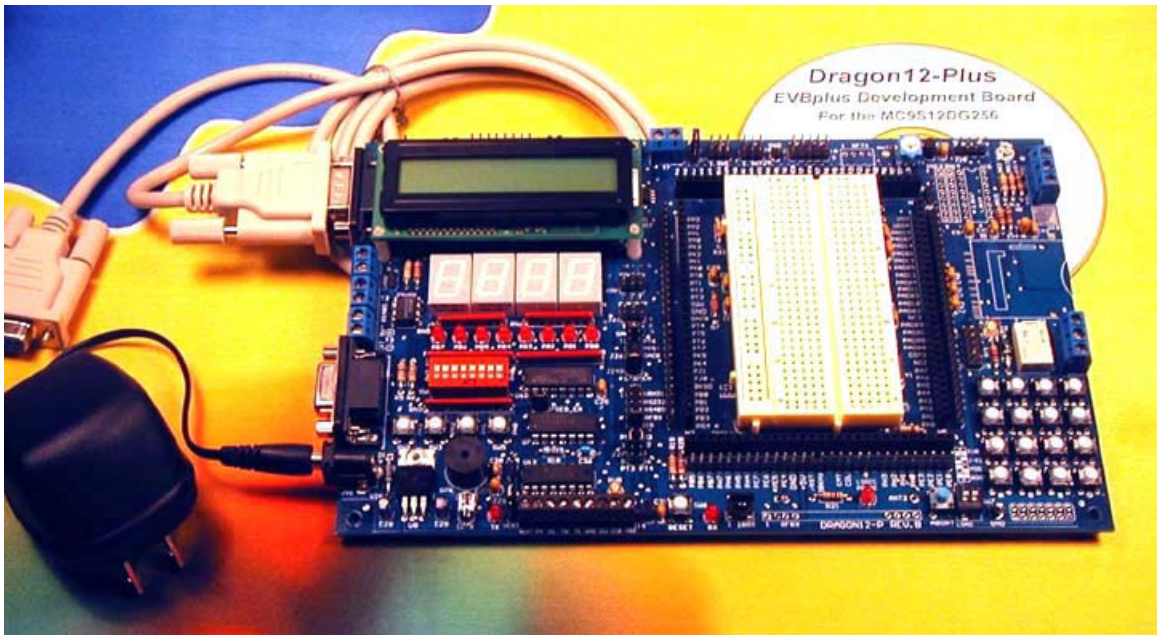
EE 337 Introduction to Microprocessors
Lab Manual V2.2

Wytec HCS12 - MC68HC12

Department of Electrical and Computer Engineering
UAB School of Engineering



Revision History:

Version 1.0 2009-08-20 original by Nicholas Christian

Version 2.0 2013-08-19 revised by Devin Sanders

Version 2.1 2013-12-12 revised by Devin Sanders

Version 2.2 2014-10-24 revised by Pieter Loubser and Jon Marstrander

Table of Contents

Introduction

Welcome to EE337

First off let me welcome you to EE337 Lab! The purpose of this lab is for you to become familiar with a typical microcontroller that is intended for an embedded system. In order to do this you must first become familiar with the fundamentals of the assembly programming language. Once you have a grasp on assembly you will take what you know about the C-language to use the microcontroller for more advanced tasks. This lab is also meant to provide a real world grasp of the ideas studied in the lecture portion of this course. By the end of this lab you will have demonstrated your ability to design, code, and debug a mixed hardware/software solution, albeit very a simple one.

As mentioned before, this is an introductory course to microprocessors. The lab assignments in this course are some-what basic, yet they lay the foundation for much larger and more complex systems. It is the authors' hope that these assignments will spark an interest that will cause you to further explore this subject on your own time.

In this lab you will be using Wytec's Evalutaion Board (PN: HCs12), which is run by an MC68HCS12 microcontroller. Information about this board can be found at http://www.evbplus.com/hcs12.html. Before coming to class you are expected to have at least visited this site once. This site gives you many useful tools such as a user's manual. It is also suggested that you familiarize yourself with the technical manual for the microcontroller itself.

Datasheet:
http://www-ece.eng.uab.edu/GVaughn/EE444/MC9S12DG256.pdf
Manual:
http://www-ece.eng.uab.edu/GVaughn/EE444/dragon12_plus_usb_9s12_manual.pdf
Schematic:
http://www-ece.eng.uab.edu/GVaughn/EE444/DR12P-USB_G_Schematic.pdf

Tools used in this course:

- Wytech Evaluation Board HCS12
- Freescale and Wytech Documentation PDFs
- Freescale Code Warrior IDE
- Example Code
- YOUR BRAIN!!!

Getting to know the HCS12

The DRAGON12-Plus from Wytec has a long list of features:

- CAN controller
- 16X2 LCD display module with LED backlight and it can be replaced by any size of LCD display module via a 16 pin cable assembly
- 4-digit, 7-segment display module for learning multiplexing technique
- 4 X 4 keypad
- Eight LEDs connected to port B
- An 8-position DIP switch connected to port H
- Four pushbutton switches
- IR transceiver with built-in 38KHz oscillator
- RS485 communication port with terminal block for daisy chaining
- speaker, driven by timer, or PWM, or DAC for alarm, voice and music applications
- Potentiometer trimmer pot for analog input
- Dual SCIs with DB9 connectors
- Dual 10-bit DAC for testing SPI interface and generating analog waveforms
- I^2C based Real Time Clock DS1307 with backup battery
- I^2C expansion port for interfacing external I^2C devices
- Dual H-Bridge with motor feedback or incremental encoder interface with quadrature output for controlling two DC motors or one Stepper motor
- Four robot servo outputs with a terminal block for external 5V
- Opto-coupler output
- DPDT form C relay
- Temperature sensor for home automation applications
- Light sensor for home automation applications
- Logic probe with LED indicator
- Fast SPI expansion port for interfacing external SPI devices
- Power-On LED indicator
- Abort switch for stopping program when program is hung in a dead loop
- MC9S12DG256 MCU includes the following on-chip peripherals:
    - 3 SPIs
    - 2 SCIs
    - 2 CANs
    - I^2C interface
    - 8 16-bit timers
    - 8 PWMs
    - 16-channel 10-bit A/D converter
- Super-fast bus speed up to 25 MHz
- The 112-Pins on-board MCU (MC9S12DG256CVPE) with 89 I/O-Pins is included
- BDM-in connector to be connected with a BDM from multiple vendors for debugging.
- BDM-out connector and a 6-in BDM cable is provided to convert this board into a HCS12/9S12 BDM or programmer. No extra hardware needed.
- Female and male headers provide shortest distance (great for high speed applications!) to every I/O pin of the MC9S12DG256

- Comes with AsmIDE and EmbbedGNU IDE under GPL (general public license)
- Pre-loaded with D-Bug12 monitor for working with AsmIDE and EmbeddedGNU
- Or pre-loaded with serial monitor for working with Code Warrior
- Supports source level debugging in C and Assembler without a BDM
- Mode switch for selecting 4 operating modes: EVB, Jump-to-EEPROM, BDM POD and Bootloader
- Auto start user programs when the board is turned on
- Fast prototyping with on-board solderless breadboard
- Many fully debugged, fairly complex 68HC12 program examples including source code, not just "Hello World" type programs
- Includes a hardware test program that simultaneously scans the keypad, plays a song, multiplexes the 4 LED seven segment display, changes display brightness by adjusting the trimmer pot and detects an object by using IR transceiver as a proximity sensor
- DC jack and terminal block for external 9V battery input
- PC board size 8.4" X 5.3"
- DB9 RS232 cable for connecting to a PC serial port
- 110V AC adapter to power the board (US and Canadian orders only)

During this course we will attempt to use as many of these features as possible, but there is no way we will be able to get to them all.

What is expected?

The student is expected to be fully prepared for each assignment. The student should do any necessary research to complete the lab prior to attempting to write code. The textbook is an excellent resource for research and the most complete source of information pertaining to the Dragon12-Plus Demo Board for the purposes of this course. The examples in the textbook are typically just that, examples. Code from the textbook often cannot be copied and pasted into your code and be expected to work for your lab. The student may also find source code on the Internet. As with any type of research, you must ALWAYS reference the sources for any work that you did not do yourself.

The student is also expected to complete some amount of code and perform some testing prior to the start of lab. When asking questions in the lab, the student is expected to have completed some research and to have some knowledge about the topics being covered in the lab. The instructor may require students to THINK about the assignment and how to answer questions on their own. You are encouraged to attend every class so that you do not miss any material that may be covered by your lab assignments. All labs are to be submitted before the end of the assigned lab time or a late penalty will be incurred. Please see the grading section for full grading details.

Students will be expected to have completed a minimum assignment BEFORE coming to lab. The minimum assignment is to answer every question for the lab and to complete any prelab assignment. This work must be shown to the lab instructor at the BEGINNING of the assigned lab time and must be typed; handwritten assignments will not be accepted. Students who do not bring the minimum assignment may be required to relinquish their lab stations to students who have completed the assignments. Students

that do not submit the prelab assignment prior to submitting the lab assignment will automatically lose points for the prelab.

Finally, the student is reminded that academic misconduct is a serious offense and will be enforced in the lab. All students are expected to do their own work and may not share code or discuss the algorithm for the lab with other students. The students may, however, discuss the proper use of tools used in the lab, such as the demo boards, Code Warrior, and program language syntax.

Lab Grading

For each lab you will be given an assignment that must be completed before the end of the assigned lab time. You will then be given a second requirement that must be completed in the presence of the lab instructor, also before the end of lab. This is to verify that you know what you are doing and to discourage academic misconduct. The second assignment will vary from person-to-person. Do not try to copy someone else's code. Anything not submitted by the end of the assigned lab time will be counted as late and 5 points will automatically be deducted. 10 additional points will be deducted for every additional day that an assignment is late, excluding weekends. The lab instructor reserves the right to alter this late penalty as necessary. It is recommended that everyone has the first requirement completed before the beginning of lab in order to avoid late penalties. Keep in mind that there are more students than lab stations in the Microprocessors lab.

You may demo your assignments to the lab instructor or professor at any time prior to the assigned lab time. Please contact your instructor for appointments and availability.

Documentation and formatting are very important parts of writing good code. Good documentation helps you remember what you did or intended to do when looking over old code. It also tells the grader how much you understand the assignment and whether you actually know what you are doing or not. Proper formatting makes your code more readable and presentable to others.

The following rubric is offered as a GUIDELINE for students to understand how their labs are graded. The instructor reserves the right to alter grading procedures at any time if necessary.

Requirement	Description	Percentage of Grade
1st Assignment	This is the assignment that is provided by the lab manual. Strive to have this completed before the assigned lab time.	20
2nd Assignment	This is the assignment that will be assigned after successfully demonstrating your project as explained in the lab manual.	25
Questions	Your labs will have accompanying questions that must also be answered before you submit your lab for grading. Do not forget to answer these questions.	20
Formatting	This has to do with the overall format of your code, including indentation, whitespace, and readability, etc…	10
Documentation	This has to do with your code documentation. Please see the attached source code for examples of how to write good comments in assembly and C.	10
Pre-Labs	You may be expected to turn in a prelab before your lab can be demoed. The prelab assignments are listed in each lab section of this manual.	15
	Total	100

Lab 0 – Introduction to Assembly

Pre-Lab Assignment

Short Description: What does it do?

This may be the easiest lab you have all semester. This lab is meant to introduce you to the basics of Assembly programming. In this assignment you are asked to look at a section of assembly code to determine what it does.

The source code is provided below. You will be asked to demonstrate that you know how to get the demo code running in the simulator and that you understand what the program is actually doing.

Hint: Look at an ASCII table!

```
; Name:   Nicholas Christian/Jon Marstrander
; Program: Lab 0a
; Version: 1.0
; Date Started: August 23, 2008
; Last Update:  NA
; Copyright © 2008, 2014
;
; Description:  This is Lab 0 for EE337.  You are to describe
;  the function of this program.
;
;
        XDEF Entry, main
        ; We use export 'Entry' as symbol. This allows us to
        ; reference 'Entry' either 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

; include derivative specific macros
        INCLUDE 'mc9s12dg256.inc'

; code section
MyCode:         SECTION
main:
Entry:

Init:
        ldaa    #$00    ; clear accumulator A
        ldab    #$00    ; clear accumulator B
Start:                  ; Start of your code

LoadVal:                ; Initialization portion
        ldaa    #$00    ; Load accum. A with a value,
                        ; pick a value, e.g. 0x71
CheckVal:               ; Meat of the program
        cmpa    #$41    ; compare to 0x41
        blt     Linus   ; What happened?
```

```
        cmpa    #$7a
        bgt     Linus   ; What happened?
        ldab    #$01
        bra     Lucy

Linus:
        ldab    #$00    ; What happened?
Lucy:
Over:
        nop
        nop
        bra     Over    ; Why?
        end
```

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) What is the penalty for cheating on your lab assignments?
2) List 3 arithmetic instructions in assembly.
3) What is a BCD? Based on this lab manual, what are the requirements for making a 100 on your lab reports?

Lab Assignment

Short Description: Math, it's what's for dinner!

In this assignment you will be asked to write your own code. Given below is a skeleton that you should use when writing your assembly programs. Be sure to include plenty of comments and the information header at the beginning of the code.

```
; Name:     Your Name
; BlazerID: blazers
; Program:  Lab 0b
; Version:  1.0
; Date:     August 23, 2008
; Description:
;

        XDEF Entry, main
        XREF __SEG_END_SSTACK
        INCLUDE 'mc9s12dg256.inc'

MyCode: SECTION
main:
Entry:
        ;
        ; Your Code Here!!! :)
        ;
        end
```

For this assignment you are asked to do some very simple math. First off you are to load five complex numbers (ex. 2+5i) into memory. You should load the real portion into the first memory slot followed by the imaginary portion in the next location. For example 0x1000 would contain '2' and 0x1001 would contain '5'. You pick these

numbers "at random," either to be initialized by the program, or hand loaded into memory at run time.

Next you are to add the complex numbers that you have put into memory into the accumulators. Accumulator A should contain the real portion of the answer and Accumulator B should contain the imaginary portion of the answer.

Hint 1: Start loading your numbers into memory at 0x1000. Ten locations will be needed, so do not go above 0x100A.

Hint 2: Run this program in the simulator, since there is no user I/O.

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) List the tools that you will use in this lab.
2) List five functions of the demo board that you would be most interested in using in this lab.
3) What is expected of you, the student, prior to the start of your lab time? What did you do to meet these expectations this week?

Lab 1 – Basic Input and Output

Pre-lab Assignment

Explain how PTIH relates to SW2-SW5. Also explain what PORTB relates to.

Hint: Use the schematics and datasheets.

Lab Assignment

Short Description: Press a switch to turn on a light

This assignment is more difficult than Lab 0, but is still rather simple. For this assignment you will be asked to turn on an LED when a button is pressed. Digital inputs and outputs are very important parts of a microprocessor system. Digital I/O allows you to sense certain things like a limit or an emergency shut off.

For this assignment you need to turn on the LEDs associated with PB0-PB3 when switches SW0-SW3 are pressed, respectively. SW0 should turn on the LED associated with PB3 and SW3 should turn on PB0 and so on.

Refer to the 'Detailed Register Map' starting on Page 25 of the 'MC9S12DP256B Device User Guide.' This document can be found with the documentation on the lab PCs. If you cannot find this document, please see the lab instructor or course instructor.

You will need to determine the memory addresses of the following:

DDRB – Data direction register for Port B
PORTB – Input / Output register for Port B
DDRH –Data direction register for Port H
PTH – Input / Output register for Port H

Below I have listed some pseudo-code for you to follow for this assignment.

```
Make Port B an output port
Make Port H an input port
loop forever and do the following:
        read switches on port H
        which switches are on?
        turn on LEDs on Port B if pressed, else turn LEDs off
```

Be sure to only look at the correct bits on Port H (Bit0 to Bit 3).

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) Explain how the binary and hexadecimal number systems were used or could be used to aid in this assignment.
2) Explain how the binary and hexadecimal number systems can be used to modify the input and output values for this assignment.

Lab 2 –Switch Debouncing

Pre-lab Assignment

Explain the purpose of the software delay that is required for this lab assignment. Also explain how the X and Y registers are used in the example code to create a 32-bit number. Explain how this is different than two 16-bit numbers.

Lab Assignment

Short Description: Turnstile Counter

This assignment is more difficult than the last, a pattern we will only vary from once this semester. In this assignment you are asked to model a turnstile counter. Each time the button is pressed (a person walks through), you are to increment the count (number of people that have gone through).

Below is a picture of what happens when a mechanical switch is pressed. As you can see there is some noise. Imagine what would happen if you were trying to count the number of times that a sensor (button) was triggered and this happened. What would happen? The count would be incorrect (especially with fast hardware).

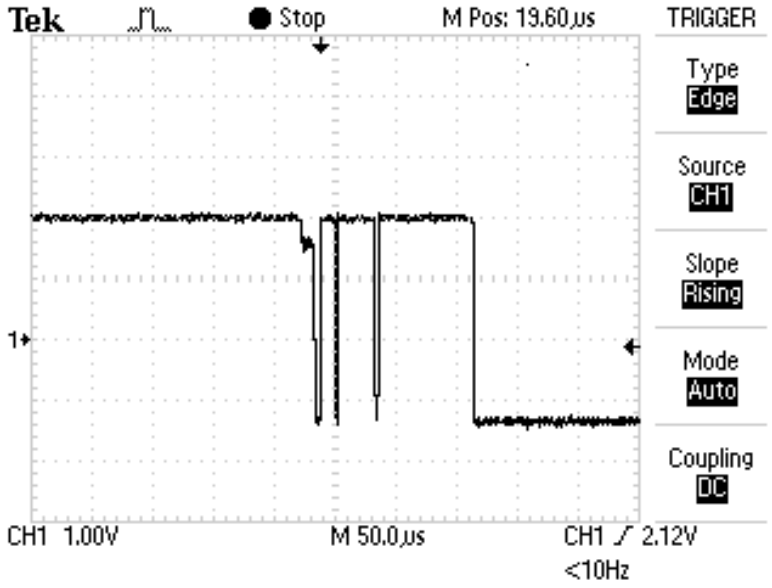


Figure 1. Bouncing Switch

So, how do we fix this problem? There are two ways, only one of which you will really try in this course. We will use a software delay method to remove the bouncing of the switch. To do this we will need to come up with a way to create a delay within our code.

Below is pseudo code example that describes one method of debouncing.

```
while(1)
{
    // start with the switch not pressed

    // wait for the switch to be pressed
    while(switch is not pressed)
        ;
    // switch has just been pressed

    take needed action

    Delay()  // wait while the switch bounces
    // switch should now be fully pressed

    // wait for the switch to be released
    while(switch is pressed)
        ;
    // switch has just been released

    Delay()  // wait while the switch bounces
    // switch should now be fully released
}
```

Well this is great, but how do we implement the delay? We need to write a delay function! So how do we do that? Let us use index registers X and Y as our delay variables (that gives us the ability to use up to 16 bits instead of only 8 bits) since these things are very fast. X is the upper byte and Y is the lower byte. Below is a code example of how to implement a delay subroutine.

```
MY_DELAY:                ; void my_delay( void )
                         ; {
        LDY #$0FFF       ;   for ( y=0x0FFF; y != 0 ; y-- )
DELAY_Y_LOOP:            ;   {
        LDX #$FFFF       ;       for ( x=0xFFFF; x != 0 ; x-- )
DELAY_X_LOOP:            ;       {
        NOP              ;           // NOP
        DEX              ;       }
        BNE DELAY_X_LOOP ;
        DEY              ;   }
        BNE DELAY_Y_LOOP ;
                         ;
        RTS              ; }
```

For this assignment you will need to implement a counter that increments the count when SWX is pressed and clears the count when SWY is pressed. The count is to be displayed via the four LEDs that were used in the last Lab. You get to choose the two switches that increment and clear (these should be chosen from the switches used in the previous lab). Be sure to use the debouncing method covered so that you do not add to your count when you do not need to.

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) Why is debouncing important?
2) How can I debounce in hardware?
3) What other electro-mechanical device might “bounce”?
4) List the number of cycles for each of the opcodes used in your delay function, then sum the total time of the delay in clock cycles.

Lab 3 – Switch Interrupts

Pre-lab Assignment

What is the difference between 'polling' and interrupts? Describe the purpose of each of the following things: PPSH, PIEH, PIFH, SEI, CLI, and RTI.

Lab Assignment

Short Description: Interrupts from switch presses

For this assignment you use interrupts to determine when a button is pressed. You will not use 'polling' like last time.

There is a very important concept associated with interrupts that you MUST understand: DO NOT 'BLOCK' WITHIN AN INTERRUPT.

You will need to write this lab two times: once in assembly code, and once more in C. This exercise will introduce the C language for this microprocessor and show you how the two languages relate to each other. It will also let you see how interrupts can be implemented in each language.

You need to pick one button to increment a binary count on the 8 LEDs on port B, and another button to clear the count on the 8 LEDs. Your program shall modify the value on the LEDs within the interrupt routine. Your interrupt routine needs to use PIFH to determine which button was pushed instead of PTH or PTIH (why?).

Note that if the switch bounces on either press or release, then the counter will change on every bounce. See if you can demonstrate this effect when you run your code.

When setting up the C project, it is very important that you use a SMALL memory model.

Here is some example assembly code that implements an interrupt for a falling edge on Port H Bit 0.

```
; ---------------------------------------------------------
; Program: Example Interrupt Code
; Version: 2014-10-20, Jon Marstrander, original coding
;
    XDEF Entry, main
    XREF __SEG_END_SSTACK
    INCLUDE 'mc9s12dg256.inc'

MyCode: SECTION
main:
Entry:

; ------------------------------

    lds     #__SEG_END_SSTACK    ; set the stack pointer

    bsr     Init_Ports

; ------------------------------

Main_Loop:
    bra     Main_Loop

; ------------------------------
; Initialize ports and interrupt hardware

Init_Ports:

    sei             ; Disable Interrupts

; Setup I/O
    ldaa    DDRH
    anda    #$FE
    staa    DDRH    ; make port H bit 0 an input port

; Setup Interrupts
; See page 244 and 245 for pieh, ppsh, and pifh
    ldaa    PPSH
    anda    #$FE
    staa    PPSH    ; set polarity
    ldaa    PIEH
    oraa    #$01
    staa    PIEH    ; enable interupt on bit 0
    ldaa    #$FF
    staa    PIFH    ; clear all interrupt flags on portH

    cli             ; Enable Interrupts

    rts

; ------------------------------
; Interrupt Routine
```

```
IRQ_ISR:

    ldaa    #$01
    staa    PIFH    ; clear interrupt flags

    rti             ; Return from Interrupt

; ------------------------------
; Vector Table

    org     $FFCC   ; port h interrupt vector see page 145
    dc.w    IRQ_ISR

; -------------------------------------------------------------
```

Here is some example C code that implements an interrupt for a falling edge on Port H Bit 0.

```
// -------------------------------------------------------------
// Program: Example Interrupt Code
// Version: 2014-10-20, Jon Marstrander, original coding
//

#include <hidef.h>        // common defines and macros
#include "derivative.h"   // processor information

void initPorts(void);

// ------------------------------

// NEVER RETURN FROM THE MAIN FUNCTION !
void main(void)
{

    initPorts();

    while(1)
        ;
}

// ------------------------------

void initPorts(void){

    DisableInterrupts;
    DDRH &= 0xFE;  // make port H bit 0 an input port

    // Setup Interrupts
    // See page 244 and 245 for pieh, ppsh, and pifh
    PPSH &= 0xFE;  // set polarity
    PIEH |= 0x01;  // enable interupt on bit 0
    PIFH = 0xFF;   // clear all interrupt flags on portH
    EnableInterrupts;
```

```
}

// ------------------------------

void interrupt VectorNumber_Vporth myISR(void){

    PIFH = 0x01;  // clear interrupt flags

}

// ------------------------------------------------------------
```

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) Why are interrupts important?
2) What does the interrupt vector used for?
3) Why are priorities important?
4) What is an ISR?
5) What is 'Blocking'?
6) Why do you not 'block' within an interrupt service routine?

Lab 4 – Timers

Pre-lab Assignment

In the example code for this lab, what are the main methods of altering the timer frequency? Explain how each method affects the frequency.

Lab Assignment

Short Description: Binary Second Keeper

In this lab assignment we start to use interrupts with timers. Timers are extremely useful in embedded systems

There is a very important concept associated with interrupts that you MUST understand: DO NOT 'BLOCK' WITHIN AN INTERRUPT. It is better to set a flag and return to your main program where you are expected to do the bulk of your processing. The buttons may only be polled within your timer interrupt code. Also the LEDs may only be set from your main code. Not complying with this will result in a zero for the lab.

For this assignment you will be expected to generate a counter on the LEDs that increments and decrements at very nearly 1 count per second. The timer that starts counting when switch zero (PortH_0) has been pressed the first time, pressing the switch again should stop the counter. Pressing switch one (PortH_1) for the first time should let the timer count up, pressing the switch again should make the timer count down. Pressing switch two (PortH_2) should preset the timer to 255 and pressing switch three (PortH_3) should preset the timer to 0. Your counter should stop when it reaches 255 seconds when counting up and it should stop when it reaches zero seconds when counting down.

All switch debouncing must be done within the timer interrupt. However, you can NOT wait within the interrupt!!! Your interrupt must get in, clear any source events, observe the inputs, decide what to do, send any needed messages, and return without delay. You must count the number of times that the interrupt observes the switch in a pressed state, and must store these values in private state variables within the interrupt. When the interrupt observes a fully pressed button, it shall send a button-event message to the main routine, using a safe message-passing interface.

The interrupt routine must also keep track of the passage of time by counting the number of times that it occurs. When a full second has passed, the interrupt must send a one-second tick message to the main routine, using a safe message-passing interface.

The interrupt shall NOT perform any operations on the LEDs.

The main routine must not look at buttons or any private data of the interrupt routine. It may only interact with the shared variables to receive a message from the safe message-passing interface. The main routine shall update the LEDs on Port B to indicate the count value, based on the button event and timer-tick message events.

This lab is not as easy as it sounds because your timers actually has to fire at an interval many times per second, and you must generate counter increments and decrements as close to one second intervals as possible. Please refer to Page 288 of the processors documentation to determine the correct values of the RTI Frequency Divide Rates.

Here is some example code that sets up a particular frequency for the RTI interrupt:

```
RTICTL =  0x78;    //set RTI Frequency
CRGINT |= 0x80;       // enable the interrupt for RTI
```

Here is some example code that implements an interrupt routine for the RTI:

```
void interrupt VectorNumber_Vrti myISR(void){

    CRGFLG = 0x80; //clear interrupt flag

}
```

So, how to write code to keep track of all of this stuff? Here is one way. First, outside of all functions, declare a global state variable:

```
int foo; // private counter state variable for the interrupt
```

Next, in the initialization code (while interrupts are turned off!) clear this variable. This is the only time you should ever access this variable outside of the interrupt itself. Think of this initialization as part of the "constructor" function.

```
foo = 0; // initialize
```

Then, within the interrupt service routine, you can include code like this to make stuff happen only every 6th time through the ISR:

```
foo++;
if( foo >= 6 )
{
    // do something every 6th time
    foo = 0;
}
```

You can repeat this trick multiple times as needed, for different things you need to keep track of. :)

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) What other applications might timers have?
2) What is the speed of the oscillator? (Refer to the schematics!!)
3) What changes would need to be made if the Oscillator speed was changed?

Lab 5 – Analog to Digital Conversion and Serial

Pre-lab Assignment

Write a short, technical paragraph on why you think it is important to learn about serial communication. Remember to cite any sources that you use for research.

Lab Assignment

Short Description: Convert It!

Once again we increase the difficulty of the labs. In this lab you will be given no examples. Since you are given no examples, you must gather all the needed information yourself.

The main portion of this assignment is to us the HCS12 Analog to Digital converter to measure the voltage generated by the potentiometer on the board. The values of voltage range from 0VDC to 5 VDC. Once you have successfully converted the voltage you are to display the 8 most-significant bits of that measured value on the 8 port-B LEDs. The MSB of the converter should appear on the PB7 LED, and a logical 1 should light each LED. Your program should hang in a loop continually repeating the measure-and-display cycle. The potentiometer used in this assignment is designated VR2 on the board. Be sure you set things up so that you read from the input generated by the potentiometer and not the external input.

For the second portion of this lab, you will communicate the measured voltage using the RS-232 serial interface. Each time through the measurement loop, you should check to see if you have received a complete command from the serial port. If you have a complete command, you should send an appropriate response back through the serial port, and then continue with the measure-and-display loop. Your measure-and display loop should not stop running while a partial command has been received. All serial characters received should be echoed back to the serial transmit port as they are received. Valid commands and their appropriate responses are listed in Table 6-1, below.

Table 6-1: Serial Commands for Lab 6

Command	Response
Raw <return> or R <return>	A/D Count = 0x#### <CRLF>
Volts <return> or V <return>	Volts = #.## V <CRLF>
(anything else...)	*** Invalid Command ***<CRLF>

In the Command column of Table 6-1, <return> indicates the "Carriage Return" character. All commands are to be case insensitive, meaning that any combination of upper and lower case letters is valid.

In the Response column of Table 6-1, <CRLF> indicates both the "Carriage Return" and "Line Feed" characters. The '#'symbol represents a numeral as part of a number. The Raw numbers should be a 16-bit, binary value, displayed in hexadecimal. In the Volts response, #.## represents the decimal value of the actual voltage measured.

For a bonus, you can add the option to display the raw binary value on the 7-segment display. In this portion you are expected to toggle between how the voltage is displayed by pressing SW5 (you should remember this from previous labs). The main (default) display is the Port-B LEDs as described above. The secondary display is the Four seven-segment displays units. You are to display the raw digital value given by the ADC unit on this display, in decimal.

Note: When toggling between displays, be sure to clear (turn off) the display you are not using.

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) What is the minimum voltage change that the ADC can detect (the resolution) for a voltage swing of 0VDC to 5VDC? To 10VDC? To 2VDC?
2) What are the maximum voltage input values that the ADC can handle?
3) What should you do if you need to read voltages that are above the maximum voltage inputs?

Bonus Lab – Do The Robot!

Pre-lab Assignment

What is a Servo motor? What is the difference between a servo motor and a DC motor? List 5 other peripheral devices that can be added to a microcontroller. Be specific.

Lab Assignment

Short Description: Robot Arm Controller

This will be the most difficult lab so far. The ability to control peripheral devices makes microcontrollers extremely versatile. There are many different types of peripheral devices and many more variations of each. The peripherals that we will use include a low-current servo motor and a joystick.

For this lab you are expected to write a program that reads the position of the joystick and moves the servo horn to the corresponding position. The position of the joystick will be read with the analog-to-digital converter. The position of the servo will be output using pulse width modulation. The student is expected to do any required research prior to attempting this lab assignment.

Examples from Text

```
//Setup AtD
void initATD(void){
  ATDCTL2 = 0x80;
  Delay();
  ATDCTL3 = 0x0A;
  ATDCTL4 = 0xE0;
}

//Setup PWM
void initPWM(void){
  PWMCLK= 0xFF;        //Set Clock SA as the clock source
  PWMPOL = 0xFF;       //Set waveform to begin high
  PWMCTL = 0x00;       //Select 8-bit PWM
  PWMCAE = 0x00;       //Select left-aligned mode
  PWMPRCLK= 0x04;      //Set clock A prescaler to 16
  PWMSCLA=0x3B;        //Set Clock SA scale factor to 59
  PWMPER5 = 0xFE;      //Set neutral PWM period on PP5
  PWMDTY5 = 0x13;      //Set neutral PWM duty cycle on PP5
  PWEN |= 0x20;        //Enable PWM on PP5
}
```

Pseudo code for I/O tasks

```
//Read Joystick
int readATD(void){
  //begin AtD conversion on desired port
  //wait for conversion to finish
  //read conversion value for desired port
  //return value
```

```
}

//Set Servo Position
void setServo(int position){
  //read AtD conversion value
  //set duty cycle to calculated value for desired PWM wave
}
```

This lab will test your knowledge of C programming much more than the previous labs. You might want to use some reference material. Since this lab is intended to also test your knowledge of C, it is left up to the student to determine how to convert from the voltage reading of the joystick to the required period and duty cycle values for the PWM.

Good Luck!

QUESTIONS TO BE ANSWERED IN COMMENTS AT THE END OF YOUR CODE

1) How does the period and duty cycle of the PWM affect the position of the servo?
2) How do you get the most accurate reading from the AtD converter?
3) How can you adjust the precision of the Servo position?

Appendix A: PWM Generation

November 13, 2013 – Devin Sanders

Introduction

The purpose of this handout is to demonstrate the use of the Dragon12-Plus Evaluation Board for the purpose of pulse-width modulation (PWM). This handout is intended to supplement the course material and provide additional information that is specific to the laboratory exercises. The student is expected to have an understanding of Digital Logic as a prerequisite to this course.

Pulse Width Modulation

Before attempting the PWM lab exercise the student should have an understanding of pulse-width Modulation. Pulse-width modulation is the changing of the pulse-width of a rectangular wave, sometimes referred to as a pulse train. The pulse-width is the amount of time that each pulse lasts. Since rectangular waves are periodic the pulse width can also be expressed as the duty cycle of the waveform, or the percentage of the period that the signal is high.

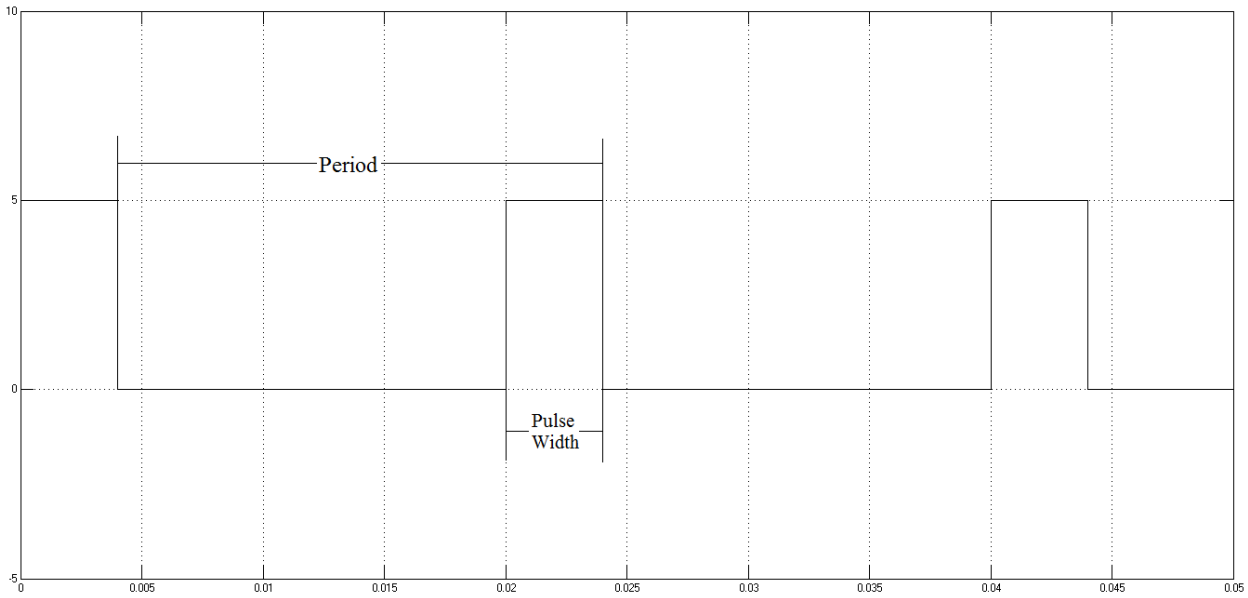


Figure 1. Pulse Train

Dragon12 PWM

This section focuses on the generation of PWM signals using the Dragon12-Plus Demo Board with the MC9212DG256 microcontroller from Freescale. The Dargon12-Plus has seven PWM output channels. The output signal of each channel is controlled by a number of different registers. Chapter 8 of *HCS12/9S12: An Introduction to Software and Hardware Interfacing*, 2nd Edition by Han-Way Huang details the use of these registers to control the PWM waveform.

The Dragon12-Plus generates the PWM signal from the 24MHz E-Clock. The board has three registers that control the clock signal to the PWM counters. The clock select (PWMPRCLK) register selects either Clock A or Clock B for use by the PWM. Each clock signal can be scaled down independently. Each PWM channel can use either Clock A or Clock B. The prescale (PWMPRCLK) and scale (PWMSCLA/B) registers scale down the frequency of the E-Clock to create a PWM clock signal within a desired frequency range.

The prescale register scales the E-Clock by powers of 2 and outputs Clock A and Clock B. The scale register further scales down the output from the prescale by a selectable value from 1-256 then divides again by 2 to output Clock SA/B. This gives the programmer very precise control over the PWM clock signal. The PWM Clock can be calculated from Equation 1.

$$PWM\ Clock = \frac{E_{CLK}}{2^{Prescale}(2 \times Scale)} \quad \text{Eq. 1}$$

The period (PWMPERx) and duty (PWMDTYx) registers are used by the PWM counters to determine the amount of time to generate each pulse. The PWM counters run at the frequency of the PWM Clock and the output flip-flop changes state when the counter matches the duty cycle or period values. Therefore, scaling the E-Clock to generate a slower PWM clock allows the period and duty cycle values to fit within the 8-bit registers, regardless of the desired PWM frequency. Equation 1 can now be expanded to determine the appropriate values for the period (P) and duty cycle (D) by expressing the desired frequency (F) as a ratio (R) of the E-Clock as in equations 2 – 4.

$$R = \frac{E_{CLK}}{F} \quad \text{Eq. 2}$$

$$T = \frac{R}{2^{Prescale}(2 \times Scale)} \quad \text{Eq. 3}$$

$$D = \%T_{High} \quad \text{Eq. 4}$$

To ensure that the desired frequency can be set with an 8-bit number the scale and prescale values should be determined by comparing equation 3 to 2^8 as in equation 5.

$$\frac{R}{2^{Prescale}(2\times Scale)} < 256 \qquad \text{Eq. 5}$$

Any scale and prescale value can be used as long as the calculated value is less than 256. Also note that care should be taken when calculating the duty cycle for the minimum and maximum values as setting the pulse-width too high or too low can cause current spikes that can damage the microcontroller or other parts of the demo board. The duty cycle values should be rounded appropriately to prevent this effect. For example, the maximum duty cycle should be rounded down to the next whole number and the minimum should be rounded up. Any torque on the motor shaft that exceeds the rated stall torque of the servo can also cause the current to spike so the servo horn should not be held or prevented from turning at any time.

Example

This section provides a functional example for creating a PWM signal to control a servo motor using the Dragon12-Plus Demo Board. Servo motors require a 20ms period to maintain a position and the position of the horn can be controlled by changing the pulse-width of the signal. The minimum pulse-width of the servo is 1ms and the maximum is 2ms. A 1.5ms pulse should therefore move the servo horn to a neutral position. Using the previously discussed equations the period and duty cycle can be calculated as demonstrated below.

$$F = \frac{1}{0.020\ s} = 50\ Hz$$

$$R = \frac{24000000\ Hz}{50\ Hz} = 480000$$

$$\frac{480000}{2\times S\times 2^P} < 256 \rightarrow \frac{480000}{2\times 59\times 2^4} \approx 254 < 256$$

This last equation means that the desired 20ms frequency can be generated by using a prescale factor of 4, a scale factor of 59 and a period of 254. Remember that the scale and prescale values can be any value as long the calculated value of this equation is less than 256. The calculated value should be as close to a whole number as possible for greater accuracy. The duty cycle can then be calculated as follows.

$$D = \frac{0.0015}{0.2} = 7.5\%$$

$$D = 0.075\times 254 \approx 19$$

Using Clock SA, 8-bit mode, left-aligned mode and polarity of 1 this PWM signal can be output from PWM 5 on the Dragon12-Plus using the following register values.

PWMCLK = 0xFF
PWMCTL = 0x00
PWMPOL = 0xFF
PWMCAE = 0x00
PWMPRCLK = 0x02
PWMSCLA = 0x3B
PWMPER5 = 0xFE
PWMDTY5 = 0x13
PWME = 0x20

References

1. Huang, Han-Way. HCS12/9S12: An Introduction to Software and Hardware Interfacing. 2 ed. 2006. PP416-441