Lab 7. OLED Device Driver for the SSD1306 (Spring 2021)

Videos see end of eBook chapter 17 for Lab videos

Preparation

Purpose

System Requirements

Procedure

- Part a - Build and Test the OLED Hardware
- Part b - Write and Test the low level I2C Driver
- Part c - Write and Test the OutString function
- Part d - Write and Test the IO functions
- Part e - Write and Test the LCD Driver

Demonstration

Deliverables

Hints

Videos see end of eBook chapter 17 for Lab videos

http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C17_Local.htm

Preparation

Read all of Chapter 7
Read the data sheet for the LCD display SSD1306.pdf EE319K web page
SSD1306_4C123 starter project
Lab7_EE319K_SSD1306 starter project

Purpose

This lab has these major objectives:

1. to interface an OLED interface that can be used to display information on the embedded system;
2. to learn how to design implement and test a device driver using busy-wait synchronization;
3. to learn how to allocate and use local variables on the stack;
4. to use fixed-point numbers to store non-integer values.

Lab 7 does not have an automatic grader. There is no simulation for the SSD1306

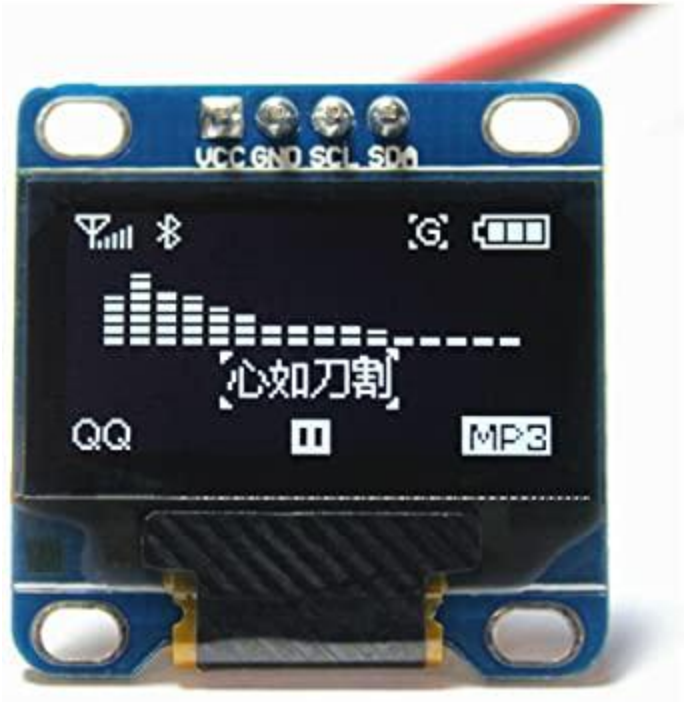


Figure 7.1. SSD1306 128 by 64 pixel black and white OLED.

The EE312H/EE319H group (Valvano's class in Spring 2021) must use **recursion** for LCD_OutDec.

System Requirements

In this lab you will create an embedded system that:

- Interfaces to the SSD1306 OLED using busy-wait
- Displays numbers in decimal format
- Displays fixed point numbers in decimal format

Procedure

The basic approach to this lab will be to first develop and debug the low level output function to the OLED using the actual board. During this phase of the project you will use the debugger to single step through your low level function. After the low level software is debugged, you will be able to see character and graphic output on the OLED. The second step is to develop and debug high level output functions to the OLED. There are many functions to write in this lab, so it is important to develop the device driver in small pieces. One technique you might find useful is **desk checking**. Basically, you hand-execute your functions with a specific input parameter. For example, using just a pencil and paper, think about the sequential steps that will occur when **`LCD_OutDec`** or **`LCD_OutFix`** processes the input 187. Later, while you are debugging the actual functions on the simulator, you can single step the program and compare the actual data with your expected data.

Part a - Build and Test the OLED Hardware

In this lab you will interface a SSD1306 OLED to the TM4C123 as shown in Figure 7.2. When wiring your OLED, look at the pin names and not the pin numbers. There are four possible I2C ports we could have used. EE319K however will use I2C3 on PD0 and PD1.

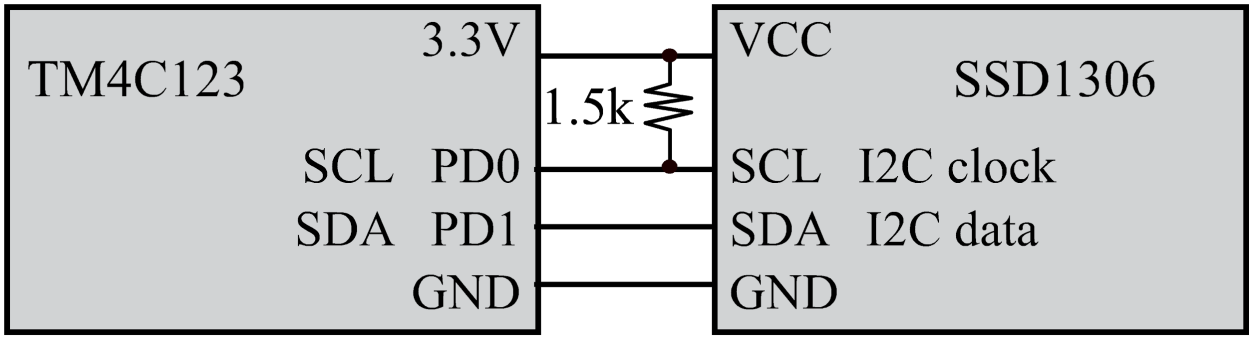


Figure 7.2. Interface connections for the SSD1306.

For LED heartbeats, you can use PF1 PF2 PF3 LED on the LaunchPad, For switch input you can use PF0 or PF4. Be careful not to also use PB6 and PB7 without first removing the R9 and R10 resistor. See Figure 1.13.3 in the book, also shown as Figure 7.3.

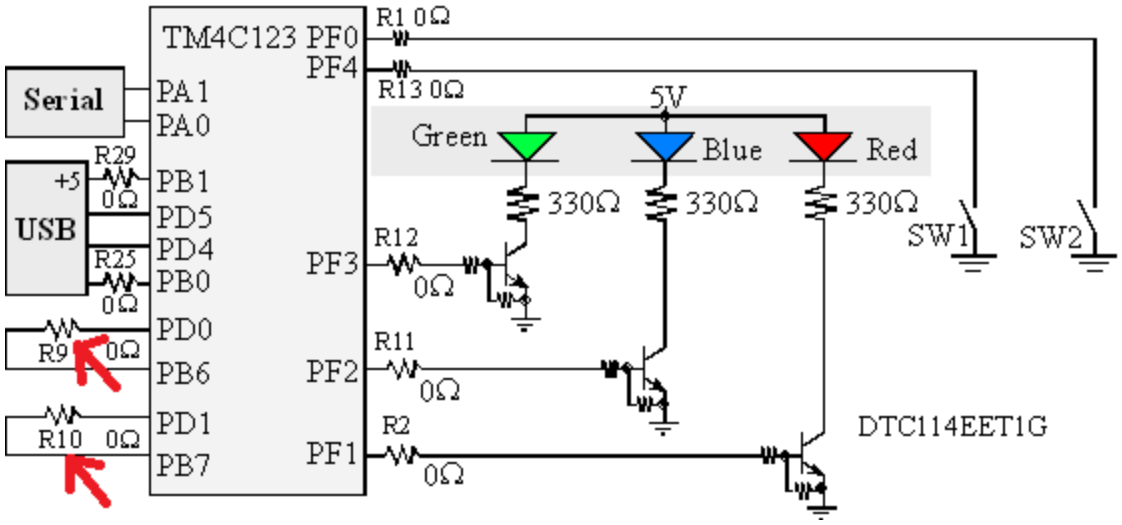


Figure 7.3. LaunchPad circuit showing PD1 PD0 are internally connected to PB7 PB6. Either do not use PB7 PB6 or remove R9 R10. see https://www.youtube.com/watch?v=MWIX7wgS9PM

You should test the hardware by running the **SSD1306_4C123** starter project. Make sure the system is configured for I2C3 (PD0 PD1). In particular, see this line in the file **inc/SSD1306.h**

#define I2C 3

Running this project will show you the types of output available for the SSD1306. There are multiple mains that demonstrate various example uses..

Part b - Write and Test the low level I2C Driver

This lab will use "busy-wait" synchronization, which means before the software issues an output command to the I2C port, it will wait until the display is not busy. In particular, the software will wait for the previous I2C command to complete. In the **LCD.s** file you will implement and test **`I2C_Send2`** which communicates with the OLED using the I2C3 interface. You will not write the initialization ritual, because it is given in the **SSD1306.c** file. However you will write this one function that outputs two bytes via I2C3 to the OLED. The first parameter is the slave address and the next two parameters are data.

```
; sends two bytes to specified slave
; Input: R0  7-bit slave address
;        R1  first 8-bit data to be written.
;        R2  second 8-bit data to be written.
; Output: 0 if successful, nonzero (error code) if error
; Assumes: I2C3 and port D have already been initialized and enabled
I2C_Send2
; --UUU--
; 1) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 2) write slave address to I2C3_MSA_R,
;     MSA bits7-1 is slave address
;     MSA bit 0 is 0 for send data
; 3) write first data to I2C3_MDR_R
; 4) write 0x03 to I2C3_MCS_R,  send no stop, generate start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 5) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 6) check for errors, if any bits 3,2,1 I2C3_MCS_R is high
;  a) if error set I2C3_MCS_R to 0x04 to send stop
;  b) if error return R0 equal to error bits 3,2,1 of I2C3_MCS_R
; 7) write second data to I2C3_MDR_R
; 8) write 0x05 to I2C3_MCS_R, send stop, no start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 9) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 10) return R0 equal to bits 3,2,1 of I2C3_MCS_R, error bits
;     will be 0 is no error
```

You should test this one function by running the **Lab7_EE319K_SSD1306** project. Put a breakpoint at the first **`IO_Touch();`** routine. The sequence of calls to **`SSD1306_Init`** **`SSD1306_OutClear`** **`SSD1306_DrawBMP`** and **`SSD1306_OutBuffer`** should draw the logo shown in Figure 7.4.

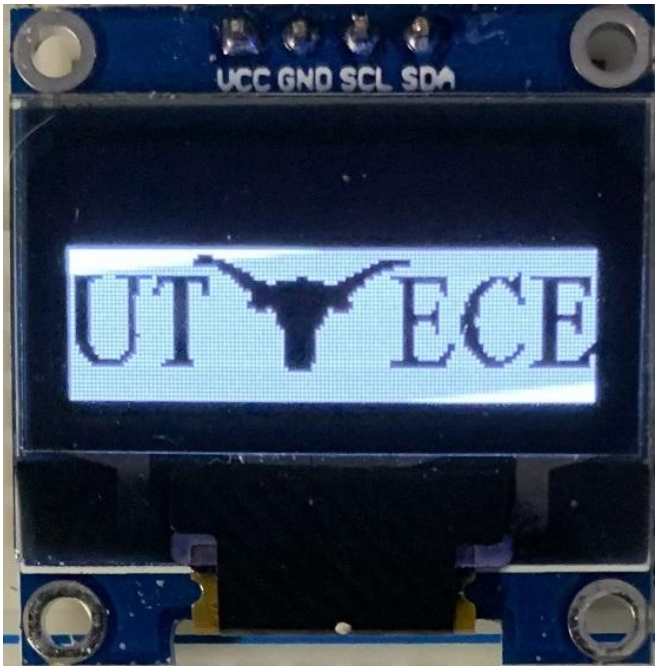


Figure 7.4. ECE logo

Part c - Write and Test the OutString function

In the Lab 7 version of **SSD1306.c** you will implement and test **`SSD1306_OutString`** which outputs a string to the OLED. You should call the existing **`SSD1306_OutChar`** function, which outputs one character to the OLED. The parameter is a pointer to a null-terminated ASCII string. *Warning*: for Labs 7 to 10, we will set the Keil compiler mode so the **`char`** type is unsigned 8 bits. This allows the driver to print special characters using codes 128 to 255. These characters are shown in Figure 7.5.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	∑
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

Figure 7.5. Special characters. These characters are not standard.

You should test this function by running the **Lab7_EE319K_SSD1306** project. Put a breakpoint at the second **`IO_Touch();`** routine. The sequence of calls to **`SSD1306_OutClear`** and **`SSD1306_OutString`** should print the *Welcome to EE319K* message.

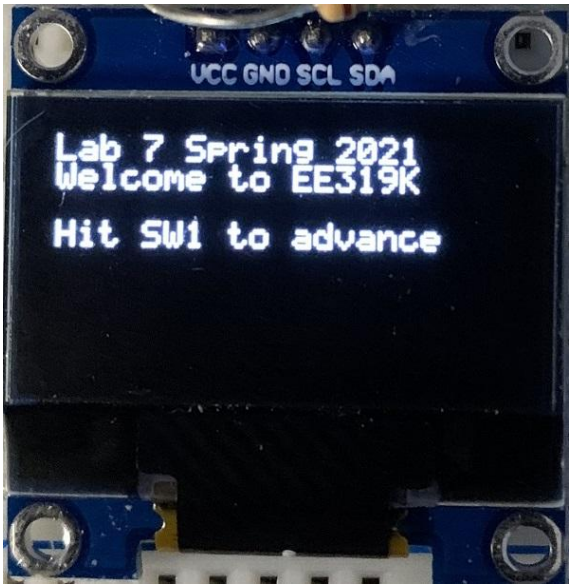


Figure 7.6. Welcome message.

Notice the style of the functions in SSD1306.c. Private functions that perform internal functions do not include **`SSD1306_`** in their names. In this lab, you are required to develop and test seven public functions (notice that public functions include **`LCD_`** or **`IO_`** in their names). The third component is a main program that can be used to test these functions. We have given you this main program, which you can find in the **Lab7Main.c** file. Please use the starter project which connects all these files.

Part d - Write and Test the IO functions

In the **IO.c** file you will implement and test the functions to interact with the user. See comments in the **IO.c** file for specific details. You are free to reorganize and redesign these three functions. For the 5ms wait function needed in **IO.c**, we suggest you use the cycle-counting approach to blind wait (like Lab 3) instead of SysTick (like Lab 5) because you will need SysTick periodic interrupts for Labs 8, 9, and 10. There is such an implementation of a delay function in CortexM.c, which you can use.

You should test the IO module functionality by running the **Lab7_EE319K_SSD1306** project without breakpoints. The program will output the logo and wait for user input, then print the *Welcome to EE319K* message.

Part e - Write and Test the LCD Driver

Implement the **LCD_OutDec** and **LCD_OutFix** functions (in **Print.s**) which print a decimal number to the screen and a fixed point number to the screen, respectively. See Table 7.1, Figure 7.7, **Print.s** and **print.h** for specific details. Your **`LCD_OutDec`** and **`LCD_OutFix`** functions **must use local variables on the stack with symbolic binding**. You can use SP or stack frame R11 for accessing the local variable. Your **`LCD_OutDec`** function will be used to output an unsigned 32-bit integer to the LCD, and your **`LCD_OutFix`** function will be used to output an unsigned 32-bit fixed-point number to the LCD If you want a challenge, you should consider writing **`LCD_OutDec`** as a recursive function (recursion is not a requirement for most of EE319K, recursion is a requirement for the EE312H section however). To observe the stack when assembly programs are running, set the memory view to observe the Stack Pointer (R13), and choose the type to be unsigned long. This will show you the top elements on the stack.

You must have at least one local variable in both `LCD_OutDec` and `LCD_OutFix`, allocated on the stack, and use symbolic binding for the local variable.

Parameter	LCD display
`0`	`0.00`
`1`	`0.01`
`99`	`0.99`
`100`	`1.00`
`999`	`9.99`
`1000 or more`	`*.**`

Table 7.1. Specification for the **`LCD_OutFix`** *function.*

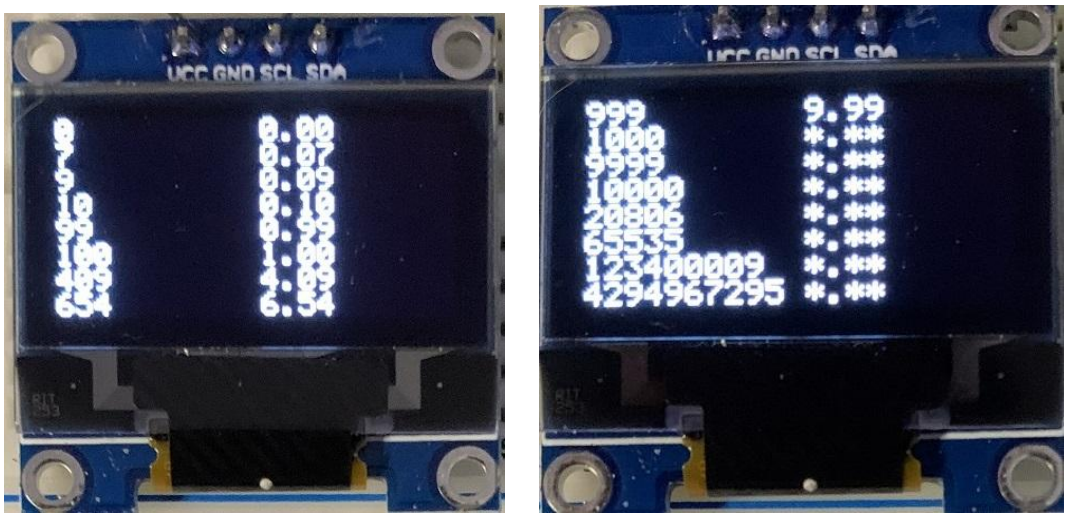


Figure 7.7. Decimal and fixed point output.

An important factor in device driver design is to separate the policies (how to use the programs, which are defined in the comments placed in the header) from the mechanisms (how the programs are implemented, which are described in the comments placed within the body of the subroutine.)

A main function that exercises your OLED driver's new decimal output facilities is provided in your project (**Lab7Main.c**). This software has two purposes. For the developer (you), it provides a means to test the driver functions. It should illustrate the full range of features available with the system. The second purpose of the main program is to give your client or customer (e.g., the TA) examples of how to use your driver. You can modify/comment this main function to help you debug your code. Your modification will not be part of your deliverable.

Demonstration

(both partners must be present, and demonstration grades for partners may be different)

There are grading sheets for every lab so you know exactly how you will be evaluated. You will be required to demonstrate the proper operation on the actual microcontroller. During demonstration to the TA, you will run your system in the debugger and show the binding, allocation/initialization, access and deallocation of the local variables. Each time a function is called, an **activation record** is created on the stack, which includes parameters passed on the

stack (none in this lab), registered saved, and the local variables. You will be asked to observe the stack in the debugger and identify the activation records created during the execution of **`LCD_OutDec`**. TAs may ask you questions on OLED interfacing, and programming. What does the SDA and SCL signals signify? What does busy-wait synchronization mean in the context of communication with the LCD? The TA will ask to see your implementation of local variables and ask you to explain the four step process (binding, allocation, access and deallocation). What are the advantages/disadvantages of using SP or R11 for the stack frame? You should be able to draw stack pictures. How does AAPCS apply to this lab? Why is AAPCS important? Why do students who use R4 without saving and restoring R4 find the SSD1306 software crashes?

Do all these well in advance of your checkout

1. **Signup for a Zoom time with a TA. If you have a partner, then both must be present**
2. **If you cannot live-stream video, create a 60-sec YouTube video and upload it**
3. **Upload your software to canvas, make sure your names are on all your software**
4. **No pdf deliverable for Lab 7**

Do all these during Zoom meeting

1. **Have Keil Lab 7 open so TA can ask about your code**
2. **Start promptly, because we are on a schedule. If you have a partner, then both must be connected**
3. **Demonstrate lab to TA (YouTube video or livestream video)**
4. **Answer questions from TA in the usual manner**
5. **TA tells you your score (later the TA will upload scores to Canvas)**

Deliverables

1. No pdf deliverable file for Lab 7.

Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

Hints

1. This Lab will not run in simulation mode.
2. It will be important to pass data into your functions in Register R0; this way you will be able to call your functions from C code later in Labs 8, 9, and 10. It is important to save and restore Registers R4-R11 if you use them.
3. You must use the cycle-counting approach to implement the blind waits (Lab 3) instead of SysTick (Lab 5) because you will need SysTick periodic interrupts for Labs 8-10.

FAQ

1. **What does I2C3_MCS_R do?**

   There is a serial interface module on the chip. When we write to I2C3_MCS_R it performs a command (3 means start/send, 4 means stop, 5 means send/stop). When we read from I2C3_MCS_R we the status register (bits 3,2,1 are error bits).

2. **Where are we supposed to use Wait function?**

   Waiting 5ms will be used in IO_Touch to debounce your switches. You will write this private function either inside IO.c or use the existing public function in CortexM.c.

3. **Does anyone know in what manner we are expected to connect the board to the LCD?**

   See Figure 7.2. Please ask the instructor or TA if you are unsure.

4. **The starter files mention setting up a switch. What exactly is this switch used for?**

   The switch, which is switch SW1 and on Port F, is used to change through the initialization screens. The screen should change from "Lab7! ..." to the longhorn when the switch is pressed and released, and then to your code when pressed and released again.

5. **For OutDec and OutFix, are we supposed to put our final result back into R0 as the output? I am confused as to how we should be "returning" the string of ASCII values that we created in OutDec and OutFix.**

   Rather than return a value you just need to call OutChar to display each integer to the screen (i.e. 123->'1' '2' '3'). We do not suggest you use OutString, rather call OutChar for each character you wish to output.

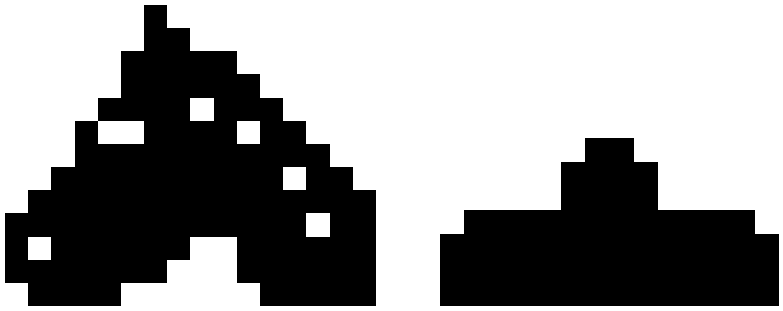
6. **Can someone explain how we can do this OutDec and OutFix part? I'm very confused..and where does the stack take place in this?**

   There is a function, SSD1306_OutChar that you should use to output characters to the OLED. It is implemented in SSD1306.c. To make use of this function, you need to somehow convert the 32-bit input number into chars and pass them in as input parameters (think AAPCS standards). If you use recursive to implement OutDec, the stack will be your friend. Even if you don't, you will probably be using more registers than just R0 - R3 so you end up having to push/pop some registers onto/from the stack. We want you guys to become familiar with stack, stack frame, local variables etc. so please follow the instruction use local variables, symbolic binding when implementing these two functions.

Extra credit for Lab 7: you may do either but not both extra credits

Option 1: 5 points extra credit Study how the font table works in **SSD1306.c**. Each ASCII character from 32 to 255 is encoded as a 6-wide by 7-tall pixel image. See **static const uint8_t ASCII[]**. Notice most of the data has bit 7 clear; this will create a 1-pixel separation between characters on different rows of the display. Most of the data has the right column clear, creating a 1-pixel separation between characters on the same row. Notice the font images are stored upside down (bit 0 is the top, and bit 6 is the bottom). Find a foreign language character (not yet implemented) that will be recognizable, displayed in 5 by 7 pixel format. Draw this character as a 5 by 7 image. Convert this image into 5 8-bit bytes (bit 7 must be clear), and replace the one of the last two lines of **ASCII[]** with the new image (the first line is n=32, and the last is n=255). You can test the new character by calling **SSD1306_OutChar(n)**, where n is the line number in **ASCII[]** that you replaced.

Option 2: 5 points extra credit Study how **SSD1306_DrawBitmap** works in **SSD1306.c**. Read the directions in **BmpConvertReadme.txt**. Using Paint observe the **test.bmp** and **ship.bmp** images.



Run **BmpConvert.exe** and convert one of the **.bmp** files to a **.txt** file. Using NotePad observe the **t.txt** data. Copy the data from the **.txt** file and paste it into **image.h**. Add calls **SSD1306_DrawBitmap** so the image is drawn on the OLED.

Using Paint draw a new black and white image with 6 or more pixels high, and more than 6 or more pixels wide. Run **BmpConvert.exe** to convert your **.bmp** file to a **.txt** file. Copy the data from the **.txt** file and paste it also into **image.h**. Add calls **SSD1306_DrawBitmap** so the image is drawn on the OLED.