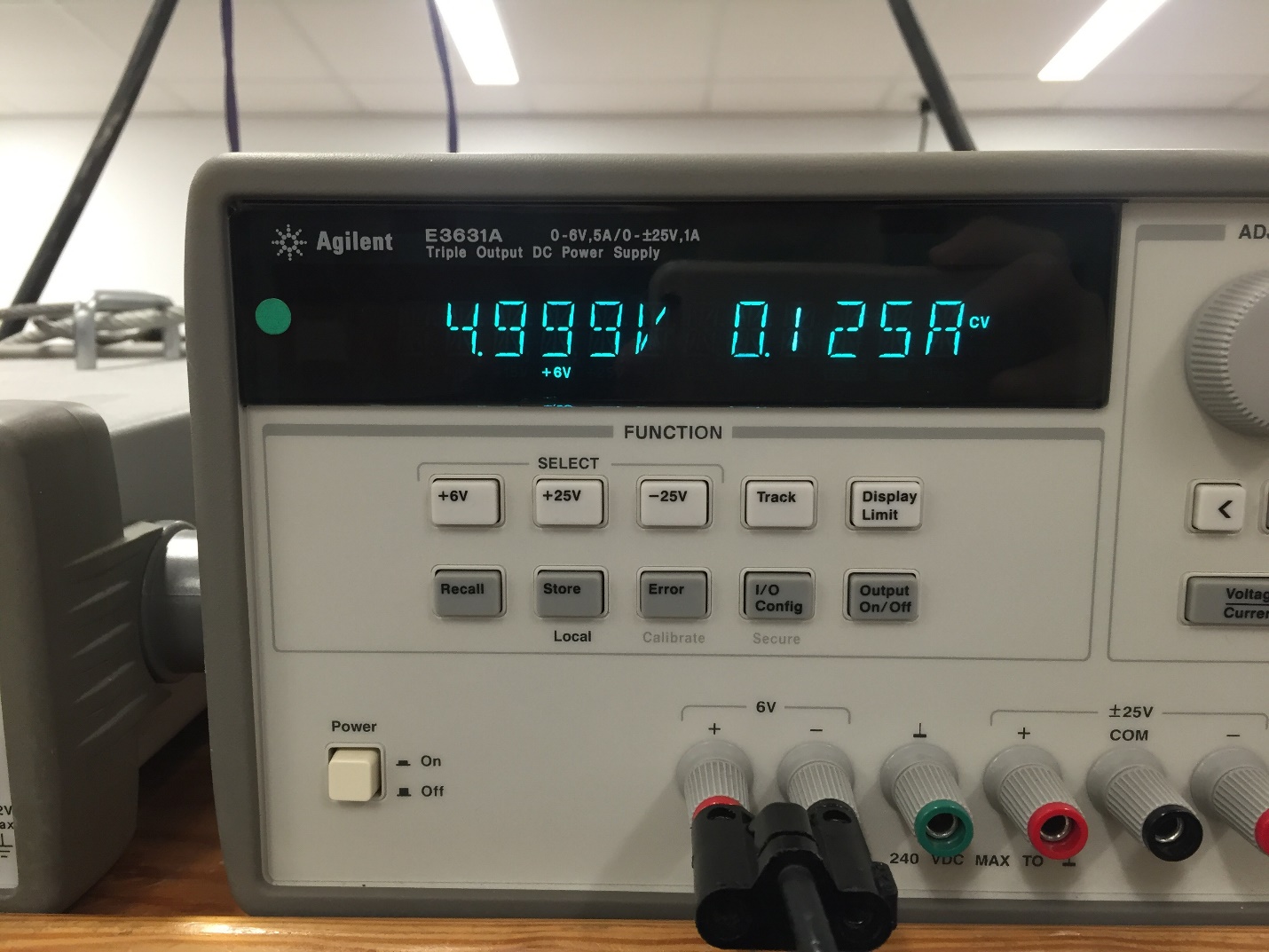
**+5.0 V vs time RMS magnitude : 5.04V**

**Input Voltage RMS to speaker:**

**Output Voltage from speaker:**



**Current Required with Alarm: 0.125A**



**Current Required without Alarm: 0.71A**

**Analysis and Discussion:**

1. One of the methods to remove critical sections is to, disable the writing of global variables so that no variables are overwritten in execution time. This can be achieved if all we use are local variables and pass in variables as parameters instead of directly writing into a global variable. Another method of removing a critical section is to disable interrupts at the time of writing into global variables. In this situation, no values would be overwritten because the interrupt I will prevent any interrupts to trigger in the critical section.
2. Our first method, digital display, takes approximately 50958 clock cycles to execute the display that updates the LCD with a new time. Our Analog display method for updating the time on the LCD takes approximately 45427 clock cycles to execute.
3. The disadvantage of updating the LCD in the background ISR would be that the program would spend a large chunk of time in the interrupt handler, which is bad programming practice. This is bad because this allows a greater chance of global values being corrupted with overwrite methods. Furthermore, this would cause certain time latencies in real time systems that could be detrimental to the function of the process.
4. We did not redraw the clock for each output. We implemented a method that would “delete” the minute and hour hand after each update. Essentially this method would redraw the previous line, in a black color that would completely overwrite the original hand. In this way we could remove the previous hand and add the new hand after each update, which reduces flicker.
5. The first way we could save power is to use a busy wait system for the main menu instead of using an interrupt, so that we could continuously poll for a button input without the use of multi-threading. Obviously we could reduce sound output or completely eliminate it to reduce battery consumption if it were absolutely necessary. We could implement a system in which the program would go to sleep if the 10 second wait is passed, instead of going to the main menu.