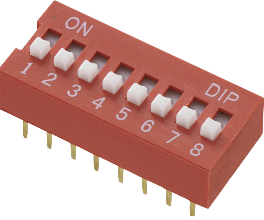
The purpose of this lab is learning the use of switches and pushbuttons and writing a code to use them to turn LEDs on/off.

1. **Pre-lab:**

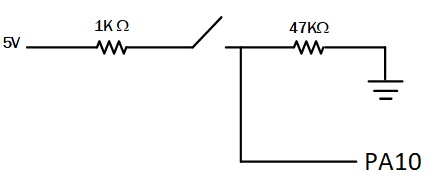
**This must be completed before coming to the lab.**

Write a program to do the following programming tasks.

**PROGRAMMING ASSIGNMENTs:**

**Task 1 Use a DIP switch to turn LED1 on/off**

Write a code to read the status of a DIP switch and turn the LED1 on or off based on the position of the switch. The program should continuously read the switch and echo its status by LED1. Since the board does not have a DIP switch, you need to connect it externally. See slides 3-84 to 3-87 to learn how to connect the switch.

Connect the switch to PA10 as shown in the picture

**Task 2 Use a pushbutton to turn LED0 on/off**

Use the pushbutton SW5 connected to PA6 to turn LED0 on/off. Note that in case of Dip switch, you can read the voltage of the microcontroller’s pin to determine the position of the switch, but in case of pushbutton, the voltage is always zero and it is 5V only momentarily when the button is pressed. So, in your code, you need to define a variable that can take one of two values where one value refers to on and the other one refers to off. You need to continuously read the pin of the pushbutton and if the button is pressed, change the value of variable (from on to off or from off to on), and then based on the value of the variable, turn LED0 on or off. The program should continuously do the above steps so that LED0 always reflect the status stored by the variable.

1. **In lab:**
2. Run the program.

**Task one works? [10 marks]**

Approved: Lab TA \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task two works? [10 marks]**

Approved: Lab TA \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Post lab:**

**Things to turn in as your Lab Report:**

1. The codes of the three programs.

## Lab 6

## Parallel I/O: Lights and 7-segment Display

### Before the Lab

**The most commonly used method of exchanging information between a microprocessor system and the outside world is through parallel I/O. The MC68HC11 micro controller has 5 parallel ports (A, B, C, D, E). Some of these ports have dual functions so they can be used as I/O ports or as special function pins like data or address bus. In the single chip mode all the ports are available. In the expanded mode, ports B and C are used for the data and address bus. In the HC11-VDK board, port D is used for serial communication and the real-time clock. Port A is reserved to be used with the internal timers in the MC68HC11 micro controller. This leaves us with only port E, which can be used either as an 8 analog input to the internal A/D or as a digital input port.**

**Another chip is used with the MC68HC11 micro controller, which is the MC68HC24 Port Replacement Unit (PRU). This chip allows us to restore ports B and C that are used by the data and address bus. Port C can be used as a programmable I/O port which means that we can define which bits to be inputs and which ones to be output. It also supports I/O with handshaking.**

**For this lab, you are going to use port E as an input port and port C as an output port. The address of port E is $100A, so by loading data from this address, you can read the state of the pins of port E. Port C will be used as an output port. To program the direction of the port, you need to write to the data direction register of port C that is located at $1007. Writing a 1 to a bit will set it as output and writing a 0 will set it as input.**

**The I/O devices are located in the satellite board which has 8-bit dip switches, 8 LEDs, two 7-segment displays and an 8-bit D/A. The input port is connected to the dip switches. The output port is connected to the LEDs, the 7-segment displays and the D/A. Jumper JPl on the board will either connect the output port to the LEDs or to the Displays.**

**Seven-segment displays are now widely used in almost all microprocessor-based instruments. A single seven-segment display can display the digits from 0 to 9 and the hex. digits A to F. Each display is composed of seven LEDs that are arranged in a way to allow the display of different digits using different combinations of LEDs. Since the display is composed of LEDs, which need high current to drive them, power consumption is very critical. Consider a panel with 4 displays and the number to be displayed is 8888. Each LED needs 20 mA. So we need a current of 20x7x4 = 560 mA. That's a lot of current compared to the current consumed by the microprocessor. Another problem is the number of components and output bits that are needed to connect the displays to the processor. We need at least 4x7 = 28 resistors and 28 output bits for the 4 displays. Is there a solution for these problems? Yes, there is, it's called MULTIPLEXING!**

**Multiplexing can solve both problems at the same time. By activating only one display at a time, power consumption will be reduced to the amount needed for only one display. Also, the number of components will be reduced by the same ratio since all the displays will share the same data bits and components. But what does multiplexing mean? It means that only one display will have current flowing through it at one time. This means that we need a switch to enable or disable current from flowing through each display. Remember that we cannot use any TTL gate to supply the current since it is much more than its normal output current that it can supply.**

**There are two types of seven-segment displays, common anode and common cathode. In common anode, all the anodes of the LEDs are connected together in a common point. By connecting this point to the supply we can activate the display. The data bits are connected to the cathodes of the LEDs. So, if one of the bits is low; current will flow through the corresponding LED and turn it ON. For the common cathode display, the common cathode is connected to the GND and the bits are connected to the anodes of the LEDs. Remember that we cannot use the output bits directly from the processor to the display since they cannot sink or source enough current. So, we need to use a driver for this job.**

**How can the 4 displays display at the same time? They cannot! But, if the multiplexing frequency is high enough, our eyes will not be able to detect the switching in the displays and they will seem to be active at the same time. Normally, a frequency above 50 Hz will be enough to create this illusion. But since the displays active only for one forth of the time, the intensity will be less than that of non-multiplexed display. Our eyes do the averaging job.**

**For this lab, you are required to write programs that:**

1. **Copy the switch settings to the LED's.  This should be a continuous process, so that when you change a switch status, the result is immediately apparent in the LEDs. You can terminate the loop simply by pressing RESET.  A nicer solution would be to detect a particular switch setting (e.g., all 0's or all 1's) and escape to SWI via a conditional branch.**
2. **Rotate a single lit LED from left to right. Each LED should be lit for a reasonable length of times (200 msec). When the light reaches the boundary of the 8 LEDs, flash all of them then continue.**
3. **Extend the previous program so that the direction of rotation depends on one of the settings of the dipswitches.**
4. **Rotate a pattern obtained from dipswitches, from right to left.**
5. **Display a two digit number stored at location $C300 on the two displays. Remember that each digit needs only one nibble to store it. Your program will use the least significant seven output bits from port C for the display data and the MSB to multiplex the displays. You should have a delay loop for multiplexing the display.**

### In the Lab

**Test the I/O ports using the MM and MD commands in the BUFFALO monitor. Assemble, download and run your programs.**

**Try different frequencies for multiplexing and note the minimum frequency that will give a stable display.**

### After the Lab

* **Generate correct and annotated listings of your programs. Provide details about how you connected the satellite board.**
* **Can we use a NPN transistor instead of the PNP transistor for the seven-segment display? Why?**
* **Estimate the power saving using this method compared to the non-multiplexed method.**
* **Is the minimum multiplexing frequency you have noted realistic? Can you explain it?**

**|** [**EE-218  Homepage**](http://eecs.vanderbilt.edu/Courses/ee218/index.html) **|** [**Syllabus**](http://eecs.vanderbilt.edu/Courses/ee218/Syllabus.htm) **|** [**Schedule**](http://eecs.vanderbilt.edu/Courses/ee218/Syllabus.htm#Topics%20and%20Schedule) **|** [**Lab News**](http://eecs.vanderbilt.edu/Courses/ee218/notices_old.html) **|** [**Faculty**](http://eecs.vanderbilt.edu/Courses/ee218/faculty.html) **|** [**Contact Information**](http://eecs.vanderbilt.edu/Courses/ee218/contact.html) **|** [**Lab Info**](http://eecs.vanderbilt.edu/Courses/ee218/Labs.html) **|** [**Project**](http://eecs.vanderbilt.edu/Courses/ee218/assignments.html) **|**

**Department of Electrical Engineering and Computer Science  
Box 1824 Station B  
Nashville, TN 37235  
Phone: 322-2771  
Fax: 343-6702**

**|** [**Search**](http://www.vanderbilt.edu/search.html) **|** [**Site Index**](http://www.vanderbilt.edu/siteindex.html) **|** [**People Finder**](http://directory.vanderbilt.edu/) **|** [**Phone Directory**](http://phonedirectory.vanderbilt.edu) **|** [**VUnet**](http://www.vanderbilt.edu/vunet/) **|** [**VUmail**](http://www.vanderbilt.edu/vumail/) **|** [**VU Library**](http://www.library.vanderbilt.edu/) **|** [**Help**](http://www.vanderbilt.edu/help.html) **|**

**Last Updated: Saturday, March 05, 2005**

**Juan J. Rodriguez-Moscoso**

absentry entry ;entry point of application

include 'mc9s12dg256.inc'

org $1000

R1 ds.b 1

R2 ds.b 1

R3 ds.b 1

;R1 EQU $1001

;R2 EQU $1002

;R3 EQU $1003

rti\_count ds.b 1

count ds.b 1

IRQ ds.b 1

org $FFF0

dc.w RTI\_ISR ;load RTI ISR vector

; main program

org $2000

entry:

;SetClk8:movb #$01,SYNR ; SYNR = 2 , PLLOSC = 48 MHz

; movb #$0f,REFDV ; REFDV = 0

;movb #$60,PLLCTL ; enable PLL

;pll1: brclr CRGFLG,#$08,pll1 ;wait until PLL locks into the target frequency

; movb #$81,CRGFLG

; movb #$80,CLKSEL ;clock derived from PLL PLLCLK

;movb #$40,CLKSEL ;clock derived from select OSCCLK

;rts

ldaa #0

staa count

staa IRQ

LDAA #$0F

STAA DDRP ;PTP as Output

ldaa #$FF

staa PTP ;turn off 7-seg

LDAA #$FF

STAA DDRJ ;Make PORTJ output, (Needed by Dragon12+)

LDAA #$0

STAA PTJ ;Turn off PTJ1 to allow the LEDs on PORTB to show data (Needed by Dragon12+)

movb #%11111111,DDRB ;set up PT0 for output

loop:

inc count

ldaa count

staa PORTB

jsr Delay

jmp loop

Delay

;PSHA ;Save Reg A on Stack

LDAA #100 ;Change this value to see

STAA R3 ;how fast LEDs Toggle

;--10 msec delay. The Serial Monitor works at speed of 48MHz with XTAL=8MHz on Dragon12+ board

;Freq. for Instruction Clock Cycle is 24MHz (1/2 of 48Mhz).

;(1/24MHz) x 10 Clk x240x100=10 msec. Overheads are excluded in this calculation.

L3 LDAA #100

STAA R2

L2 LDAA #240

STAA R1

L1 NOP ;1 Intruction Clk Cycle

NOP ;1

NOP ;1

DEC R1 ;4

BNE L1 ;3

DEC R2 ;Total Instr.Clk=10

BNE L2

DEC R3

BNE L3

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

; PULA ;Restore Reg A

RTS

RTI\_ISR:

movb #$81,CRGFLG ;clear RTIF by writing a 1 to it.

inc rti\_count

ldaa rti\_count

cmpa #8 ;check if count =8

bne RTI\_done ;if no, we are done

clr rti\_count ;if yes clear count and

;ldaa #$FF

;staa PTP

inc count

ldaa count ; --- togle PT0

;eora #1

staa PORTB

RTI\_done: rti

IRQISR:

inc IRQ

staa CRGFLG

eora #%10000000

ldaa CRGFLG

rti