**Experiment #2: Danger! Keep Right!**

**ECE 367 – Microprocessor Design (Spring 2013)**

**PROFESSOR:** Robert Becker

**T.A.:** Chenjie Tang

MWF – 10:00AM – 11:50PM

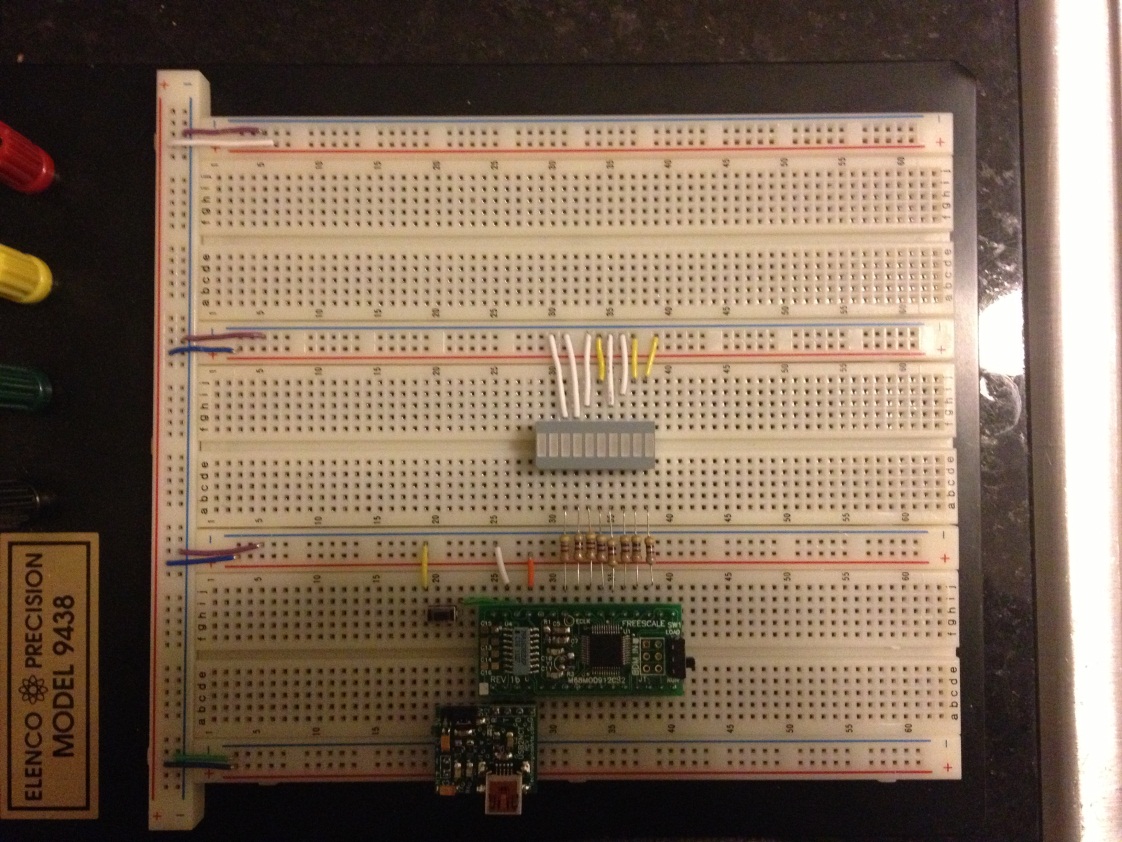
T Lab: 8:00AM – 10:50AM

Prepared by: Mitchell Hedditch

UIN: 677318273

Date Prepared: Sunday, February 3rd, 2013

Date Submitted: Tuesday, February 5th, 2013



1. Logic Diagram
2. Schematic Diagram
3. 9S12 Assembler Program

; University of Illinois at Chicago, Dept. of Electrical and Computer Engineering

; ECE 367 -Microprocessor-Based Design

; Semester: Spring 2013

; Experiment Title: Danger! Keep Right!

; Date: 2/2/2013

; Updated 2/4/2013

; Version: 1

; Programmer: Mitchell Hedditch

; Lab Session: Tuesday 8AM-10:50AM

; Define symbolic constants

PortA EQU $00 ; PortA address (relative to Regbase i.e. offset)

PortT EQU $0240 ; PortT offset (actual address of PortT)

DDRA EQU $02 ; PortA Data Direction control register offset

DDRT EQU $0242 ; Actual Data Direction Register for PortT

INITRG EQU $11

INITRM EQU $10

CLKSEL EQU $39

PLLCTL EQU $3A

CRGFLG EQU $37

SYNR EQU $34

REFDV EQU $35

COPCTL EQU $3C

TSCR1 EQU $46

TSCR2 EQU $4D

TIOS EQU $40

TCNT EQU $44

TC0 EQU $50

TFLG1 EQU $4E

Regbas EQU $0000 ; Register block starts at $0000

;

; The ORG statment below would normally be followed by variable definitions

; There are no variables needed for this project.

; THIS IS THE BEGINNING SETUP CODE

;

ORG $3800 ; Beginning of RAM for Variables

;

; The main code begins here. Note the START Label

;

ORG $4000 ; Beginning of Flash EEPROM

START LDS #$3FC0 ; Top of the Stack

SEI ; Turn Off Interrupts

MOVB #$00, INITRG ; I/O and Control Registers Start at $0000

MOVB #$39, INITRM ; RAM ends at $3FFF

;

; We Need To Set Up The PLL So that the E-Clock = 24MHz

;

BCLR CLKSEL,$80 ; disengage PLL from system

BSET PLLCTL,$40 ; turn on PLL

MOVB #$2,SYNR ; set PLL multiplier

MOVB #$0,REFDV ; set PLL divider

NOP ; No OP

NOP ; NO OP

PLP BRCLR CRGFLG,$08,PLP ; while (!(crg.crgflg.bit.lock==1))

BSET CLKSEL,$80 ; engage PLL

;

;

;

CLI ; Turn ON Interrupts

;

; End of setup code. You will always need the above setup code for every experiment

; Begin Code

; Initialize the 68HC11

LDY #Regbas ; Initialize register base address

; Note that Regbas = $0000 so now <Y> = $0000

; Setup the data directon for PortA and PortT

BSET DDRA,$FF ; PortA pins are outbound

BSET DDRT,$FF ; PortT pins are outbound

BSET PortT,$00 ; Make Sure all PortT pins are low

; Start the program loop

LOOP: LDAA #$A0 ; Load hex 15 ($0F) into accum A

STAA $3800 ; Store 15 into a mem position

LDAA #$05 ; Load 1 in accum A to be used for subtraction later

STAA $3802 ; Store the 1 value in mem

LDX #TABLE ; Initialize index X to beginnng of the table

NXT: LDAA $3800 ; Load accum A with the value stored at $3800 (decreases)

LDAB 1, X+ ; Load accumulator B with the value at X and post increment

CPX #TABLE+17 ; Compare index X to the value

BEQ LOOP ; If we are at the end of our table, then restart the loop

STAB PortT ; Output the results to port t

SUBA $3802 ; Subtract 1 from accum A

STAA $3800 ; Store our new accum A value to mem

STAA $3804 ; Store accum A in mem location #3804 for timer delay change

TIMER: JSR Sec\_Delay ; Jump to subroutine Sec\_Delay

LDAA $3804 ; Load the value from $3804 into accum A

CMPA $00 ; If A is 0 then

BEQ NXT ; Branch to light the next LED

DECA ; If not, decrement accum A by 1

STAA $3804 ; Store accum A in mem location #3804

JMP TIMER ; Continue displaying LEDs

Sec\_Delay:

LDAA #100 ; Outer Loop counter - 1 clock cycle

A1: LDY #100 ; Inside Loop Counter 2 clock cycles

A0: LBRN A0 ; 3 clock cycles \

DEY ; 1 clock cycles | 8 clock cycles in loop

LBNE A0 ; 4 clock cycles /

DECA ; 1 clock cycles

BNE A1 ; 3 clock cycles

RTS ; Return from subroutine - 5 clock cycles

; when we get here we have

; ([(8\*30000) + (2) + (1) + (3)]\*100) + 1 + 5

; 24000606 clock cycles or approx 1 sec.

; Have the Assembler put the solution data in the look-up table

ORG $5000 ; The look-up table is at $5000

TABLE: DC.B $80, $C0, $E0, $F0, $F8, $FC ; Define data to be stored.

DC.B $FE, $FF, $00, $FF, $00, $FF ; i.e. the solutions

DC.B $00, $FF, $00, $FF ; This line includes the end of the blinking and the stop code

; End of code

; Define Power-On Reset Interrupt Vector - Required for all programs!

; AGAIN - OP CODES are at column 9

ORG $FFFE ; $FFFE, $FFFF = Power-On Reset Int. Vector Location

FDB START ; Specify instruction to execute on power up

END ; (Optional) End of source code

; Labels start in the first column (left most column = colunm 1)

; OP CODES are at column 9

; COMMENTS follow a ";" symbol

; Blank lines are allowed (Makes the code more readable)

1. USER MANUAL
   1. This system displays a “Keep Right Arrow” by lighting LEDs from successively from left to right and then blinking all lit LEDs four times.
   2. To operate the counter:
      1. Make sure the system is plugged into power via the USB cord to computer.
      2. To begin the program simply plug into power and press the black reset button on the breadboard (this is the only button available to the user).
      3. The system will continue to cycle as long as it connected to power.
2. Conclusion.
   1. How well does your project meet the specifications?
      1. The project meets all specifications by incrementally lighting all 8 LED’s from left to right.
      2. It exceeds the specifications by decreasing the time between each successively lit LED.
   2. What were the most difficult issues in realizing the system?
      1. My first difficulty was in finding a method to determine when I was at the end of the cycle to restart. Initially I used an indicator in my look-up table. Eventually, saving lines of code, I was able to use the CPX command to directly compare my index X to the location on the table I wanted.
   3. Were you able to add extra features? If so, explain them.
      1. Yes, the time between the lighting of each successive LED decreases by a factor of .03.
   4. What would you have done differently if you were to do this project again?
      1. I don’t know if I would have done anything differently, but I ran out of time to implement the arrow, cover, and ability to reverse direction of the arrow.
   5. What did you learn from working on this project?
      1. I learned quite a bit. I became familiar with different addressing modes, subtraction methods, branching, data storage and retrieval, and comparison operators.