Assignment 4: PRU Morse Code

- May be done **individually** or in **pairs**.
- Do not give your work to another student, do not copy code found online without citing it, and do not post questions about the assignment online.
  - Post questions to the course discussion forum.
  - You may use any code you or your partner have written for this offering of ENSC 351.
- Submit deliverables to CourSys: https://coursys.sfu.ca
- See the marking guide for details on how each part will be marked.

1. Morse Code

In this assignment you will create a Linux application which uses the PRU to flash out Morse Code on an LED.

Morse Code (see Wikipedia)

- Morse code is a way of transmitting text (such as a message "Hello World") as short and long beeps, or in our case, short and long flashes on an LED. The short beeps/flashes are called "dots", and the long ones called "dashes".
- A "dot" is the basic unit of time.
- A "dash" is three dot-times long.
- Spacing
  - Each dot/dash is separated by one dot-time. During this time the LED is off.
  - Two letters in a message are separated by three dot-times. During this time the LED is off.
  - Trim extra whitespace (‘ ’, ‘\n’, ‘\r’, ‘\t’) from the end of a message; do not have any wait after the last dot/dash of the message is flashed out.
  - Each whitespace is equal to seven dot-times total (no additional 3 dot-time inter-character delay).
- See the Wikipedia link for more explanation of timing.
- Morse code does not differentiate between upper and lower case letters.

2. Folder Structure & Names

- Create a new folder to store your assignment 4 code (you may name it anything you like).
  - Have a subdirectory for your Linux app: `as4-linux/`
  - Have a subdirectory for your PRU app: `as4-pru/`
- Submit a single TAR file containing all of your project (Linux, PRU, and `makefile`).
- When the TA extracts this TAR file it must have a Makefile in the root directory which builds and deploys the Linux application (by running the makefile in `as4-linux/`) and copies the PRU code (`as4-pru/`) to the NFS folder.
- The PRU code folder must have a makefile which can build (`make`) and install (`make install_PRU0`) the code.
- Naming:
  - The compiled Linux application must be:
    `$(HOME)/cmpt433/public/myApps/`**`as4-morsecode`**
  - Your Makefile must copy your `as4-pru/` folder to become:
    `$(HOME)/cmpt433/public/pru/as4-pru/`

In your assignment 4 directory (named anything you like), you must have the following structure. Please name the folders (`as4-linux` and `as4-pru`) as shown. You will have more files beyond what is shown, especially for the Linux app!

```
(host)$ cd myAs4Director/
(host)$ tree
.
├── as4-linux
│   ├── makefile
│   ├── myLinuxCode.c
│   ├── myManyModules.c
│   └── myManyModules.h
├── as4-pru
│   ├── AM335x_PRU.cmd
│   ├── makefile
│   ├── myPruCode.c
│   └── resource_table_empty.h
└── makefile
```

From within your assignment 4 project directory, generate an archive for submission with the following command (in bold):

```
(host)$ cd myAs4Director/
(host)$ tar cvzf as4.tar.gz *
as4-linux/
as4-linux/myManyModules.c
as4-linux/myManyModules.h
as4-linux/makefile
as4-linux/myLinuxCode.c
makefile
as4-pru/
as4-pru/makefile
as4-pru/myPruCode.c
as4-pru/resource_table_empty.h
as4-pru/AM335x_PRU.cmd
```

To build your submission, we will:

1. Extract your archive
   `(host)$ tar -xf as4.tar.gz`
2. Run your makefile:
   `(host)$ make`
3. Build and install your PRU code:
   `(bbg)$ cd /mnt/remote/pru/as4-pru/`
   `(bbg)$ make`
   `(bbg)$ make install_PRU0`
4. Run your Linux app (as root so it can access `mmap()`)
   `(bbg)$ sudo /mnt/remote/myApps/as4-morsecode`

You can make no assumptions about either the current user's name, or where we will extract your code, so don't use relative paths to get to the above locations; use `$(HOME)` instead.

You may find the following makefile to be useful in the root of your project:

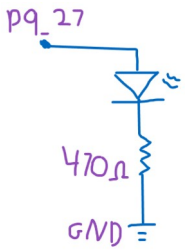
```
all: nested-linux pru-copy

nested-linux:
	make --directory=as4-linux

pru-copy:
	mkdir -p $(HOME)/cmpt433/public/pru/
	cp -r * $(HOME)/cmpt433/public/pru/
	@echo "DO NOT MODIFY: COPY ONLY" > $(HOME)/cmpt433/public/pru/COPY_ONLY
	@echo ""
	@echo "You must build the PRU code on the target, then install it:"
	@echo "(bbg)$$ cd /mount/remote/pru/as4-pru"
	@echo "(bbg)$$ make"
	@echo "(bbg)$$ sudo make install_PRU0"
```

3. Wiring

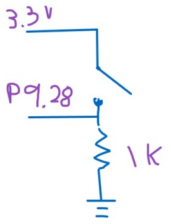
The following is the required hardware configuration for completing this assignment.

- **8x8 LED Matrix** (see I2C Guide)
  (Same as previous assignments)
  - P9.01 (`Ground`) to "-" pin on the LED Matrix
  - P9.03 (`3.3V`) to "+" pin on the LED Matrix
  - P9.17 (`I2C1-SCL`) to "C" pin on the LED Matrix
  - P9.18 (`I2C1-SDA`) to "D" pin on the LED Matrix
  - Orient display with pins at the bottom, and display pointing out the top of the breadboard. (This is 180 degree rotation: text on the "backpack" board upside down).

- **LED (for PRU)**
  - P9.27 (`GPIO`) Connect the LED in series with a ~470 ohm resistor to ground.
  - P9.01 (`Ground`)



- **Button (for PRU)**
  - P9.03 (`3.3V`) Connect a button in series with a ~1k ohm (pull-down) resistor to ground.
  - P9.01 (`Ground`)
  - P9.28 (`GPIO`) Connect the GPIO input pin between the button and resistor.



For the LED matrix, you may assume that all GPIO pins are already exported, and are configured for GPIO on the Linux processor. However, **you may not assume any needed pins are configured for GPIO by the PRU.**

You Linux app must run the `config-pin` commands required for the PRU GPIO pins.

4. Features

4.1 Keyboard Input

Repeatedly prompt the user to type some text and flash it out as Morse code.

- Display a prompt on the terminal ('>') and allow user to type in a line of text.
  See hints section for use of `getline()` and `fflush()`.
- If the user just presses ENTER without any text entered the application must exit gracefully, shutting down smoothly. It's OK to leave the PRU running.
- If the user enters some text, then:
  1. Trim whitespace from the end of the input string.
  2. Display how many characters are in the input after trimming, and print the string of text to the screen.
  3. Encode the message into Morse code and display the encoded Morse code message to the screen.
     - For each dot-time, print out an 'X' if the light will be on, or print out a '_' if the LED will be off.
  4. Send message or encoded Morse code to the PRU to flash out (see below).
  5. Wait until the PRU is finished flashing, then prompt the user for another line of text.

4.2 Morse Code Generation

Your program (Linux and PRU) must somehow work together to convert the text into Morse code. You *may* use the provided Morse code encoding file (see the course website). You may ***not*** use code from another source for processing or generating Morse code. You are expected to write the code that does the conversion.

- If your message contains multiple whitespaces in a row, you can choose to either have 7-dot times off per whitespace, or just one 7-dot times off total time.
- Ignore characters which are neither whitespace nor letters (A-Z).
- Input must be case insensitive.

4.3 PRU Flashing

- Your PRU code must be responsible for flashing out the actual Morse code.
- The dot-time depends on if the user is currently pressing the button.
  - If the button is not pressed, the dot-time is 300ms.
  - If the button is pressed, the dot-time is 1000ms.

  The dot-time is decided per dot-time being transmitted, so the user may press and release the button multiple times during a transmission if they like.
- When finished flashing out a message, the LED must end off.
- Your Linux app must transfer to the PRU either the text of the message, or the Morse code encoding of the message. It will likely use shared memory for this transfer.
- The PRU app must tell the Linux app what part of the message is currently being transmitted so that it can display the status on the terminal and LED matrix (see below).

4.4 Print to Terminal while Flashing

- While the PRU is flashing out the message, have the Linux app print to the terminal (on one line):
  - an '`X`' when the PRU starts a dot time with the LED on.
  - a '_' when the PRU starts a dot time with the LED off.
- After the messages finished flashing, print a linefeed.

4.5 8x8 LED Matrix

Use the 8x8 LED matrix to show the current and upcoming dot times.

- While the PRU has no message to flash out (waiting for next input string), the 8x8 LED matrix should be blank.
- While the PRU is flashing out dots and dashes, have the Linux app show on the 8x8 LED matrix a waveform of upcoming dot times.
  - Each column of the display relates to one dot time.
  - The left most column represents the current dot time; the remaining 7 columns are the 7 future dot times.
  - If the LED will be turned on during a dot time, turn on the bottom half (4 rows) of the column.
  - If the LED will be turned off during a dot time, turn off all the LEDs in the column.
- As the PRU flashes out the pattern, the display updates in real-time.

Image 1: Displaying start of a dash, followed by two dots.

4.6 Memory Testing

We will run Valgrind on your code to look for incorrect memory accesses and leaks. When your program ends, Valgrind must not find any definitely lost blocks of memory. Still in use or possibly lost is fine.

4.7 Suggestions and Hints

- You may reuse code from your Assignments 1, 2, and/or 3 (from ENSC 351, this semester) which either you or your partner wrote.
- Make C modules for each input and output device. Give them easy to use .h files and test each module before integrating it with the rest of your application.
- If you print something to the terminal but don't want to print a linefeed yet (say the ">" for the prompt, or the "_" and "X" characters while the Morse code is flashed out) you will need to flush the file buffer after each `printf()`:

```
fflush(stdout);
```

- Use `getline()` to read in from the keyboard:

```
char *buff = NULL;
size_t sizeAllocated = 0;
size_t numCh = getline(&buff, &sizeAllocated, stdin);

// Now use buff[] and it's numCh characters.
// You can ignore sizeAllocated.
// ..<your code here>..

// Cleanup from getline()
free (buff);
buff = NULL;
```

- To communicate between the Linux app and the PRU, define a shared struct containing all the info you need to transmit.

- Have the Linux app convert the input text into Morse code. For example, create an array of bool values, where each element corresponds to one dot time and represents the LED being on (true) or off (false).
  - You can assume that any message will require less than 2048 dot times.
  - You can pre-allocate the maximum size array in your struct
  - As with all array, you'll need to pass how much valid data you have. Include this in your struct.
- Have a flag that indicates if there is data to flash out. Your Linux app can set this and PRU clear it.
- The PRU has to report its current progress, which depends on timing and if the user presses the button. Include in your struct a field for the PRU to list the current dot time that it's flashing out (i.e., an index into your encoded Morse code dot-times).

5. Screen shot

```
debian@BeagleBone:~$ sudo /mnt/remote/myApps/as4-morsecode
[sudo] password for debian:
Beginning Morse Code!!
> hello world
Flashing out 11 characters: 'hello world'
X_X_X_X___X___X_XXX_X_X___X_XXX_X_X___XXX_XXX_XXX_______X_XXX_XXX___XXX_XXX_XXX___X_XXX_X___X_XXX_X_X___XXX_X_X
X_X_X_X___X___X_XXX_X_X___X_XXX_X_X___XXX_XXX_XXX_______X_XXX_XXX___XXX_XXX_XXX___X_XXX_X___X_XXX_X_X___XXX_X_X
> SOS
Flashing out 3 characters: 'SOS'
X_X_X___XXX_XXX_XXX___X_X_X
X_X_X___XXX_XXX_XXX___X_X_X
> s o s
Flashing out 5 characters: 's o s'
X_X_X_______XXX_XXX_XXX_______X_X_X
X_X_X_______XXX_XXX_XXX_______X_X_X
> e i s h t m o
Flashing out 13 characters: 'e i s h t m o'
X_______X_X_______X_X_X_______X_X_X_X_______XXX_______XXX_XXX_______XXX_XXX_XXX
X_______X_X_______X_X_X_______X_X_X_X_______XXX_______XXX_XXX_______XXX_XXX_XXX
> eishtmo
Flashing out 7 characters: 'eishtmo'
X___X_X___X_X_X___X_X_X_X___XXX___XXX_XXX___XXX_XXX_XXX
X___X_X___X_X_X___X_X_X_X___XXX___XXX_XXX___XXX_XXX_XXX
>
Shutting down application.
Done!
debian@BeagleBone:~$
```

See course website for a video of product in operation.

6. Deliverables

Submit the TAR file to CourSys (https://coursys.sfu.ca/) as described in Section 2.

Since the assignment can be done individually or in pairs, if you are working individually you'll still need to create a group in CourSys to submit the assignment.

Remember that all submissions will automatically be compared for unexplainable similarities from both this semester, and previous semesters!

