Alarm O’Clock

CS122A: Fall 2017

Andrew Apostol

Table of Contents

[**Introduction**](#_n9o44neiquk5) **2**

[**Hardware**](#_vqmbsuo5p330) **3**

[Parts List](#_xtqh2tzibtrr) 3

[Block Diagram](#_8m8fex2vr9el) 4

[Pinout (For each microcontroller/processor)](#_8yptvhjzojq8) 4

[**Software**](#_w9543qe124on) **4**

[**Implementation Reflection**](#_yfnwo0ojgf6a) **5**

[Milestone](#_2j4av972xaj8) 6

[Completed components](#_tp9dnsse4u9p) 6

[Incomplete components](#_99jbzhpy2eks) 6

[**Youtube Links**](#_qj6ffofi3ean) **6**

[**Testing**](#_stprv61uxgvu) **7**

[**Known Bugs**](#_grlzpb6vy2cq) **19**

[**Resume/Curriculum Vitae (CV) Blurb**](#_rvkpcxu3jywt) **19**

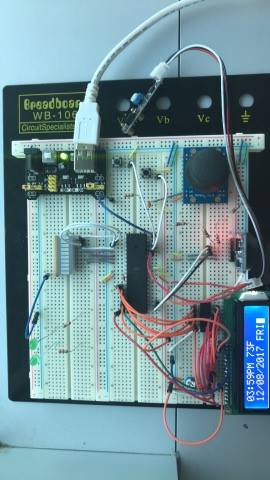
[**Future work**](#_qbv6f31drpex) **19**

[**References**](#_1txxevdbm0qb) **19**

[**Appendix**](#_i5qunlt5l708) **20**

# Introduction

The project is an alarm clock that can tell the time, day of week, date, and weather, and set off a programmable alarm. The time, date, day, and temperature are all read from a real time clock. Through the menu, the temperature can be changed between F and C, time can be changed between 12 hours and 24 hours, and an alarm can be set. The alarm is turned off by the heartbeat sensor sensing a heartbeat.



Github link: https://github.com/aapos001/Alarm-O-Clock

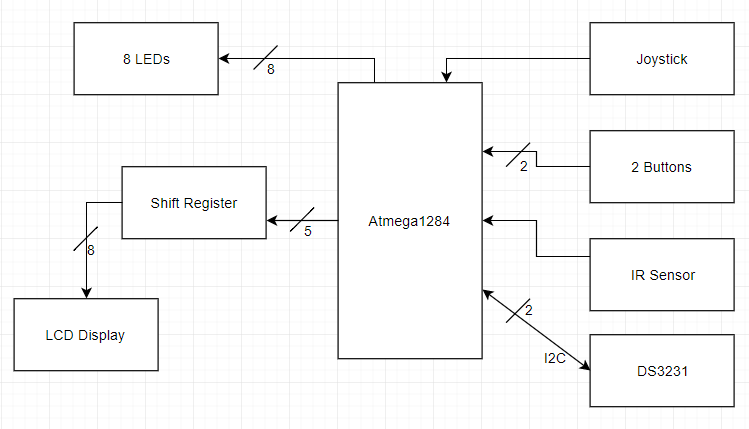
# Hardware

## Parts List

The hardware that was **used** in this project is listed below. The equipment that was not taught in this course has been bolded. *Include part numbers when available*.

* ATMega1284 microcontroller (1)
* Joystick (1)
* Push buttons (2)
* LEDs (8)
* **DS3231 (1)**
* 16 x 2 LCD display (1)
* SN74HC595N Shift Register (1)
* 330Ω Resistor (9)
* 1000Ω Resistor (4)
* 103 Capacitor (2)
* 101 Capacitor (1)
* **IR sensor** (1)

## Block Diagram

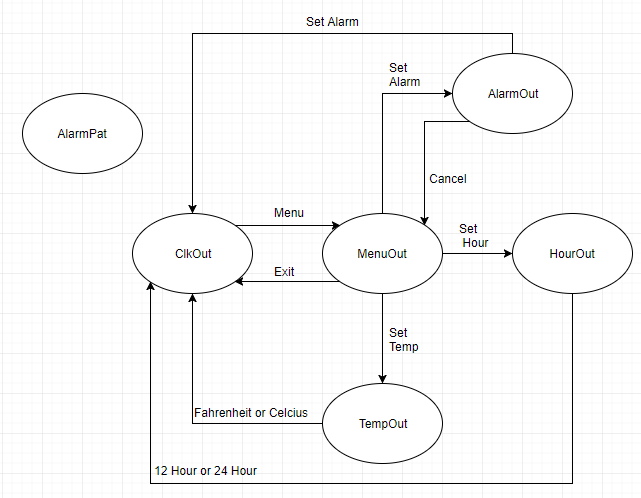


## Pinout (For each microcontroller/processor)

* **Inputs**
  + Joystick A0
  + 2 Buttons A2, A3
  + Heartbeat sensor A4
  + DS3231 SDA C1
* **Outputs**
  + Shift Register D0-D4
  + LCD Display D5-D6
  + DS3231 SCL C0
  + LEDs B0-B7

# Software

The software designed for this project was implemented using the PES standard. The overall design as a task diagram is included below.



The ClkOut task outputs the current time, temperature, date, month, year, and day.

The MenuOut task outputs the choices for settings such as set alarm, set temperature, and set hour mode.

The AlarmOut task allows the user to set an alarm.

The HourOut task allows the user to change the temperature display to Fahrenheit or Celcius.

The TempOut task allows the user to change the hour display to 12 hour mode or 24 hour mode.

The AlarmPat task continuously checks whether the alarm set is equal to the current time. It outputs an alarm when the alarm set and current time are equal.

# Implementation Reflection

I am satisfied with the project that I have completed because what is done runs smoothly. The best part of the project is how the tasks work together. Instead of one large task that controlled the output of the LCD, I split each part of the LCD output into different tasks. I treated the tasks similar to an OS, where only 1 tasks was running at any time because they all shared the LCD display. This allowed me to sandbox and bug test each task individually.

If I were to do this project again, there are a few things I would do differently. In terms of software, I would change the tasks into triggered state machines. Currently, the tasks poll until they are given “admin” privileges. Next, I would add the left and the right parts so that the user can move through the hours and minutes of alarm setting more easily. At the moment, only the up and down parts of the joystick are necessary. Although commented, the code is somewhat messy. To fix this, I would try to use more #define and declare functions for repetitive code in some tasks. Lastly, I would try to implement the DS3231 alarm function. The DS3231 has an alarm function that is not used in this project. I store the set alarm in a global variable and compare this value to the time I output.

In terms of user interface, I would label the buttons to make it more intuitive what they do.

## Milestone

The milestone was to be able to read from the DS3231, output the reading onto the LCD, and have a programmable alarm that can be turned off by the heartbeat sensor. I fell short of the milestone due to careless mistakes. In the beginning, I was having trouble outputting onto the LCD display through the shift register. I was clearing the control line during the transmission to the shift register, so the LCD display did not display anything. At some point during week 8, the atmega1284p’s PORTD stopped working. This caused a lot of headache when debugging because I focused on finding the bugs in the code. Afterwards, I spent a long time trying to get readings from the DS3231. The i2c communication on the atmega1284 calls for a pullup resistor. I thought that I could use the internal pullup resistors in the atmega1284, but the datasheet says specifically to use external ones.

## Completed components

I finished the 70-80 project. I ran out of time to work on the 80-90 project.

## Incomplete components

I did not complete the 80-90 project. As stated in the milestone, I ran into problems with the LCD display and shift register, atmega1284p, and DS3231.

# Youtube Links

**Make sure they are publicly viewable!**

* Short video: https://www.youtube.com/watch?v=U533-srnK-c
* Longer video: https://www.youtube.com/watch?v=99J2WPDwORY

# Testing

Joystick

To test the joystick, I set up an LED bar and set specific patterns for different voltage readings. I took the voltage readings that corresponded to pushing the joystick all the way up or all the way down and used them in the project.

**Joystick up and down**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | 0x00 on LED bar |
| Move joystick up | if(ADC > 750) { 0xF0 } else if(ADC > 600) {0x30} |
| None | 0x00 |
| Move joystick down | if(ADC < 200) { 0x0F } else if(ADC < 350)  {0x03} |
| None | 0x00 |

Buttons

To test the buttons, I set up an LED to turn on when the button was pressed and off when it was not pressed.

**Button presses**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | LED off |
| Button press | LED on |
| None | LED off |

LCD Display

To test the LCD display, I hardcoded a set of outputs that would test each output function. In between each output was a delay of about 5s to see which function was or was not working.

**LCD and shift register**

|  |  |
| --- | --- |
| Input | Expected Output |
| LCD\_Init() | LCD display turns on |
| LCD\_DisplayString(1, “Hello World”) | “Hello World” starting on cursor 1 |
| LCD\_Cursor(2) | Cursor pointed on cursor 2 |
| LCD\_ClearScreen() | Clear the screen |

**LCD with buttons**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | LCD display turns on |
| Button 1 | Display “menu” |
| None | LCD\_ClearScreen() |
| Button 2 | Display “exit” |
| None | LCD\_ClearScreen() |

Clock Task

This task displays the time, temperature, date, and day of the week. The time and temperature are dependent on global variables set by other tasks. The menu task can be moved to from clock task. The menu task tests test the transition to the menu task.

**Change temperature and hour**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “11:00PM 55F  12/4/2017 MON” |
| Change temperature to Celsius | Display “11:00PM 13C  12/4/2017 MON” |
| Change temperature to Fahrenheit | Display “11:00PM 55F  12/4/2017 MON” |
| Change hour mode to 24 hour mode | Display “23:00 55F  12/4/2017 MON” |
| Change hour mode to 12 hour mode | Display “11:00PM 55F  12/4/2017 MON” |

DS3231

The DS3231 communicates with the atmega1284 through i2c. To read from the DS3231, the time must be set manually on the clock first. To test the clock, the registers were set to Sunday 11:59:00PM 12/31/2017. Once 60 seconds pass, the minute, hour, day date, month, and year will all increment.

**RTC**

|  |  |
| --- | --- |
| Input | Expected Output |
| Sunday 11:59:00PM 12/31/2017 | Display “11:59PM 75F  12/31/2017 SUN” |
| Time ticks for 60 seconds | Display “12:00AM 75F  01/01/2018 MON” |

To test the temperature, 5C was displayed as initial temperature. Then the DS3231 temperature sensor was read from.

**Temperature**

|  |  |
| --- | --- |
| Input | Expected Output |
| 5C | Display “11:00PM 5C  12/4/2017 MON” |
| Read from temperature register | Display “11:00PM 23C  12/4/2017 MON” |

To test the 12 hour and 24 hour conversion, the registers in the DS3231 were set to 12 hour initially. The test changes the register to 24 hour mode and then back to 12 hour mode.

**12 hour 24 hour conversion**

|  |  |
| --- | --- |
| Input | Expected Output |
| 12 hour mode in DS3231 hour register | Display “11:00PM 55F  12/4/2017 MON” |
| Change to 24 hour mode | Display “23:00 55F  12/4/2017 MON” |
| Change to 12 hour mode | Display “11:00PM 55F  12/4/2017 MON” |

Menu Task

The menu moves between the clock task, set alarm task, set temperature task, and set time task. Moving between each task involves a handoff of “admin” privileges that allows the task to display. If the display shows more than one task’s output, then the handoff did not work properly. This means that moving between the menu task and task A is not affected by moving between the menu task and task B.

First, I tested whether the display moves between the clock task and menu task with the buttons. Then, I tested whether the display will move between display options in the menu. The joystick moves the menu up and down. Each state should be able to go back to the clock task.

**Enter and leave menu**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Button 2 | Exit menu, display clock |

**Scroll through menu, then exit menu from each display**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Up on joystick | No change |
| Down on joystick | Display “1. Alarm  2. F/C <-” |
| Up on joystick | Display “Menu  1. Alarm <-” |
| Down on joystick | Display “1. Alarm  2. F/C <-” |
| Down on joystick | Display “2. F/C  3. 12H/24H” |
| Down on joystick | No change |
| Up on joystick | Display “1. Alarm  2. F/C <-” |
| Button 2 | Exit menu, display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Down on joystick | Display “1. Alarm  2. F/C <-” |
| Down on joystick | Display “2. F/C  3. 12H/24H” |
| Button 2 | Exit menu, display clock |

To test the menu options, I went to each menu option.

**Enter and leave set alarm**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Button 1 | Exit menu, display set alarm |
| Button 2 | Exit set alarm, display menu |

**Enter and leave set temperature**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Down on joystick | Display “1. Alarm  2. F/C <-” |
| Button 1 | Exit menu, display set temperature |

**Enter and leave set hour**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display clock |
| Button 1 | Exit clock, display “Menu   1. Alarm <-” |
| Down on joystick | Display “1. Alarm  2. F/C <-” |
| Down on joystick | Display “2. F/C  3. 12H/24H” |
| Button 1 | Exit menu, display set temperature |

The set temperature task and set hour task do not return to menu, so the test just checks whether they can be moved to through menu.

Setting Alarm Task

The set alarm task toggles through the hours and minutes based on whether the time is currently displayed as 12 hours or 24 hours. In the 12 hour clock alarm, the hours go up to 12 and an AM/PM bit can be set. In the 24 hour clock alarm, the hours go up to 24. The minutes go to 59 for both settings. This task can go back to the menu task and reset the task variables or set the alarm and go to the clock task. The set alarm is held in a global variable and is only changed when a new alarm is programmed.

To test this task, I started with moving back to menu immediately. The set alarm task moves back to the menu task and resets the variables for this task. Afterwards, I tested if the alarm task will set the alarm and move to the clock task.

**Enter set alarm, then reset back to menu**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-” |
| Button 1 | Exit menu, display “Set Alarm   12:00AM”  In 24 hour mode,  Exit menu, display “Set Alarm  12:00” |
| Button 2 | Exit set alarm, display menu |

**Set an alarm**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00AM”  In 24 hour mode,  Exit menu, display “Set Alarm  12:00” |
| Button 1 | Cursor to minutes |
| Button 1 | Cursor to AM/PM  In 24 hour mode,  Exit set alarm, display clock, alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit set alarm, display clock, alarm\_hour, alarm\_min on LED bar |

The buttons move between the hours, minutes, and possible AM/PM display. Pressing button 1 on the hours should move the cursor to the minutes. Button 2 exits to the menu.

From the minutes, button 1 moves the cursor to the AM/PM bit in 12 hour mode or saves the alarm and gives “admin” to the clock task in 24 hour mode. Button 2 moves the cursor back to the hours.

In 12 hour mode, The AM/PM bit is flipped regardless of up or down on the joystick. Button 1 saves the alarm and gives “admin” to the clock task. Button 2 returns to the minutes.

**12 hour mode move between hours, minutes, and AM/PM (set alarm)**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00AM”  In 24 hour mode,  Exit menu, display “Set Alarm  12:00” |
| Button 1 | Cursor to minutes |
| Button 2 | Cursor to hours |
| Button 1 | Cursor to minutes |
| Button 1 | Cursor to AM/PM  In 24 hour mode,  Exit set alarm, display clock, alarm\_hour, alarm\_min on LED bar |
| Button 2 | Cursor to minutes(12 hour only) |
| Button 1 | Cursor to AM/PM(12 hour only) |
| Button 1 | Set the alarm variable, exit set alarm, display clock, alarm\_hour, alarm\_min on LED bar(12 hour only) |

**12 hour mode move between hours, minutes, and AM/PM (exit to menu)**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00AM” |
| Button 1 | Cursor to minutes |
| Button 1 | Cursor to AM/PM |
| Button 2 | Cursor to minutes |
| Button 2 | Cursor to hours |
| Button 2 | Exit set alarm, display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |

**24 hour mode move between hours and minutes (exit to menu)**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00” |
| Button 1 | Cursor to minutes |
| Button 2 | Cursor to hours |
| Button 2 | Exit set alarm, display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |

The joystick will increment or decrement the hours. I tested the buttons in this state, then the joystick. Starting with the 12 hour setting, I make sure the alarm\_hour’s range is from 1 to 12. Pushing up on the joystick on 12 should overflow to 1. Pushing down on 1 should underflow to 12. In the 24 hour setting, 23 overflows to 0 and 0 underflows to 23.

I tested the minutes in a similar fashion. 59 overflows to 0 while 0 underflows to 59.

To test the variables, I set an alarm and then display the alarm\_hour variable on an LED bar. Then I repeated this for the minutes and AM/PM bit. This process is repeated for the 24 hour setting.

**12 hour mode increment and decrement the hours**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00AM” |
| Up on joystick | Overflow to 1, display “01” for hours |
| Up on joystick | Increment to 2, display “02” for hours |
| Down on joystick | Decrement to 1, display “01” for hours |
| Down on joystick | Underflow to 12, display “12” for hours |
| Down on joystick | Decrement to 11, display “11” for hours |
| Up on joystick | Increment to 12, display “12” for hours |
| Down on joystick x2 | Decrement from 12 to 10, display “10” for hours |
| Button 1 x3 | Move to minutes,  move to AM/PM,  set the alarm variable, exit set alarm, display clock, and alarm\_hour, alarm\_min, AM/PM bit on LED bar |

**24 hour mode hour increment and decrement hours**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “Menu   1. Alarm <-”  and alarm\_hour, alarm\_min on LED bar |
| Button 1 | Exit menu, display “Set Alarm   12:00” |
| Up on joystick x11 | Increment from 12 to 23, display “23” for hours |
| Up on joystick | Overflow to 00, display “00” for hours |
| Down on joystick | Underflow to 23, display “23” for hours |
| Down on joystick x2 | Decrement from 23 to 21, display “21” for hours |
| Button 1 x3 | Move to minutes,  move to AM/PM,  set the alarm variable, exit set alarm, display clock, and alarm\_hour, alarm\_min on LED bar |

Setting Temperature to F/C Task

The set temperature task flips a temperature bit that determines whether F or C should be output on the clock. This task gains admin from the menu. There are 2 options, button 1 for Fahrenheit and button 2 for Celsius. After choosing an option, this task sets the temperature bit and gives “admin” to the clock task.

To test this task, I output the temperature bit on an LED. The LED turns on for F and off for C.

**Set temperature bit**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “1. Alarm  2. F/C <-”  Default F, so LED is on |
| Button 1 | Exit menu, display “ L:Fahrenheit  R:Celsius” |
| Button 2 (R) | Exit set temperature, display clock, turn off LED |
| Button 1, down on joystick, button 1 | Exit clock, display menu  Display “1. Alarm  2. F/C <-”  Exit menu, display “L:Fahrenheit  R:Celsius” |
| Button 1 (L) | Exit set temperature, display clock, turn on LED |

Setting Time to 12H/24H Task

This task is exactly the same functionally as the set temperature task. There is 1 bit that determines whether 12 or 24 hour is displayed.

**Set hour bit**

|  |  |
| --- | --- |
| Input | Expected Output |
| None | Display “2. F/C  3. 12H/24H <-”  Default 12 hour, so LED is on |
| Button 1 | Exit menu, display “ L:12 Hour  R:24 Hour” |
| Button 2 (R) | Exit set hour, display clock, turn off LED |
| Button 1, down on joystick x2, button 1 | Exit clock, display menu  Display “2. F/C  3. 12H/24H <-”  Exit menu, display “L:12 Hour  R:24 Hour” |
| Button 1 (L) | Exit set hour, display clock, turn on LED |

Alarm Task

The alarm will go off when the set alarm variable is equal to the current time. To turn off the alarm, a heartbeat must be sensed on the IR sensor for at least 3 seconds. This task is not connected to any other task because both the set alarm variable and current time are global variables. The alarm will go off on an LED bar regardless of the other tasks.

To test this, an alarm was set normally and turned off with the IR sensor.

**Alarm test**

|  |  |
| --- | --- |
| Input | Expected Output |
| Set alarm to 12:01AM | 0x00 on LED bar, time is 12:00AM |
| Time ticks to 12:01AM | 0xFF -> 0x00 -> 0xFF .. on LED bar |
| Touch IR sensor | No change |
| Hold IR sensor for 2 seconds, then let go | No change |
| Hold IR sensor for 3 seconds | 0x00 on LED bar |

To test robustness, the alarm was set in 24 hour mode, then the time was changed to 12 hour mode. This was tested both ways.

**Alarm test, changed hour mode 12 to 24**

|  |  |
| --- | --- |
| Input | Expected Output |
| Set alarm to 12:01AM | 0x00 on LED bar, time is 12:00AM |
| Change time to 24 hour mode | Time is 00:00 |
| Time ticks to 00:01 | 0xFF -> 0x00 -> 0xFF .. on LED bar |
| Touch IR sensor | No change |
| Hold IR sensor for 2 seconds, then let go | No change |
| Hold IR sensor for 3 seconds | 0x00 on LED bar |

**Alarm test, changed hour mode 24 to 12**

|  |  |
| --- | --- |
| Input | Expected Output |
| Set alarm to 00:01 | 0x00 on LED bar, time is 00:01 |
| Change time to 12 hour mode | Time is 12:00AM |
| Time ticks to 12:01AM | 0xFF -> 0x00 -> 0xFF .. on LED bar |
| Touch IR sensor | No change |
| Hold IR sensor for 2 seconds, then let go | No change |
| Hold IR sensor for 3 seconds | 0x00 on LED bar |

# Known Bugs

The temperature output has 2 bugs.

The first bug is not displaying any values above 100. I did not remember to output a third digit that would represent the hundreds digit within the display temperature code. The can easily be fixed by modding temperature by 100 and displaying that value before the tens digit.

The second bug is not displaying negative values. This can be resolved easily as well. The DS3231 temperature register stores the value as a two’s complement binary value. If the MSB is 1, then output a negative sign. If 0, then do nothing.

# Resume/Curriculum Vitae (CV) Blurb

From this Alarm O’Clock project, I learned time management. I learned how to communicate with an RTC through i2c. I also learned how to shift register into a LCD display. Finally, I learned how to use the ADC on the atmega1284.

# Future work

For the next feature, I would add an FM radio. The radio would be turned on and off with another input button. Tuning the radio frequency can be another choice on the menu. The radio could be integrated with the alarm clock and play when the alarm goes off.

The next step for this project would be to simply add another input button and create a new task that the menu can move admin into.

The case would be rectangular. The display would take up most of if not all of the face. The joystick would be replaced with 2 buttons with arrows on top to make the Alarm O’Clock less bulky. There would be 4(5 with the radio) buttons across the top that were labeled for clarity. The IR sensor would be on one of the sides. The speakers would be on either side of the buttons on top.

The PCB would reduce the size of the project. The DS3231 and IR sensor PCBs would have to be somehow integrated into it.

# References

Heartbeat Sensor Inspiration

<https://www.youtube.com/watch?v=2_c0yE9QHNI>

I2C Library

<https://github.com/g4lvanix/I2C-master-lib>

DS3231 Library

<http://www.instructables.com/id/Clock-Calender-Using-DS3231GLCDAtmega32/>

# Appendix

Refer to html files for state machines