P442 Alarm Clock Profile Report

Authors: Ching Ng, Brian Rak
Customer: Bryce Himebaugh

Overview

This report outlines the high level design for the P442 Alarm Clock (from now on referred to as “clock”, or “the clock”) for customer Bryce Himebaugh. This report will touch on design considerations, background, high level technical details, assumptions, and finally the deliverables.

The objective of the report is to inform the reader of design considerations in creating Alarm Clocks from scratch. In addition, hardware and software requirements will be outlined and specified. Finally design and functional requirements will be specified and estimated.

Background

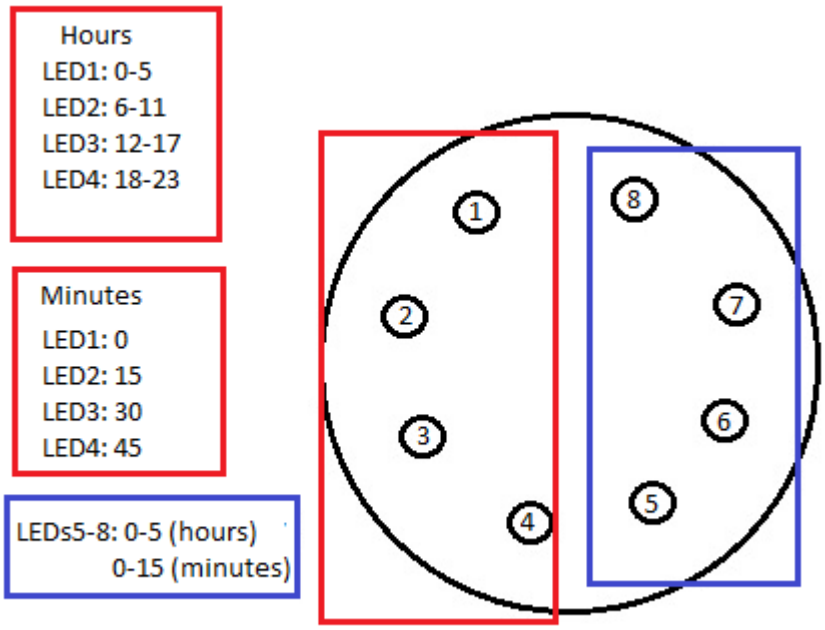
The requirements of the clock are as follows:

Design Requirements
Must use real time clock on the STM32
Must use LEDs as the time display
Must use a General Purpose Input Output (GPIO) Buzzer
Must be possible to set alarm directly from the board
Must be possible to set the time/date from a workstation when the board is connected via USB
Make a reasonable effort to prioritize battery life
May use additional GPIO buttons

To accomplish these objectives the authors will utilize pair programming technique to minimize errors and further facilitate the debugging process. The clock will be written in the C programming language and compiled using the arm-none-eabi compiler. No operating system will be used on the board itself, but the Red Hat Enterprise Linux and command line tools will be used for development and downloading to the clock. Also, Python or Bash (TBD), will be used to create the command line application to set the clock.

The clock that will be used has the capacity to store seconds, hours, days of the week, days of the month, month, and year. The clock does not provide an alarm by itself, but instead must be programmed and wired correctly for use.

Below is a (crude) implementation of the clock.



The clock uses an interesting and concise numbering system. The left side is represented in quadrants. Hours are broken up in quadrants of count 6, and minutes are broken up into quadrants of count 15 since they are in base 24 and base 60 respectively. The lights light up according to their respective quadrant in the legend above.

The right hand side of the clock is represented in binary up to 0xf. However, 0xe is the last value needed since 0xf will start a new quadrant in the quadrant category. Minutes and hours can be toggled or displayed concurrently. In the concurrent display mode, hours will be dimmed. Both approaches will be used since the user may not be able to tell the exact time due to overlapping of LEDs.

Examples of Time (Concurrent time mode, Discrete Time Mode displays fully lit lights):

20:43

LED 4 is dimmed (18) + LED 7 is dimmed (2) : LED 3 is lit (30) + LED 8, 6, 5 are lit (13)

2:10

LED 1 is dimmed (0) + LED 7 is dimmed (2) : LED1 is lit (0) + LED 5, 7 are lit (10)

Assumptions

We will use the following set of assumptions:

- Working STM32f30x board will be provided
- The arm-none-eabi compiler and debugger will be provided
- Appropriate GPIO Libraries and STM32f30x Libraries will be provided
- The customer, Bryce Himebaugh, will be open for consultation and clarification during normal business hours

- Peripherals, batteries, quartz clock and other necessary hardware not listed here will be provided
- The user is able to read basic binary format 0-f 0-1111 to read the display on the clock
- The user is able to hear the buzzer on the alarm clock
- The user will not expose the clock to hazardous environments or moisture
- The clock will not endure any *unreasonable* physical force on it
- The ARM board will provide a working serial port communicating at 9600 baud
- Customer is capable of using a Command Line Interface (CLI) to set the clock
- Customer is able to operate basic commands in a Linux environment to operate the clock
- Customer has Python installed
- The device will have continuous power via battery or USB
- The device's 9v battery will last no more than 6 months of continuous operation

Weight and size will be approximately a quarter pound and approximately the dimensions of the STM board. Wires will protrude from the board and it will be up to the customer to provide the casing for the board if desired.

There will be a cost that is no greater than the components of the board. The cost is based on market price of the STM32f30x board (basic model), STM Real Time Clock and Crystal, 1 Button, Less than 10 Male to Male wires. 1 9v battery, and 1 Universal Asynchronous Receiver and Transmitter (UART) plugin. There is no cost for the programmers due to lack of expertise in the field. Credit in the form of grades should be provided.

There will be a relatively small total development time of 1 week. The main development time of the drivers is estimated at 2 days. The application should be developed in two days. The serial connection script written in either bash or Python will be estimated at less than one day. Testing will last approximately 1 day. The customer will have the opportunity to test the application and clock on the last day of testing.

Objectives / Deliverables

The following Deliverables will be set:

Objective	Deliverable
Battery Life	Minimum battery life of 6 month based on usage assumptions.
Battery Availability	Design uses commercial 9v battery, available at any retail store selling batteries
Alarm Set	Alarm is able to be set via the user buttons
Clock Visibility	Clock is able to be viewed in two modes, concurrent and discrete using 8 LEDs
Clock Reset	Clock will able to be set and reset programmatically from a remote workstation tethered via USB
Clock Set	Clock will able to be set by user
Code Visibility	Project will have limited open source with the customer, closed source to the public

Setting the alarm

The users will be using two buttons to set the time for the alarm. One will be the user button and one will be the extra button that we add on breadboard. User can switch from the time display mode to set hour mode by holding down the blue user button, it will goes into set hour mode and all led will go off, because we are incrementing it from zero. Press one more time then we will go from set hour mode to set minute mode; The user can keep switching from set hour and set minute by pressing the user button. If the user holding down the user button again, then the alarm will be saved and we will go back to time display mode. By setting the hours and minutes, user will be using the extra button on the breadboard and each time the user press it will increment the unit (hour/minute) by one.