Lab 10. Embedded System Design

Preparation

Use the starter project, paste in your ADC and LCD solution code from a previous lab (collect images to display on the ST7735 LCD) https://www.dropbox.com/s/oirc5dyp8dymqvc/Lab10_EE319K.zip?dl=1
See some art and sounds: http://users.ece.utexas.edu/~valvano/Volume1/Lab10Files
Example game art can be found at https://opengameart.org/

Restrictions on components

For this lab you may use any additional components in addition to the components given to you. For example, you can use enclosures, bigger buttons, buzzers, joysticks, speakers, amplifiers etc. The only restriction is that you must write all the low-level driver software yourself.

Spring 2022 Game proposal link: Here

Restrictions on software

Except for software found in the EE319K book, on Valvano's website (valvanoware), on EE319K web site (EE319Kware), and provided by TAs in git, **you must write all the software for your Lab 10.**

Exception to the restriction: if you wish to use a different display and there exists a graphics driver similar in functionality to the code provided to you in ST7735.c, then you can ask a TA to waive the restriction for this graphics driver. You MUST get TA approval before you start to use the display.

Timetable

- Friday, 4/22, 11pm, submit a proposal, edit the googledoc shown in the section **System Requirements for Design Your Own Project**. All teams must submit a proposal, even if you are doing Space Invaders, Connect Four or Pipe Dreams**.**
- Tuesday-Wednesday-Thursday,5/3,5/4,5/5, Lab 10 checkout in regular checkout slot
- Deadline for Late checkout is Friday 5/6 by 5pm - any TA, any office hour
- Deadline for letting us know the "youtube link" (submission on Canvas) is Friday 5/6 by 5pm
- **After 5pm Friday 5/6 there can be no more late submissions**
- First Round Voting (on Canvas) starts 5/9 2pm and ends Friday 5/13 noon
- YouTube voting is just for fun to choose the best game for all EE319K/EE319H

Teams

Lab 10 will be performed in teams of two keeping the same partner as Labs 7, 8, and 9.

Purpose

The objectives of this lab are: 1) design, test, and debug a large C program; 2) to review I/O interfacing techniques used in this class; and 3) to design a system that performs a useful task. There are many options for Lab 10. One option is to design a 80's-style shoot-em up game like **Space Invaders**. Another option is to design a turn-based like **Connect Four**. All students are required to propose a project similar in scope to these other options. Good grades will be given to projects that have simple implementations, are functionally complete, and are finished on time. Significant grade reductions will occur if the project is not completed on time.

Interrupts must be appropriately used to control the input/output, and will make a profound impact on how the user interacts with the game. You could use an edge-triggered interrupt to execute software whenever a button is pressed. You could output sounds with the DAC using a fixed-frequency periodic interrupts. You could decide to move a sprite using a periodic interrupt, although the actual LCD output should always be performed in the main program.

Base System Requirements for all projects

- There must be at least two buttons (could be ones on-board or external) and one slide pot (a joystick counts as two slide pots). Buttons and slide-po(tentiometer) must affect game play. The slide-pot must be sampled by the ADC at a periodic rate.
- There must be at least three sprites/images on the LCD display that move in relation to user input and/or time.
- There must be sounds (at least two) appropriate for the game, generated by the DAC developed in Lab 6. However, the interrupt can be of a fixed period (e.g., 11025Hz for sampled audio extracted from a .wav file).

- The score should be displayed on the screen (but it could be displayed after the game action).
- At least two interrupt ISRs must be used in appropriate manners.
- The game must be both simple to learn and fun to play.
- It must run in at least two languages.
- Creating a YouTube video of your finished project is worth 10 points out of the 100; Max 90 points awarded at checkout. If you prefer an alternative to YouTube like Vimeo that works too.

Labs 1 to 9 contribute to 24% of the EE319K total grade. Labs 1-9 count for 2.67% each and. Lab 10 counts for 6%. All labs together count for 30%.

System Requirements for Space Invaders, Asteroids, Missile Command, Centipede, Snood, or Defender

You will design, implement and debug a 80's or 90's-style video game. You are free to simplify the rules but your game should be recognizable. Buttons and the slide pot are inputs, and the LCD and sound (Lab 7) are the outputs. The slide pot is a simple yet effective means to move your ship.

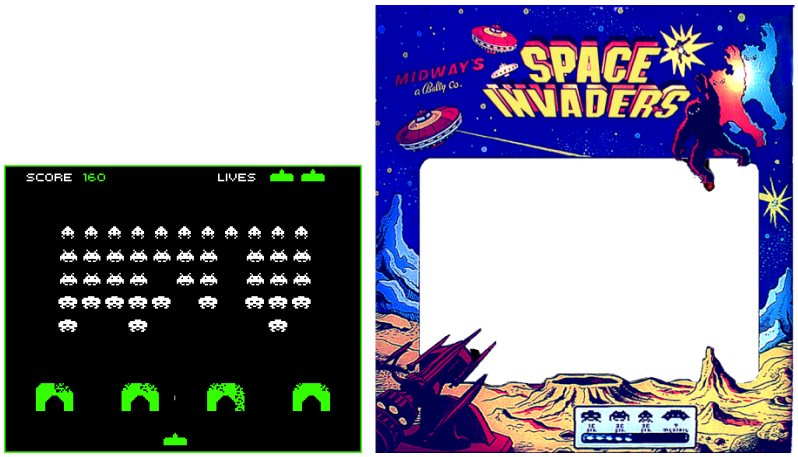


Figure 10.1. Space Invaders
http://www.classicgaming.cc/classics/spaceinvaders/index.php

System Requirements for Connect Four

You will design, implement and debug a Connect Four game. Buttons and slide pots are inputs, and the LCD and sound (Lab 7) are the outputs. The slide pot is a simple yet effective means to select which column to enter your chip. There are two possible modes for the game. In the required mode, a human plays against artificial intelligence running on the microcontroller. In addition to the above requirements, Connect Four must also satisfy

- The colored pieces must move on the LCD.
- You must implement at least a 6-row by 7-column board, but are allowed to create more rows or columns.
- Aside from the size of the board, you must implement the official Connect Four rules.
- You must check for wins horizontal, vertical, and diagonals. You should also check for a draw.
- You must not allow illegal moves.
- The game must have a man versus machine mode.

For more fun, you have the option of creating a two computer mode, where the AI in one microcontroller plays against a second AI running on a second microcontroller. A similar fun option is having two microcontrollers that

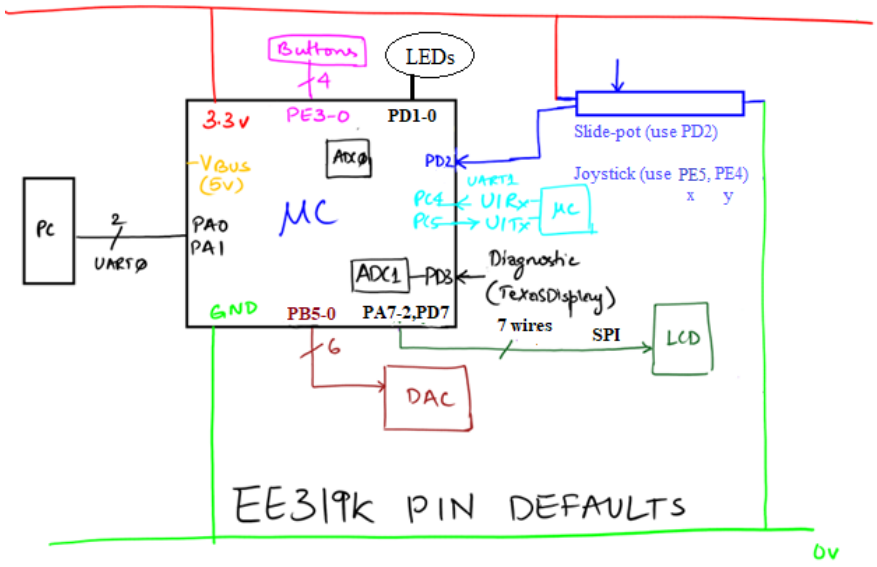
allow one human looking at one display to play against another human looking at a second display. The boards are connected via the UART1 ports, like Lab 9, but with full duplex communication. When running in two-computer mode, you need to specify baud rate, communication protocol, and determine which side goes first. Both computers should recognize that a winner has occurred. If you implement a two-computer option, it must be done in addition to the required man versus machine mode.

All students must create a proposal for their project

The first step is to make a proposal. You can find previous proposals on the proposal page.

Other groups are allowed to solve similar or identical games as those proposed by other students. Good projects require students to integrate fundamental EE319K educational objectives such as I/O interfacing, data structures, interrupts, sound, and the effective implementation of real-time activities. The project **need not be** a game, but the project must satisfy the System Requirements for all games listed above. Any TA will approve the proposal by adding a comment to your proposal.

These connections are recommended (so TAs can run your code on their hardware) but not required.



Procedure

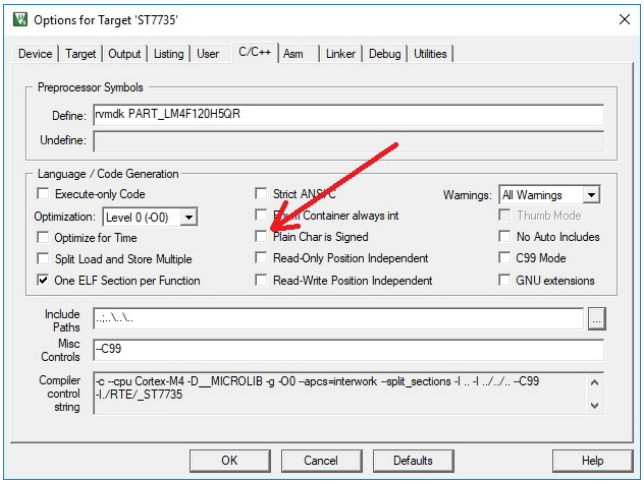
Part a - Design Modules

In a system such as this each module must be individually tested. Your system will have four or more modules. Each module has a separate header and code file. Possible examples for modules include slide pot input, button input, LCD, and sound output. For each module design the I/O driver (header and code files) and a separate main program to test that particular module. Develop and test sound outputs as needed for your game. There should be at least one external switch and at least one slide pot.

Part b - Make the game globally aware

One of the ABET criteria is to develop "**an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors**". In EE319K/EE319H we will address this criteria by designing our project with a global awareness. In particular, each game must be playable in at least two languages. The startup screen must allow the user to select the language. Within the game each output string could be implemented as an array of strings, where the language is used to index into the array. A more elegant approach is to place all strings into a 2-D array data structure, where one index is the string name, and the other index is the language. This second approach will simplify adding additional languages and provide for a more consistent feel for the string outputs throughout the game.

To get extended ASCII to display on your LCD, you will need to make the **char** data type be **unsigned 8 bits** (0 to 255). Within Keil, execute options, select the C/C++ tab and unclick the "Plain Char is Signed" button



Run test code like this to see one way to output a set of phrases in multiple languages. Notice **\x** escape sequences are used to specify extended ASCII as needed for the language. Also notice the weird string in Language_French. One cannot combine all of Français into one string: **"Fran\x87ais"** because **\x87a** would be considered one character and not two. Run **main2** in the starter project to execute this example.

```
typedef enum {English, Spanish, Portuguese, French} Language_t;
Language_t myLanguage=English;
typedef enum {HELLO, GOODBYE, LANGUAGE} phrase_t;
const char Hello_English[] ="Hello";
const char Hello_Spanish[] ="\xADHola!";
const char Hello_Portuguese[] = "Ol\xA0";
const char Hello_French[] ="All\x83";
const char Goodbye_English[]="Goodbye";
const char Goodbye_Spanish[]="Adi\xA2s";
const char Goodbye_Portuguese[] = "Tchau";
const char Goodbye_French[] = "Au revoir";
const char Language_English[]="English";
const char Language_Spanish[]="Espa\xA4ol";
const char Language_Portuguese[]="Portugu\x88s";
const char Language_French[]="Fran\x87" "ais";
const char *Phrases[3][4]={
  {Hello_English,Hello_Spanish,Hello_Portuguese,Hello_French},
  {Goodbye_English,Goodbye_Spanish,Goodbye_Portuguese,Goodbye_French},
  {Language_English,Language_Spanish,Language_Portuguese,Language_French}
};
```

```
  for(phrase_t myPhrase=HELLO; myPhrase<= GOODBYE; myPhrase++){
    for(Language_t myL=English; myL<= French; myL++){
        ST7735_OutString((char *)Phrases[LANGUAGE][myL]);
      ST7735_OutChar(' ');
        ST7735_OutString((char *)Phrases[myPhrase][myL]);
      ST7735_OutChar(13);
    }
  }

    l = 128;
    while(1){
       for(int i=0;i<8;i++){
       ST7735_SetCursor(0,i);
         ST7735_OutUDec(l);
         ST7735_OutChar(' ');
         ST7735_OutChar(l);
        l++;
    }
  }
}
```

Generalize this output of phrases, that uses the global language variable **myLanguage** to output a phrase in the appropriate language.

```
void ST7735_OutPhrase(phrase_t message){
```

You will increase the number of phrases (and potentially decrease the number of languages)

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	

Part c - Generate Sprites

In the game industry an entity that moves around the screen is called a *sprite*. You will find lots of sprites in the **SpaceInvadersArt** folder of the starter project. You can create additional sprites as needed using a drawing program like Paint or get them online (https://opengameart.org/). Most students will be able to complete Lab 10 using only the existing sprites in the starter package. You can create your own sprites by following the instructions in **BmpConvert16Readme.txt**. Use **Paint** to create the image, save as 16-bit BMP images (65536 colors). Use **BmpConvert16.exe** to create a text file. Copy the contents of the text file into your code. Use **ST7735_DrawBitmap** to send the image to the LCD. For more details read section 10.1 in the book.

Part d - Sound

Some of you will find it easier to get/create sampled sound for your game. Sound generated in Lab6 is called synthesized sound which involved generating sound of a particular frequency based on an instrument (sine) table. Sampled sound is sound that has been recorded and stored in a file like a .wav or .mp3 file. You can either create sounds yourself or download them from online sites like this: https://opengameart.org/. To use sounds from a file, you will need to convert them to C declarations of digital audio samples much like the image bitmap arrays store pixels. There is a sample project (WavPlay.zip) on the class website and an associated video (link) that shows you how to create (WC.m) these arrays and use them to produce sound.

Part e - Extend Random Number Generator

The starter project includes a random number generator. A C version of this function can be found in the book as Programs 2.6, 2,7, and 3.12. To learn more about this simple method for creating random numbers, do a web search for **linear congruential multiplier**. The book example will seed the number with a constant; this means you get exactly the same random numbers each time you run the program. To make your game more random, you could seed the random number sequence using the SysTick counter that exists at the time the user first pushes a button (copy the value from NVIC_ST_CURRENT_R into the private variable M). You will need to extend this random number module to provide random numbers as needed for your game. For example, if you wish to generate a random number between 1 and 5, you could define this function

```
uint32_t Random5(void){
  return ((Random32()>>24)%5)+1;  // returns 1, 2, 3, 4, or 5
}
```

It is important to shift by a large number (>>24) because the bottom bits are not very random. Seeding it with 1 will create the exact same sequence each execution. If you wish different results each time, seed it once after a button has been pressed for the first time, assuming SysTick is running

```
Seed(NVIC_ST_CURRENT_R);
```

Part f - Test Subsystems

When designing a complex system, it is important to design, implement and test low-level modules first (parts a,b,c,d). In order to not be overwhelmed with the complexity, you must take two working subsystems and combine them. Do not combine all subsystems at the same time. Rather, we should add them one at a time. Start with a simple approach and add features as you go along.

Part g - Optional Feedback: http://goo.gl/forms/rBsP9NTxSy

Competition Structure

Step 1 - Checkout and Grade (5.4% of EE319K total grade)

Checkout occurs in your usual lab checkout slot. You need to meet the base requirements and be able to answer questions about your code and hardware. Your TA will give you a sheet, which is your admission ticket to the in-class competition. Your Lab 10 checkout grade is similar to how Labs 1 to 9 were graded.
Game meets requirements. Each requirement has a corresponding score next to it:

· There must be at least two buttons (could be ones on-board or external). Buttons must affect play a role (10%)
· The slide pot must be sampled by the ADC and must play a role in the game/project (10%)
· There must be at least three sprites/images on the LCD display that move in relation to user input and/or time (20%)
· There must be sounds appropriate for the game/project, generated by the DAC developed in Lab 6. However, the interrupt can be fixed period (15%)
· Some numerical data (score for example) should be displayed on the screen (10%)
· Two languages (10%)
· At least two interrupt ISRs must be used in appropriate manner (15%)

Step 2 - YouTube Competition (0.6% for YouTube posting)

Participation in the YouTube competition is NOT optional, counting for 10% of the lab 10 grade. The posted video must be limited to a minimum of 2 minutes and a maximum of 5 minutes. All teams in the competition receive the full competition points. *Fun will be had by all.*

Watch some videos from Spring 2021
https://utexas.instructure.com/courses/1311774/quizzes/1594174

Extra fun

Here are some features you may implement for extra fun:

- Edge-Triggered Interrupts with software debouncing
- Edge-Triggered Interrupts with hardware debouncing
- Multi-player game with multiple controllers
- UART used for communication with two LaunchPads
- More than 4 awesome sound effects connected to game events
- Multiple levels in game with demonstrable rise in intelligence and difficulty (however, the class demo only has about 2 minutes of playtime, so make it get to the good stuff within 2 minutes)
- <please do not attempt double buffering with this display> ~~Double-buffering part of the screen (two RAM buffers, one with image being sent to LCD and other being created by software to be output next) - Writing your pixel data to a virtual buffer and periodically updating the entire screen; this is very hard to implement with the ST7735 at 20 frames/sec~~
- Layering of graphics, showing one image over top other images (clipping)
- Use of any specialized hardware (note you must write all software). An accelerometer for example.
- Joysticks can be used for the design of your own game option. The interface is simply two slide pots, which you will need to connect to two ADC channels. You will need a different sequencer from SS3 so that your software can trigger the ADC and collect conversions from two channels. See the **ADCSWTriggerTwoChan_4C123** project.
- Digital Signal Processing. You must explain what it does and how it works.
- Software implementation of volume control using a 6-bit DAC
- Design and implement an 8-bit DAC using an R-2R ladder
- Soldered on the solder-board
- Solved a distributed computing problem in a multi-player game. Each player has its own thread and the computer executes the threads independent of each other. The threads may be running on different LaunchPads or running on the same LaunchPad, independent of each other.