Lab 3 – Alarm Clock
Preparation
Stefan Bordovsky and Kevin Yee

Requirements document

1. Overview

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

The objectives of this project are to design, build and test an alarm clock. Educationally, students are learning how to design and test modular software and how to perform switch/keypad input in the background.

1.2. Process: How will the project be developed?

The project will be developed using the TM4C123 board. There will be switches or a keypad. The system will be built on a solderless breadboard and run on the usual USB power. The system may use the on board switches and/or the on board LEDs. Alternatively, the system may include external switches. The speaker will be external. There will be at least 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?

EE445L students are the engineers and the TA is the client. Students are expected to modify this document to clarify exactly what they plan to build. Students are allowed to divide responsibilities of the project however they wish, but, at the time of demonstration, both students are expected to understand all aspects of the design.

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

The system will use the TM4C123 board, a ST7735 color LCD, a solderless breadboard, and be powered using the USB cable.

1.5. Terminology: Define terms used in the document.

Power budget, device driver, critical section, latency, time jitter, and modular programming. See textbook for definitions.

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 the team to keep its EE445L lab solutions secure.

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.

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.

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.

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 debounced, 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.

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?

A lab report described below is due by the due date listed in the syllabus. This report includes the final requirements document.

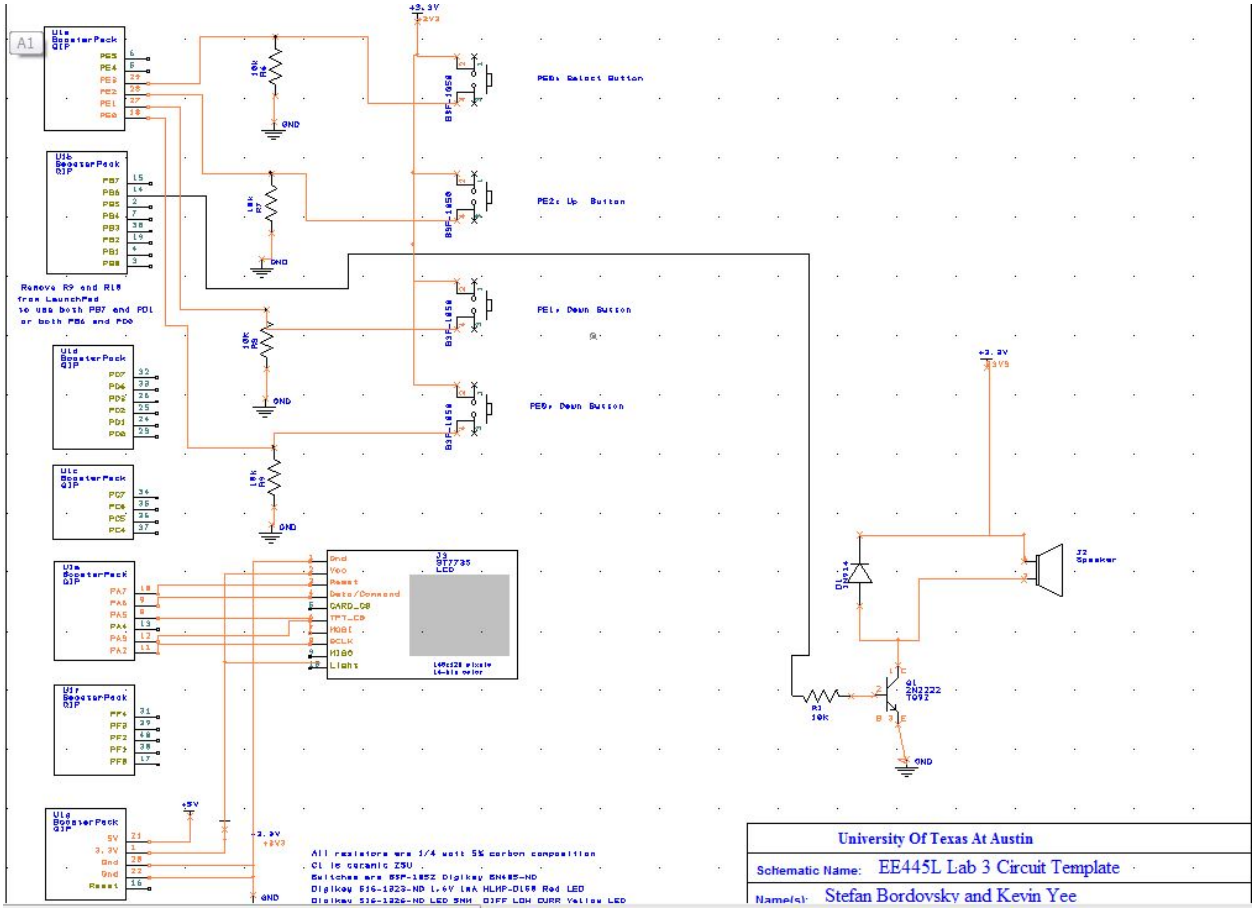
3.2. Audits: How will the clients evaluate progress?

The preparation is due at the beginning of the lab period on the date listed in the syllabus.

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

There are three deliverables: preparation, demonstration, and report.

Hardware Design



PF4: Speaker, PE0: Down Key, PE1: Up Key, PE2: Select Key, PE3: Menu Key

Starter Files:
FIFO.h

```
// FIFO.h
// Runs on any LM3Sxxx
// Provide functions that initialize a FIFO, put data in, get data out,
// and return the current size.  The file includes a transmit FIFO
// using index implementation and a receive FIFO using pointer
// implementation.  Other index or pointer implementation FIFOs can be
// created using the macros supplied at the end of the file.
// Daniel Valvano
// May 2, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
      Programs 3.7, 3.8., 3.9 and 3.10 in Section 3.7
```

```
#ifndef __FIFO_H__
#define __FIFO_H__

long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value

// Two-index implementation of the transmit FIFO
// can hold 0 to TXFIFOSIZE elements
#define TXFIFOSIZE 16 // must be a power of 2
#define TXFIFOSUCCESS 1
#define TXFIFOFAIL    0
typedef char txDataType;

// initialize index FIFO
void TxFifo_Init(void);
// add element to end of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Put(txDataType data);
// remove element from front of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Get(txDataType *datapt);
// number of elements in index FIFO
// 0 to TXFIFOSIZE-1
uint32_t TxFifo_Size(void);

// Two-pointer implementation of the receive FIFO
// can hold 0 to RXFIFOSIZE-1 elements
```

```
#define RXFIFOSIZE 16 // can be any size
#define RXFIFOSUCCESS 1
#define RXFIFOFAIL    0

typedef char rxDataType;

// initialize pointer FIFO
void RxFifo_Init(void);
// add element to end of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Put(rxDataType data);
// remove element from front of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Get(rxDataType *datapt);
// number of elements in pointer FIFO
// 0 to RXFIFOSIZE-1
uint32_t RxFifo_Size(void);

// macro to create an index FIFO
#define AddIndexFifo(NAME,SIZE,TYPE,SUCCESS,FAIL) \
uint32_t volatile NAME ## PutI;    \
uint32_t volatile NAME ## GetI;    \
TYPE static NAME ## Fifo [SIZE];        \
void NAME ## Fifo_Init(void){ long sr;  \
  sr = StartCritical();                 \
  NAME ## PutI = NAME ## GetI = 0;      \
  EndCritical(sr);                      \
}                                       \
int NAME ## Fifo_Put (TYPE data){       \
  if(( NAME ## PutI - NAME ## GetI ) & ~(SIZE-1)){  \
    return(FAIL);      \
  }                     \
  NAME ## Fifo[ NAME ## PutI &(SIZE-1)] = data; \
  NAME ## PutI ## ++;  \
  return(SUCCESS);     \
}                       \
int NAME ## Fifo_Get (TYPE *datapt){  \
  if( NAME ## PutI == NAME ## GetI ){ \
    return(FAIL);      \
  }                     \
  *datapt = NAME ## Fifo[ NAME ## GetI &(SIZE-1)];  \
  NAME ## GetI ## ++;  \
  return(SUCCESS);     \
```

```
}                   \
unsigned short NAME ## Fifo_Size (void){  \
 return ((uint32_t)( NAME ## PutI - NAME ## GetI ));  \
}
// e.g.,
// AddIndexFifo(Tx,32,unsigned char, 1,0)
// SIZE must be a power of two
// creates TxFifo_Init() TxFifo_Get() and TxFifo_Put()

// macro to create a pointer FIFO
#define AddPointerFifo(NAME,SIZE,TYPE,SUCCESS,FAIL) \
TYPE volatile *NAME ## PutPt;    \
TYPE volatile *NAME ## GetPt;    \
TYPE static NAME ## Fifo [SIZE];        \
void NAME ## Fifo_Init(void){ long sr;  \
  sr = StartCritical();                 \
  NAME ## PutPt = NAME ## GetPt = &NAME ## Fifo[0]; \
  EndCritical(sr);                      \
}                                 \
int NAME ## Fifo_Put (TYPE data){       \
  TYPE volatile *nextPutPt;             \
  nextPutPt = NAME ## PutPt + 1;        \
  if(nextPutPt == &NAME ## Fifo[SIZE]){ \
    nextPutPt = &NAME ## Fifo[0];       \
  }                                     \
  if(nextPutPt == NAME ## GetPt ){      \
    return(FAIL);                       \
  }                                     \
  else{                                   \
    *( NAME ## PutPt ) = data;          \
    NAME ## PutPt = nextPutPt;          \
    return(SUCCESS);                    \
  }                                     \
}                                       \
int NAME ## Fifo_Get (TYPE *datapt){    \
  if( NAME ## PutPt == NAME ## GetPt ){ \
    return(FAIL);                       \
  }                                     \
  *datapt = *( NAME ## GetPt ## ++);    \
  if( NAME ## GetPt == &NAME ## Fifo[SIZE]){ \
    NAME ## GetPt = &NAME ## Fifo[0];   \
  }                                     \
  return(SUCCESS);                      \
```

```
}                                   \
unsigned short NAME ## Fifo_Size (void){\
  if( NAME ## PutPt < NAME ## GetPt ){  \
    return ((uint32_t)( NAME ## PutPt - NAME ## GetPt + (SIZE*sizeof(TYPE)))/sizeof(TYPE)); \
  }                                 \
  return ((uint32_t)( NAME ## PutPt - NAME ## GetPt )/sizeof(TYPE)); \
}
// e.g.,
// AddPointerFifo(Rx,32,unsigned char, 1,0)
// SIZE can be any size
// creates RxFifo_Init() RxFifo_Get() and RxFifo_Put()

#endif //  __FIFO_H__
```

Switch.h

```
// Switch.h
// This software configures the off-board Switch keys
// Runs on LM4F120 or TM4C123

// Lab number: 6
// Hardware connections

// Header files contain the prototypes for public functions
// this file explains what the module does

// **************Switch_Init**********************
// Initialize Switch key inputs, called once
// Input: none
// Output: none
void Switch_Init(void);

// **************Switch_In**********************
// Input from Switch key inputs
// Input: none
// Output: 0 to 7 depending on keys
// 0x01 is just Key0, 0x02 is just Key1, 0x04 is just Key2
uint32_t Switch_In(void);

void Timer2Arm(void);

void init_switchmain(void);
```

Clock.h

```
#include <stdio.h>

void format_setTime(char*,int);

void format_Time(char* timeStringBuffer);

void format_And_Output_Time(char* timeStringBuffer, uint8_t* minute_hand, uint8_t*
hour_hand);
```

LCD.h

```
// LCD.h
// Runs on LM4F120/TM4C123
// A software function to drive graphics for Lab 3 Alarm Clock.
// Stefan Bordovsky and Kevin Yee
// February 7, 2017

#include <stdint.h>
#include "PLL.h"
#include "../inc/tm4c123gh6pm.h"
#include "ST7735.h"

void ST7735_XYplotInit1(int32_t minX, int32_t maxX, int32_t minY, int32_t maxY);

/**************ST7735_XYplot***************
 Plot an array of (x,y) data
 Inputs:  num    number of data points in the two arrays
          bufX   array of 32-bit fixed-point data, resolution= 0.001
          bufY   array of 32-bit fixed-point data, resolution= 0.001
 Outputs: none
 assumes ST7735_XYplotInit has been previously called
 neglect any points outside the minX maxY minY maxY bounds
*/
void ST7735_XYplot1(uint32_t num, int32_t bufX[], int32_t bufY[]);

/**************ST7735_Translate***************
 Translate an array of (x,y) data from fixed-point to LCD-mapped values.
 Inputs:  num    number of data points in the two arrays
          bufX   array of 32-bit fixed-point data, resolution= 0.001
          bufY   array of 32-bit fixed-point data, resolution= 0.001
 Outputs: none
 assumes ST7735_XYplotInit has been previously called
 neglect any points outside the minX maxY minY maxY bounds
```

```
*/
void ST7735_Translate1(uint32_t num, int32_t bufX[], int32_t bufY[]);

//************* ST7735_Line*****************************************
//  Draws one line on the ST7735 color LCD using Bresenham's Line Algorithm.
//  Inputs: (x1,y1) is the start point
//          (x2,y2) is the end point
// x1,x2 are horizontal positions, columns from the left edge
//              must be less than 128
//              0 is on the left, 126 is near the right
// y1,y2 are vertical positions, rows from the top edge
//              must be less than 160
//              159 is near the wires, 0 is the side opposite the wires
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
// Note: This is an implementation of Bresenham's algorithm inspired by an example
//                    from rosettacode.org
void ST7735_Line1(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);

// Find an index into the minute hand circle buffer for the current minute hand.
uint16_t get_Minute_Hand_Index(uint8_t minutes);

// Find an index into the hour hand circle buffer for current hour hand.
uint16_t get_Hour_Hand_Index(uint8_t minutes, uint8_t hours);

// Write out digital time.
void draw_Time(void);

// Draw clock face.
void draw_Clock(void);

// Draw clock hands.
void draw_Hands(uint8_t minutes, uint8_t hours);

// An edgy clock animation for future use.
void animate_Clock(void);

// A local delay function for clock animation.
void DelayWait1Millisecond(uint32_t n);

// SysTick.h
// Runs on LM4F120/TM4C123
// Provide functions that initialize the SysTick module, wait at least a
```

```
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// LM4F120 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  If you are using this module, you may need more
// precise timing, so it is assumed that you are using the PLL to set
// the system clock to 50 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// September 11, 2013

/* This example accompanies the books
   "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2014
   Volume 1, Program 4.7

   "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.11, Section 2.6

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

 #include <stdint.h>

extern volatile uint32_t Time;
// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(uint32_t period);

// Set current time on clock.
```

```
void SysTick_Set_Time(uint8_t hours, uint8_t minutes, uint8_t seconds, uint8_t meridian);

// Increment current time by one dSecond.
void incrementTime(void);

// Increment time every time handler is called.
void SysTick_Handler(void);

/*
// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 20 nsec for 50 MHz clock)
void SysTick_Wait(uint32_t delay);

// Time delay using busy wait.
// This assumes 50 MHz system clock.
void SysTick_Wait10ms(uint32_t delay);
*/
```