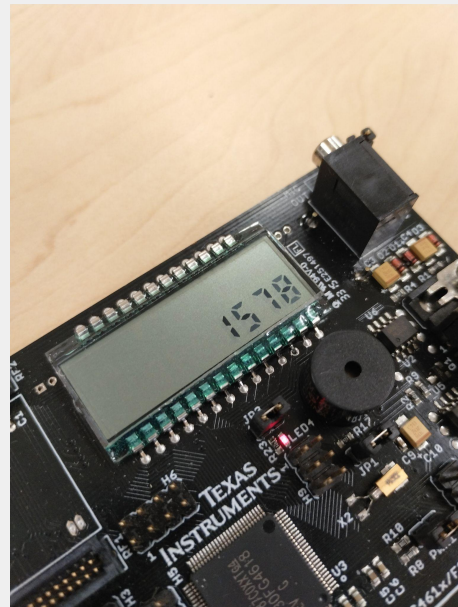


## *Report on:*

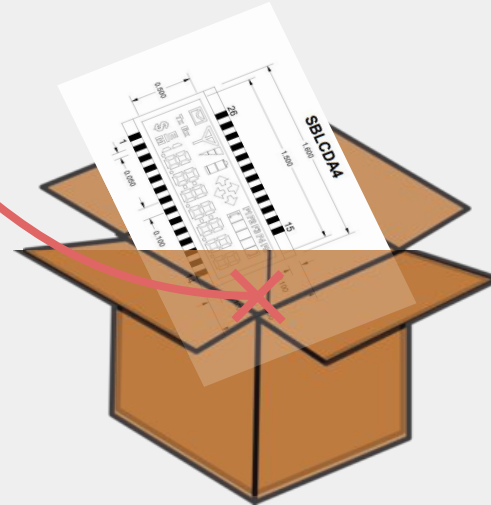
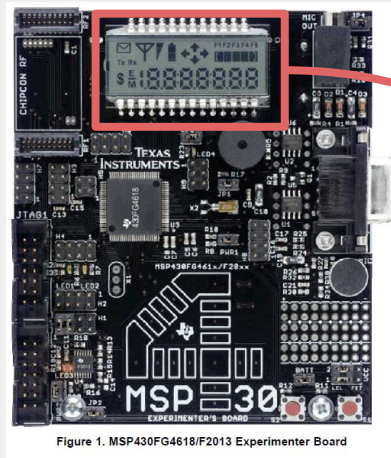
C-Wrapper for the on-board LCD  
on the MSP430FG4618



Original Presentation:  
*Intro*

# C/ASM Wrapper

-for the onboard LCD with the MSP430x.



# Original Presentation: Executive Summary

## What is it?

A **wrapper** is an **instance** which **encapsulates** another **instance** in order to not directly expose the **internal components** of that inner-instance while **providing a means to access it**.

It is analogous to an **ATM**, which provides a way to **interface** with the **bank's services** and your **account** while not having **to be inside the bank** itself.

## What's the problem?

```
int n;
for (n=0;n<LCD_SIZE;n++){
    // initialize the segment memory to zero to clear the LCD
    // writing a zero in the LCD memory location clears turns
    // off the LCD segment
    // Including all of the special characters
    // This way or
    *(LCDSeg+n) = 0;
    // LCDSeg[n]=0;
}
// Port 5 ports 5.2-5.4 are connected to com1, com2, com3 of LCD and
// com0 is fixed and already assigned
// Need to assign com1 - com3 to port5
P5SEL = 0x1c; // BIT4 | BIT3 | BIT2 = 1 P5.4, P.3, P5.2 = 1
// Used the internal voltage for the LCD bit 4 = 0 (VLCDEXT=0)
// internal bias voltage set to 1/3 of Vcc, charge pump disabled,
// page 26-25 of MSP430x4xx user manual
LCDAVCTL0 = 0x00;
// LCDs28-LCDs0 pins LCDs0 = lsb and LCDs28 = MSB need
// LCDs4 through LCDs24
// from the experimenter board schematic the LCD uses S4-S24,
// S0-S3 are not used here
// Only use up to S24 on the LCD 28-31 not needed.
// Also LCDACCTL not required since not using S32 - S39
// Davis's book page 260
// page 26-23 of MSP430x4xx user manual
LCDAPCTL0 = 0x78;
// The LCD uses the ACLK as the master clock as the scan rate for
// the display segments
// The ACLK has been set to 32768 Hz with the external
// 32768 Hz crystal
// Let's use scan frequency of 256 Hz (This is fast enough not
// to see the display flicker)
// or a divisor of 128
// LCDREQ division(3 bits), LCDMUX (2 bits), LCDSON segments on,
// Not used, LCDON LCD module on
// 011 = freq /128, 11 = 4 mux's needed since the display uses for
// common inputs com0-com3
// need to turn the LCD on LCDON = 1
// LCDSON allows the segments to be blanked good for blinking but
// needs to be on to
// display the LCD segments LCDSON = 1
// Bit pattern required = 0111 1101 = 0x7d
// page 26-22 of MSP430x4xx user manual
LCDACCTL = 0x7d;
}
```

Attempting to utilize the onboard LCD requires a decent amount of overhead dedicated to not only understanding the esoteric inner-workings of the LCD but also a significant amount of time just to configure it, let alone operating/driving it.

While an undeniably useful feature -provides feedback on status, produces a more effective HMI, allows for rapid prototyping-learning to program the LCD could be as much of a project as whatever we are trying to implement which incorporates it.

# Original Presentation: Executive Summary cont.

## What's the fix?

A **wrapper** that **encapsulates** the **configuration and operation/driving** of the **onboard LCD**, abstracting the more esoteric, low-level and machine-oriented programming of the LCD with a **more human-readable, high-level interface** which **provides greater quality-of-life and ease-of-access**.

## What's the timeline look like?

### R&D: (~4-5 hrs)

- Preliminary Research
- Block Diagram
- API Specification
- Final Report

### Coding: (20+ hrs)

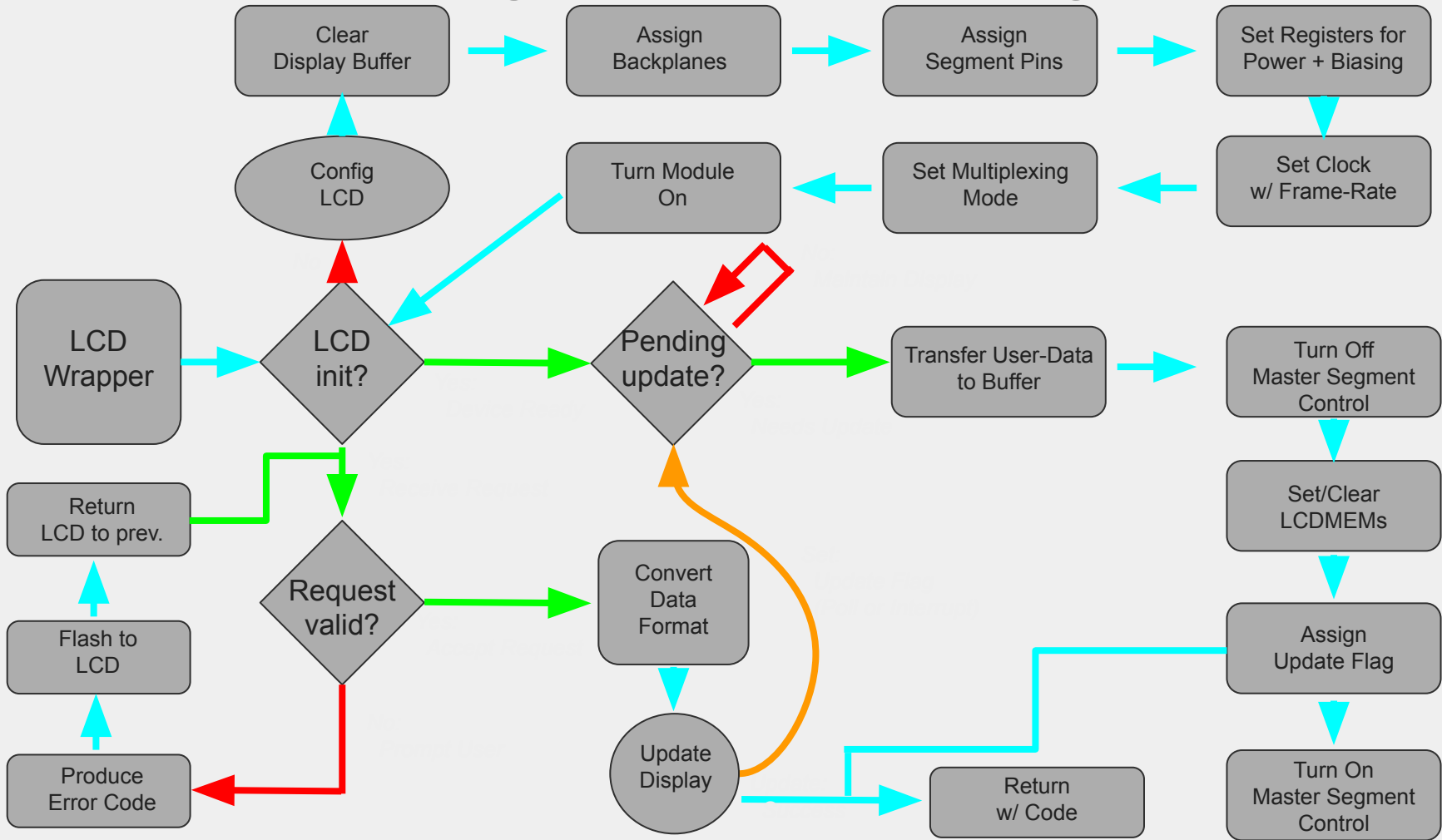
- Configuration/Initialization (2 hrs)
  - LCD set-up
  - URI abstraction
  - state variables (data structures)
- Request-Handling (4 hrs)
  - request validation
  - access/mutation functions
  - returns "cleanly"

### Debugging: (~2-4 hrs)

- Exception-Handling
- Optimization
- "Bulletproofing"
- Operates on User-Data (12+ hrs)
  - data conversion to usable format
  - stores display data
  - drives LCD (refreshes old/new data)
- Exits "cleanly" (2 hrs)
  - program end/device reset returns to well-defined state

COM:			3	2	1	0	3	2	1	0	LCD MSP430		
display	MSP430	LCD	$S_{n+1}$				$S_n$				pin	pin	
memory	pin	pin									P25	S24	
LCDM13	S25	P26	MEM	MIN	ERR	DOL	8BC	RX	TX	ENV			
LCDM12	S23	P24	A0	A1	A2	ANT	BB	B0	B1	BT	P23	S22	
⋮	⋮	⋮									⋮	⋮	
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			

## Original Presentation - *Block Diagram*



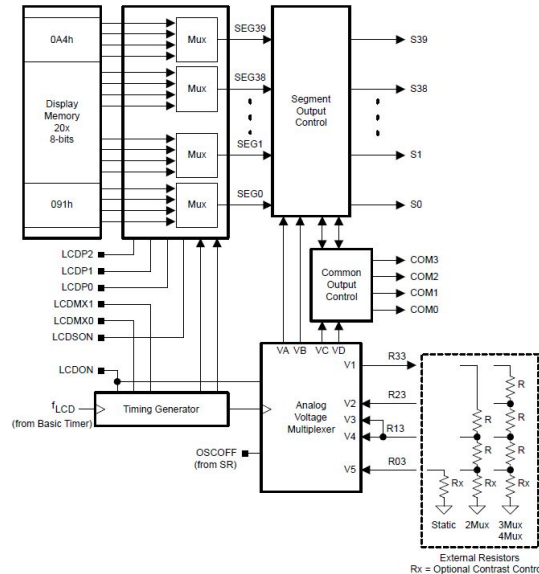
## Original Presentation - *Risks*

Priority	Description	Mitigation
#1	C/ASM are <i>functional</i> languages: they lack the visibility modifiers found in <i>OOP</i> languages like C++ to prevent unintended tampering.	1.) User-Trust 2.) “Adapter Pattern” Architecture 3.) System Access Token
#2	I have never written an API before nor code that was a component for integration as opposed to an application entirely.	1.) Always a first time for everything 2.) Look at public GitHub repos for examples on how to structure
#3	Development time is truly ambiguous due to the lack of thorough documentation on the LCD. Additionally, an API specification needs to be written to make it even useful to others.	1.) Browse through header files, especially <intrinsics.h> 2.) Source other educational institutions whom use the MSP430.
#4	I am essentially making a C++ “stream” for stdio; complexity depends on robustness.	1.) Start with tracking register updates; move to character buffers.
#5	I do not have access to the MSP430 for more thorough testing outside of school.	1.) Find if emulation for LCD works on CCS and have on PC. 2.) Test-Driven Development and/or shell

# Original Presentation: *Material List*

*So what's needed?*

Figure 25-1. LCD Controller Block Diagram



Time with the board; time with the codebase.

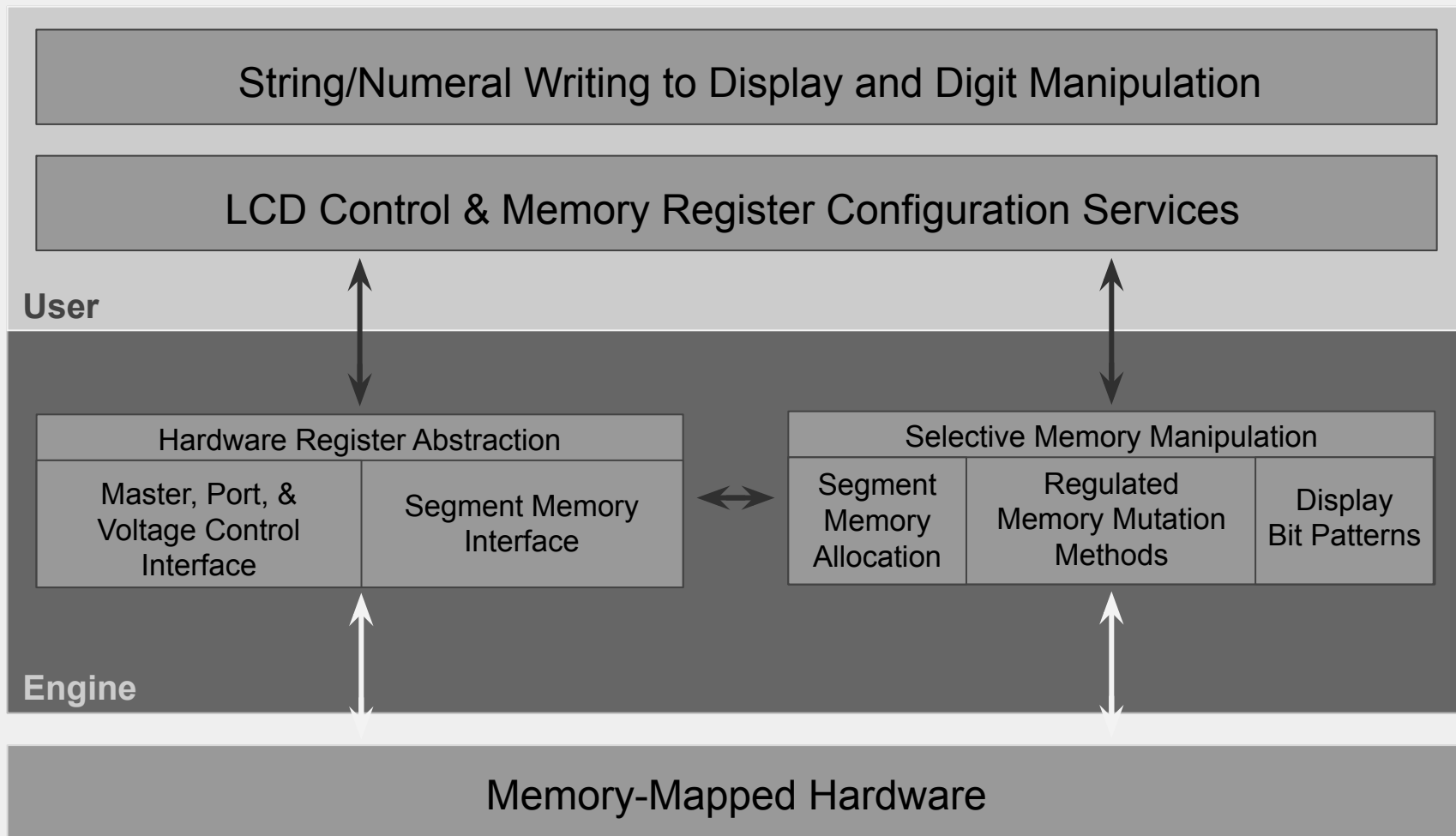
All of the hardware needed is integrated into the board:

LCD A

1. LCD
2. Clock ← ACLK
3. Chain of Resistors ←
4. Variable Resistor (optional)
5. Charge Pump ← Inter-changeable
6. Intermediate Bias Selection

While driving the onboard LCD requires repurposing other board-integrated features, difficulty lies in properly managing the control registers for these, protecting them for user-disruption, and making it extensible to driving external displays.

# Block Diagram





# Schematic

## MEMORY INTERFACE

```

LCD_MEM  mems[MAX];
LCD_DIG*  dig;

m_init(LCD_MEM*);
m_all(bool val, LCD_MEM*);

lcd_all(bool val,
        unsigned int start, unsigned int end);

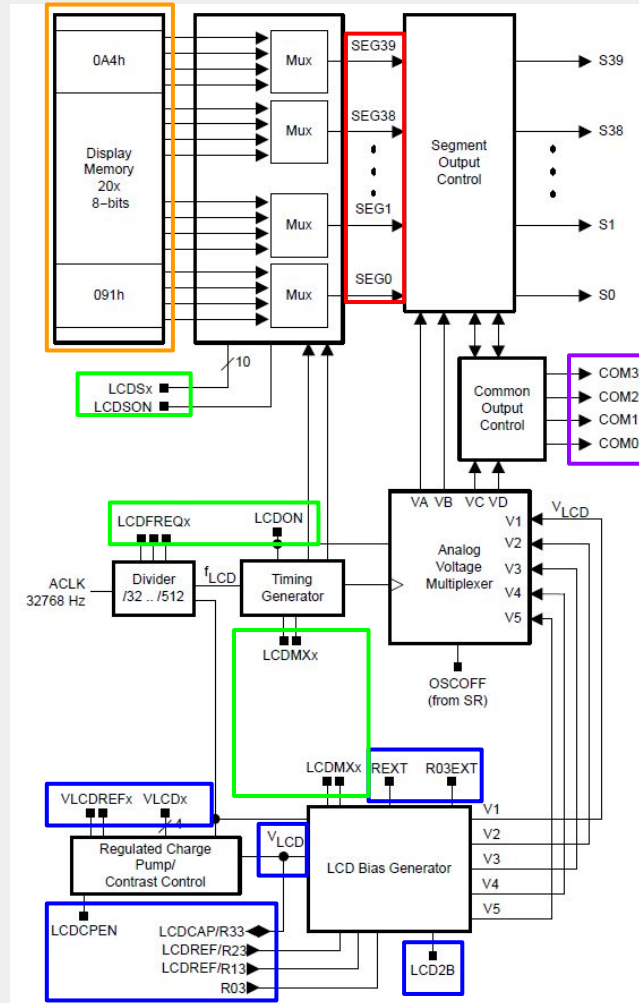
rwrite(DIGIT d, NUMBER n);
write(const unsigned char* c,
      unsigned int len);
writeNum(unsigned int num)
    
```

## LCD CONFIG

```

volatile LCDACTL_REG*  ctrl;

bool lcd_freq(unsigned int f);
bool lcd_mux(unsigned int m);
bool lcd_segsOn(bool t);
bool lcd_on(bool t);
    
```



## SEGMENT PINS CONFIG

```

volatile LCDAPCTL0_REG*  port0;
volatile LCDAPCTL1_REG*  port1;

segPins(unsigned int pin,
        bool val, bool cascade);
    
```

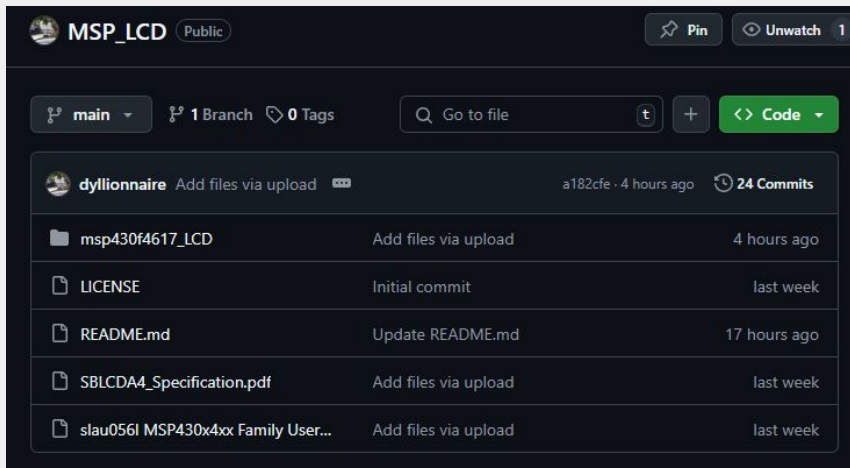
## BACKPLANE CONFIG

```
void lcd_init(void);
```

## VOLTAGE CONFIG

```
void lcd_init(void);
```

# Code



The screenshot shows the GitHub repository page for 'MSP\_LCD' by user 'dyllionnaire'. The repository is public and has 1 branch and 0 tags. The file list includes:

File	Commit Message	Time
msp430f4617_LCD	Add files via upload	4 hours ago
LICENSE	Initial commit	last week
README.md	Update README.md	17 hours ago
SBLCD44_Specification.pdf	Add files via upload	last week
slau056l MSP430x4xx Family User...	Add files via upload	last week

[https://github.com/dyllionnaire/MSP\\_LCD/tree/main](https://github.com/dyllionnaire/MSP_LCD/tree/main)

According to Git, between code additions and deletions, a total of 1,673 lines of code were written, with a final codebase of 583 lines.

For reference, it takes 64 lines of 9-point font to fill a page; thus, it would take 10 pages to print the header file alone!

```
/*
    Asserts that all segments at a memory address are either set or cleared
*/
bool m_all(bool val, LCD_MEM* mem)
{
    if ( !mem
        &&( (TOP < (unsigned char*)(mem->dig))
          &&( (BASE > (unsigned char*)(mem->dig)) )
        )
        return false; // Invalid address

    mem->dig->reg = val ? 0xFF : 0x00;
    return true;
}

/*
    Maps the next available memory address for the LCD segment map to the parameterized pointer
    RETURNS:
        - "false" = no more memory addresses for LCD available
        - "true" = pointer parameter has been allocated a memory address and had its value
    NOTES:
        - (?) change return type to 'int' for distinct numeric error-codes, assert '0' as clear
*/
bool m_init(LCD_MEM* mem)
{
    static unsigned int count = 0; // used as offset for LCDMEM address

    // Check for available addresses (cannot exceed 20)
    if ( count >= ((TOP - BASE) + 1) )
        return false;

    // Assert the LCDMEMx address as the LCD_DIG custom-type to provide more streamlined inter
    mem->dig = (LCD_DIG*)( BASE + count );
    mem->id = count++;

    // According to User Guide, the initial state of the memory registers are "unchanged;"
    // thus, they are cleared to render display blank.
    m_all(0, mem);

    return true;
}
```

*Code excerpt handling memory allocation*

## Known Issues

<i>Issue</i>	<i>Description</i>	<i>Mitigation</i>
#1	No support for other display icons beyond the “7” digits in 7.1 segments.	Alter macros for MEMTOP and MAX as variables; make bit-field for memory mapping of icons for each register.
#2	Device only operates in 4-MUX mode; thus, if the segment pins are used elsewhere functionally, code is inoperable.	Acceptable under some circumstances; otherwise, would need an event listener to swap pin function as needed.
#3	No support for external displays.	While no function support, the memory interfaces are fully usable, assuming proper address assignment.
#4	Only current supports unsigned integers without formatting (does not add indicators of binary or hex value).	Would need to check for signed bits and prepend a minus sign bit pattern; type check for floating point; mode check for representation.
#5	Only accepts <i>literal</i> write arguments; does not support <i>symbolic</i> writes, e.g. binary, BCD, hex.	Incorporate mode flag into numeral parser, decode accordingly.
#6	No formatted printing intrinsically supported; only right-aligned writing with digit persistence for non-overwritten characters.	Current options are available though hacky at best; implemented ranged printing with padding for formatting

## Parts List

Only two:

- ❏ MSP430FG4618

- ❏ *Unsure of how well the code works with rest of MSP430Fx4x family, as the code has proven to be both compiler and platform dependent.*

- ❏ Header File

- ❏ [https://github.com/dyllionnaire/MSP\\_LCD/tree/main](https://github.com/dyllionnaire/MSP_LCD/tree/main)
  - ❏ *Add to project folder alongside main.c*

```

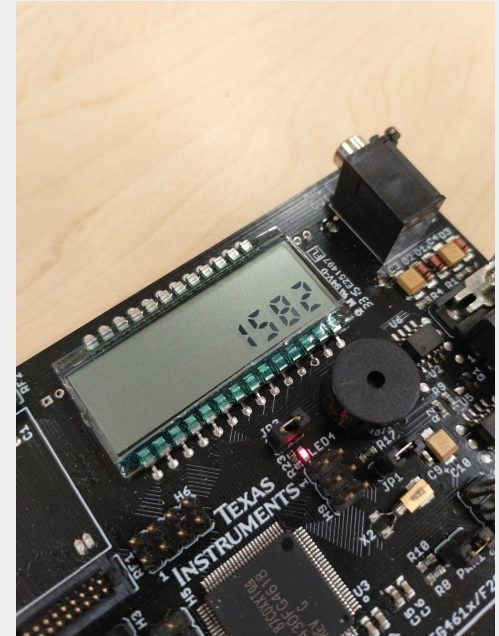
1 #include <msp430.h>
2 #include <lcd.h>
3
4 union RTC_CTL
5 {
6     unsigned char reg;
7     struct
8     {
9         unsigned char RTC_FG : 1;
10        unsigned char RTC_IE : 1;
11        unsigned char RTC_TEVx : 2;
12        unsigned char RTC_MODEx : 2;
13        unsigned char RTC_HOLD : 1;
14        unsigned char RTC_BCD : 1;
15    } flags;
16 };
17 typedef union RTC_CTL RTC_CTL;
18 volatile unsigned char* RTC_CTL_ADDR = (unsigned char*)(0x0041);
19
20 unsigned int MINUTES = 0;
21
22 /**
23  * Seconds-based counting program to utilize the custom wrapper object for
24  * in order to create an animated element for use in a presentation of the
25  *
26  * Utilizes the RTC-counter functions, allowing for a total number of counts
27  */
28 int main(void)
29 {
30     WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
31
32     lcd_init();
33     lcd_all(0,1,MAX);
34
35     volatile RTC_CTL* ctrl = (RTC_CTL*)(RTC_CTL_ADDR);
36     ctrl->flags.RTC_HOLD = 1;
37
38     ctrl->flags.RTC_FG = 0;
39     ctrl->flags.RTC_IE = 1;
40     __bis_SR_register(GIE);
41
42     ctrl->flags.RTC_BCD = 0;
43     ctrl->flags.RTC_TEVx = 0;
44     ctrl->flags.RTC_MODEx = 3;
45
46     ctrl->flags.RTC_HOLD = 0;
47
48     RTCNT2 = 0;
49     RTCNT1 = 0;
50     while(1)
51     {
52         writeNum( MINUTES + RTCNT1 );
53
54         return 0;
55     }
56
57 #pragma vector = BASICTIMER_VECTOR
58 __interrupt void BASICTIMER_ISR (void)
59 {
60     MINUTES = 60*RTCNT2;
61 }

```

## Demo

Since the project doesn't have necessarily flashy product, in order to display the supported character range, a real-time clock has been configured in order to count seconds and print that count to the LCD using the built-in writeNum() function.

As evidenced from the main.c associated with the presentation, the wrapper only needs two lines at minimum to print.



# Revision Ideas

<i>Idea</i>	<i>Description</i>
<b>#1</b>	Allow for signed integers: this would require either losing a digital of representation to represent a minus sign that prepends to the numeral or utilizes the minus sign segment on the LCD display already (this, however, is fixed to the opposite end of the LCD from the 7-segment region and thus not relative like the aforementioned).
<b>#2</b>	Allow for floating point numbers: easiest implementation using current functions and no refactoring would be to use fixed-point arithmetic; this, however, brings into question the level of precision for the floating point allowed, as you are only allowed, with 7 digits to have 7 significant figures. Could have scrolling animation to allow for full display, but this would require a display buffer to allow for sequential writes for new frames as well as a timer function to update.
<b>#3</b>	Allow for non-7-segment icon display: the LCD has a whole host of icons currently not supported functionally (though still accessible with some “hacking” of the wrapper). This could then have it’s only tailored printing function, depending of the use case for the icon (e.g. the “battery” icon for the LCD could be operated using a function that allows for setting a reference value in a static local variable and then allowing for subsequent invocations using a read value, displaying the proper amount of “battery” segments depending on the percentage of reference-to-read).
<b>#4</b>	Animation support: could allow printing functions for strobe-like flashing, text scrolling (allow for “rotations” from ends of display to “wrap-around” or not), ranged animation (allow for animation which selectively affects certain digits), contrast controls (fade-in/fade-out), animation buffers for printing sequential frames, etc.
<b>#5</b>	Allow for more functional control over voltage registers to adjust display settings. Currently, the viewing angle is kept to the default properties of the LCD (~80° @ ⅓ bias w/ AVcc) as described in the datasheet.
<b>#6</b>	Allow for external component integration for the onboard LCD with functional support and configuration native to the wrapper as opposed to being possible but not directly supported -would free internal component and power consumption of MSP430 itself.

**Rubric**

- Complexity Level [1...10] \_\_\_\_\_
- Proposal [1...20] \_\_\_\_\_
- Project Meets Course Objectives [1..20] \_\_\_\_\_
- Clear Functional Description of Design [1..20] \_\_\_\_\_
- Effort (team size, outside of class effort) [1..10] \_\_\_\_\_
- Report [1...20] \_\_\_\_\_