HC908GP32 Evaluation Board

Lab Manual

Table of Contents

LAB INTRODUCTION

This manual is designed to introduce the new student to the microcontroller in a laboratory environment. It is assumed that you have some background in microcontroller architecture and programming, although this is not a prerequisite. A good textbook on microcontrollers as well as some familiarity with assembly or C programming should provide enough background to step through the exercises contained in this manual. Most of the labs here have been originally developed using assembly language.

The HC908GP32 is part of Motorola line of 8-bit microcontrollers. Newer versions of these devices have been introduced over the past few years; however, the basic methods and skills learned in these labs will be directly applicable to a wide variety of microcontroller products designed with various memory configurations, drive ability and power usage. As your skills become more advanced, real-world issues such as design for low-power consumption, wireless and internet connected systems can be developed.

Before beginning the first labs, there is some hardware setup that will need to occur. You will need a standard 9-pin RS232 cable connected to the COM port of our PC. We also recommend using CodeWarrior Standard Edition for your software platform. CodeWarrior offers a free version (limited to 32Kbytes for C programming and unlimited assembly) of its Integrated Development Environment simulator and debugger.   The GP32 microcontroller can be programmed and debugged using this setup. See the “Prerequisites” section immediately below for the URL to download this software.

Prerequisites

Download ‘Special Edition: CodeWarrior for Microcontrollers’ here: http://www.freescale.com/webapp/sps/site/overview.jsp?code=CW_SPECIALEDITIONS&tid=CWH

This program is free of charge; however, the special edition is limited to 32K code size for C programming. For assembly language, there is no code size limit.

You will also need to following equipment for evaluation board setup:

A. RS-232 9-pin cable to connect your PC with the evaluation board.

B. Wall mount power supply with a voltage range from 6Vdc to 30Vdc. The on-board voltage regulator will regulate this voltage to the required 5Vdc needed for board operation. Insure that the jumper pin at location U81 is installed. A supply capability of 500mA should be sufficient.

C. A 4x4 keypad + ribbon cable. Insure that Pin1 from the keypad matches Pin1 as labeled on the evaluation board. Since a 2x8 cable is used, insure that the correct row of pins is connected on each end.

An LCD display + ribbon cable. Depending on your version of evaluation board, you may have either a 2x16 or 2x40 LCD display. Either will work fine for the labs. Pin connectors on the evaluation board are available for either a 1x16 LCD header or 2x8 header. Either will work depending on the type of LCD configuration used. Again, insure that Pin1 on the board lines up with Pin1 on the LCD.

Setting up the Software Environment

Once CodeWarrior SE is installed on your PC, it will need to be properly configured for use with the evaluation board. The basic software development process should take place as follows:

- Write source code using your IDE (Integrated Development Environment)
- Compile the source code into machine code
- Link this code to form an executable file
- Debug your code as needed
- Re-compile, link and develop for final executable product

In order to become familiar with the IDE environment, we will start with a simple file and go through the steps to develop a program. After completing this exercise, you should be able to beginning writing your own programs starting with the first lab.

Start CodeWarrior IDE. In a Windows XP environment, you can click Start -> All Programs and .. ***Freescale CodeWarrior -> CW for Microcontrollers V6.2 -> CodeWarrior IDE*** (substitute in your own version of CodeWarrior). You will get a startup dialog window as shown here. Click on the 'Create New Project' button. This brings up a new window where the GP32 microcontroller option can be selected. Select 'Mon08 Interface' on the right. This allows the current user configuration with the serial port of the evaluation board. References will be provided at the end of this document for more extensive information on these options as well as using the IDE in general.

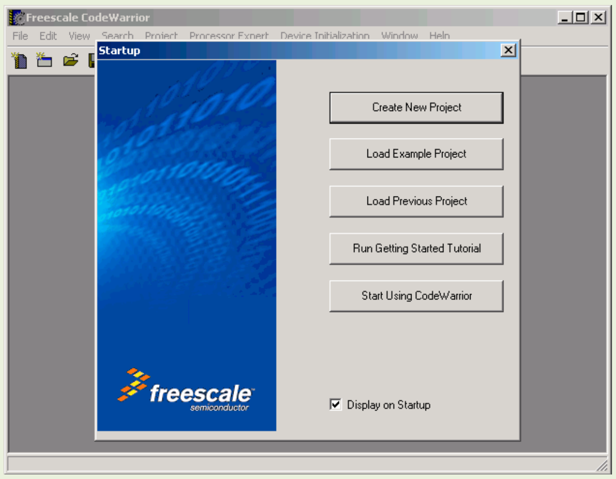
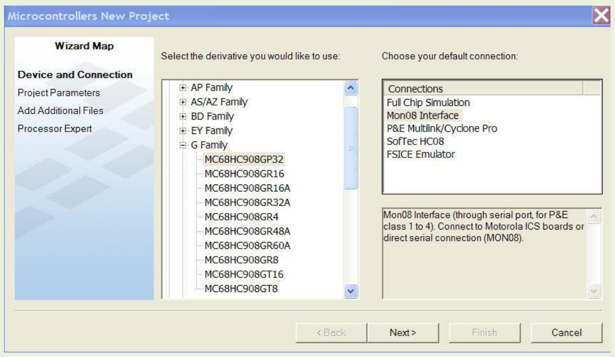


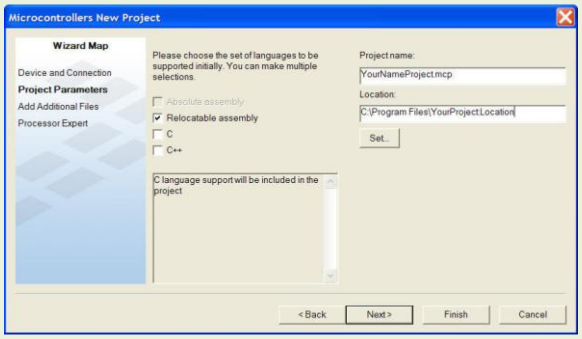
Figure 1: Create New Project Window



Mon08 Interface (through serial port, for P&E class 1 to 4). Connect to Motorola ICS boards or direct serial connection (MON08).

Figure 2: Select GP32 Template

Click Next. This opens new window shown below. Check the appropriate checkbox for using relocatable assembly. Also, name your file and the directory where your file is to be stored.


Microcontrollers New Project

Wizard Map: Device and Connection; **Project Parameters**; Add Additional Files; Processor Expert

Please choose the set of languages to be supported initially. You can make multiple selections.

- [ ] Absolute assembly
- [x] Relocatable assembly
- [ ] C
- [ ] C++

C language support will be included in the project

Project name: YourNameProject.mcp

Location: C:\Program Files\YourProjectLocation

Figure 3: Programming Type Window

Click Finish. The main project window appears. Double-click on 'Main.asm'. Freescale provides a program template for the HC08GP32 device, which appears in the section on the right. You are now ready to write you first code.

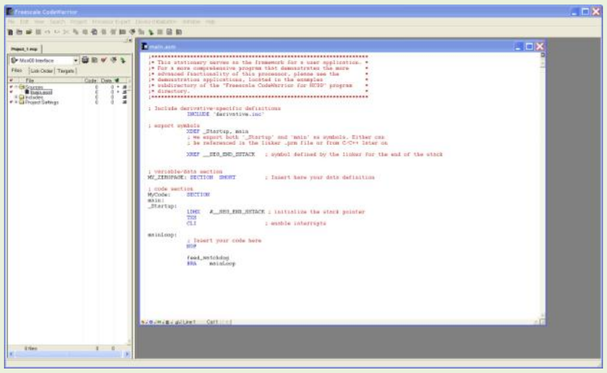
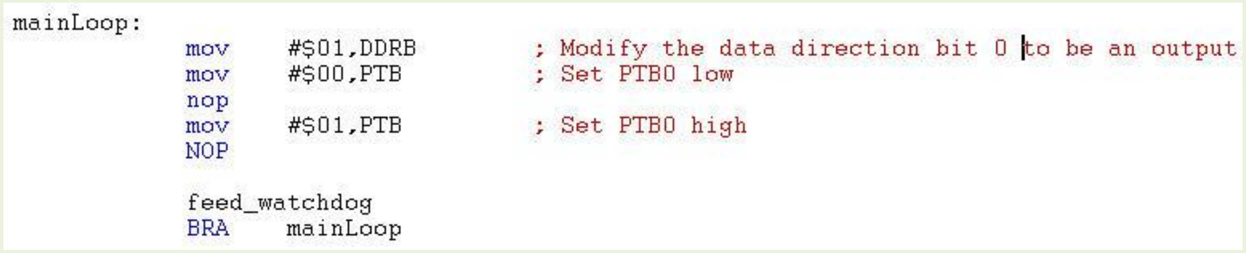

```
;**************************************************************
;* This stationery serves as the framework for a user application. *
;* For a more comprehensive program that demonstrates the more    *
;* advanced functionality of this processor, please see the       *
;* demonstration applications, located in the examples            *
;* subdirectory of the "Freescale CodeWarrior for HC08" program   *
;* directory.                                                     *
;**************************************************************

; Include derivative-specific definitions
            INCLUDE 'derivative.inc'

; export symbols
            XDEF _Startup, main
            ; we export both '_Startup' and 'main' as symbols. Either can
            ; be referenced in the linker .prm file or from C/C++ later on

            XREF __SEG_END_SSTACK   ; symbol defined by the linker for the end of the stack


; variable/data section
MY_ZEROPAGE: SECTION  SHORT         ; Insert here your data definition

; code section
MyCode:     SECTION
main:
_Startup:
            LDHX   #__SEG_END_SSTACK ; initialize the stack pointer
            TXS
            CLI                     ; enable interrupts

mainLoop:
            ; Insert your code here
            NOP

            feed_watchdog
            BRA    mainLoop
```

Figure 4: Source Code Window

Let's start with a simple program to toggle an I/O pin on and off. We can use PORTB (PTRB), bit 0. Your code in the assembly window should appear as below. Only the mainLoop section was modified with the addition of 3 lines of code.


```
mainLoop:
            mov     #$01,DDRB        ; Modify the data direction bit 0 to be an output
            mov     #$00,PTB         ; Set PTB0 low
            nop
            mov     #$01,PTB         ; Set PTB0 high
            NOP

            feed_watchdog
            BRA     mainLoop
```

Once your code has been added, click the 'Make' icon, the grey icon (second from the right). This compiles your program (hopefully) without error. Now click on the debug icon (green arrow). A new window appears that is overlayed by the connection manager. Click on 'Add A Connection'.

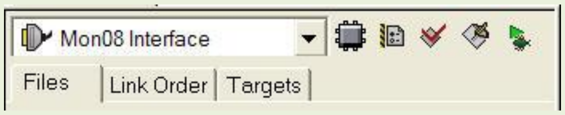
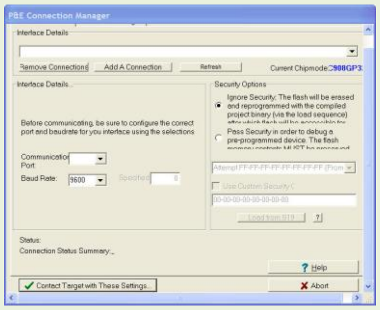


Figure 5: Source Code Processing


PBE Connection Manager

Interface Details

Current Chipmode: C908GP3

Interface Details...

Before communicating, be sure to configure the correct port and baudrate for you interface using the selections

Communication Port:

Baud Rate: 9600

Specified: 0

Security Options

(●) Ignore Security: The flash will be erased and reprogrammed with the compiled project binary (via the load sequence)

( ) Pass Security in order to debug a pre-programmed device. The flash

Attempt FF-FF-FF-FF-FF-FF-FF-FF (From

[ ] Use Custom Security (

00-00-00-00-00-00-00-00

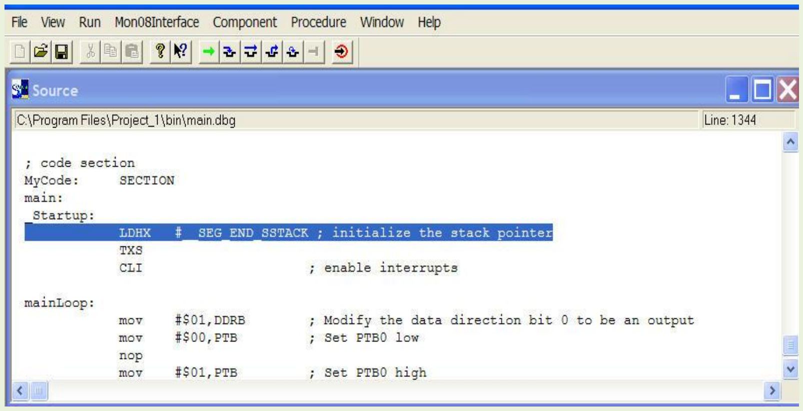
Status:

Connection Status Summary:_

Figure 6: Communication Configuration

Choose Class 3 (custom). The Communications Port is COM1 in this case (by may vary according to your PC configuration), set at a Baud Rate of 9600. If your PC is configured to use a different COM port, you can adjust this entry accordingly.

Click 'Contact Target at These Settings'. A RESET or POWER CYCLE window will then appear. Reset the microprocessor before clicking OK. The GP32 evaluation board can be reset by pushing (and releasing) the reset button on the board located at U72. Click OK when the window requesting the program download appears. You should now see the entire debug window. The source code screen is shown on the left. Click on the single step icon to step through your program. With a voltmeter, you can measure the output voltage change on PTB0 as each line of code is incremented.


File View Run Mon08Interface Component Procedure Window Help
Source
C:\Program Files\Project_1\bin\main.dbg
Line: 1344

```
; code section
MyCode:     SECTION
main:
 _Startup:
            LDHX   #__SEG_END_SSTACK ; initialize the stack pointer
            TXS
            CLI                      ; enable interrupts

mainLoop:
            mov    #$01,DDRB         ; Modify the data direction bit 0 to be an output
            mov    #$00,PTB          ; Set PTB0 low
            nop
            mov    #$01,PTB          ; Set PTB0 high
```

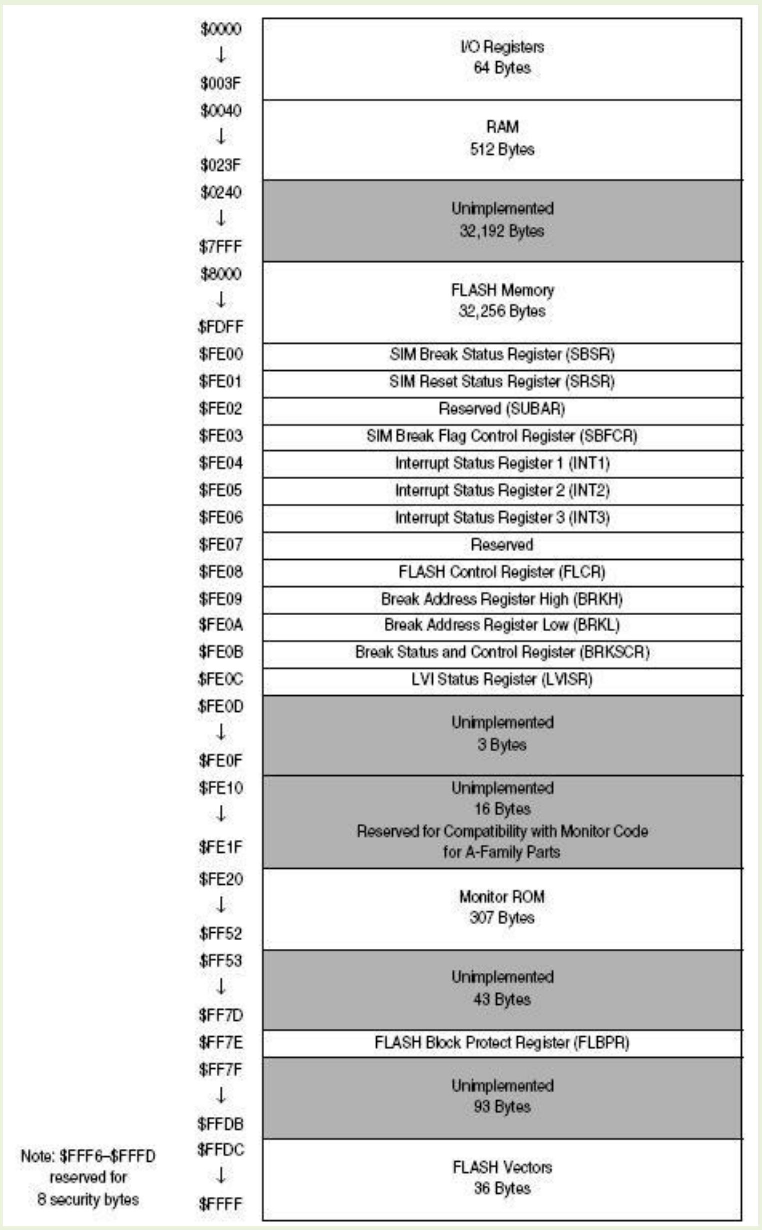
Figure 7: Source Code Debugging Window

Single Step Icon

HC908GP32 Features

The datasheet for the GP32 can be found here:
http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC68HC908GP32.pdf

A copy of the HC908GP32 memory map is shown here for reference. FLASH memory, where your program will reside, begins at address $8000. RAM address space begins at $0040. As we proceed through some of the more advances lab experiments, you will need to refer to this table for locating addresses of interest.



$0000 ↓ $003F	I/O Registers 64 Bytes
$0040 ↓ $023F	RAM 512 Bytes
$0240 ↓ $7FFF	Unimplemented 32,192 Bytes
$8000 ↓ $FDFF	FLASH Memory 32,256 Bytes
$FE00	SIM Break Status Register (SBSR)
$FE01	SIM Reset Status Register (SRSR)
$FE02	Reserved (SUBAR)
$FE03	SIM Break Flag Control Register (SBFCR)
$FE04	Interrupt Status Register 1 (INT1)
$FE05	Interrupt Status Register 2 (INT2)
$FE06	Interrupt Status Register 3 (INT3)
$FE07	Reserved
$FE08	FLASH Control Register (FLCR)
$FE09	Break Address Register High (BRKH)
$FE0A	Break Address Register Low (BRKL)
$FE0B	Break Status and Control Register (BRKSCR)
$FE0C	LVI Status Register (LVISR)
$FE0D ↓ $FE0F	Unimplemented 3 Bytes
$FE10 ↓ $FE1F	Unimplemented 16 Bytes Reserved for Compatibility with Monitor Code for A-Family Parts
$FE20 ↓ $FF52	Monitor ROM 307 Bytes
$FF53 ↓ $FF7D	Unimplemented 43 Bytes
$FF7E	FLASH Block Protect Register (FLBPR)
$FF7F ↓ $FFDB	Unimplemented 93 Bytes
$FFDC ↓ $FFFF	FLASH Vectors 36 Bytes

Note: $FFF6–$FFFD reserved for 8 security bytes

Figure 8: GP32 Memory Map

Lab 1: Racing LEDs

Objective:

With this first exercise, the student will write a short program to control the 8 diodes (U92 to U99) using the 74HC595 serial-in / parallel-out shift register. The objective will be to develop a "racing" condition whereby one LED seems to chase the next LED as each device is lit in sequence. The 74HC595 is used here to minimize microcontroller I/O usage. The 74HC595 devices can also be cascaded. This will be shown in subsequent lab experiments.

Reference Material: 74HC595 Datasheet

http://www.nxp.com/acrobat_download/datasheets/74HC_HCT595_4.pdf

The relevant portions of the schematic needed for this experiment is shown below. Three microcontroller I/O pins will be used.

PTA1 – Data
PTB3 – Storage Register Clock Input (Latch)
PTB2 – Shift Register Clock Input (Clock)

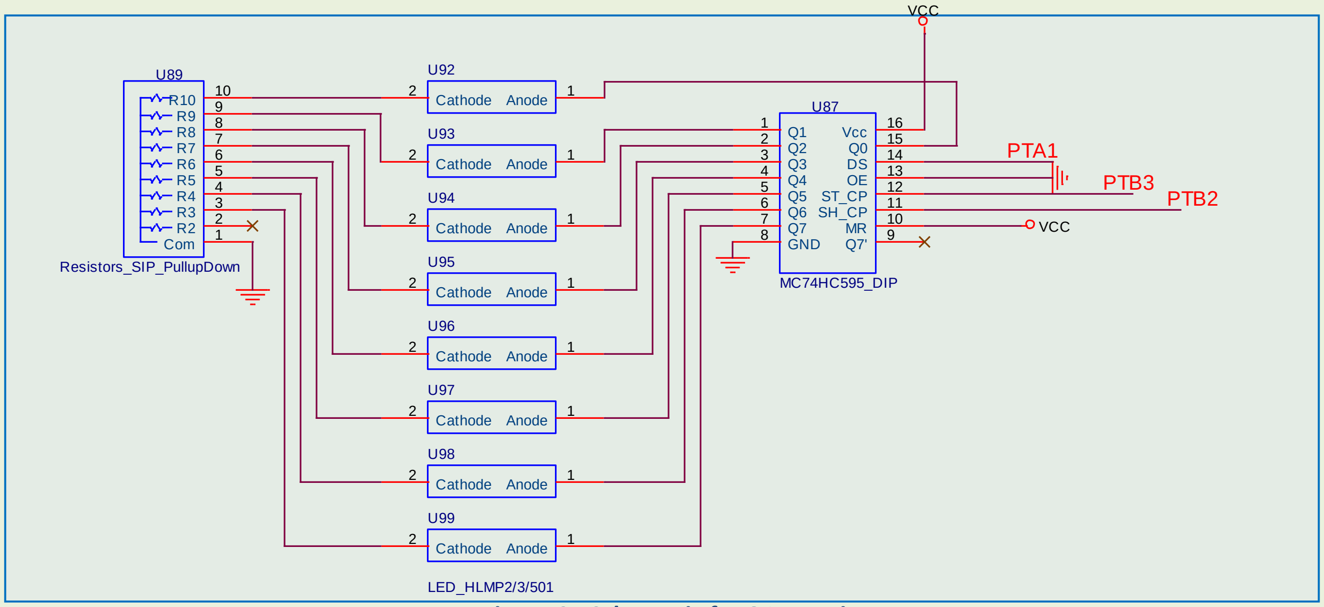


Figure 9: Schematic for 8 LED Driver

Data is clocked in to the shift register. Once 8 bits are sent, they are latched. One LED is lit per cycle, then rotated so that the next LED in sequence is activated. The accumulator 'A' can be used to rotate the bit that is high during each cycle (rola/rora). The basic structure of your program might look as follows:

Possible Implementation of “Racing LEDs” Algorithm

- Initialize Ports A and B by clearing; set Carry Bit (SEC), clear accumulator
- Declare a variable (Counter) to count down each cycle so 8 bits are transmitted
- Enable PTB2, PTB3 and PTA1 as outputs (use data direction registers)
- Begin Loop:
    - Decrement Counter
    - Rora (rotate right thru carry)
    - Branch to code to clock in a ‘0’ if Z=1 (Zero bit in CCR)
    - Otherwise, clock in a ‘1’
    - Iterate loop until all 8 bits are transmitted
    - Activate latch
    - Reinitialize Counter and send in next 8 bits

Lab 2: Playing the Speaker

Reference Material: 74HC04 Datasheet
http://www.standardics.nxp.com/products/hc/datasheet/74hc04.74hct04.pdf

Objective:

In this exercise, you will play a note on the speaker using I/O pin PTD4. A simple algorithm can be used to pulse the speaker with a square wave, thus creating a sound. More advanced applications might include playing a song on the evaluation system with the use of a look-up table with different pitched notes and modulating the frequency accordingly. Here we will start with the creation of one sound.

The relevant portions of the circuit are shown below. The PTD4 drives the 74HC04 buffer, which provides a maximum I_o of ±25mA. Given the inductive loading presented by the speaker, a diode clamp is placed across the terminal for transient protection.

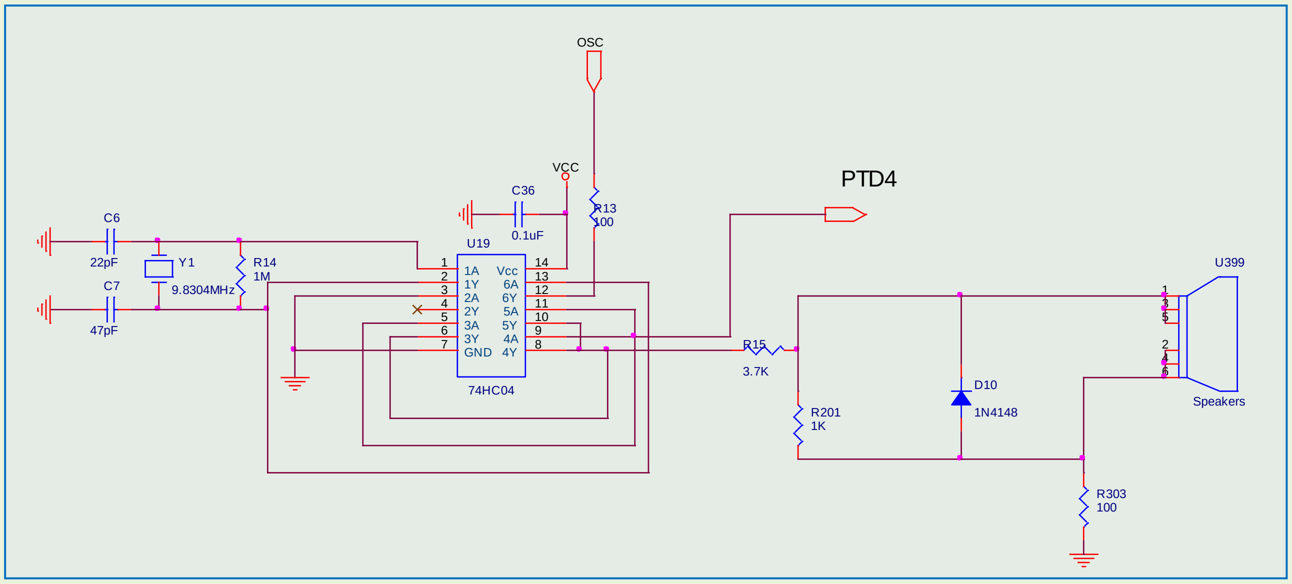


Figure 10: Schematic for Speaker Driver

Possible Implementation for Speaker Driver Code

- Set Data Direction Resistor for Bit 4 of Port D as an output
- Drive PTD4 high
- Put in a delay (delay 1)
- Drive PTD4 low
- Additional Delay (delay 2)
- Repeat

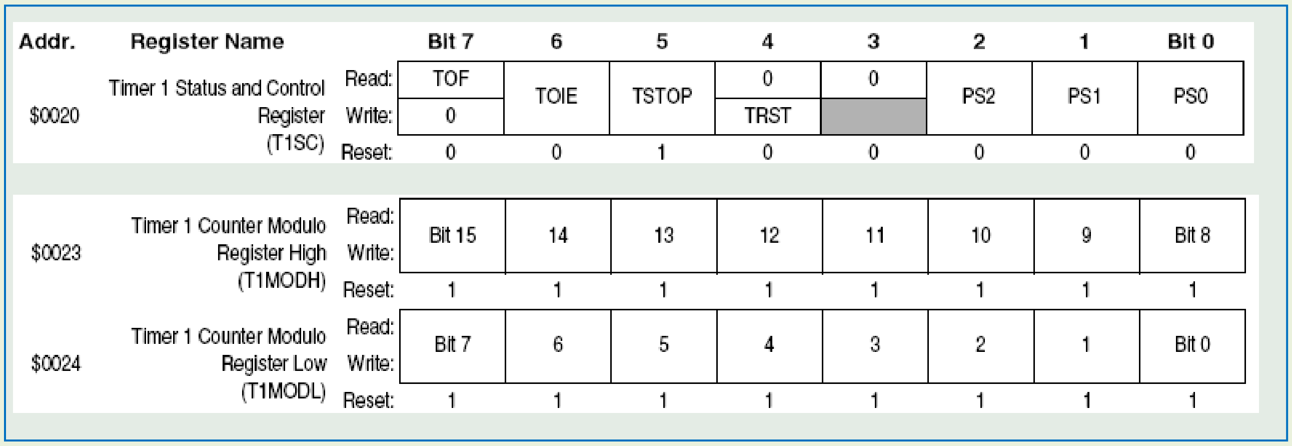
As an additional exercise, you can experiment by modifying the duration of each delay. This will affect the frequency or “pitch” of the sound, allowing the play of different tones. Additionally, you can make the duration of delay 1 different from delay 2. This modifies the duty cycle of each waveform, producing the net effect of varying the volume of the sound by modifying the duty cycle of the signal.

Lab 3: Flashing LEDs

Objective

In this lab, you will become familiar with the use of the HC08GP32 Interrupt feature. In addition, we make use of one of the microcontroller's built-in timers to set the time the LEDs cycle on and off.

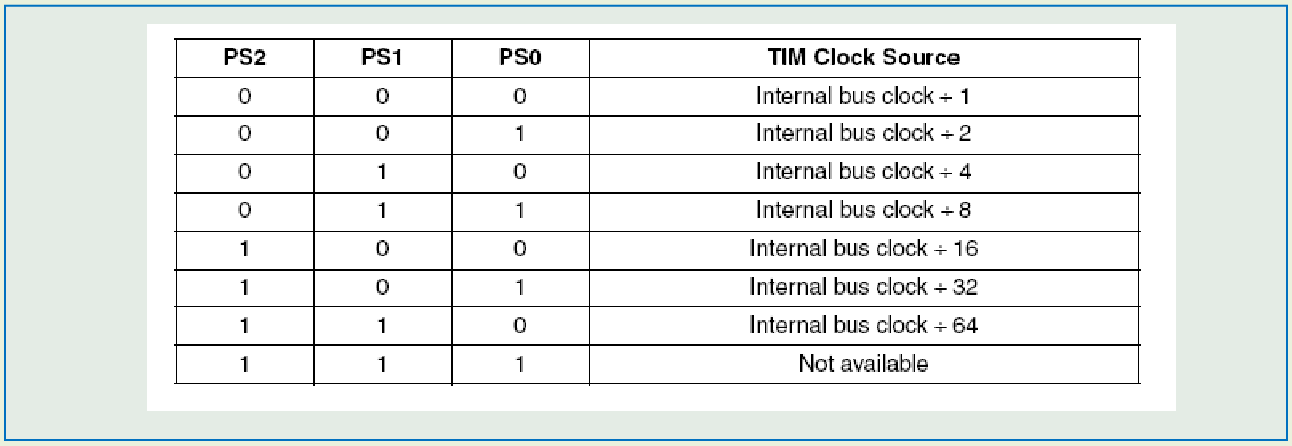
The HC08GP32 timer registers that will be used are shown below.



Addr.	Register Name		Bit 7	6	5	4	3	2	1	Bit 0
$0020	Timer 1 Status and Control Register (T1SC)	Read:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
		Write:	0			TRST	▨			
		Reset:	0	0	1	0	0	0	0	0
$0023	Timer 1 Counter Modulo Register High (T1MODH)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
		Reset:	1	1	1	1	1	1	1	1
$0024	Timer 1 Counter Modulo Register Low (T1MODL)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
		Reset:	1	1	1	1	1	1	1	1

Figure 11: Timer Registers

T1MODH and T1MODL together comprise a 16-bit register used as a free-running counter. When the counter reaches a certain value, an event can take place as defined in the program. Additionally, interrupts can be set to run a service routine. The Timer Status and Control Register (T1SC) controls whether an interrupt is set, whether the counter is active and other counter features. The GP32 data sheet will provide more information. The 3 low-order bits in T1SC provide prescaling for the counter by dividing the internal bus clock a set amount as given in the table below. The internal bus clock of the microcontroller is 25% of the external frequency set by the crystal. With the crystal operating at 9.83MHz, the internal bus clock is 2.46MHz.



PS2	PS1	PS0	TIM Clock Source
0	0	0	Internal bus clock ÷ 1
0	0	1	Internal bus clock ÷ 2
0	1	0	Internal bus clock ÷ 4
0	1	1	Internal bus clock ÷ 8
1	0	0	Internal bus clock ÷ 16
1	0	1	Internal bus clock ÷ 32
1	1	0	Internal bus clock ÷ 64
1	1	1	Not available

Figure 12: Clock Frequency Configuration

Interrupt Vector Addresses

For the HC908GP32, interrupt vector addresses begin at location $FFDC. The interrupt vector for TIM 1 is located at $FFF2 and $FFF3. You will need to initialize this location with the address where your program will continue. Each time the interrupt is serviced, the program will jump back to the address located in its register.

Possible Implementation of Flashing LEDs Code

- Initialize I/O Ports, set inputs/outputs as needed
  (PTA1=Data, PTB2=SH_CP, PTB3=ST_CP)
- Initialize Timer, set needed values into T1SC, T1MODH, T1MODL
- Clear Interrupts
- Set up infinite loop (this is where interrupt should return with each iteration)
  Loop: BRA Loop
- Set up variable to change state after each interrupt
  COM Your_variable_name
- Turn On/Off LEDs depending on state of Your_variable_name
- Clear Interrupt and repeat

Lab 4: 4-Digit Seven-Segment Display

Reference Material: 2-Digit Display Datasheet
http://www.datasheetcatalog.org/datasheet/hp/HDSP-5507.pdf

Objective:

The objective of this lab is to write a number to each of the four seven-segment displays. Each number will be displayed simultaneously. We will make use of a single 74HC595 serial-to-parallel converter to drive all displays. To accomplish this task, each digit can be pulsed briefly in sequence. Since this process occurs quickly, each number will appear to be lit simultaneously. The relevant portions of the circuit are shown below.

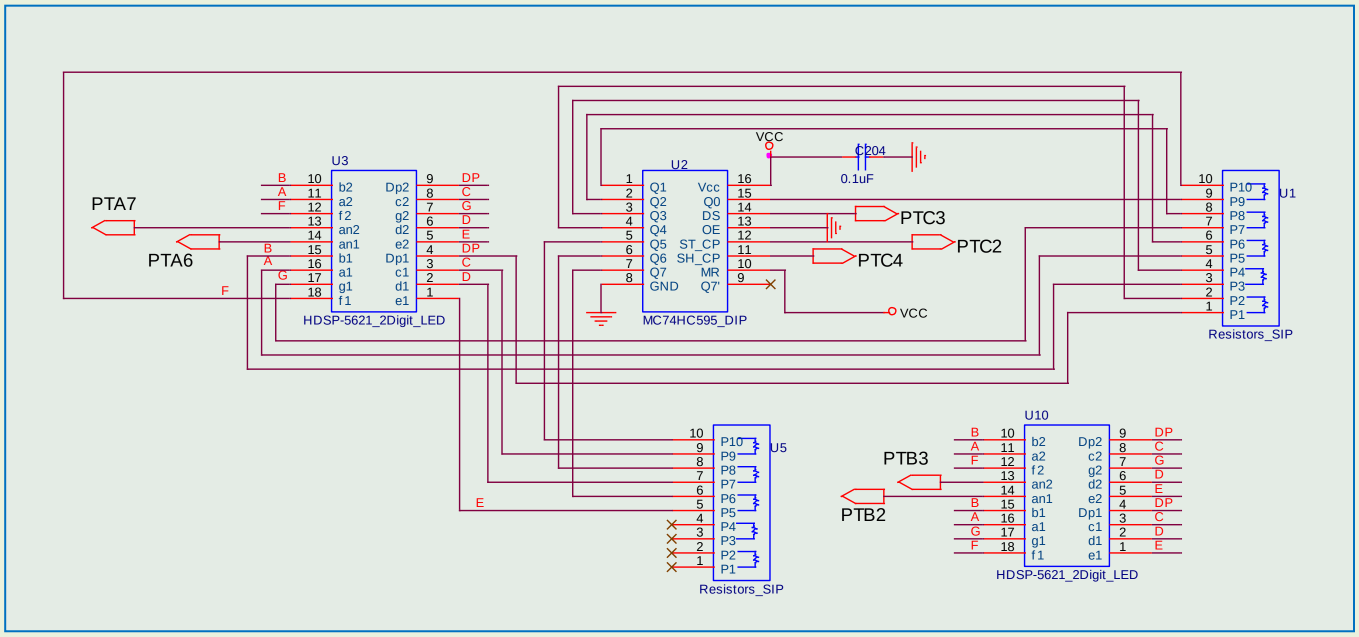


Figure 13: Schematic for 2-Digit LED Driver

For each digit LED, there is a corresponding anode which is common to all other diodes comprising each segment. By switching on/off each common anode (AN1/AN2), the display can be controlled. The pin-out configuration for the 2-digit LED structure is shown below.

PIN	FUNCTION
1	E Cathode No. 1
2	D Cathode No. 1
3	C Cathode No. 1
4	DP Cathode No. 1
5	E Cathode No. 1
6	D Cathode No. 2
7	G Cathode No. 2
8	C Cathode No. 2
9	DP Cathode No. 2
10	B Cathode No. 2
11	A Cathode No. 2
12	F Cathode No. 2
13	Digit No. 2 Anode
14	Digit No. 1 Anode
15	B Cathode No. 1
16	A Cathode No. 1
17	G Cathode No. 1
18	F Cathode No. 1

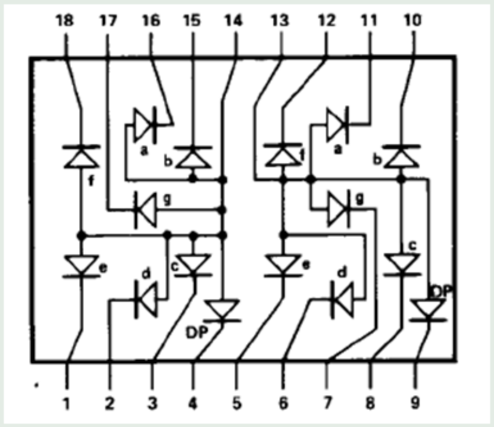


Figure 14: 2-Digit LED Pinout

This table shows the various pin-outs for the 7-segment LED displays.   See the configuration on the right with the various 'Q' designations (outputs from the 74HC595 converter). Since we are using a common anode typology, setting a '0' to any segment results in lighting the LED. For example, in order to display the number '4' on the LED, the number $D4 will need to be output.

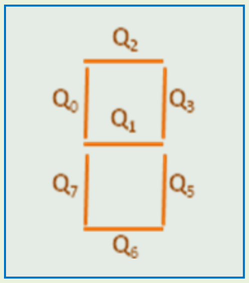


Figure 15: 7-Segment Display Mapping

Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
1	1	0	1	0	1	0	0

Figure 16: 7-Segment Display Mapping for Number '4'

You can make a look-up table comprising the other numbers for your display.

Possible Implementation for 7-Segment LED Display

- Initialize all I/O pins to be used
- Load the accumulator with the number to be displayed
- Transmit the 8 bits into the 74HC595 and latch
- Turn on AN1
- Turn off AN1
- Transmit next 8 bits representing 2nd number
- Turn on AN2
- Turn off AN2
- …do the same for AN3 and AN4
- Loop back and repeat

Lab 5: Keypad

Objective:

This lab introduces the 4x4 keypad. Each key has a momentary contact switch connected to an intersection of row and column wires. When a key is pressed, the corresponding row and column is shorted. The microcontroller can continually scan the rows/columns to identify the short and therefore the corresponding key. This method is widely used due to the savings in numbers of I/O pins required. See the illustration below for the keypad setup.

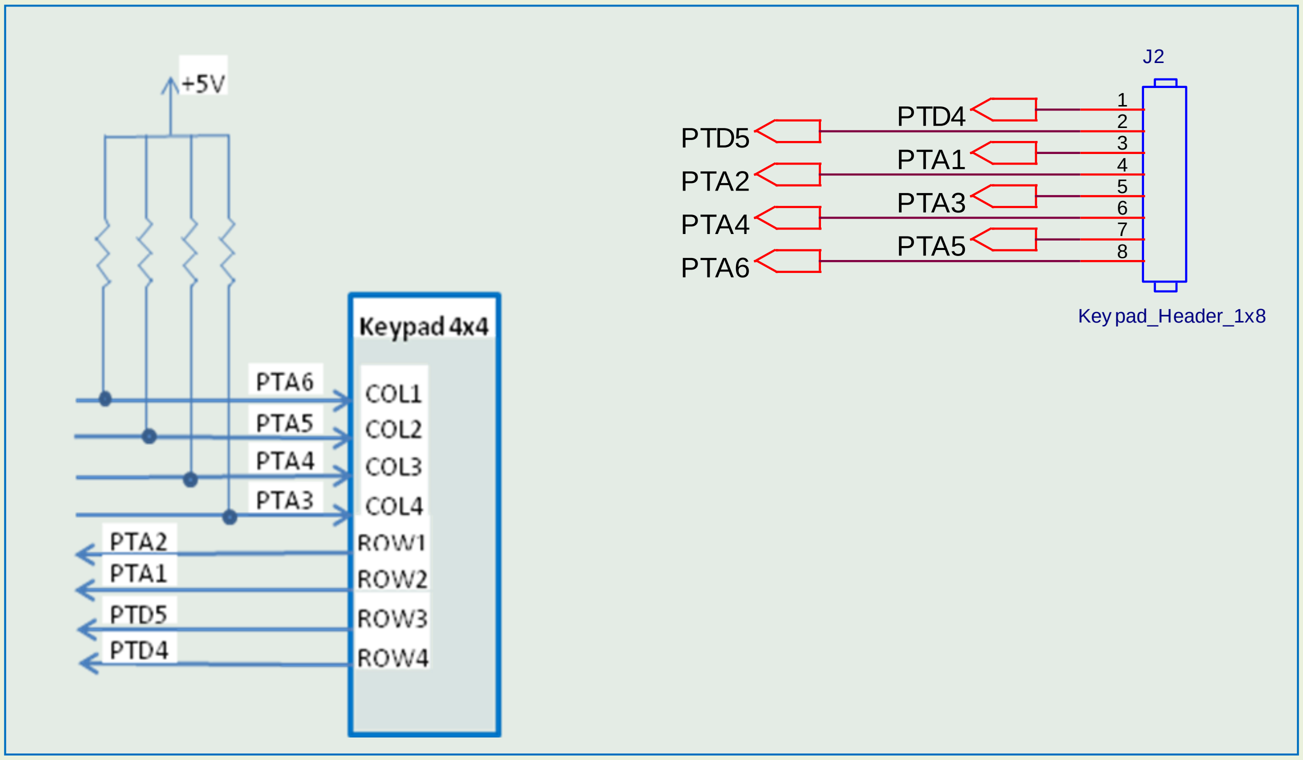
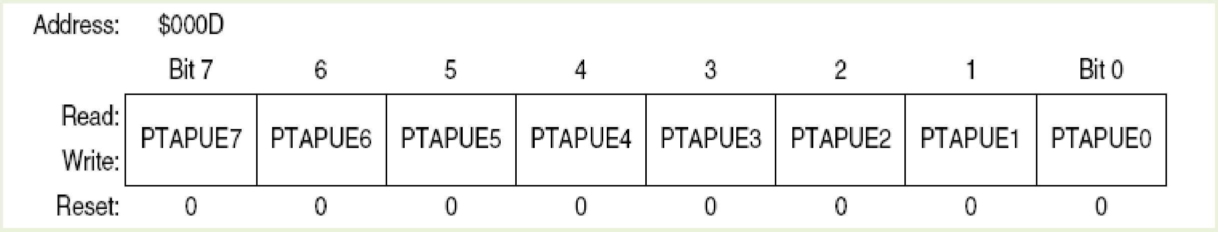


Figure 17: Keypad Configuration

After all outputs are set high, each input (row) is set to '0' in sequence to determine if a key was pressed. This cycle continues repeatedly until a key is pressed. A lookup table can be used to identify each key with a number (or letter). This character can then be stored into a variable to test the functionality of your code.

The pull-up resistors shown in the figure above for the columns can be enabled internally within the microcontroller. See the PTAPUE register below for the correct register to use. For each pull-up resistor enabled (with a logic '1' written to the register), the corresponding DDRA register bit will need to be set as an input with a logic '0'.



Address: $000D

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
Reset:	0	0	0	0	0	0	0	0

Figure 18: Internal Pull-Up Resistor Register

Possible Implementation for 4x4 Keypad

Initialize all I/O pins used

Enable pull-up resistors

Set outputs (rows) high

Load Index Register with look-up table starting address

Set Row 1 to '0'

Search to see if key was pressed

If not, set Row 1 to '1' and Row 2 to '0'

Repeat for each row

If key pressed, search look-up table for number using H:X

Lab 6: Analog-to-Digital Converter

Objective:

In this lab, the HC08GP32's ADC will be used. Using the on-board potentiometer located at U15, a voltage will be dialed in between 0 and 5 volts. This voltage will then be displayed in hex format using the 8 individual LEDs. The digital scale representing this voltage range is 00h to FFh. For example, a voltage of 2.5V would be represented as 128d or 80h. In binary form, this translates to 0100 0000. With a 2.5V output, the 7th diode (D11) will be the only one lit. The relevant circuit is shown on the right.

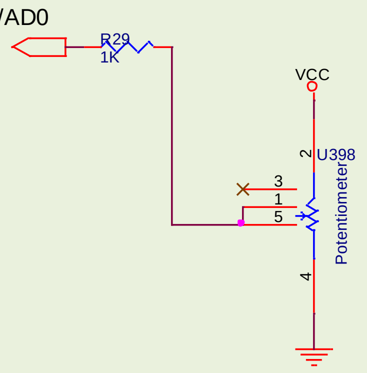
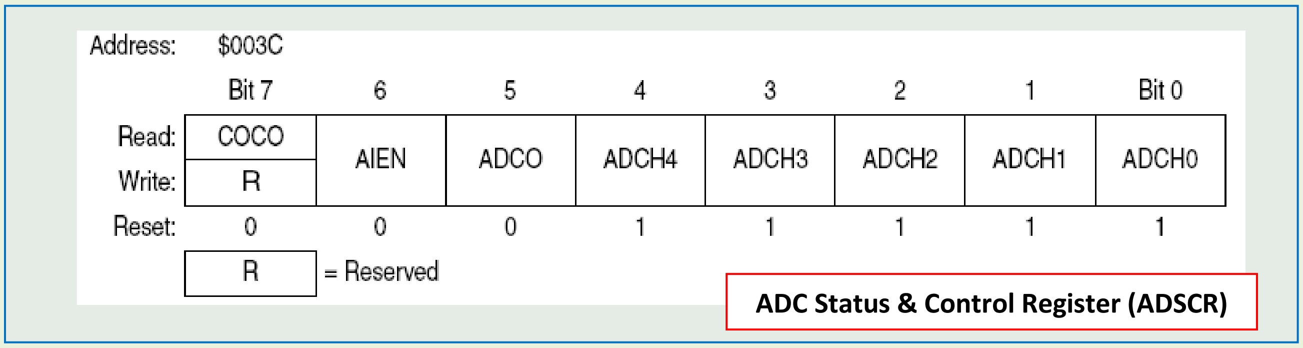


FIgure 19: ADC Schematic

The ADC pins on the HC08GP32 are shared with PTB0 to PTB7 (AD0 to AD7 respectively). We use the ADC Status and Control Register (show below) to set the parameters for the ADC.



Address: $003C

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
Write:	R							
Reset:	0	0	0	1	1	1	1	1

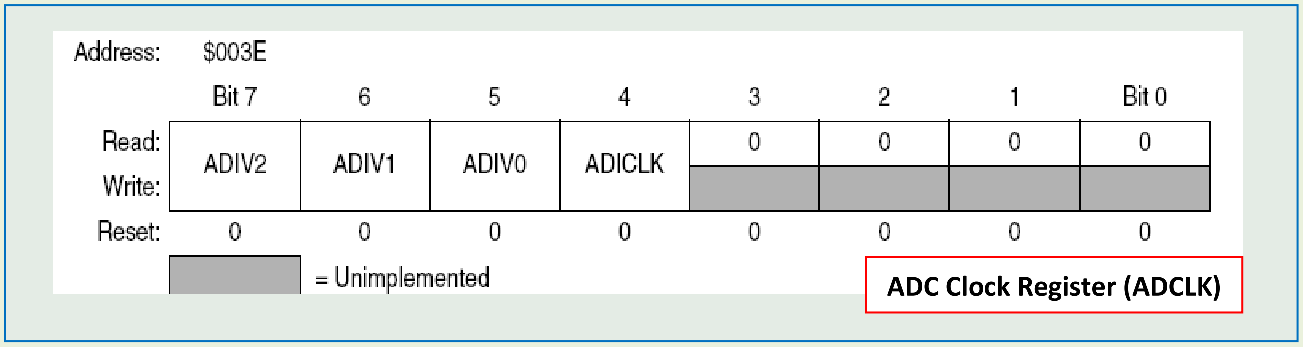
R = Reserved

ADC Status & Control Register (ADSCR)

Figure 20: ADSCR Status and Control Register for ADC

This register allows settings for continuous conversions, flags being set when conversions are completed, channel select bits, etc. See the HC08GP32 datasheet for more information.

There is also an ADC register called ADCLK. This register is used to set the clock frequency for the ADC. It is recommended by the manufacturer that the clock frequency be set to 1 MHz (divide by 8). This can be done using appropriate settings for ADIV0, ADIV1 and ADIV2.



Address: $003E

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
Write:					▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

ADC Clock Register (ADCLK)

Figure 21: ADC Clock Register

The resulting measurement is then written into the ADR (ADC Data Register).

Possible Implementation for ADC and LED Display

- Initialize ADSCR and ADCLK
- Begin ADR conversion
- Delay and read ADR into accumulator
- Shift accumulator contents into 74HC595 converter for LED display

Lab 7: DS1307 Real Time Clock Display

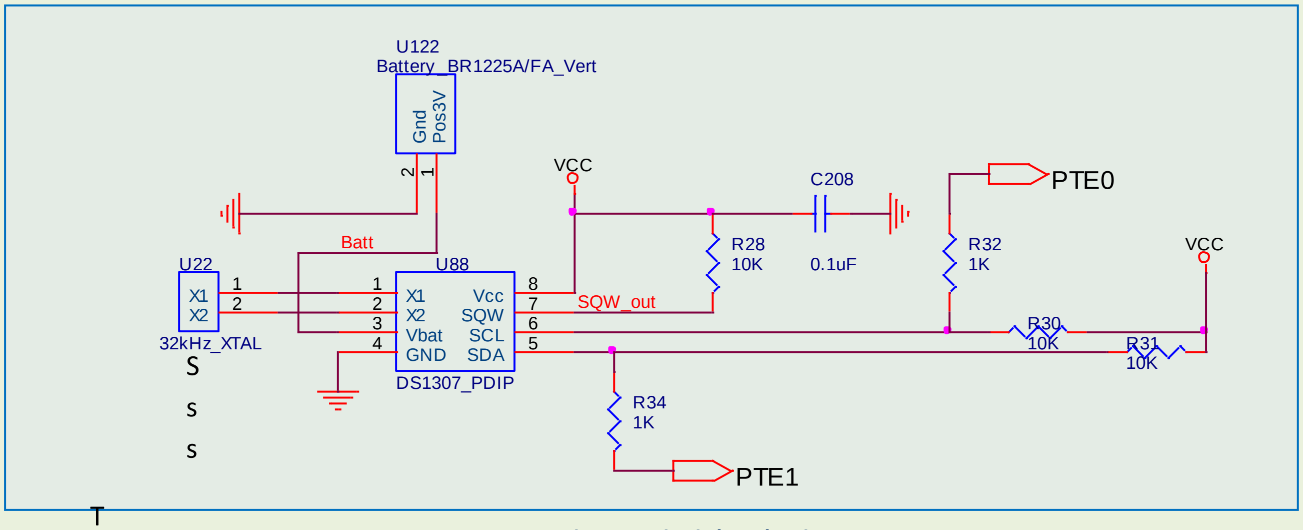
Objective:

This lab will use the DS1307 RTC. For this experiment, a timer will be created and displayed on the 2-digit LEDs. The lower 2 digits will show seconds, while the third digit displays minutes. To use the I/O pins efficiently, a single 8-bit output is used for all 3 digits by pulsing each digit individually. The DS1307 uses an I^{2}C interface. In order to communicate with the device, we will use PTE0 and PTE1 in a "bit bang" manner to clock in the data.

Reference Material: DS1307 Datasheet

http://datasheets.maxim-ic.com/en/ds/DS1307.pdf

The relevant schematic section for this circuit is shown below. HC08GP32 ports PTE0 and PTE1 will be used to drive the RTC SCL and SDA pins respectively.



T

Figure 22: Real-Time Clock (RTC) Schematic

The I^{2}C interface uses 2 bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (CLK). These lines are configured with pull-up resistors. Data is initially transferred by the microcontroller by a change in state of the data line (from HIGH to LOW), while the clock is kept HIGH. Further information on the I^{2}C protocol can be found in the DS1307 datasheet.

Each time the DS1307 is written or read in the start condition, the RTC's address must be sent by the microcontroller (master). The address is 1101 0000 or $D0 for a write. Alternatively, 1101 0001 or $D1 is sent if the microcontroller expects to read the RTC registers. An illustration of this method is shown below.

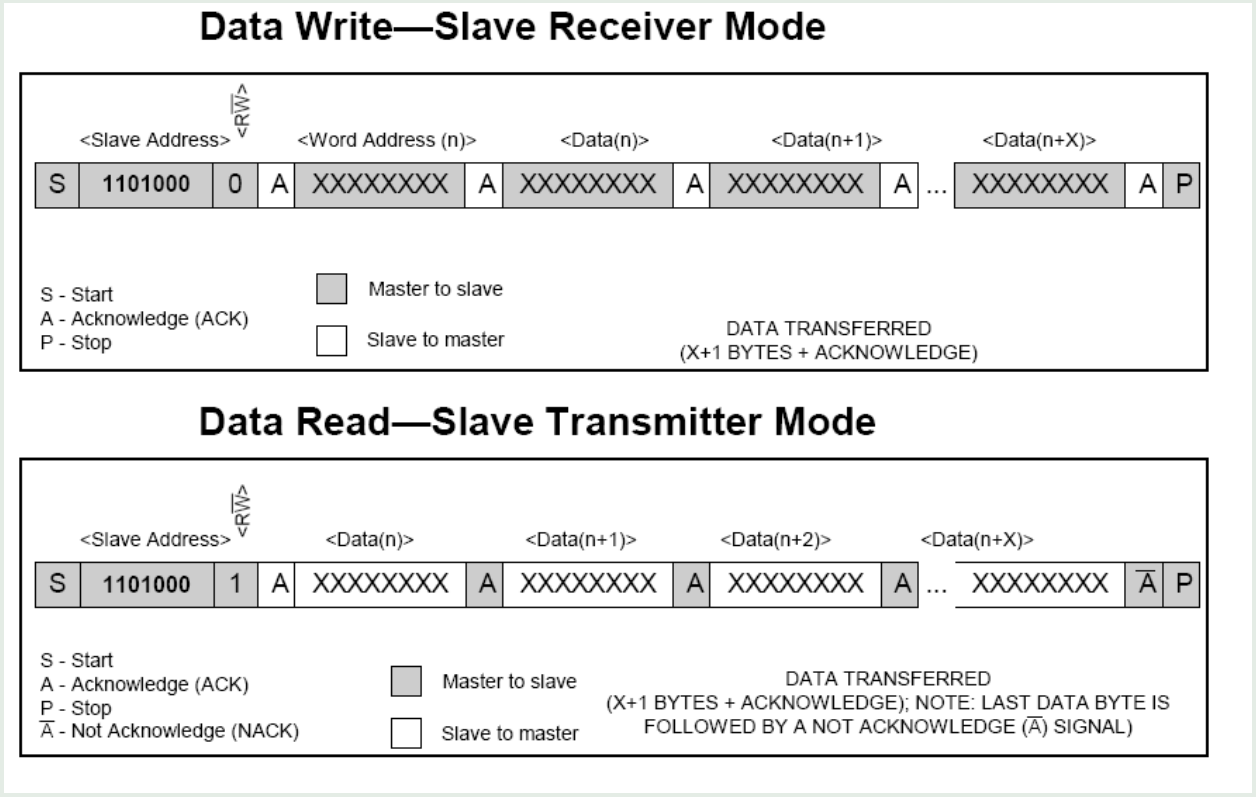
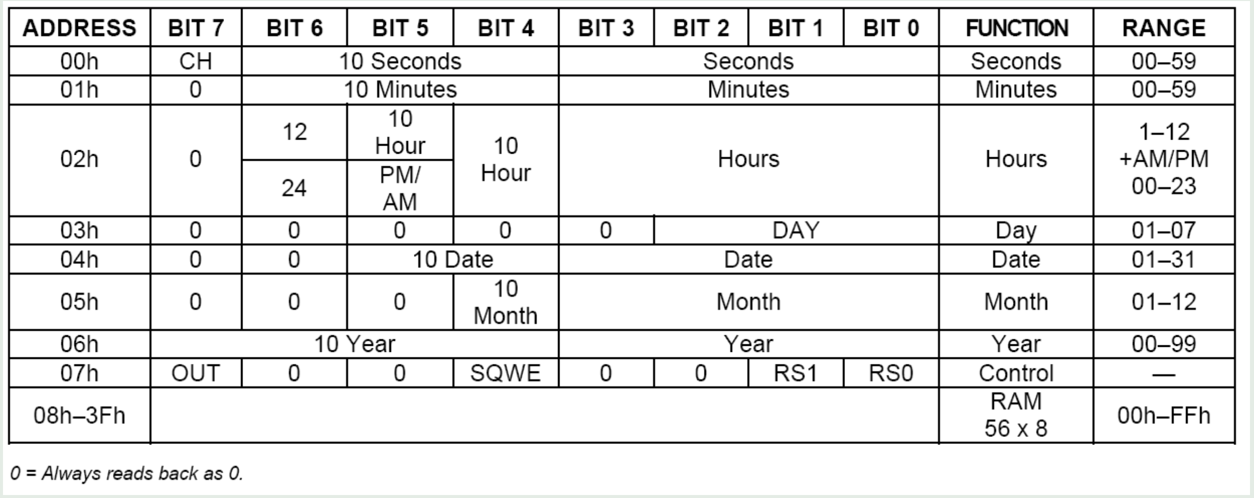


Figure 23: Read/Write Instructions to RTC

The RTC registers available with the DS1307 are shown below. RTC registers are available at locations 00h to 07h. The DS1307 also maintains available RAM located at address locations 08h to 3Fh. For this exercise, we will only be reading the seconds and minutes register for display on the 7-segment LEDs.



ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM
		24	PM/ AM							00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

Figure 24: Register Mapping for DS1307 RTC

Keep in mind that the register pointer for the DS1307 will need to be reset each time that seconds are read. Otherwise, it will increment to $3F with each byte written or read.

Although not implemented in this exercise, the DS1307 also maintains a square wave generator. The control for this function is located in the Control Register (07h). Here, the square wave can be enabled at various user-selectable output frequencies. A header is located on the evaluation board for use of the square wave function.

Possible Implementation for DS1307 RTC Display

- Initialize. Make PTA6,PTA7,PTB2,PTC2,PTC3,PTC4 outputs
  Initialize RTC. Bring PRE0, PTE1 outputs low
- Start RTC. (SDA/SCL high, then bring SDA low; SCL then brought low)
- Load accumulator with DS1307 WRITE address ($D0) and send to RTC
- Clock in $00 several times to initialize first few registers (seconds/minutes)
- Clock in STOP byte to end serial transfer
- Initialize pointer to point to first register $00
- Clock in START byte
- Clock in READ byte ($D1)
- Read seconds register and store in variable location
- Read minutes register and store in variable location
- Clock in Last RX byte
- Clock in STOP byte to end serial transfer
- Initialize point to point to first register $00
- Display seconds/minutes on 7-segment LED display
- Repeat loop to refresh seconds/minutes

Lab 8:   LCD Display

Reference Material: HD44780 LCD Controller Data Sheet
http://web.media.mit.edu/~ayah/documents/hd44780u.pdf

Objective:

In this lab, you will create a 2 line message for display on the evaluation board's LCD display. The HD44780 controller chip is commonly used on many character LCDs. Knowledge of the HD44780 will serve as a good foundation for developing your code.

A copy of the relevant portions of the LCD circuit is shown here. This graphics show the 1x16 pin configuration (J3). Identical pin-outs are used for the 2x8 LCD port (J44) located adjacent to the 1x16. To minimize I/O usage, we will use only 4 pins to input the data: PTB3, PTB4, PTB5 and PTB6. Two control lines are used, RS (Register Select) and EN (Enable). The RW (Read/Write) line is tied low, which sets up the LCD as a write-only device.

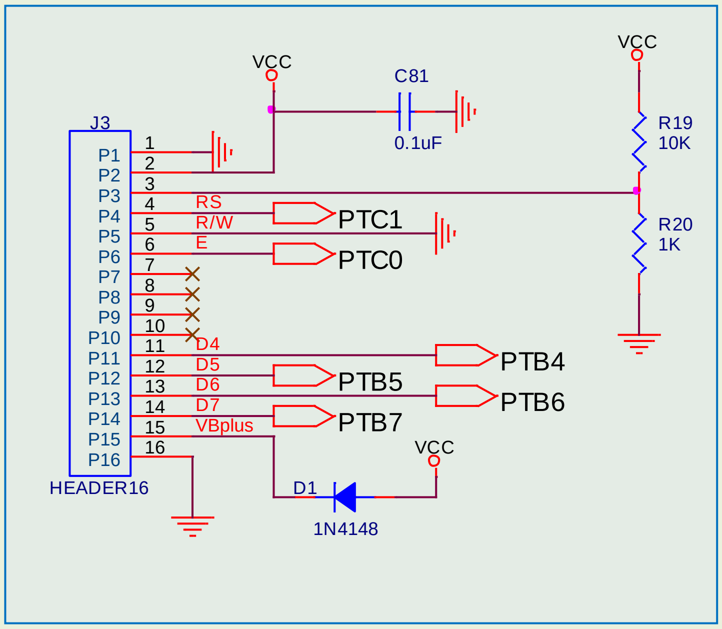
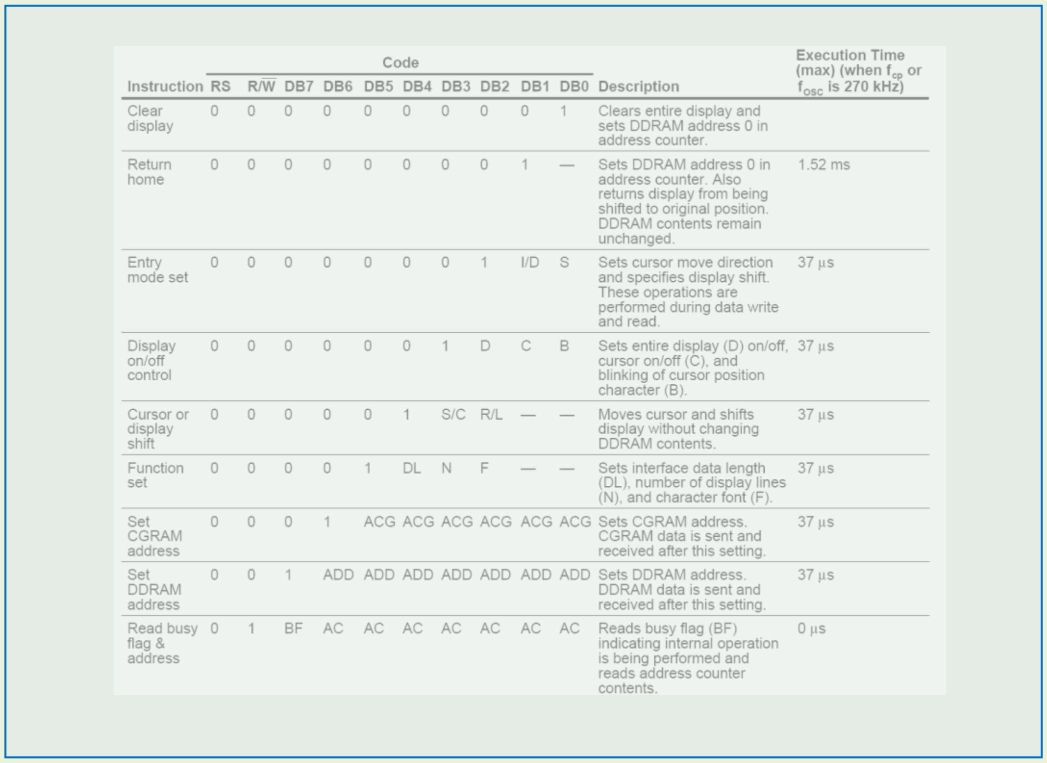


Figure 25: Schematic for LCD Circuit

The EN line is used to prepare the LCD for receiving data. To send data to the LCD, this line must be low. With EN low, you can set the RS line or begin sending data. When ready to transmit, bring EN high. At the end of the transmission, EN is brought low.

When RS is low, the data is considered to be a command or instruction. For sending data to be displayed, keep RS high. A set of commands/instructions is shown in the table below.



Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Figure 26: LCD Function Instructions

For initializing the LCD, a good tutorial for the sequence of bytes to transmit can be found here: http://www.doc.ic.ac.uk/~ih/doc/lcd/initiali.html

Keep in mind the delay times needed between transmissions. You can use the timer (TIM) algorithm coded in earlier labs for this purpose.

To display your message, you may want to store your ASCII string into consecutive bytes of memory with the directive 'FCC'. Code would look something like this:

```
msg1   FCC  '  My Message 1  '    ;  show message top line
msg2   FCC  '  My Message 2  '    ;  show message bottom line
```

The first message can then be loaded into the H:X register with characters displayed by moving each character in sequence to be transmitted. A useful instruction might be -
mov x+,PTB

PTB can then be shifted one bit at a time for sending to the LCD.

Possible Implementation for DS1307 RTC Display

- Initialize I/O pins
- Initialize LCD for using 4-bit interface, 2 lines and 5x7 format. To input data, each 4-bit set is entered twice to make up the 8-bit instruction/dataset
- Load first message for printing to line 1 of the LCD into H:X
- Transmit data to the LCD. The ASL instruction may be helpful for this.
- Assume 16 characters per line
- Jump to the next line and transmit message #2
- *Optional:* set RESET low and turn cursor off
- *Note:* for all these instructions, insure that the proper delay times are observed.

Lab 9: Temperature Measurement & Display

Objective:

To measure temperature in °C for display on 2 digits of the 7-segment display. The DS1822 thermometer will be used. Produced by Dallas Semi, this device uses only one I/O pin. Thermometer resolution is user selectable for 9-bit to 12-bit accuracy. To simplify the coding for the display, only 8 bits will be used (ignoring any fractional resolution lower than 1 degree). Additionally, it will be assumed that only positive temperatures are being measured, i.e. ≥ 0 °C. Precise timing control will need to be utilized to communicate with this device.

Reference Material: DS1822 Data Sheet

http://datasheets.maxim-ic.com/en/ds/DS1822.pdf

A copy of the relevant circuit for driving the DS1822 is shown here. As mentioned, there is only one I/O pin needed. In some cases, the DS1822 can be powered by the I/O line itself; however, in this example we will use the 5V supply. A weak pullup resistor is attached to the data line to keep the port high in the absence of external input. Once a temperature measurement is taken, it is stored in the 2-byte temperature register located in the lower part of the "scratchpad" memory.

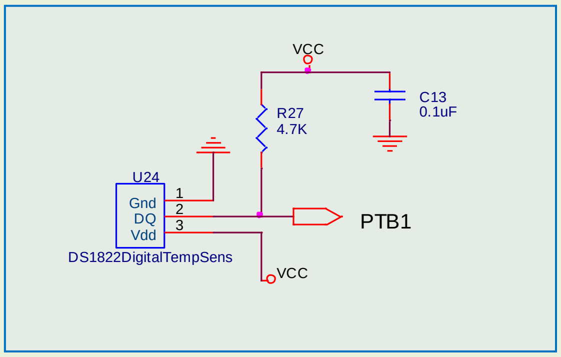
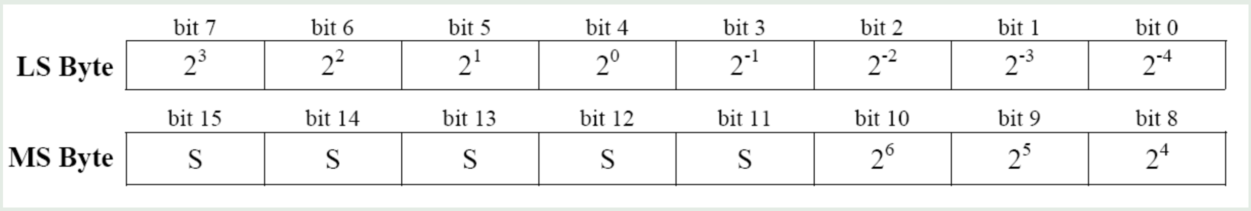


Figure 27: DS1822 Temperature Circuit Schematic

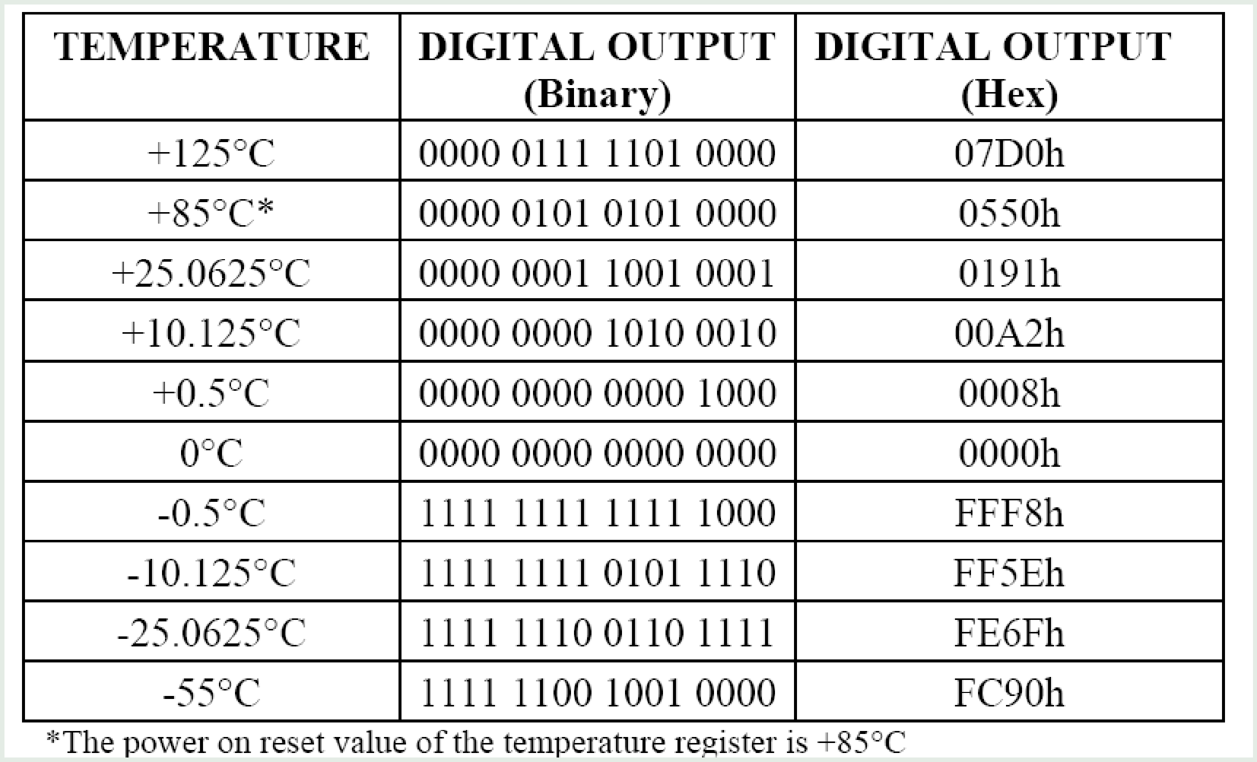


	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

Figure 28: DS1822 Temperature Registers

The temperature register format is shown here.

During the DS1822's initialization, any temperature resolution can be set (12-bit to 9-bit). However, we will only be using 8-bits for display. This includes bits 11 to bit 4 inclusive. Since bit 11 is '0' for all positive temperatures, only the 7 bits from bits 10 to 4 will be relevant for this application. See the temperature table below for more information.



TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C*	0000 0101 0101 0000	0550h
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FE6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power on reset value of the temperature register is +85°C

Figure 29: DS1822 Temperature Table

As shown, the least significant 4 bits only represent fractional degrees and can be ignored. The temperature can then be directly read from the hex output. For instance, the highest temperature 07D0h represents 125 °C (after stripping off the lower 4 bits). Once the temperature is read, a binary-to-BCD conversion can be done for display.

In the initial debug phase of coding this example, it might be helpful to do a read and display of the DS1822 temperature register without actually doing a temperature conversion. As seen with the asterisk in the above table, the DS1822 powers up to a default temperature setting of +85 °C. Once this temperature can be reliably displayed, you could proceed with the next step of generating an actual temperature.

There are 3 steps which must be followed for any transaction sequence. These include:
Initialization
ROM Command
Function Command

The ROM Command can be skipped. However, CCh must be sent after initialization to inform the device that ROM commands are being skipped. You could then proceed to issue a Function Command.

Before the temperature measurements can be displayed, they will need to be converted to BCD (Binary Coded Decimal). For example, the hex number 1Eh (30d) cannot be displayed as is on the 7-segment

LED display as a decimal number. However, once converted to BCD, each digit (4 bits each) can be shown.

The Binary-to-BCD conversion can take place by dividing the binary by 100d or 64h (or an alternate power of 10 depending on the size of the number to be converted).   The remainder is then divided by the next lower power of 10. The quotients are then captured for eventual display of the decimal equivalent. See the table on the right for an illustration of converting 07Dh to its decimal equivalent of 125d. In this example, 125 is divided by 100 to give a quotient of 1 with a remainder of 25. The numbers in the table here represent the hex equivalent of this division.

		Quotient		Remainder
07D/64	=	**1**	+	19
19/A	=	**2**	+	5
5/1	=	**5**	+	0

Possible Implementation for DS1307 RTC Display

Initialize I/O, clear registers A, X, H and whatever variables are to be used

Reset the DS1822. Write ‘0’ on the data line, delay and read the ACK. Confirm reset functional.

Transmit CCh (skip ROM)

Transmit 44h (compute temperature)

Reset DS1822.

Transmit CCh (skip ROM)

Transmit BEh (instruct DS1822 that a read scratchpad will follow)

Read lower temperature register and store in variable

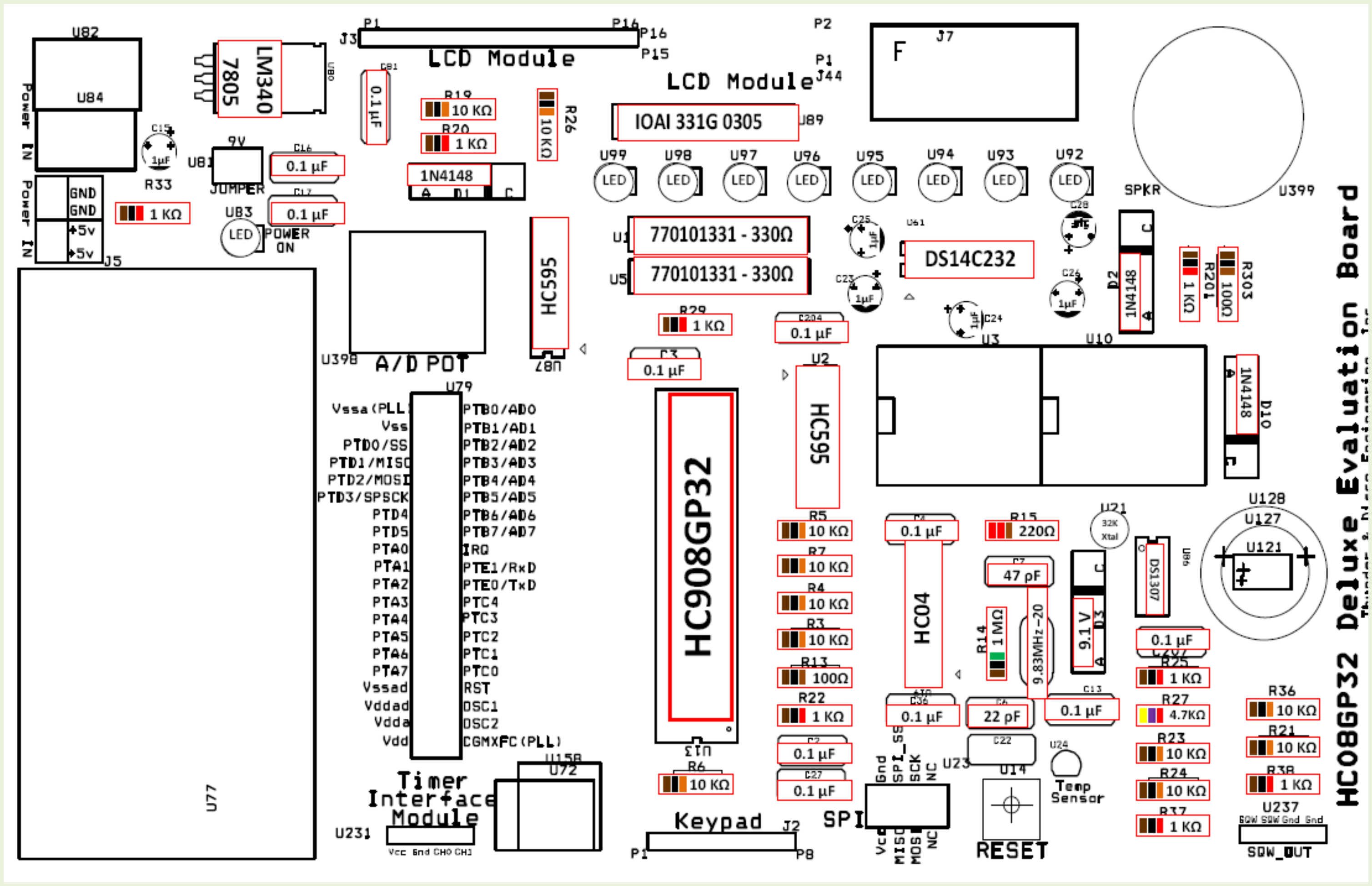
Read higher temperature register and store in variable

Prepare temperature measurements for display
(you should have the temperature number stored in hex format in a variable)

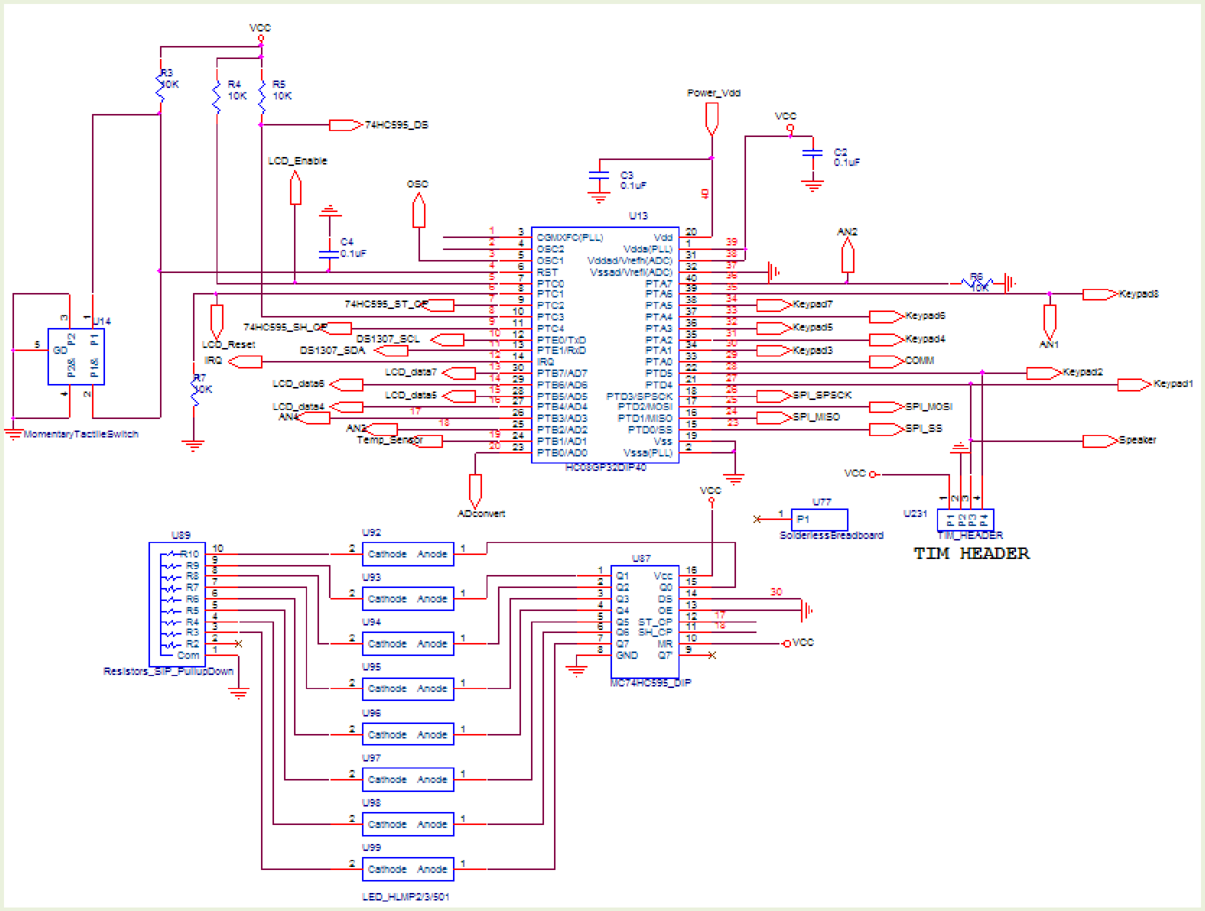
Perform a binary-to-BCD conversion

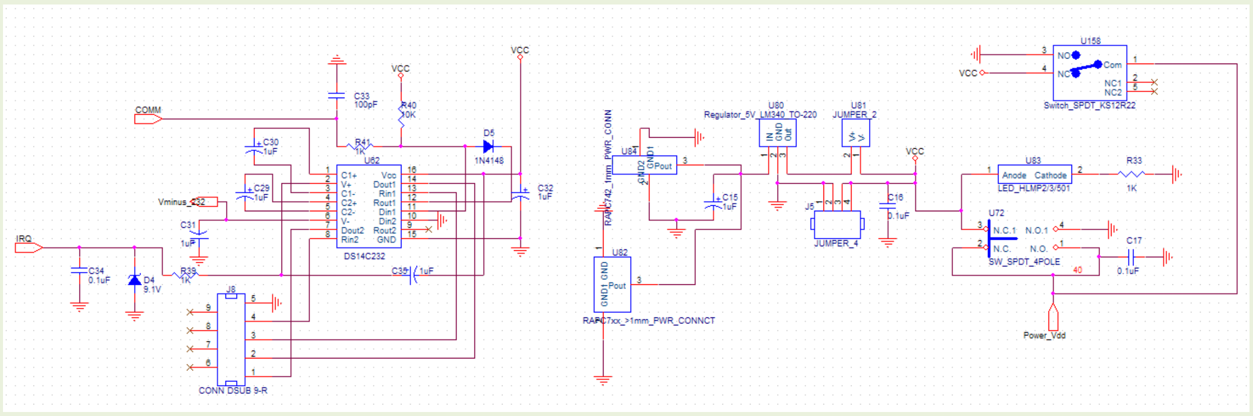
Display

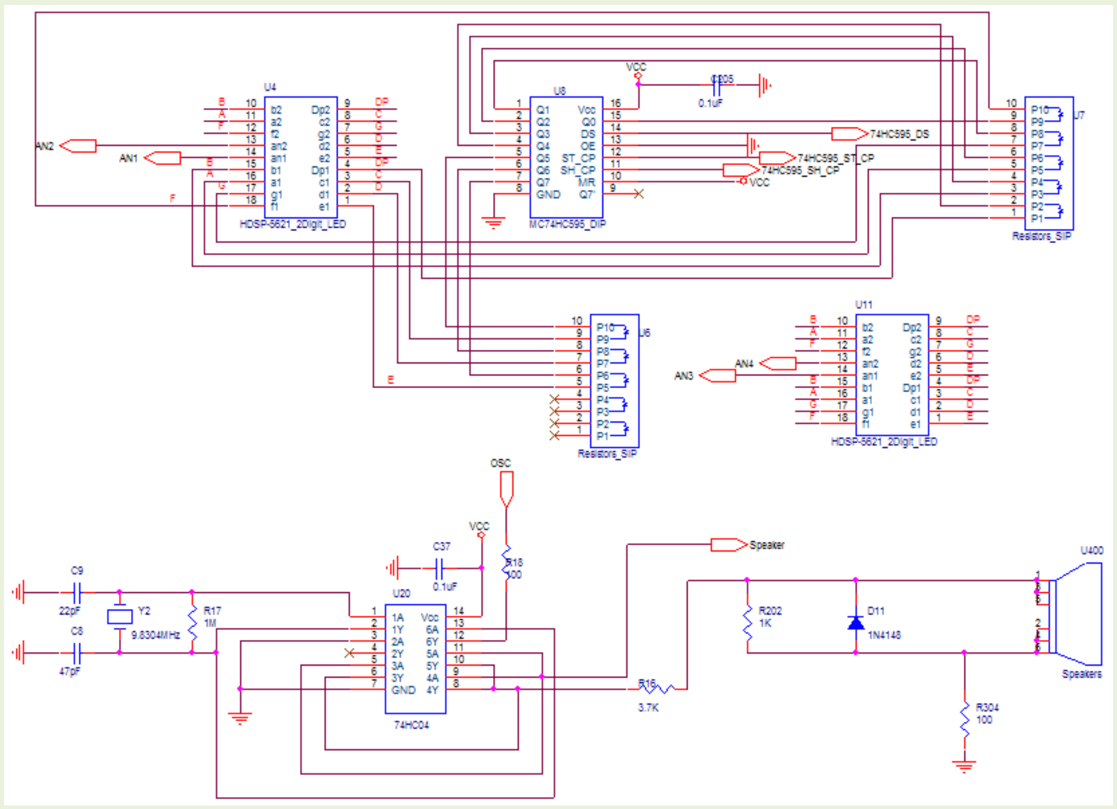
Appendix 1: Evaluation Board Outline



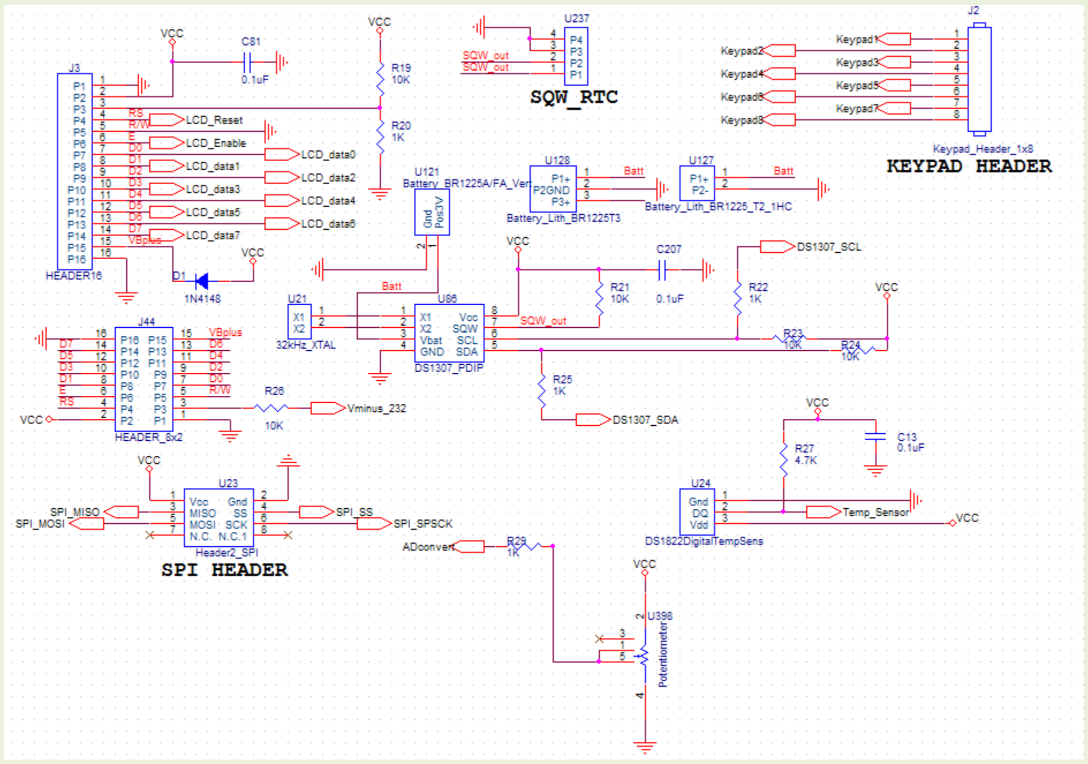
Appendix 2: Schematics







U4
HDSP-5621_2Digit_LED
U8
MC74HC595_DIP
U7
Resistors_SIP
U6
Resistors_SIP
U11
HDSP-5621_2Digit_LED
AN1
AN2
AN3
AN4
74HC595_DS
74HC595_ST_CP
74HC595_SH_CP
VCC
C205
0.1uF
OSC
Speaker
U400
Speakers
U20
74HC04
C9
22pF
C8
47pF
Y2
9.8304MHz
R17
1M
C37
0.1uF
R18
100
R202
1K
D11
1N4148
R16
3.7K
R304
100



SQW_RTC
KEYPAD HEADER
SPI HEADER
J3
HEADER16
C81
0.1uF
R19
10K
R20
1K
LCD_Reset
LCD_Enable
LCD_data0
LCD_data1
LCD_data2
LCD_data3
LCD_data4
LCD_data5
LCD_data6
LCD_data7
D1
1N4148
U237
SQW_out
J2
Keypad_Header_1x8
Keypad1
Keypad2
Keypad3
Keypad4
Keypad5
Keypad6
Keypad7
Keypad8
U121
Battery_BR1225A/FA_Vert
U128
Battery_Lith_BR1225T3
U127
Battery_Lith_BR1225_T2_1HC
Batt
U21
32kHz_XTAL
U86
DS1307_PDIP
C207
0.1uF
R21
10K
R22
1K
R23
10K
R24
10K
R25
1K
DS1307_SCL
DS1307_SDA
J44
HEADER_8x2
R26
10K
Vminus_232
U23
Header2_SPI
SPI_MISO
SPI_MOSI
SPI_SS
SPI_SPSCK
ADconvert
R29
1K
U398
Potentiometer
U24
DS1822DigitalTempSens
R27
4.7K
C13
0.1uF
Temp_Sensor