Lab 9. Embedded System Design (Spring 2025)

Preparation
Restrictions on components
Spring 2025 Game proposal link (due 11pm Friday 4/11):
Restrictions on software
Timetable for grading (steps to get 100 on Lab 9)
Timetable for in-class competition
Teams
Purpose
Base System Requirements for all projects
    System Requirements for Space Invaders
    All students must create a proposal for their project
Procedure
    Part a - Design Modules
    Part b - Make the game globally aware, use main1 to test
    Part c - Generate Sprites, use main2 to test
    Part d - Interface switches and LEDs, use main3 to test
    Part e - Sound, use main4 to test
    Part f - Implement the system, use main5
    Part g - Optional Feedback: http://goo.gl/forms/rBsP9NTxSy
Grading Structure
    Step 1 - Checkout and Grade (6% of ECE319K total grade)
    Step 2 - Submit all .c or .cpp files that you wrote to Canvas
    Extra fun (but no extra credit)
    Lab9 Support

Preparation

Read all of Chapter 9 in the textbook. Use the starter project, paste in any code from previous labs as needed . The CCS the starter project for Lab 9 is in the original downloads

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 2025 Game proposal link (due 11pm Friday **4/11**):

Submit Lab 9 proposal

Restrictions on software

Except for software found in the ECE319K book, on Valvano's website (valvanoware), on ECE319K web site (valvanoware), and provided by TAs, **you must write all the software for your Lab 9.**

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.

There are online tutorials for RayTracing (rendering 3-D worlds onto a 2-D screen). I like this one https://lodev.org/cgtutor/raycasting.html However this is a very difficult problem and you cannot just copy code, you will be required to change it to fixed point and to understand everything.

Timetable for grading (steps to get 100 on Lab 9)

- Friday, 4/11, 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**.**
- Tuesday-Wednesday-Thursday, 4/22-4/24, Lab 9 checkout in regular checkout slot. You may also get a checkout done by a TA during their office hours.
- Deadline for Late checkout is Friday 4/25 by 5pm - any TA, any office hour.

Timetable for in-class competition

- The competition is held during the last class time. It is optional but strongly recommended that you enter the competition.
- To enter your team in the competition (the format of the competition depends on your instructor)
- ~~The first 12 teams (in each section) to sign up will be allowed to demo their lab during the lecture. The signup sheets are available from each professor~~
- ~~Both students must attend class and be ready to setup/play/describe.~~
- ~~To win the competition you will have to get the highest votes from your peers who watch you demo your lab.~~

Teams

Lab 9 will be performed in teams of two keeping the same partner as Labs 4-8.

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 9. 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**. A third option is to design and implement a product that does something useful, like a **Heart-Rate Monitor**. 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

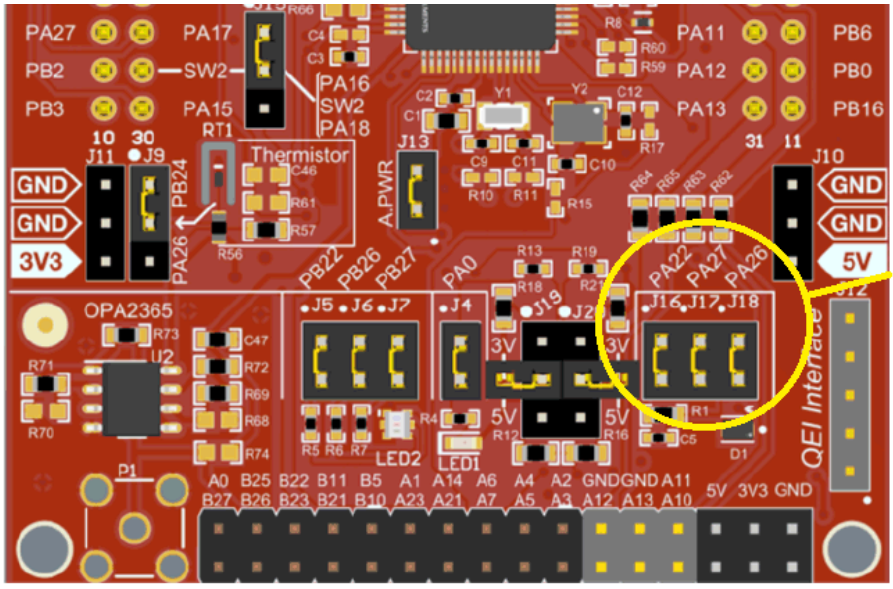
move a sprite using a periodic interrupt, although the actual LCD output should always be performed in the main program.

Base System Requirements for all projects

Each requirement has a corresponding score next to it:

- There must be at least two external buttons (not counting the on-board switches). Buttons must affect game play. (10%)
- There must be at least one slide-pot. The slide-pot must be sampled by the ADC at a periodic rate according to the Nyquist Theorem, and must play a role in the game/project. A joystick counts as two slide pots). (15%)
- 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 at least two sounds appropriate for the game, generated by the DAC developed in Lab 5, or the built-in 12-bit DAC. However, the interrupt can be of a fixed period (e.g., 11025Hz for sampled audio extracted from a **.wav** file). (20%)
- The score should be displayed on the screen (but it could be displayed after the game action). (10%)
- 2 to 3 minutes video showcasing their game. You have the option to opt out of the public playlist. (5%)
- At least two interrupt ISRs must be used in appropriate manners. (10%)
- The game must be both simple to learn and fun to play. (0%)
- It must run in at least two languages, displaying text appropriate for the game. (10%)

Labs 1 to 8 contribute to 24% of the ECE319K total grade. Labs 1-8 count for 3% each and. Lab 9 counts for 6%. All labs together count for 30% of your total grade.



If using PA22 PA27 or PA26 remove J16, J17, J18

System Requirements for Space Invaders

You will design, implement and debug an 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 are the outputs. The slide pot is a simple yet effective means to move your ship.

Figure 9.1. Some images for Space Invaders
http://www.classicgaming.cc/classics/spaceinvaders/index.php

For more information on details of Space Invaders, see the example proposal on the proposal signup page.

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 ECE319K 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.

The **second step** will be for the TAs to approve the proposal by commenting on the googledoc.

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, switch input, LCD, LED output, 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.

- how many switches do you want?
- how many LEDs do you want?
- which LCD you are using?
- do you want to have a joystick (which you have to buy)?
- there is a 12-bit DAC on PA15; you are allowed to use it instead of your 5 bit DAC

Decide which pins to use. These pins are already used
- PA11, PA10 UART to PC
- PA22 UART2, PA8 UART1 to other microcontroller Lab 8
- PB6, PB7, PB8, PB9, PB15, PA13 SPI/GPIO to LCD
- PB18 ADC to slidepot
- PB0, PB1, PB2, PB3, PB4 to 5-bit DAC
- PA15 12-bit DAC output
- PA18, PB21 to LaunchPad switches

- PA0, PB22, PB26, PB27 to LaunchPad LEDs

If you plan to use PA22, PA27 and PA26, please remove the jumpers J16, J17, J18 to disconnect the light sensor.

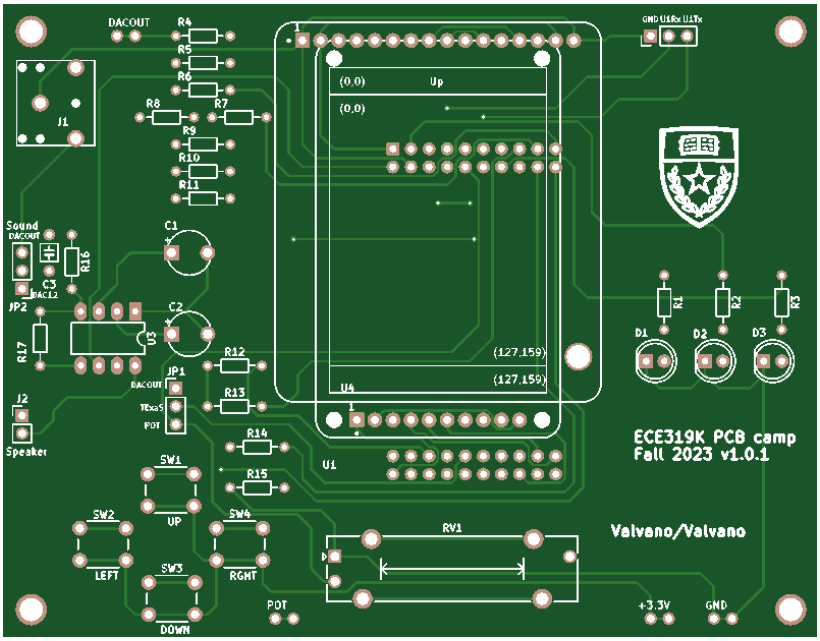


Figure 9.2. Example PCB from PCB camp

When selecting pins for your inputs and outputs, be aware of the many jumpers on the LaunchPad, see Figure 9.3. For example, **if you wish to use PA26, then you should remove jumper J18**.

Table 2-1. LaunchPad Kit Header Description and Jumper Shunt Installation (continued)

Jumper	Description	Default setting	Connected Pin or Signal	Low Power Measurement Recommendation
J4	Red LED1	Installed	PA0 to LED	OFF to disconnect pin from LED
J5	RGB LED2 – blue channel	Installed	PB22 to RGB LED	OFF to disconnect pin from RGB LED
J6	RGB LED2 – red channel	Installed	PB26 to RGB LED	OFF to disconnect pin from RGB LED
J7	RGB LED2 – green channel	Installed	PB27 to RGB LED	OFF to disconnect pin from RGB LED
J8	S1 Button and BSL invoke	Installed	PA18	OFF to avoid current from external pulldown depending on pin configuration
J9	Thermistor signal selection	(1) short to (2) PB.24	PB24 to thermistor circuit	OFF to disconnect from thermistor circuit
J10	5V power header	NA	5V, GND	No external connection
J11	3V3 power header	NA	3V3, GND	No external connection
J12	QEI interface header	NA	PA29, PA30, PB14, 3V3, GND	No external connections
J13	Analog power – power to thermistor and OPA2365	Installed	3V3	OFF to disconnect power to thermistor, and OPA2365 circuits
J14	SW1 selection to BP header – PA9/ PB23	(1) short to (2) PB.23	PB23 to J1.3	Don't care if not used in BoosterPack connector
J15	SW2 selection to BP header – PA16/ PA18	(1) short to (2) PA16	PA16 to J3.29	Don't care if not used in BoosterPack connector
J16	OPA0_OUT for light sensor circuit	Installed	PA22 to light sensor	OFF to disconnect from photodiode circuit
J17	OPA0_IN0- for light sensor circuit	Installed	PA27 to light sensor	OFF to disconnect from photodiode circuit
J18	OPA0_IN0+ for light sensor circuit	Installed	PA26 to light sensor	OFF to disconnect from photodiode circuit
J19	PA0 Open-drain IO pullup	(1) Short to (2) 3.3 V	PA0 to 3V3	OFF if pin initialized as output low or input with pullup/pulldown
J20	PA1 Open-drain IO pullup	(1) Short to (2) 3.3 V	PA1 to 3V3	OFF if pin initialized as output low or input with pullup/pulldown
J21	UART0_TX selection	(1) short to (2) XDS_UART function	PA10 to XDS	OFF to disconnect pin
J22	UART0_RX selection	(1) short to (2) XDS_UART function	PA11 to XDS	OFF to disconnect pin
J23-J28	MSPM0G3507 pin extension header (unlabeled – bottom of board)	NA	See schematic for details	No external connection

Figure 9.3. LaunchPad jumpers

Part b - Make the game globally aware, use main1 to test

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 ECE319K/ECE319H 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.

Run test code like the following 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. Test your language code using main1. 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	

Figure 9.4. Extended ASCII.

Part c - Generate Sprites, use main2 to test

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 9 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 9.1 in the book. Modify main2 so it tests all the game art you plan to use.

To implement Spaceinvaders, you do not need to generate images, they are provided in the starter code.

Part d - Interface switches and LEDs, use main3 to test

Interface the switches to the microcontroller, and write code in Switch.c to initialize and read the switches. Feel free to modify how the switch software operates.

Interface the LEDs to the microcontroller, and write code in LED.c to initialize and output to the LEDs. Feel free to modify how the switch software operates. Because of the possible read-modify-write critical section, please only use DOUTSET31_0, DOUTCLR31_0, or DOUTTGL31_0 to affect the LEDs.

Modify main3 so it tests your switches and LEDs.

Part e - Sound, use main4 to test

Some of you will find it easier to get/create sampled sound for your game. Sound generated in Lab5 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.

To implement Spaceinvaders, you do not need to generate new sounds, they are provided in the starter code.

Part f - Implement the system, use main5

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

Grading Structure

Step 1 - Checkout and Grade (6% of ECE319K 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 9 checkout grade is similar to how Labs 1 to 8 were graded. Game **MUST** meet requirements.

Step 2 - Submit all .c or .cpp files that you wrote to Canvas

One set of files per team should be submitted. We will run the code through a plagiarism checker, so please do not share code with others.

Extra fun (but no extra credit)

Here are some features you may implement for extra fun:

- Edge-Triggered Interrupts with software debouncing
- Edge-Triggered Interrupts with hardware debouncing
- Multiplayer 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 on the same ADC (both on ADC0 or both on ADC1). See the **ADCSWTrigger** project.
- Digital Signal Processing. You must explain what it does and how it works.
- Software implementation of volume control using an 8-bit DAC
- Design and implement an 8-bit DAC using an R-2R ladder
- Soldered on the solder-board or soldered onto a PCB
- 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.

Lab9 Support

Here are some resources (projects, howtos, videos) that are either already in your CCS lab 9 project folder or posted on the ECE319K site that should help you with your Lab9. These are in no particular order and may or may be relevant to your specific lab9:

1. Convert BMP files into code that can be displayed on ST7735R LCD:
   BmpConvert16.exe, BMPConvert16Readme.txt
   a. C++ code https://utexas.box.com/shared/static/0gb79l40rjl4xls2h1pi6xq09ese9em3.cpp
2. Matlab/Octave script to convert wav files into C declaration with n-bit sound:
   Script: WC.m ; Video
3. Edge-Triggered Interrupts which cause buttons to interrupt: see the **EdgeTriggeredInt**
4. C Structs + Enums + Modules (header files): CStructsModules.zip
5. Timer interrupts: see the project **TimerPeriodicInt**
6. JoyStick with two ADCs: see the project **ADCSWTrigger**