NAME : _____

DATE : _____

Microcontrollers
CRN: 605-182

SCORE : _____ / **35**

LAB: Utilizing the on-board LCD for the MSP430FG4618

OBJECTIVES:

- ☐ Understand ***memory-mapped hardware*** through the LCD_A controller
- ☐ Learn how ***LCDs*** function through configuring control registers in memory

INTRODUCTION:

A ***L**iquid-**C**rystal **D**isplay*, or **LCD**, is a type of display which uses electrical modulation to alter the path of light traveling through a transparent *liquid crystal* trapped between two layered planes; “liquid crystal” is a substance which has the physical properties that it can both flow like a *liquid* AND arrange its molecules in a common direction like a solid (*crystal*) -in other words, it can have a uniform structure while also being fluid in that structure.

Anywhere there’s a screen, there’s a good chance it’s a LCD: flat-screens, smartphones, hand-held gaming consoles, camera viewfinders, digital signage, wristwatches, and so on. The benefits of a LCD over its predecessor -the ***C**athode-**R**ay **T**ube*, or **CRT**- are its:

1. Energy efficiency, as they are driven by *electrical fields* and NOT *current.*
2. Slim and compact profile, i.e. dimensions.
3. Resistance to *burn-in* or *image persistence*, permanent damage done to a screen by prolonged use.

It should be noted that some displays which LOOK like LCDs are oftentimes another display variety known as *OLEDs*, like those in use by Apple for their iPhone’s “Retina” displays.

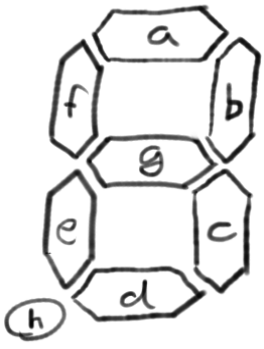
EXERCISE (1) : CONFIGURING THE ON-BOARD LCD

Please begin by starting a new C project in Code Composer Studio. Ensure that the main.c file *looks like the picture below* ; the comments and comment blocks do NOT need to be included, as they are only there to demonstrate the location in our main program where the coding sections from the exercises are to be typed in.

```
#include <msp430.h>

#define     SW1     (P1IN&BIT0)

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // Memory addresses used to print values to the LCD (see memory map)
    unsigned char* mem = (unsigned char*)(&LCDM3);

    // Clear memory addresses beforehand to get rid of junk values
    int i;
    for ( i=0; i<20; i++ )
        LCDMEM[i] = 0x00;

    /*
     * ================ LCD_A CONFIG ================
     */

    /*
     * ================ LCD_A ON ================
    */

    /*
     * ================ LCD_A MEMORY ================
     */


    return 0;
}
```

Associated with each exercise are a set of *technical questions* in *multiple-choice* form and *comprehension questions*, for which you will need to write a *short response*. You will find answers to the technical questions in the text and implement those answers into your code, while the comprehension questions are short-response, evaluating topical knowledge.

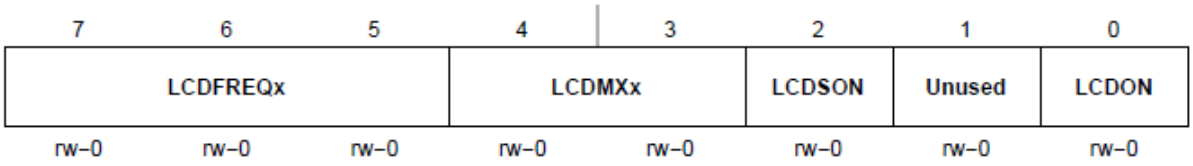
The TI MSP430FG4618 contains functionality to operate an external LCD component or an integrated LCD found directly on the board. This on-board display is a *segmented LCD*, which means that symbols are displayed by turning “on” (opaque) or “off” (transparent) *segments*, usually labeled as segments **a - g** with an optional **h** or **dp**, in order to create a pattern representing our symbol.



Typical layout for a digit in a 7-segment display

First and foremost, we must configure the *master control register* for the LCD_A-controller; the composition of that control register is:

LCDACTL, LCD_A Control Register



7	6	5	4	3	2	1	0
LCDFREQx			LCDMXx		LCDSON	Unused	LCDON
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

It is imperative that we understand two, essential features of this register:

1. This register is at *an address stored in the MCU's memory*, similar to how variables are just a value stored at an address somewhere in memory.
    - The use of the term "register" here is NOT a register in the traditional sense, like those used by the processor; they are, however, used LIKE registers by the LCD_A-controller.
    - These controller registers are said to be *memory-mapped*, meaning they possess a referential-place in memory so that they may ALWAYS be found using that address *mapped* (assigned) in memory.
    - Devices that lack internal registers and instead use registers from memory are called *memory-mapped hardware*.

2. "LCDFREQx," "LCDMXx," "LCDSON," and "LCDON" are known as *flags*.
    - "Flags" are a set of bits responsible for signifying the conditions of a resource in our system, e.g. hardware in our microcontroller.
    - Flags are assigned like any other variable, though the values we assign to them are special BINARY codes which indicate certain *condition parameters*.
    - Flags usually signify important operating conditions; if you assign a value to a flag, make sure it ONLY affects that flag and NOT the entirety of the register.

Please type in the following code *verbatim* in the section "**LCD_A CONFIG"**. You will receive errors due to the presence of "x;" the values of "x" will be the solutions to future technical questions.

```
LCDACTL |= LCDxMUX;
LCDACTL |= LCDFREQ_x;

// Selects COM1-3 pins (COM0 on by default)
P5SEL |= (0x10|0x08|0x04);

LCDAPCTL0 |= x;
```

QC1 (4 pts)
Please describe, in your own words, the relation between flags and memory-mapped hardware:

_____

_____

_____

LCDxMUX

The “MUX” stands for “multiplexing,” which may remind you of a “multiplexer” combinational circuit -this is because every *segment* on the display is associated with both a *segment pin* and a *common (*COM*) pin* on the MSP430, and these pins are *multiplexed* in order to turn on or off segments based on the *combination* of these pins that are ON or OFF.

In its “LCD1MUX” mode, otherwise known as *static MUX*, the LCD_A-controller utilizes **1** segment per **1** pin (the MUX) using only *COM**0***. Given that we only have **36** segments (see MSP430 pinout) to use and **8** segments per *digit*, we get:

$$D = 36 * (S / P) / 8$$

-where “D” = number of digits we can use and “(S / P)” being the segment-to-pin (MUX) ratio. Please do note that there are only FOUR available MUX configurations, ranging 1-4.

QT1 (1 pt)
The most number of full digits we can represent in static MUX mode is:

A. 2
B. 4
C. 16
D. 32

QT2 (1 pt)
What MUX number would we choose in order to have 14 digits possible for display? (Note: you CANNOT have partial digits)

A. 1
B. 2
C. 3
D. 4

Please type in your answer for **QT2** as the missing “x” in **LCDxMUX**.

QC2 (3 pts)
Why do we have different MUX modes? (HINT: find segment pins on data sheet)

_____

_____

_____

_____

LCDFREQx

LCDs are driven to update their contents or ***frames*** on a regular basis using a *clock* signal, whose period dictates the rate at which it updates; this becomes the LCD's *frequency*, which is better known as its ***refresh rate***. The **LCDFREQx** flag selects the LCD's frequency, and it does so as a *function* of the clock running the LCD_A controller, with the value of **LCDFREQx** dictating the *clock divider* value that adjusts the frequency. In other words:

$$f_{LCD} = (\textit{clock speed} = \textbf{32KHz}) \,/\, (\textit{clock divider})$$

-as the LCD_A-controller shares the ***ACLK*** system clock for its timing, running at **32KHz**.

How do we determine what value we want for f_{LCD}? First, we must understand that the thing a LCD is refreshing are its *frames*, whose duration is the TOTAL period it takes to display a COMPLETE *render* (image) to the display. Because of how different MUX modes work, we are unable to display all segments at once and instead have to display *partial renders* of the complete image at any one time; however, due to the human phenomenon of ***persistence of vision***, we are unable to detect these partial renders and thus see only *the complete render altogether*. Interestingly enough, this same effect occurs with fluorescent lights, hence why slow-motion video under these lights seems to produce a *strobing* or flashing effect.

So, in essence, we are using a MUX mode to build **4** partial frames, which involves us spending **2** processor cycles to turn a segment on-then-off. As a convention, flickering tends to be imperceptible in the **30+Hz** for refresh rates. Using those factors, we have:

$$f_{LCD} = (2 \text{ cycles}) * \text{MUX} * (\text{refresh rate})$$

-which, when combining the two equations, yields:

$$(clock\ divider) = \frac{32KHz}{(2\ cycles) * MUX * (refresh\ rate)}$$

QT3 (1 pt)
For a MUX = 4 and a desired refresh rate of 30Hz,
which value below for (clock divider) gives the CLOSEST value to f_{LCD} = 240Hz?

A. 32
B. 128
C. 256
D. 384

Please type in your answer for **QT3** as the missing "x" in **LCDFREQx**.

LCDAPCTL0/1

LCDAPCTL0, LCD_A Port Control Register 0

7	6	5	4	3	2	1	0
LCDS28	LCDS24	LCDS20	LCDS16	LCDS12	LCDS8	LCDS4	LCDS0†
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

† Segments S0–S3 on the MSP430FG461x devices are disabled from LCD functionality when charge pump is enabled.

LCDAPCTL1, LCD_A Port Control Register 1

7	6	5	4	3	2	1	0
Unused						LCDS36	LCDS32
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Both the segment- and common-pins (with the exception of *COM**0***) all share their pins with other ports, meaning that we use these registers to dictate whether the segment pins are PORT pins (logical LOW) or LCD pins (logical HIGH). For any "*LCDSx*" flag, we are able to control the function of groups of segment pins in fours starting from segment pin "x"; therefore, LCDS8 enables segment pins 8, 9, 10, and 11.

Note the caption below LCDAPCTL0: segment pins 0-3 (LCDS0) are NOT enabled when the charge IS enabled; as we are using the built-in charge for the LCD, we will NOT be able to control LCDS0, and we will have to offset our pin usage by **3** in order to access the total number of pins we actually want.

QT4 (1 pt)
*What flags in LCDAPCTL0 do we have to set in order to turn on our first 7 digits? (NOTE: **2** segment pins per digit in 4-MUX)*

A. Flag 3
B. Flags 3, 2, 1, 0
C. Flag 14
D. Flags 4, 3, 2, 1

QT5 (1 pt)
*What code do we need in order to set our flags in LCDAPCTL0 for the previous question **(QT4)**?*

A. 0x14
B. 0xAC
C. 0x41
D. 0x1E

Please type in your answer for **QT5** as the missing "x" value assigning **LCDAPCTL0**.

LCDSON & LCDON

"**LCDSON**" stands for "**LCD S**egments **O**n" and is used to dictate whether any all segments can *render* (display) or not -useful in strobe effects- while "**LCDON**" simply turns the LCD_A-controller itself; they are simply 1-bit flags that accept a logical LOW or HIGH representing OFF or ON, respectively.

QC3 (2 pts)
Why might a designer have allowed for the ability to independently turn off the display, with LCDSON, and the controller, with LCDON?

_____

_____

_____

_____

Please type the following code into the section labeled **"LCD_A ON"**:

```
LCDACTL |= (LCDSON | LCDON);
```

LCDAVCTL0/1

Thankfully, for our purposes of driving the on-board LCD at its most minimal configuration, we do NOT have to do anything with either voltage control 0 or 1, as the *default* configuration (i.e. all flags zeroed/cleared) is the most optimal, given our scenario!

EXERCISE (2) : DISPLAYING DIGITS ON THE LCD

Remember how the LCD_A control registers were referred to as *"memory-mapped hardware*," which meant that we controlled the hardware via values in the MSP430's memory? Well, in a similar manner, the digits and icons found on the integrated LCD are ALSO memory-mapped, meaning that the act of *printing* on the display can be done through assigning values to their associated memory registers:

⋮	⋮	⋮								⋮	⋮
LCDM4	S7	P11	DP2	2E	2G	2F	2D	2C	2B	2A	P12	S6
LCDM3	S5	P13	DP1	1E	1G	1F	1D	1C	1B	1A	P14	S4
		Bit:	7	6	5	4	3	2	1	0		

The above picture is the memory map for ONLY the 4-MUX mode; other modes have different mappings entirely! Of particular note, you will find that each register itself represents **one** digit and its associated bits, the bits of which are the segments used in that digit -in other words, **1** byte for **1** digit, or **8** bits for all **8** segments per digit. Referring back to the segment layout picture at the beginning of **Exercise (1)**, to print characters to the LCD all you need to do is:

1. Find which segments for your digit need to be ON or OFF to represent your character
2. Using the memory map, set/clear bits corresponding to your ON/OFF segments in the associated digit's memory register

Same as before in **Exercise (1)**, copy the following code as it is shown here:

```
// Prints a character in the first digit (0th index is 1st digit)
mem[0] = (0x01 | 0x02 | 0x04 | 0x08 | 0x40 | 0x10);

P1DIR &= ~BIT0;
while(1)
{
    if ( SW1 == 0 )
    {
        // Switch de-bouncing
        for ( i=2000; i>0; i-- );

        if ( SW1 == 0 )
            mem[1] = (0x01 | 0x40 | 0x20 | 0x10);
    }
}
```

Evident by the code, we are writing to the 1st digit (which is associated as the "0th" memory register) the ***bit pattern***, or the binary code associated with a character representation, of the character we wanted printed. Once we have assigned a bit pattern value to the LCD digit's memory, the LCD_A controller will update the screen accordingly and, most importantly, *automatically*. Of note is that the *DP* or "*decimal point*" segment in the memory map corresponds to the "*h*" segment in the segment layout picture at the start of **Exercise (1)**.

QT5 (1 pt)
Please run your code; what is the character displayed on the screen?

A. 0
B. 5
C. 2
D. 7

QT6 (1 pt)
Press SW1 on the MSP430 board; what Is the character on the screen now?

A. A
B. F
C. 5
D. 2

If your code is producing unexpected results, please consult your previous answers for the multiple choice sections (as those answers should have been the ones inserted in your code) as well as double-checking that the code you copied from the previous sections matches the excerpts in those sections. Additionally, make sure your code is in “Free Run” mode after debugging and not caught on a breakpoint or suspended operation.

OT6+7 (8 pts)

For the following question, please perform the following steps:

1. *Choose any two numbers between 1-9 and write them in the spaces below; additionally, pick a digit between 1-7 to display them at (NOTE: your digit place is based on its index in mem’s array and will need to offset your digit’s index by minus 1).*
2. *Write out the segments needed to display these numbers (refer to segment layout diagram on page 2) in the blank space after their respective numbers.*
3. *Write out their binary representation in LCDMEM down below (refer to the memory scheme at the beginning of **EXERCISE (2)**).*
4. *Incorporate these bit patterns into your code by altering the “mem” variable assignments; write down what digit place you expect them to occupy below.*

1ST NUMBER : _____

DIGIT : _____

SEGMENTS :

_____

BIT PATTERN :

_____

2ND NUMBER : _____

DIGIT : _____

SEGMENTS :

_____

BIT PATTERN :

_____

QC4 (12 pts)

After running your code and verifying that the displayed pattern at their corresponding digits matches your answers to the previous section, present your working display pattern to your instructor(s) and receive their sign-off below:

INSTRUCTOR INITIALS : _____