Lab 10. Embedded System Design

Preparation

Use the starter project on SVN, paste in your LCD.s and print.s solution (place BMP images on the LCD)

Restrictions on cost

To win the competition, you are restricted to only use parts that were given to you as part of one of the other labs, plus additional resistors to make more bits in your DAC. If you wish to use extra parts not given as one of the labs, such as joy sticks, speaker, and amplifiers, you may do so, but will be disqualified from winning the competition, unless specifically approved by your professor.

Teams

Lab 10 will be performed in teams of two. However, you are free to select a new partner for Lab 10, or to remain with your regular partner. If you are not working with your regular partner, the old partner and your TA must be notified by the first Friday occurring after your Lab 9 due date. This advanced warning will give your old partner time to find an alternative partner. The new partner must have the same lecture time, but could be in a different Lab section. Let the TAs know the new partnership so we can create a new SVN folder.

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 four options for Lab 10. Option 1 is to design a 80's-style shoot-em up game like **Space Invaders**. Option 2 is to design a turn-based like **Connect Four**. Option 3 is to design a **Pipe Dream** game as specified below. The fourth option is to propose an alternative project similar in scope to these other options. If you would like to propose a project prepare a one-page description by the first Monday after your Lab 9 due date. **If your project is approved please print the proposal and have a TA post in the lab.** 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 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.

System Requirements for all games

- There must be at least one externally-interfaced button and one slide pot. Buttons and slide pot must affect game play. The slide pot must be sampled by the ADC.
- There must be at least three images on the LCD display that move in relation to user input and/or time.
- There must be sounds appropriate for the game, generated by the DAC developed in Lab 7. However, the interrupt can be fixed period.
- The score should be displayed on the screen (but it could be displayed before or after the game action).
- At least two interrupt ISRs must used in appropriate manners.
- The game must be both simple to learn and fun to play.

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

You will design, implement and debug 80's or 90's-style video game. You are free to simplify the rules but your game should be recognizable as one of these six simple games. 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.

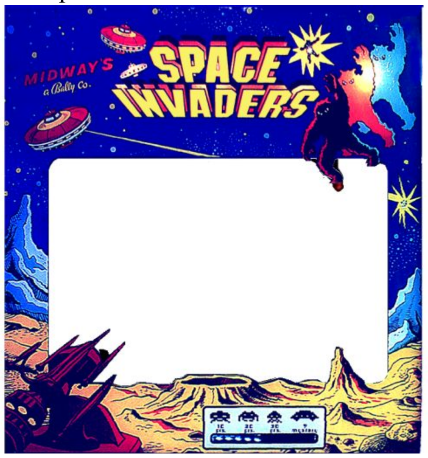

Figure 10.1. Space Invaders
http://www.classicgaming.cc/classics/spaceinvaders/index.php

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 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 for 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 mode.

System Requirements for Pipe Dream

Pipe Dream is a game played on a grid. The player receives pipe pieces that go on the grid. Possible pieces are a 90 degree turn, a straight piece, and a cross (two straight pieces on top of each other.) Players are given these pieces randomly and cannot rotate them, and they cannot replace any already placed pieces. Which piece is to be placed next is shown to the player. The object of the game is to construct a path for water to flow using the pipe pieces given from a designated starting point on the grid to another designated end point before time runs out. Players receive a score based on how long their final path of piping was, but if they fail to connect the two points the game will be over. Inputs will select a grid space, and a single button to place the next piece available in the selected space. There will be a sound for a pipe section being placed, as well as game over and win sounds. This game must satisfy the **System Requirements for all games** listed above. Possible additions include (but not required):

- Use two slide pots for moving
- Add a "queue" of pieces that shows not only the next piece to be placed, but the next 3-5 pieces as well
- Add obstacles on the grid where pipes may not be placed (or can be placed with a score penalty)
- Add already placed pipe sections that give bonus points if used in the solution
- Instead of a time limit, have a flow of water start running through the pipes. If the water gets to an open pipe the player loses, and if the water gets to the end point the player wins.

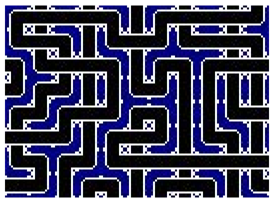
Figure 10.2 A pipe dream screen.

System Requirements for Design Your Own Project

The first step is to make a proposal. In the Lab10Files folder you will find a proposal for Super Breakout (SuperBreakout.doc). If selected your game will be added as an official choice that you and other EE319K students would be free to choose. Good projects require students to integrate fundamental EE319K educational objects such as I/O interfacing, data structures, interrupts, sound, and the effective implementation of real-time activities. This project must satisfy the System Requirements for all games listed above.

Any TA is authorized to approve an alternate game. **If the alternate game is approved the one-page description should be printed and posted in the lab, and thus available for other students to implement.**

Valvano has eight 2-axis joysticks. These joysticks can be checked out on a first come first served basis, and used for the design 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. To checkout a joystick, first you must get your design your own game approved by your professor, and then come to Valvano. These joysticks must be returned.

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 - 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 Lab10Files directory of the starter project. You can create additional sprites as needed using a drawing program like Paint. Most students will be able to complete Lab 10 using only the existing sprites in the starter package. Because of the way pixels are packed onto the screen, we will limit the placing of sprites to even addresses along the x-axis. Sprites can be placed at any position along the y-axis. Having a 2-pixel black border on the left and right of the image will simplify moving the sprite 2 pixels to the left and right without needing to erase it. Similarly having a 1-pixel black border on the top and bottom of the image will simplify moving the sprite 1 pixel up or down without needing to erase it. You can create your own sprites using Paint by saving the images as 24-bit BMP images. Make all your images very small so you do not use up your allotment of 32k on the free version of the compiler. Figure 10.3 is an example BMP image. Because of the black border, this image can be moved left/right 2 pixels, or up/down 1 pixel. Use the **BmpConvert16.exe** program to convert the BMP image into a two-dimensional array that can be displayed on the LCD using the function `ST7735_DrawBitmap()`. For this part of the procedure you will need to write main programs for drawing and animating your sprites.

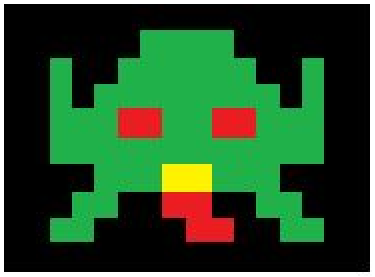

Figure 10.3. Example BMP file. Each is 24-bit color, 16 pixels wide by 10 pixels high.

Program 10.1 shows an example image in C program format. This image is 16 pixels wide by 10 high, requiring 320 bytes of ROM storage. Notice the bug in **BmpConvert16.exe** (you could delete the extra comma at the end)

```
const unsigned short Scary4[] = {
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,// bottom row
 0x0000, 0x0000, 0x4D84, 0x4D84, 0x0000, 0x0000, 0x0000, 0x0000,
 0x20FD, 0x20FD, 0x0000, 0x0000, 0x4D84, 0x4D84, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x4D84, 0x4D84, 0x0000, 0x0000, 0x20FD,
 0x20FD, 0x0000, 0x0000, 0x4D84, 0x4D84, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x4D84, 0x4D84, 0x4D84, 0x079F,
 0x079F, 0x4D84, 0x4D84, 0x4D84, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x4D84,
 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x0000, 0x0000,
 0x0000, 0x0000, 0x4D84, 0x4D84, 0x4D84, 0x20FD, 0x20FD, 0x4D84,
 0x4D84, 0x20FD, 0x20FD, 0x4D84, 0x4D84, 0x4D84, 0x0000, 0x0000,
 0x0000, 0x0000, 0x4D84, 0x0000, 0x4D84, 0x4D84, 0x4D84, 0x4D84,
 0x4D84, 0x4D84, 0x4D84, 0x4D84, 0x0000, 0x4D84, 0x0000, 0x0000,
 0x0000, 0x0000, 0x4D84, 0x0000, 0x0000, 0x4D84, 0x4D84, 0x4D84,
 0x4D84, 0x4D84, 0x4D84, 0x0000, 0x0000, 0x4D84, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x4D84, 0x4D84,
 0x4D84, 0x4D84, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // top row
};
```
Program 10.1. Example image written as a C constant allocated in ROM.

In this case the height for this sprite is 10 pixels. The `ST7735_DrawBitmap()` function assumes the entire image will fit onto the screen, and not stick off the side, the top, or the bottom. As mentioned earlier, you must save the BMP as a 24-bit color image. **BmpConvert16.exe** program will reduce the 24-bit color into 5-6-5 RGB 16-bit format. The 16-bit color will be displayed on the ST7735 LCD. The 0xFFFF is maximum brightness or white, and the color 0x0000 is off or black. The BMP format will pad extra bytes into each row so the number of bytes per row is always divisible by 8. However the **BmpConvert16.exe** program will remove the padding an create a simple linear array of the 2-D image.

Part c - Extend Random Number Generator

The starter project includes a random number generate. 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
```
unsigned long Random5(void){
  return ((Random32()>>24)%5)+1;  // returns 1, 2, 3, 4, or 5
}
```
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 d - Test Subsystems

When designing a complex system, it is important to design implement and test low-level modules first (parts a,b,c). In order to not be overwhelmed with the complexity, you must take two working subsystems and combine them. Do not combinate 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.

Commit to SVN the final version of the software that implements your game. All source files that you have changed or added should be committed to SVN. Please do not commit other file types.

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

Competition Structure

Step 1 - Certification

Do this before lab the last week of classes

You will show the TA your program operation on your TM4C123 board. The purpose of certification is to certify the game meets specifications, and to guarantee the SVN is committed. Certification will occur during the lab periods and office hours. There will not be time to certify during the class competition. If your system runs and meets the specifications, it will be invited to the design competition. The TA will place your game into one or two categories. The **wonderful** category will have a maximum of 100 points, and the **supreme** category has a possible grade of 110 points. We expect most projects to be invited. The details of the competition will be explained in class. The dute date is 2 hours before the start of the last lecture class of the semester. Late checkouts are handled by the TA in the usual way. All late checkouts must be completed by Friday 3pm, and materials checked out from the ECJ lab counter must be completed by 5pm Friday.

Step 2 - Design Competition

Occurs during the last class period of the semester (details announced in class)

For each project you will print one page (size 12 font) of some of your Lab 10 code representing software that you think is well-written and creative. You will pull power from a computer to run your game. One team member will be stationed at your game to help explain and run your system. The other team member will be given a grading sheet and will visit/evaluate 8 other projects. The graders should bring their own headphones to minimize the chance of spreading germs. The graders will spend 4 minutes with each project, and then rotate to the next project. The grading is a rank order and will be secret. The scores should be based on software quality, fun and function. *Fun will be had by all.*

Step 3 - Final Competition

Occurs during the last week of classes (details announced in class)

The top scoring game from each class will be invited to the finals. In addition, the TAs will choose a second game from each class. This will make a total of 10 games for the final. All students from EE319K are invited: cookies and carrots provided. Each person entering the room will be given ONE voting card. If you are invited to compete, bring your game and a 30 second story of something special about your game. At noon we will write the names of the 6 groups on the white board making 6 columns

Between at 3:05-3:45 we will demonstrate the games

1. Set up, 1 minute
    a. Plug the audio jack into your device (we will have amplified speakers)
    b. Position the LCD under the document camera
    c. Plug the USB cable into power (we will have USB power)
2. Game play 3 minutes, it is best to talk narrate while you play, helpers allowed
3. Summarize major features so audience will remember you, 30 seconds
4. Disconnect, 30 seconds (at this point you should ceremoniously vote for yourself)

If you have not been invited to compete, and you are proud of your game, bring it and a power source. Set it up and invite others to play. We expect and encourage talking and walking around.

After the demonstrations have completed, students who wish to vote will cast their vote by placing their ONE card under/by the name of their favorite project

Hint

1. Say you want to convert a wav file **(blah.wav**) you want to use in your game. Here are the sequence of steps you can do in matlab to make this happen:
    a. Read the file and extract the sample as well as the frequency:
    ```
    [Spls, fs] = wavread('blah.wav');
    ```
    b. Downsample it, to get it to the frequency you want (11.025 kHz):
    ```
    Spls = downsample(Spls, round(fs/11025));
    ```
    c. Adjust the samples (originals are fractions between -1 and 1) to range between 0 and 15 (4-bit)
    ```
    Spls = round((Spls+ 1)* 7.5);
    ```
    d. write it to a file so you can cut and paste it into your C code:
    ```
    file = fopen('blah.txt', 'w');
    fprintf(file, 'unsigned char blah[] = {');
    fprintf(file, '%d,', Spls);
    fprintf(file, '};\n');
    ```

That's it, you should have a file (called **blah.txt**) with a declaration you can cut and paste in your code. There may be an extra commas in the output that you would have to remove. You could make a convert.m file with these statements inside a subroutine and have a converter that can be invoked for conversion from matlab. (From Mitch Crooks)