Ronald Macmaster and Parth Adhia

EID: Rpm953 and Pda375

09/23/2016

# EE 445L – Lab 3: Alarm Clock

## 1.0: Objectives:

**Revised Requirements document**

## 1. Overview

1.1. Objectives: Why are we doing this project? What is the purpose?

**Our primary objectives for this project are to design, build and test an alarm clock. We are learning how to design and test modular software and how to perform switch / keypad input in the background.**

**Additionally, we will develop a graphics driver for our LCD to plot lines and circles, and we will also design hardware / software interfaces and drivers for a couple of switches and speaker. We will measure the supply current necessary to operate our impeded system, and implement an alarm clock with periodic interrupts.**

**Our project will also require us to satisfy extra open-ended requirements to the original design. We have formulated these extra requirements and presented them later in the requirements document. Finally, we will explore the deeper design principles of modularity, information hiding, coupling, and cohesion.**

1.2. Process: How will the project be developed?

The project will be developed using the TM4C123 board. The system will utilize multiple control switches constituting a keypad. The system will be built on a solderless breadboard and run on the usual USB power. The system may additionally use the on-board switches and / or the on-board sound. However, the system **will** include an external **speaker**. Our system will consist of four hardware/software modules: switch/keypad input, time management, LCD graphics, and sound output. The process will be to design and test each module independently from the other modules. After each module is tested, the system will be built and tested.

1.3. Roles and Responsibilities: Who will do what? Who are the clients?

Ronald Macmaster and Parth Adhia are the engineers and the Dylan Zika (TA) is the client. We have modified this document to clarify exactly what we plan to build. We are allowed to divide responsibilities of the project however we wish, but, at the time of demonstration, both of us are expected to understand all aspects of the design.

*Who will do what?*

**Ronald Macmaster will be responsible for drafting the requirements document and the lab report. He will also create the software design diagrams that will consist of a call graph and a data-flow graph. He will draft up the interface files for the various software modules and create program**

**skeletons for the various driver files. Finally, he will supervise and contribute to the software development process and submit all of the assignment documentation.**

**Parth Adhia will be responsible for drafting the hardware design documentation. This will require a schematic diagram of the external system hardware that should be built using the PCB Artist program. He will write a significant portion of the software drivers and assemble the external system hardware circuit. He is responsible for providing the speaker, BJT, and tactile switches as well. Finally, he will edit and contribute to the various project documentation as fit.**

1.4. Interactions with Existing Systems: How will it fit in?

The system will use the TM4C123 board, a ST7735 color LCD, and a solderless breadboard. It will operate on power through the usual USB cable. Additionally, the system will interface an external keypad composed of four tactical switches and a 32 Ohm speaker.

The TM4C123 contains the main processing unit, the LCD will display the digital alarm clock output, and the breadboard will serve as a base for our external hardware devices. The keypad will be used to service user input, and the speaker will perform the alarm output function.

1.5. Terminology: *We herein define the following key terms:*

**Power budget:**

The **power budget** of an embedded system can be represented by the following concept: Suppose $E$ is the total battery lifetime in amp-hours of the system's battery and $t_{life}$ is the desired operating lifetime of the system; then the average current the system is allowed to draw is represented by $I_{avg} <= ( E / t_{life} )$.

**Device driver:**

A **device driver** consists of the software routines that provide the functionality of an I/O hardware device. The device driver does not need to hide what type of I/O module it is or which device implements it.

**Critical section:**

When two threads have shared access to global data or an I/O port, and one of the accesses is a write, this gives birth to a **critical section** in the lower priority thread. Usually, this occurs when the write access is a multistep, non-atomic sequence of read-modify-write, write-write, or write-read.

**Latency:**

In the context of a real-time system, **latency** is the time between the request for service from an I/O device and the time the device is actually serviced.

**Time jitter:**

The **time jitter** of a system is the deviation from the true periodicity of a periodic signal or routine. In the context of our system, we will observe the **time jitter** of our system to prove it's real-time. It can be calculated as the difference between the largest and smallest time differences between signals or interrupt service routine executions.

**Modular Programming:**

The **modular programming** approach is a style of software development that divides the software problem into distinct and independent modules. The modules are as small as possible and relatively

independent of one another. Each module consists of a self-contained software task with clear entry and exit points.

1.6. Security: How will intellectual property be managed?

The system may include software from Tivaware and from the book. No software written for this project may be transmitted, viewed, or communicated with any other EE445L student past, present, or future (other than the lab partner of course). It is the responsibility of our team to keep our EE445L lab solutions secure. We will manage our source code over a private git repository hosted by GitHub Inc.

## 2. Function Description

2.1. Functionality: What will the system do precisely?

The clock must be able to perform five functions. 1) It will display hours and minutes in both graphical and numeric forms on the LCD. The graphical output will include the 12 numbers around a circle, the hour hand, and the minute hand. The numerical output will be easy to read. 2) It will allow the operator to set the current time using switches or a keypad. 3) It will allow the operator to set the alarm time including enabling/disabling alarms. 4) It will make a sound at the alarm time. 5) It will allow the operator to stop the sound. An LED heartbeat will show when the system is running.

**In addition to the five functions above, the system will also implement an on-screen animation when the alarm sounds. It will also offer the option to set more than one alarm.**

2.2. Scope: List the phases and what will be delivered in each phase.

Phase 1 is the preparation; phase 2 is the demonstration; and phase 3 is the lab report. Details can be found in the lab manual.

**For preparation, we will deliver an updated requirements document, schematic diagram of our hardware system, and a call and data-flow graph that models our software design. We will also demonstrate possession of the necessary hardware components and reasonable progress on the various software modules. The software will be written and compiled by the preparation date.**

**For checkout demonstration, we will begin by demonstrating the features of our LCG graphics driver. We will also present our digital alarm clock software and system. The clock will be stand-alone (powers on and off) and the alarm clock will run with a changeable alarm setting.**

**The final documentation for this project will be written up in a lab report and submitted over Canvas for grading by Midnight on Friday, September 23rd.**

2.3. Prototypes: How will intermediate progress be demonstrated?

A prototype system running on the TM4C123 board, ST7735 color LCD, and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration and lab report presentations.

2.4. Performance: Define the measures and describe how they will be determined.

The system will be judged by three qualitative measures. First, the software modules must be easy to understand and well-organized. Second, the clock display should be beautiful and effective in telling time. Third, the operation of setting the time and alarm should be simple and intuitive. The system should not have critical sections. All shared global variables must be identified with documentation that a

critical section does not exist. Backward jumps in the ISR should be avoided if possible. The interrupt service routine used to maintain time must complete in as short a time as possible. This means all LCD I/O occurs in the main program. The average current on the +5V power will be measured with and without the alarm sounding.

2.5. Usability: Describe the interfaces. Be quantitative if possible.

There will be two to four switch inputs. In the main menu, the switches can be used to activate 1) set time; 2) set alarm; 3) turn on/off alarm; and 4) display mode. The user should be able to set the time (hours, minutes) and be able to set the alarm (hour, minute). Exactly how the user interface works is up to you. After some amount of inactivity, the system reverts to the main menu. The user should be about to control some aspects of the display configuring the look and feel of the device. The switches MUST be de-bounced, so only one action occurs when the operator touches a switch once.

The LCD display shows the time using graphical display typical of a standard on the wall clock. The 12 numbers, the minute hand, and the hour hand are large and easy to see. The clock can also display the time in numeric mode using numbers.

The alarm sound can be a simple square wave. The sound amplitude will be just loud enough for the TA to hear when within 3 feet. **When sounding the alarm, the alarm will also play a short animation on the screen. In addition to the hours and minute hands on the microcontroller, it will also display an alarm status and a seconds hand.**

2.6. Safety: Explain any safety requirements and how they will be measured.

The alarm sound will be VERY quiet in order to respect other people in the room during testing. Connecting or disconnecting wires on the protoboard while power is applied may damage the board.

## 3. Deliverables

3.1. Reports: How will the system be described?

**The final lab report is due on Friday, September 26th at Midnight. This report will include the final requirements document. The outline of the lab report is as follows:**

**A) Objectives (final requirements document)**

**B) Hardware Design (External hardware schematic)**

**C) Software Design (Updated software modules, Call graph, and Data-flow graph)**

**D) Measurement Data**
**(Supply and Speaker V vs. T plots, RMS magnitudes, Current required with and without alarm.)**

**E) Analysis and Discussion (short answers to a couple design questions)**

3.2. Audits: How will the clients evaluate progress?

**Clients will be presented with a preparation the week before before the final checkout.**
**The lab report will be presented at conclusion of the project.**

3.3. Outcomes: What are the deliverables? How do we know when it is done?

There are three deliverables: preparation, demonstration, and report. **Most of our work and design will be documented in the final lab report.**

## 2.0 Hardware Design:

In addition to the usual ST7735 LCD screen, our system will interact with multiple user input switches and a 32 Ohm speaker. The four tactile switches will be implemented with positive logic, and the speaker will be interfaced through a BJT.

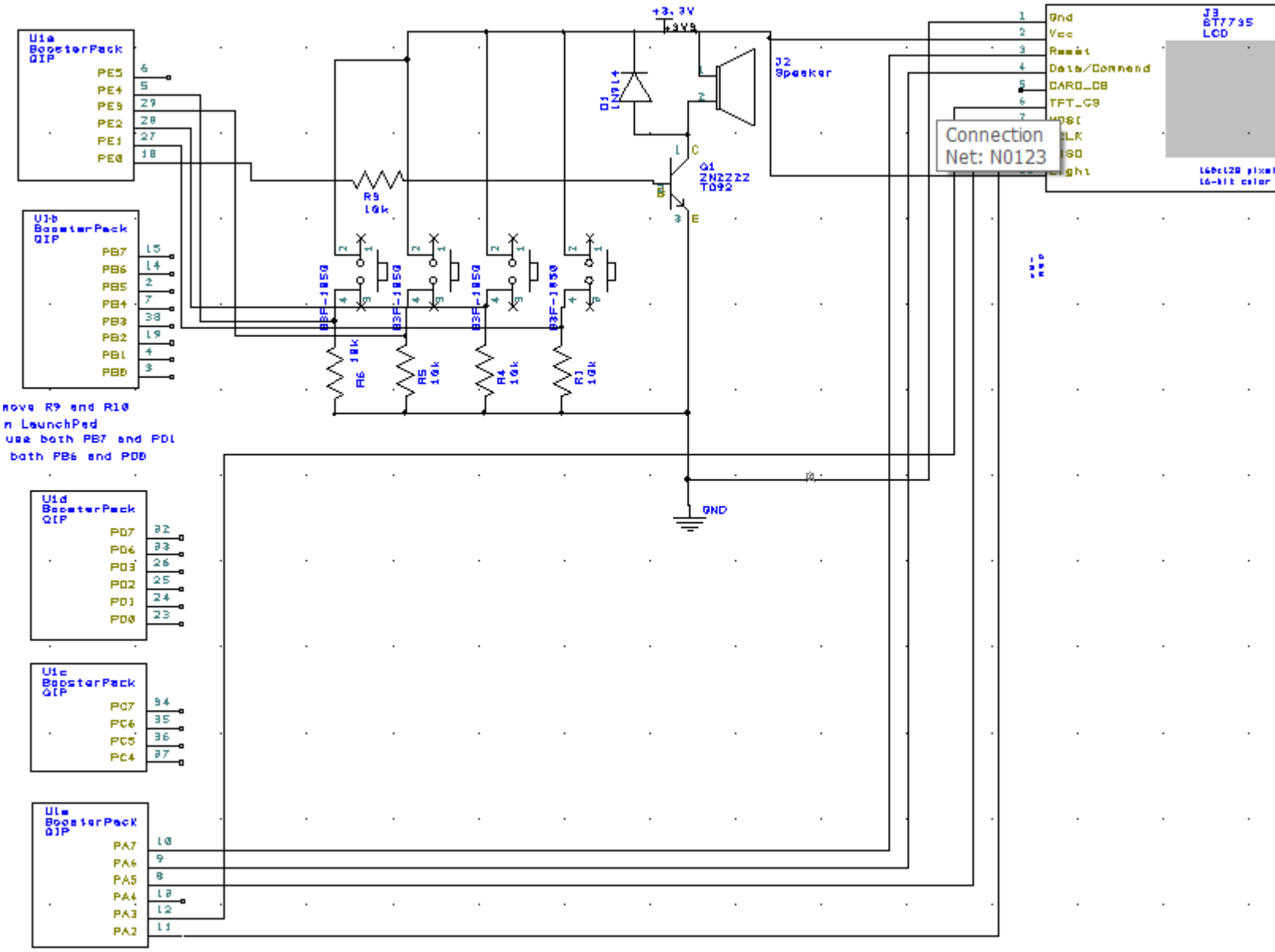



**Figure 2.1:** *Hardware Schematic for our Alarm clock system. The switches utilize positive logic to implement the keypad user interface. The speaker is interfaced through the BJT and PE0 on our protoboard.*

## 3.0: Software Design:

We have decided to proceed with the original software architecture presented in the lab manual. All of our required and additional functionality can be implemented using four basic modules — switch control, LCD graphics, time management, and speaker control. Submitted along with this report are the interface and driver files (.h and .c) for our software modules (Timer, ST7735, Speaker, and Switch).

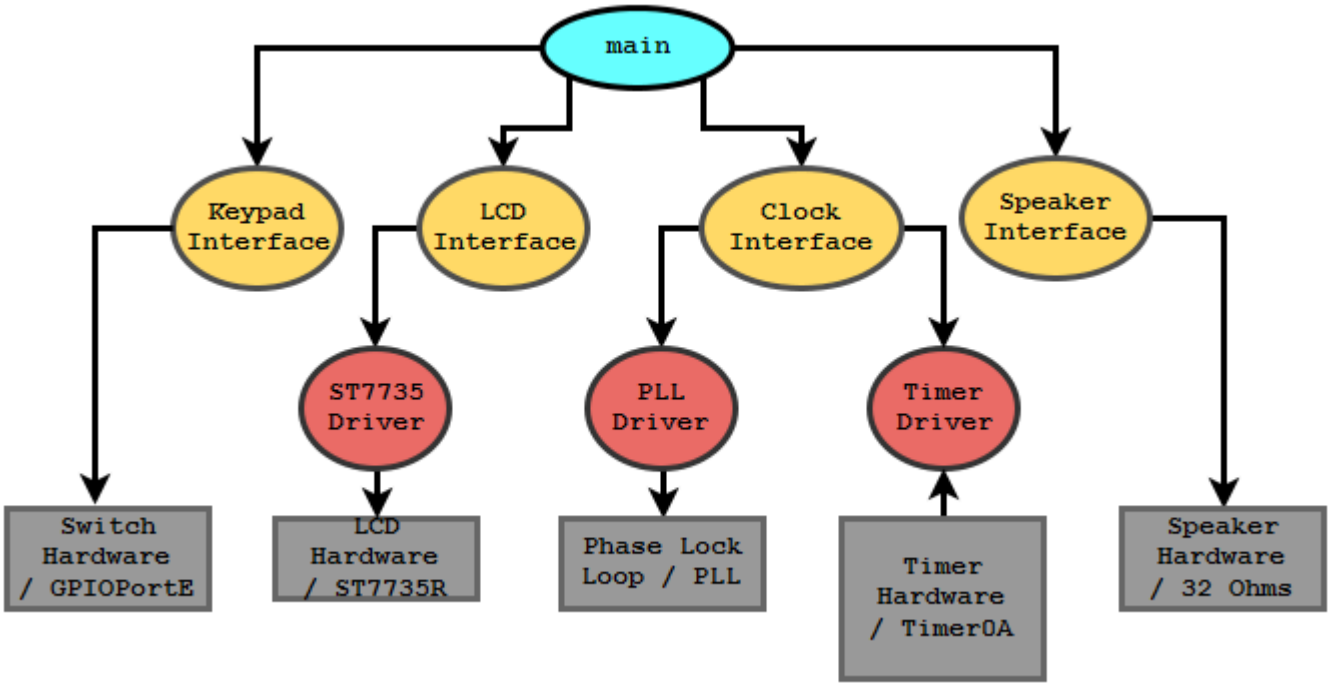



**Figure 3.1:** *Call graph for our Alarm clock system. A main driver program manages hardware through secondary software modules.*

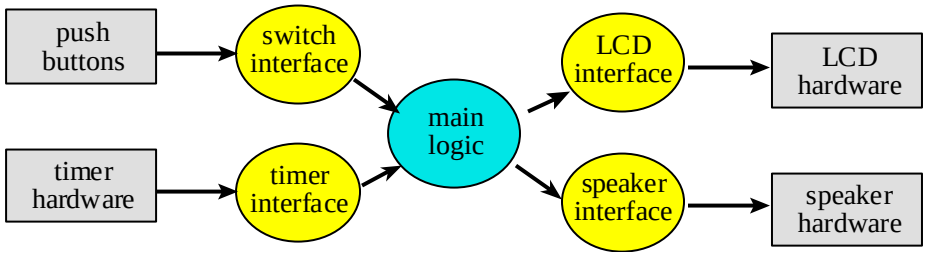



**Figure 3.2:** *Data-flow graph for our Alarm clock system. Hardware service requests are from the timer and switch interfaces. The main driver logic outputs processed input to the LCD and speaker interfaces.*

## 4.0: Measurement Data:

*Build and test any external hardware needed. Debug each module separately. Debug the overall alarm clock. Measure how long it takes to update the graphical time on the LCD. Identify all shared I/O ports and global variables. I.e., list all the permanently allocated variables that have read or write access by more than one thread. Next, consider what would happen if the interrupt occurred between any two instructions of the main program. Remember high priority interrupts can suspend lower priority ISRs. Look for critical sections, and if you find any remove them. Document in your lab manual that each shared object is not critical. During checkout, the TA may ask you to prove that your system has no critical sections.)*

We have one shared global variable (ReadyForInput). The shared access to this global variable does NOT create a critical section because there are no read, modify, write accesses to it. Even if ReadyForInput is overwritten, the results are not catastrophic. The user probably cannot press a button at a rate greater than 16Hz.

Updating the graphical time on the LCD only requires changing the hour, minutes, and seconds hands, and it takes **~41.379ms**.

### 4.1: Supply Voltage vs. Time Plots

Plot the +5 and +3.3 supply voltages versus time and record the rms magnitudes

**Figure 4.1)** 5.0 V power supply voltage. RMS Voltage: 5.04V (DC)

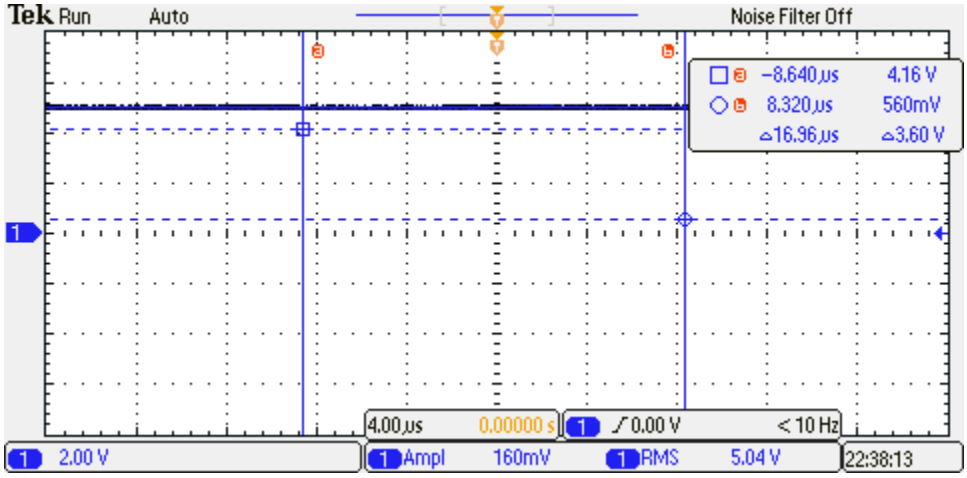
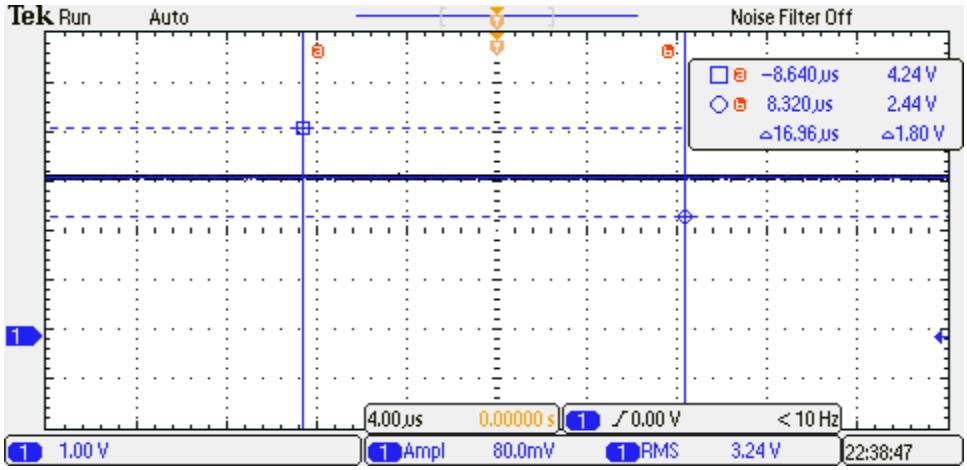



**Figure 4.2)** 3.3 V power supply voltage. RMS Voltage: 3.24 V (DC)



## 4.2: Speaker Voltage vs. Time Plot

Plot the speaker voltage (or output voltage) versus time during an alarm sound

**Figure 4.3)** PWM Output on PB6

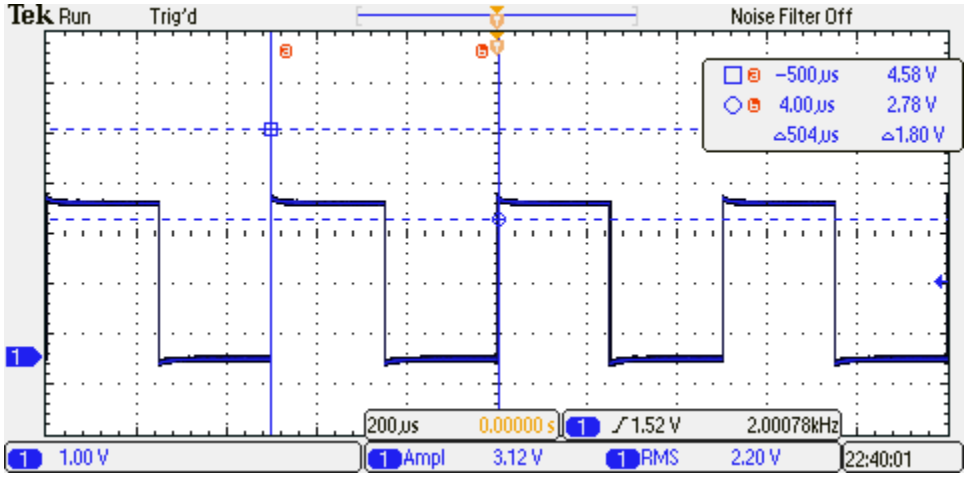
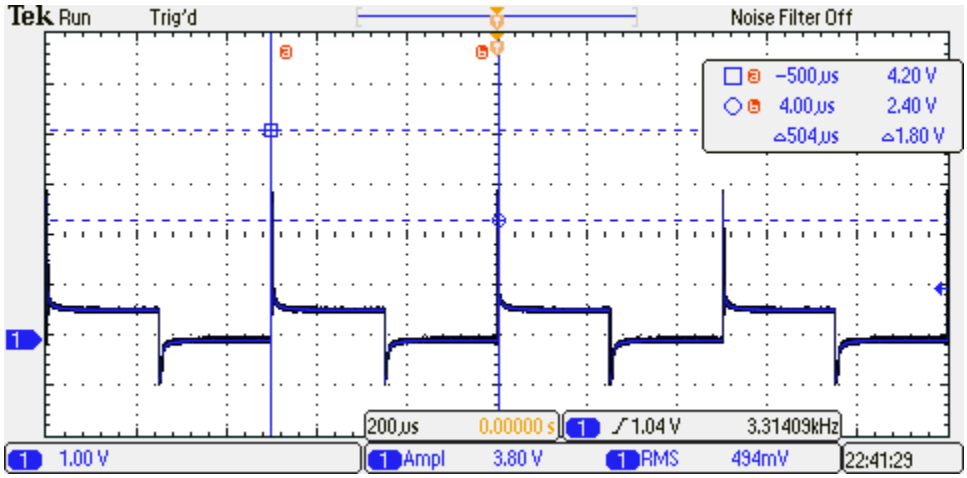



**Figure 4.4)** Speaker Voltage vs. Time




## 4.2: Power Analysis

Measurements of current required to run the alarm clock, with and without the alarm

Without the alarm clock: **72 mA**

With the alarm: **111 mA**

## 5.0: Analysis and Discussion:

*1) Give two ways to remove a critical section.*

We can remove critical sections by **removing shared access between threads** or **making the vulnerable sequences atomic**. That involves utilizing bit-specific addressing, adjusting thread priority, packing global data, or disabling interrupts.

*2) How long does it take to update the LCD with a new time?*

For a graphical time update, all we need to do is redraw the three clock hands. We sampled multiple trials of this portion of software, and we found that it takes **41.3788375 ms** to update the LCD with a new graphical time.

*3) What would be the disadvantage of updating the LCD in the background ISR?*

LCD updates can take a couple of milliseconds to complete. Putting an LCD update in a background ISR with a priority that's too high will hog the system resources from the other lower-priority threads. The LCD ISR will greatly increase the latency of lower priority threads and severely impact the real-time classification of our system.

*4) Did you redraw the entire clock for each output? If so, how could you have redesigned the LCD update to run much faster, and create a lot less flicker?*

No, we did not redraw the entire clock for each output. All we had to do was redraw the clock hands to update the time. That involves writing the background color over the previous hand and drawing the new hand. When in digital mode, all we have to do is reprint the clock string. Our LCD does not have much flicker at all, if any.

*5) Assuming the system were battery powered, list three ways you could have saved power.*

We could save power by decreasing the voltage it takes to power our system, using a greater resistance to limit the flow of current through the speaker, and decreasing the frequency of our interrupts to limit transistor switching.