ECE 263
Spring 2018

Lab 5: Digital Alarm Clock

Date Submitted: 1 May 2018

Lab Section: 01
Lab Group: 3
We, the undersigned, certify that we contributed to the generation of this report and attest to the validity of the data herein:

Jacob Aubertine
Michael Benker

Table of Contents

Abstract

The objective of this lab is to create a working digital alarm clock utilizing the Real Time Clock (DS3231) Breakout Board, a 4x4 Button Matrix Breakout Board, a 7-Segment Display Breakout Board (MAX7221), a potentiometer and the ATmega328P Xplained Mini Board. This lab implements and wraps up content relevant to the entire semester while introducing serial peripheral interface (SPI) and two-wire interface (TWI).

Introduction

This lab will implement knowledge of timers, digital I/O, waveform generation, analog to digital converter and serial communication to create a digital alarm clock. This alarm clock will set and save a user-selected alarm time, count and detect the real time and sound an alarm using a buzzer when the alarm time is met. The 4x4 button board serves as a user interface. The Digital Alarm Clock User Manual further explains the details of user interaction with the Digital Alarm Clock.

Methods

Materials:

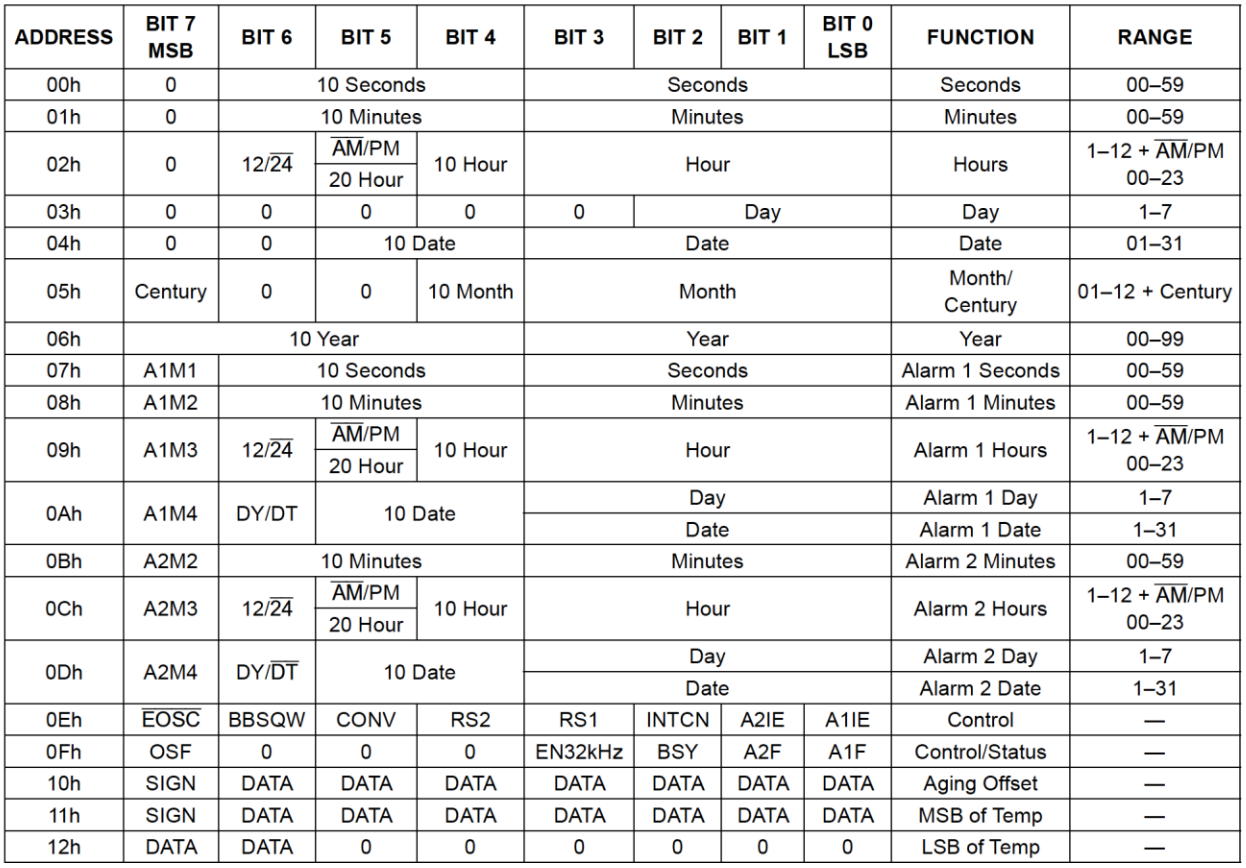
- Atmel Studio 7.0 software
- ATmega328P Xplained Mini board
- Analog Discovery
- Waveforms software
- Breadboard
- Piezo buzzer breakout board
- 4x4 button matrix breakout board
- 7 segment display breakout board with MAX7221
- Real Time Clock (DS3231) breakout board
- Potentiometer

Since this lab served as a final project, it combined topics covered previously in other labs with newer topics from lectures. These newer topics included SPI (serial peripheral interface) and TWI (two-wire interface, a variant of I^2C, or inter-integrated circuit.). SPI was used for the MAX7221 peripheral, while TWI was used for the DS3231. For simplicity and modularization, libraries were created containing functions for each interfacing method and peripheral. Once these libraries were built, they were included in the main program and the functions could be called. This avoided having a main file with dozens of functions that take up many lines of code.

The first library worked on was for the TWI. These functions only dealt with the TWI within the AVR, not the DS3231 peripheral, and are listed in the **Appendix**. The TWI_Init functions initialize the TWI at certain frequencies, using no prescaler and setting the enable bit. Since the DS3231 used a 400 kHz I^2C interface, the 400 kHz version of the function was called in the main program. TWI_Start sent the start condition for the transmission, and TWI_Stop sent the stop condition. TWI_SendData sent a byte of data by writing it to the TWI data register and then setting the TWI enable bit. TWI_ReceiveData receives data from the data register, but additional logic was needed. If there was an acknowledge, the enable acknowledge bit in the control register was written high, but if there was not then this bit was left alone. The arguments included a pointer to the variable storing the received data, and an acknowledge bit. Since the data and status register both had to be returned, the pointer was used for the data and the status register was returned normally. TWI_SLA sent the slave address, along with a read or write bit. It used the TWI_SendData function to transmit the address of the slave, with the LSB a 1 if reading or 0 if writing. There were also two other functions for multiple exchanges, TWI_ReceiveMulti and TWI_SendMulti, but they were unused in the main program. They acted similar to the single byte versions, except the data was held in an array.

Next, the DS3231 library was created, holding functions that used the previous TWI functions. To communicate with the peripheral, all of the functions began with sending a start condition and slave address, and ended with sending a stop condition. The slave address of the DS3231 was 1101000, which was left aligned, becoming D0 in hex. Before sending data, the address (shown in **Table 1**) of the DS3231 register was sent, indicate which one should be pointed to. The SetControlRegister function sent values to be written to the clock's control register. By setting the INTCN bit an external pin of the DS3231 would change when an alarm went off. GetStatus received the value of the clock's status register, but only the two alarm flags were needed. The value of the flag for alarm 2 specifically was returned in the Alarm2Flag function, which calls GetStatus. The functions that 'set' registers (SetSeconds, SetMinutes, etc.) all had an argument that was the value to be written to that register. SetDateTime was slightly different, as it took arguments for the minutes, hours, date, and month and set all of them. These were the four values displayed on the 7 segment displays. The 'get' functions were the same, except they returned the values of the registers instead of setting them. SetAlarm2 was slightly more complex, as bit 7 the alarm 2 minute and hour registers was for the mask bits, determining when to trigger the alarm. Bit 6 of the hour register was a 0 to use it as a 24 hour clock. Alarm2OnOff was contained additional logic, as this function was created after starting the main program. First, the data contained in the control register is received, because just writing values to it would overwrite the other important data it held. Based on the argument , the control register data was combined with a 1 or a 0 to either turn on or off the alarm 2 interrupt, essentially turning on or off the alarm.

Table 1: DS3231 Timekeeping Registers



ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{\text{AM}}$/PM	10 Hour	Hour				Hours	1–12 + $\overline{\text{AM}}$/PM
			20 Hour							00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{\text{AM}}$/PM	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{\text{AM}}$/PM
			20 Hour							00–23
0Ah	A1M4	DY/DT	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{\text{AM}}$/PM	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{\text{AM}}$/PM
			20 Hour							00–23
0Dh	A2M4	DY/$\overline{\text{DT}}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{\text{EOSC}}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

The SPI library contained far fewer functions than the TWI library. SPI_Init initialized SPI within the AVR by setting the SPI control register and a bit in the status register. This enabled SPI, had a %2 prescaler, used mode 00 for polarity and phase (meaning data was sampled on the leading edge of the SCK signal), and had default data order. Port B was also configured in this function by setting the MOSI, SCK, and SS (slave-select) pins as outputs, and having the SS pin output a 1. SPI_SetCS acted as the chip select, outputting either a 0 or 1 on the SS pin. Since it was active low, a 0 was output to begin a transmission, and a 1 was output to end it. SPI_Send puts whatever 8 bit value was in the argument into the SPI data register, beginning a transmission. It then waits until the interrupt flag is set, indicating the transmission was completed. Finally, the new contents of the data register are returned because an SPI transmission swaps data.

Table 2: MAX7221 Register Address Map

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

The MAX7221 library included the SPI library. All of the functions for the peripheral began with bringing the chip select low, then sending the value of the command, then sending the data, then bringing the chip select back high and returning the new contents of the data register. The commands are shown in **Table 2**. The DecodeMode function made it so that all 8 digits used decode mode. This means that sending a single digit number with a digit selected would display that number. ScanLimit sets how many digits can be displayed out of the 8 on the board. The command for displaying a certain number of digits was one less than the number of digits, meaning the command 0xX5 turned on 6 digits. The argument for the function was the number of desired digits to display, and so that number was decremented by 1 for the command. The Intensity function controlled how bright the display was by outputting a value between 0 and F, representing dimmest to brightest respectively. SetDigit was one of the more important functions, as it was how any of the information was able to be displayed. The first argument indicated the digit that was to be set, while the second was the number it was to display. **Figure 1** shows the number of the digits and their order on the board. SetAllDigits called SetDigit eight times within the function, but took an array of eight values as an input. The order of the array values and the order in which they were turned on were the same as the digits in **Figure 1** from left to right. If the values to be displayed were all known, then it was simpler to call this function.

Finally, the OnOff function turned on or off the MAX7221, with a 1 meaning normal operation and a 0 meaning shutdown mode.

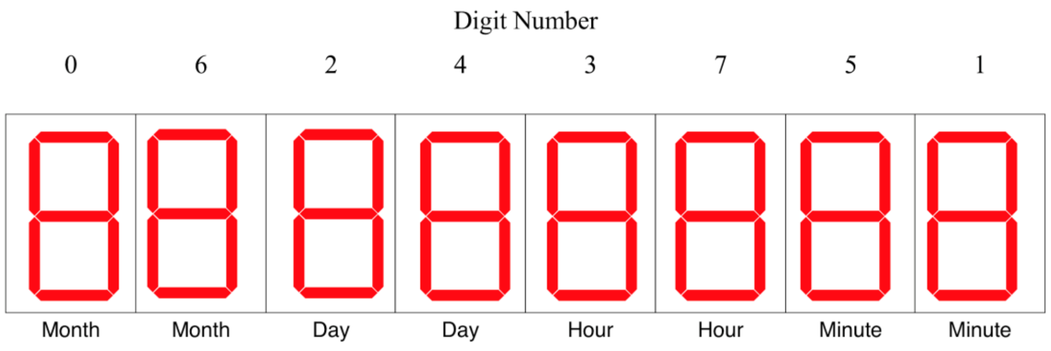


Figure 1: Seven segment digit usage

In the main functions, all of these libraries were brought together. The same button configuration was used from the previous Digital Music Keyboard lab, with a structure keeping track of the row and column of the button pressed. ButtonSetup set bits 0-3 of Port C as inputs, and bits 0-3 of Port D as outputs, corresponding to the columns and rows respectively (**Figure 2**). SPI and TWI were initialized, with TWI being at 400 kHz. The MAX7221 was brought out of shutdown mode, and its intensity was set for max brightness. Decoding was enabled for all digits, and the scan limit was set to display all 8 digits. A variable *alarm* is declared and initialized to 0. This kept track of the status of if the alarm was set or not; a 1 meant set, and a 0 meant not set.

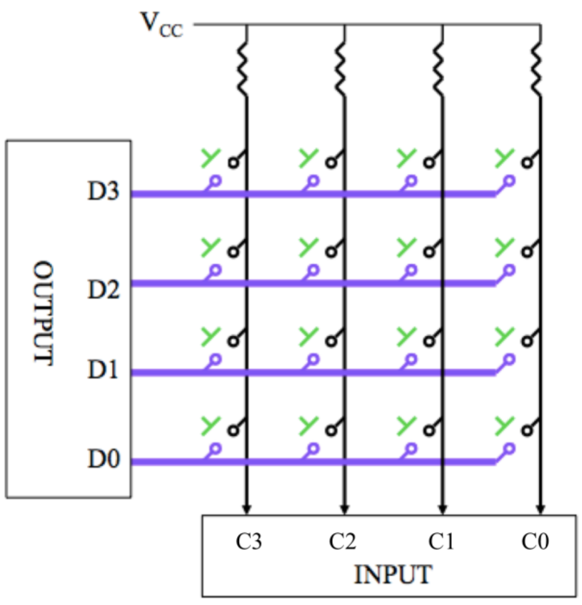


Figure 2: Circuit diagram for 4x4 button matrix

The rest of the program was contained within a while loop, meaning it ran continuously. The ClockToDisplay function was the first function called. It received the time from the real time clock and then displayed it on the 7 segment displays. Additional logic was present for the hours because the data was stored in BCD format in the clock. If the '20' bit was high in the clock, the tens digit for the hour would display a 2, and same with the '10' bit. Next, it the alarm 2 flag was checked to see if the alarm was going off. If it was, the buzzer would play a tone and continue to until any button was pressed. When a button was pressed, the buzzer would turn off and the alarm 2 interrupt would be set to 0 within the DS3231, turning the alarm off. Then, the ButtonPressed function was called to determine which, if any, button was pressed. When a number was needed from the keypad, the ButtomToNum function was called. If the button pressed was any of the 10 that represented numbers, it would return that number. The only buttons that would have an effect in the while loop initially would be Set Date/Time, Alarm Set, and Alarm On/Off (the button layout is shown in **Figure 3**).

7	8	9	ALRM SET
4	5	6	ALRM ON/OFF
1	2	3	SET DATE/ TIME
BACK	0	USR 2	USR 3

Figure 3: Keypad Layout

ButtonSetDateTime wrote values to the timekeeping registers of the DS3231. First, the displays of all of the digits were blanked. This allowed the user to see which button they had just pressed (as it would be displayed), and which values were still needed. The values were taken in left to right order according to the display, asking for the tens place and then the ones place of months, days, hours, and minutes. There were measures in place to not send a value greater than the maximum for the DS3231 register If, for example, the user attempted to enter in a 5 for the tens place of months, the code would change it to the maximum real value of 1 (since the 12th month is the last). This digit would still be displayed, but it would not be written to the clock, and when the ClockToDisplay function was called again, the correct time from the clock would be shown. ButtonAlarmSet acted nearly the same as the previous function, except the user only entered in the hours and minutes and those were written to the alarm 2 register. Finally, if the Alarm

On/Off button was pressed, it would toggle the status of the alarm using the value stored in the *alarm* variable.

Experimental Results

The final lab results demonstrated some level of operation of the code with respect to the MAX7221 digital display, the button board and the DS3231, however the desired results were not met.

Two different MAX7221 Digit Display Boards were tested using the code, both resulting in varying behavior. The first MAX7221 board displayed two different light intensities at the same time, often displaying random lines on only the least significant digit, the one's place on the minute display. The first board was lit long enough to witness a reaction to inputs on the button board. The second MAX7221 displayed a uniform lighting of all digits, although the display was highly sensitive to touch and likely had a damaged connection.

Waveforms Analog Discovery was used to ensure that data was being sent to the MAX7221 and DS3231. **Figure 4** below shows the SPI signals to the MAX7221 and TWI signals to the DS3231.

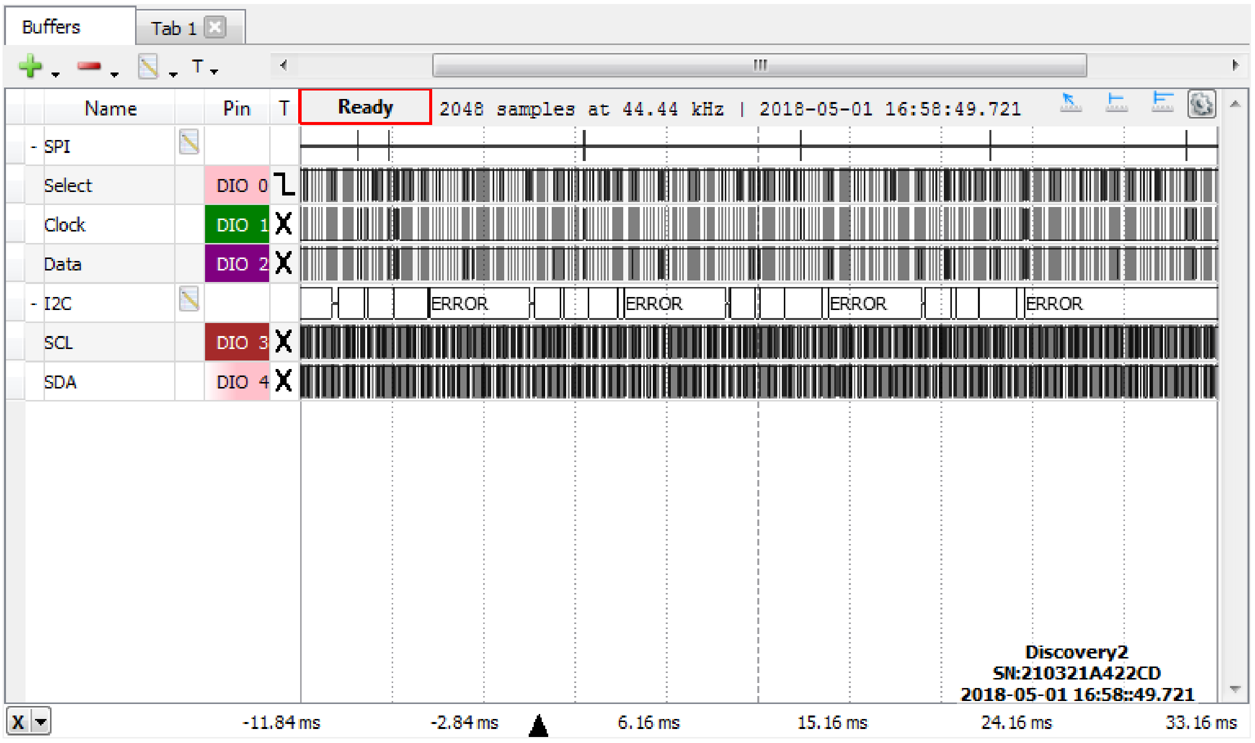


Figure 4: Waveforms data for SPI and I2C (TWI)

Discussion

Issues were encountered when using TWI, specifically when sending the start condition and waiting for the flag to know the transmission was complete. Sometimes, this flag would not be cleared, meaning that it would get stuck in an infinite loop waiting for it to be set again. Even though this code was from the datasheet, the waiting was eventually commented out and a short delay was added instead. This allowed the program to actually run, although the delay was likely more than was needed.

The potentiometer and ADC were not used in during the final test, as it was simpler to just set the display to be its brightest. When stepping through the code, sometimes it would skip over necessary parts. This was difficult to debug, as there were no compilation errors and it appeared as if the code was properly written. The button board is expected to have functioned properly, as the code used for the board was tested in the previous lab.

The Waveforms data does not appear to be displaying correct outputs, especially because I2C says 'error.' All of the signals switch from high to low rapidly, with no discernable packets. For SPI, the chip select should have gone low, the data line should have sent data, and then the chip select line would go back high. Instead, they all oscillate. This did display something on the MAX7221, but the digits displayed did not match what should have been displaying.

Conclusion

The Digital Alarm Clock did not function as expected. Despite numerous attempts at debugging the code, the final result did not leave the user with an operating digital alarm clock as outlined in the User Manual. Although the lab results were not met, the methods taken suggest that a finished, functioning digital alarm clock may not be entirely out of reach, were further steps to be taken to debug the issues related to interaction with the MAX7221. The essential components of the Digital Alarm Clock, namely the DS3231, the MAX7221, the button board and of course the ATmega328P had demonstrated some level of interaction and bits and pieces of an expected result. Furthermore, despite the digital alarm clock not achieving acceptable levels of operation, numerous detailed issues related to aesthetic and comfortable use of the digital alarm clock are addressed in the code. For instance, the digital alarm clock addresses the topical issue of whether or not the user should be able to see the digits entered while inputting the date and time. In conclusion, while the lab implemented some level of each of the elements required for proper function of the lab including SPI and TWI interfaces, the core objective of constructing a digital alarm clock was not met.

Reflection

In contrast to all previous labs, despite even more efforts made in debugging the code to achieve the desired function, we were unable to get the code working as desired. In the coding stages, we had discussed a number of finer points that we wanted to address with the user interface, but given time constraints, we had focused efforts on debugging the program so that the basic elements of the digital alarm clock could work, such as displaying the correct digits. The MAX7221s were not easy to come by and the two available worked differently with our program. All components are needed in order to ensure that the code is working correctly. It would have been helpful if the essential equipment was more available outside of lab time. We were satisfied with the program, even though it ultimately did not work as expected. With more time we would have likely made more progress, but most of it would probably be debugging the physical components.

Appendix: Source code

Below is the source code used in the lab, with custom libraries containing necessary functions afterwards. The .h files were not included as they just contain prototypes for the functions.

Main file:

```
/*
 * ece263_lab5_v1.c
 *
 * Created: 4/22/2018 3:04:17 PM
 * Authors : Jacob Aubertine, Michael Benker
 */

#include <avr/io.h>
#include <util/delay.h>
#include "DS3231.h"
#include "MAX7221.h"
#include "twi.h"

struct whichButton{
	uint8_t column;
	uint8_t row;
};

void ButtonSetup();
void ButtonPressed(struct whichButton *ptr);
void Timer0Setup();
void ButtonPressed(struct whichButton *ptr);
void ButtonToNum(struct whichButton *ptr, uint8_t *buttonnum);
void ButtonSetDateTime(struct whichButton *ptr);
void ClockToDisplay();
void ButtonAlarmSet(struct whichButton *ptr);

int main(void)
{
	ButtonSetup();
```

```
struct whichButton button;
struct whichButton *ptrButton = &button;

SPI_Init();
TWI_Init400kHz();

DS3231_SetControlRegister();

MAX7221_OnOff(1);           // ensures MAX7221 not in shutdown mode
MAX7221_Intensity(0x0F);   // brightest
MAX7221_DecodeMode();       // enables decoding for all digits
MAX7221_ScanLimit(8);       // display all 8 digits

uint8_t alarm = 0;

while (1)
{
       // display date and time
       ClockToDisplay();

       // if the alarm is going off
       if ( DS3231_Alarm2Flag() ){
              TCCR0B |= (1 << CS20);
              TCCR0B &= ~(0b11 << CS00);
              while((button.column == 0) && (button.row == 0))
                     ButtonPressed(ptrButton);
              TCCR0B &= ~(0b111 << CS00);
              DS3231_Alarm2OnOff(0);
              alarm = 0;
       }

       // while ( (button.column == 0) && (button.row == 0) )
       ButtonPressed(ptrButton);

       // set date/time (user 1)
       if ( button.column == 4 && button.row == 3 ){
              ButtonSetDateTime(ptrButton);
       }

       // alarm set button
       else if ( button.column == 4 && button.row == 1 ){
              ButtonAlarmSet(ptrButton);
       }

       // alarm on/off
       else if ( button.column == 4 && button.row == 2 ){
              alarm ^= 1;
              DS3231_Alarm2OnOff(alarm);
       }
}
```

```
}

void ButtonSetup(){
       DDRC &= 0xF0; // sets bits 0-3 of Port C as inputs
       DDRD |= 0x0F; // sets bits 0-3 of Port D as outputs
       PORTC |= 0x0F;       // enables pull-up resistors
       PORTD &= 0xF0;       // sets bits 0-3 of Port D to 0 to detect button press
}

void Timer0Setup(){
       TCCR0A = (0b01 << COM0A0) | (0b10 << WGM00);     // toggles on compare match, CTC
mode
       TCCR0B = (0 << WGM02) | (0b000 << CS00);         // no prescaler, timer0 stopped
initially
       DDRD |= (1 << PORTD6);      // sets OC0A as output (PD6)
       OCR0A = 70;
}

void ButtonPressed(struct whichButton *ptr){
       PORTD &= 0xF0;       // sets bits 0-3 of Port D to 0 to detect button press
       if( (PINC & (1 << PINC3)) == 0 ){                // column 1
                     ptr->column = 1;
              PORTD = ~(1 << PORTD3);
              if( (PINC & (1 << PINC3)) == 0 )                 // row 1
                     ptr->row = 1;
              PORTD = ~(1 << PORTD2);
              if( (PINC & (1 << PINC3)) == 0 )                 // row 2
                     ptr->row = 2;
              PORTD = ~(1 << PORTD1);
              if( (PINC & (1 << PINC3)) == 0 )                 // row 3
                     ptr->row = 3;
              PORTD = ~(1 << PORTD0);
              if( (PINC & (1 << PINC3)) == 0 )                 // row 4
                     ptr->row = 4;
       }
       else if( (PINC & (1 << PINC2)) == 0 ){          // column 2
                     ptr->column = 2;
              PORTD = ~(1 << PORTD3);
              if( (PINC & (1 << PINC2)) == 0 )                 // row 1
                     ptr->row = 1;
              PORTD = ~(1 << PORTD2);
              if( (PINC & (1 << PINC2)) == 0 )                 // row 2
                     ptr->row = 2;
              PORTD = ~(1 << PORTD1);
              if( (PINC & (1 << PINC2)) == 0 )                 // row 3
                     ptr->row = 3;
              PORTD = ~(1 << PORTD0);
              if( (PINC & (1 << PINC2)) == 0 )                 // row 4
                     ptr->row = 4;
       }
```

```
	else if( (PINC & (1 << PINC1)) == 0 ){		// column 3
			ptr->column = 3;
		PORTD = ~(1 << PORTD3);
		if( (PINC & (1 << PINC1)) == 0 )		// row 1
			ptr->row = 1;
		PORTD = ~(1 << PORTD2);
		if( (PINC & (1 << PINC1)) == 0 )		// row 2
			ptr->row = 2;
		PORTD = ~(1 << PORTD1);
		if( (PINC & (1 << PINC1)) == 0 )		// row 3
			ptr->row = 3;
		PORTD = ~(1 << PORTD0);
		if( (PINC & (1 << PINC1)) == 0 )		// row 4
			ptr->row = 4;
	}
	else if( (PINC & (1 << PINC0)) == 0 ){		// column 4
			ptr->column = 4;
		PORTD = ~(1 << PORTD3);
		if( (PINC & (1 << PINC0)) == 0 )		// row 1
			ptr->row = 1;
		PORTD = ~(1 << PORTD2);
		if( (PINC & (1 << PINC0)) == 0 )		// row 2
			ptr->row = 2;
		PORTD = ~(1 << PORTD1);
		if( (PINC & (1 << PINC0)) == 0 )		// row 3
			ptr->row = 3;
		PORTD = ~(1 << PORTD0);
		if( (PINC & (1 << PINC0)) == 0 )		// row 4
			ptr->row = 4;
	}
	else{
		ptr->column = 0;
		ptr->row = 0;
	}

	_delay_ms(20);
}

void ButtonToNum(struct whichButton *ptr, uint8_t *buttonnum){
	if ( ptr->column == 2 && ptr->row == 4 )
		*buttonnum = 0;
	else if ( ptr->column == 1 && ptr->row == 3 )
		*buttonnum = 1;
	else if ( ptr->column == 2 && ptr->row == 3 )
		*buttonnum = 2;
	else if ( ptr->column == 3 && ptr->row == 3 )
		*buttonnum = 3;
	else if ( ptr->column == 1 && ptr->row == 2 )
		*buttonnum = 4;
	else if ( ptr->column == 2 && ptr->row == 2 )
```

```
              *buttonnum = 5;
       else if ( ptr->column == 3 && ptr->row == 2 )
              *buttonnum = 6;
       else if ( ptr->column == 1 && ptr->row == 1 )
              *buttonnum = 7;
       else if ( ptr->column == 2 && ptr->row == 1 )
              *buttonnum = 8;
       else if ( ptr->column == 3 && ptr->row == 1 )
              *buttonnum = 9;
}

// month, day, hour, minute
void ButtonSetDateTime(struct whichButton *ptr){
       uint8_t tens;
       uint8_t ones;
       uint8_t cleardigits[] = {0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF};
       MAX7221_SetAllDigits(cleardigits);

       // Mm dd hh mm
       while ( (ptr->column == 0) && (ptr->row == 0) )
              ButtonPressed(ptr);
       ButtonToNum(ptr, &tens);
       MAX7221_SetDigit(0, tens);
       // mM dd hh mm
       while ( (ptr->column == 0) && (ptr->row == 0) )
              ButtonPressed(ptr);
       ButtonToNum(ptr, &ones);
       MAX7221_SetDigit(6, ones);

       if (tens > 1)
              tens = 1;
       DS3231_SetMonth((tens << 4) | ones);

       // mm Dd hh mm
       while ( (ptr->column == 0) && (ptr->row == 0) )
              ButtonPressed(ptr);
       ButtonToNum(ptr, &tens);
       MAX7221_SetDigit(2, tens);
       // mm dD hh mm
       while ( (ptr->column == 0) && (ptr->row == 0) )
              ButtonPressed(ptr);
       ButtonToNum(ptr, &ones);
       MAX7221_SetDigit(4, ones);

       if (tens > 3)
              tens = 3;
       DS3231_SetDate((tens << 4) | ones);

       // mm dd Hh mm
       while ( (ptr->column == 0) && (ptr->row == 0) )
```

```
		ButtonPressed(ptr);
	ButtonToNum(ptr, &tens);
	MAX7221_SetDigit(3, tens);
	// mm dd hH mm
	while ( (ptr->column == 0) && (ptr->row == 0) )
		ButtonPressed(ptr);
	ButtonToNum(ptr, &ones);
	MAX7221_SetDigit(7, ones);

	if (tens > 2)
		tens = 0b10;
	DS3231_SetHours((tens << 4) | ones);

	// mm dd hh Mm
	while ( (ptr->column == 0) && (ptr->row == 0) )
		ButtonPressed(ptr);
	ButtonToNum(ptr, &tens);
	MAX7221_SetDigit(5, tens);
	// mm dd hh mM
	while ( (ptr->column == 0) && (ptr->row == 0) )
		ButtonPressed(ptr);
	ButtonToNum(ptr, &ones);
	MAX7221_SetDigit(1, ones);

	if (tens > 5)
		tens = 5;
	DS3231_SetMinutes((tens << 4) | ones);
}

void ClockToDisplay(){
	uint8_t minutes;
	uint8_t hours;
	uint8_t date;
	uint8_t month;

	minutes = DS3231_GetMinutes();
	MAX7221_SetDigit(1, minutes & 0x0F);
	MAX7221_SetDigit(5, minutes >> 4);

	hours = DS3231_GetHours();
	MAX7221_SetDigit(7, hours & 0x0F);
	if ( ( (hours & (1<<5) )>>5)  == 1)
		MAX7221_SetDigit(3, 2);
	else if ( ( (hours & (1<<4) )>>4)  == 1 )
		MAX7221_SetDigit(3, 1);
	else
		MAX7221_SetDigit(3, 0);

	date = DS3231_GetDate();
	MAX7221_SetDigit(4, date & 0x0F);
```

```
        MAX7221_SetDigit(2, date >> 4);

        month = DS3231_GetMonth();
        MAX7221_SetDigit(6, month & 0x0F);
        MAX7221_SetDigit(0, (month & (1<<4)) >> 4);
}

void ButtonAlarmSet(struct whichButton *ptr){
        uint8_t tens;
        uint8_t ones;
        uint8_t hours;
        uint8_t minutes;
        uint8_t cleardigits[] = {0xF,0xF,0xF,0xF,0,0,0,0};
        MAX7221_SetAllDigits(cleardigits);

        // mm dd Hh mm
        while ( (ptr->column == 0) && (ptr->row == 0) )
        ButtonPressed(ptr);
        ButtonToNum(ptr, &tens);
        MAX7221_SetDigit(3, tens);
        // mm dd hH mm
        while ( (ptr->column == 0) && (ptr->row == 0) )
        ButtonPressed(ptr);
        ButtonToNum(ptr, &ones);
        MAX7221_SetDigit(7, ones);

        if (tens > 2)
                tens = 0b10;
        hours = ((tens << 4) | ones);

        // mm dd hh Mm
        while ( (ptr->column == 0) && (ptr->row == 0) )
        ButtonPressed(ptr);
        ButtonToNum(ptr, &tens);
        MAX7221_SetDigit(5, tens);
        // mm dd hh mM
        while ( (ptr->column == 0) && (ptr->row == 0) )
        ButtonPressed(ptr);
        ButtonToNum(ptr, &ones);
        MAX7221_SetDigit(1, ones);

        if (tens > 5)
                tens = 5;
        minutes = ((tens << 4) | ones);

        DS3231_SetAlarm2(minutes, hours);
}
```

twi.c (library)

```c
#include "twi.h"

void TWI_Init100kHz(){
	TWBR = 72;    // 100 kHz @ 16 MHz
	TWSR = (0b00 << TWPS0);
	TWCR = (1 << TWEN);
	DDRC |= (1<<PORTC5)|(1<<PORTC4);
}

void TWI_Init400kHz(){
	TWBR = 12;    // 400 kHz @ 16 MHz
	TWSR = (0b00 << TWPS0);
	TWCR = (1 << TWEN);
	DDRC |= (1<<PORTC5)|(1<<PORTC4);
}

void TWI_Init1MHz(){ // UNTESTED DO NOT ATTEMPT
	TWBR = 0;     // 1 MHz @ 16 MHz
	TWSR = (0b00 << TWPS0);
	TWCR = (1 << TWEN);
	DDRC |= (1<<PORTC5)|(1<<PORTC4);
}

uint8_t TWI_Start(){
	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
	//while ( !(TWCR & (1<<TWINT)) );
	_delay_ms(1);
	return (TWSR & 0xF8);
}

void TWI_Stop(){
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
}

// slave address is left justified
// 1101000 ==> 0xD0
uint8_t TWI_SLA( uint8_t addr, uint8_t RWbar){
	return TWI_SendData( addr | RWbar );
}

uint8_t TWI_SendData( uint8_t data ){
	TWDR = data;
	TWCR = (1<<TWINT) |  (1<<TWEN);
	return (TWSR & 0xF8);
}

uint8_t TWI_ReceiveData( uint8_t *data, uint8_t ack ){
	if (ack)
		TWCR = (1<<TWINT) |  (1<<TWEN) | (1<<TWEA);
	else
```

```
                TWCR = (1<<TWINT) |  (1<<TWEN);
        *data = TWDR;
        return (TWSR & 0xF8);
}

// receive multiple bytes
void TWI_ReceiveMulti( uint8_t *data, uint8_t len ){
        // receive lots of bytes with ack, last byte nack
        uint8_t i;
        for (i = 0; i < len-1; i++){
                TWI_ReceiveData( &(data[i]), 1 );
        }
        TWI_ReceiveData( &(data[len-1]), 0 );
}

void TWI_SendMulti( uint8_t *data, uint8_t len ){
        // transmit lots of bytes
        uint8_t i;
        for (i = 0; i < len; i++){
                TWI_SendData( data[i] );
        }
}
```

DS3231.c (library)

```
#include "DS3231.h"

void DS3231_SetControlRegister(){
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 0);
        TWI_SendData(0x0E);
        TWI_SendData( 0b100 << 0 );
        TWI_Stop();
}

uint8_t DS3231_GetStatus(){
        uint8_t status;
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 0);
        TWI_SendData(0x0F);
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 1);
        TWI_ReceiveData(&status, 0);
        TWI_Stop();
        return status;
}

uint8_t DS3231_Alarm2Flag(){
        uint8_t A2F;
        A2F = DS3231_GetStatus();
```

```
	return ((A2F&(1<<1)) >> 1);
}

void DS3231_SetSeconds(uint8_t seconds){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x00);
	TWI_SendData(seconds);
	TWI_Stop();
}

void DS3231_SetMinutes(uint8_t minutes){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x01);
	TWI_SendData(minutes);
	TWI_Stop();
}

void DS3231_SetHours(uint8_t hours){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x02);
	TWI_SendData((0 << 6) | hours);
	TWI_Stop();
}

void DS3231_SetDay(uint8_t day){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x03);
	TWI_SendData(day);
	TWI_Stop();
}

void DS3231_SetDate(uint8_t date){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x04);
	TWI_SendData(date);
	TWI_Stop();
}

void DS3231_SetMonth(uint8_t month){
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x05);
	TWI_SendData(month);
	TWI_Stop();
}
```

```
void DS3231_SetDateTime(uint8_t minutes, uint8_t hours, uint8_t date, uint8_t month){
	DS3231_SetMinutes(minutes);
	DS3231_SetHours(hours);
	DS3231_SetDate(date);
	DS3231_SetMonth(month);
}

uint8_t DS3231_GetSeconds(){
	uint8_t seconds;
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x00);
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 1);
	TWI_ReceiveData(&seconds, 0);
	TWI_Stop();
	return seconds;
}

uint8_t DS3231_GetMinutes(){
	uint8_t minutes;
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x01);
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 1);
	TWI_ReceiveData(&minutes, 0);
	TWI_Stop();
	return minutes;
}

uint8_t DS3231_GetHours(){
	uint8_t hours;
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x02);
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 1);
	TWI_ReceiveData(&hours, 0);
	TWI_Stop();
	return hours;
}

uint8_t DS3231_GetDay(){
	uint8_t day;
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x03);
	TWI_Start();
```

```
        TWI_SLA(DS3231_ADDR, 1);
        TWI_ReceiveData(&day, 0);
        TWI_Stop();
        return day;
}

uint8_t DS3231_GetDate(){
        uint8_t date;
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 0);
        TWI_SendData(0x04);
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 1);
        TWI_ReceiveData(&date, 0);
        TWI_Stop();
        return date;
}

uint8_t DS3231_GetMonth(){
        uint8_t month;
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 0);
        TWI_SendData(0x05);
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 1);
        TWI_ReceiveData(&month, 0);
        TWI_Stop();
        return month;
}

uint8_t DS3231_GetYear(){
        uint8_t year;
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 0);
        TWI_SendData(0x06);
        TWI_Start();
        TWI_SLA(DS3231_ADDR, 1);
        TWI_ReceiveData(&year, 0);
        TWI_Stop();
        return year;
}

void DS3231_GetDateTime(uint8_t *minutes, uint8_t *hours, uint8_t *date, uint8_t *month){
        *minutes = DS3231_GetMinutes();
        *hours = DS3231_GetHours();
        *date = DS3231_GetDate();
        *month = DS3231_GetMonth();
}

void DS3231_SetAlarm2(uint8_t minutes, uint8_t hours){
	// A2M = 0b100, alarm when hours and minutes match
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x0B);
	TWI_SendData( (0 << 7) | minutes); // sets A2M2 to 0
	TWI_SendData( (0 << 7) | (0 << 6) | hours);
	TWI_Stop();
}

void DS3231_Alarm2OnOff( uint8_t On_OffBar){
	uint8_t CtrlReg;
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);
	TWI_SendData(0x0E);
	TWI_Start();
	TWI_SLA(DS3231_ADDR, 1);
	TWI_ReceiveData(&CtrlReg, 0);

	TWI_Start();
	TWI_SLA(DS3231_ADDR, 0);

	if (On_OffBar)
		TWI_SendData( CtrlReg |= (1 << 1) );	// sets Alarm 2 interrupt enable
	else
		TWI_SendData( CtrlReg &= ~(1 << 1) );	// clears Alarm 2 interrupt enable

	TWI_Stop();
}
```

spi.c (library)

```
#include "spi.h"

void SPI_Init(){
	// clk/2, mode = 00 (polarity/phase), default data order

	DDRB |= (1 << MOSI) | (1 << SCK) | (1 << SS);
	PORTB |= (1 << SS);
	SPCR = (1 << SPE) | (1 << MSTR) | (0b00 << SPR0) | (0b00 << CPHA);
	SPSR = (1 << SPI2X);
}

void SPI_SetCS(uint8_t val){
	if (val)
		PORTB |= (1 << SS);
	else
		PORTB &= ~(1 << SS);
}

uint8_t SPI_Send(uint8_t val){
```

```c
	SPDR = val;
	while ( !(SPSR & (1 << SPIF)) );  // wait for SPIF
	return SPDR;
}
```

MAX7221.c (library)

```c
#include "MAX7221.h"

uint8_t MAX7221_DecodeMode(){
	uint8_t data;
	SPI_SetCS(0);
	SPI_Send(0x09);
	data = SPI_Send(0xFF);              // decode for all digits
	SPI_SetCS(1);
	return data;
}

// number of digits to display
uint8_t MAX7221_ScanLimit(uint8_t digits){
	uint8_t data;
	SPI_SetCS(0);
	SPI_Send(0x0B);
	data = SPI_Send(digits - 1);
	SPI_SetCS(1);
	return data;
}

// intensity from 0 to F
uint8_t MAX7221_Intensity(uint8_t intensity){
	uint8_t data;
	SPI_SetCS(0);
	SPI_Send(0x0A);
	data = SPI_Send(intensity);
	SPI_SetCS(1);
	return data;
}

uint8_t MAX7221_SetDigit(uint8_t digit, uint8_t value){
	uint8_t data;
	SPI_SetCS(0);
	SPI_Send(digit + 1);
	data = SPI_Send(value);
	SPI_SetCS(1);
	return data;
}

void MAX7221_SetAllDigits(uint8_t *digits){
	/*
	uint8_t i;
	for (i = 0; i < 8; i++){
		MAX7221_SetDigit(i, digits[i]);
	}
	*/

	MAX7221_SetDigit(0, digits[0]);
	MAX7221_SetDigit(6, digits[1]);
	MAX7221_SetDigit(2, digits[2]);
	MAX7221_SetDigit(4, digits[3]);
	MAX7221_SetDigit(3, digits[4]);
	MAX7221_SetDigit(7, digits[5]);
	MAX7221_SetDigit(5, digits[6]);
	MAX7221_SetDigit(1, digits[7]);
}

uint8_t MAX7221_OnOff(uint8_t On_OffBar){
	uint8_t data;
	SPI_SetCS(0);
	SPI_Send(0x0C);
	data = SPI_Send(On_OffBar);
	SPI_SetCS(1);
	return data;
}
```