Assignment 4: PRU + Linux - Find the Dot

- This assignment is to be done **individually or in pairs**.
- Post questions to Piazza or Discord.
- Do not give your work to another student/group, do not copy code found online, and do not post questions about the assignment to other online forums.
  - **You may use all sample code Dr. Brian has posted on the course website this semester, or the code posted for this assignment.**
- See the marking guide for details on how each part will be marked.

1. Application Description

Create a game where the user angles the board up/down and left/right to find a hidden dot. The user will be guided by the NeoPixel light strip, use the joystick to shoot the dot, and hear sounds from the buzzer.

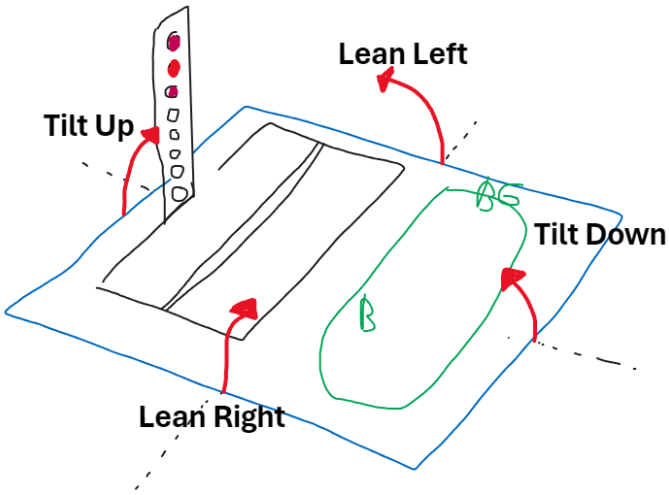
The sections below cover the required functionality in the rough order that it is suggested you implement the application in the hopes of making the assignment less time-consuming to implement. For more of a challenge, read through the whole assignment and then implement features in your own order.

1.1 NeoPixel

- Have the PRU control your NeoPixel LED strip.
- Use shared memory to allow the Linux application to control what the LED displays.

1.2 Point and Accelerometer

- When the application starts up, randomly pick a "point" somewhere on in the range of X:[-0.5, +0.5], Y:[-0.5, +0.5]
- Use the Zen cape's accelerometer's readings to determine which way the board has been rotated (uses gravity). This determines which way the board is "pointing". The figure to the right shows the idea of rotating the board.



  - Hold the hardware so that the BBG is near you and the breadboard away from you.
  - When the bread-board end (with NeoPixel light strip) is tilted/angled up, treat it as pointing up. When tilted down, treat it as pointing down.
  - When the board is tilted to the left treat it as pointing left. When tilted right, treat it as pointing right.
  - From the BBG's rotation, determine where the board is "pointing". Level gives ~0.0; ~45 degrees angle gives a value of +/- 0.5; and 90 degree gives a value of +/-1.0
- Control what is displayed on the NeoPixel LED strip based on where the board is pointing relative to the randomly chosen point:

- Colour indicates where the point is left or right:[1]
  - If the point is to the left (i.e., need to rotate it left), use a red colour.
  - If the point is to the right (i.e., need to rotate it right), use a green colour.
  - If the point is centred left-right, use a blue colour.
- On LEDs indicate where the point is up or down:
  - Use three LEDs to draw a target showing approximately where the chosen point is.
    - Make the middle of the three LEDs brighter than the other two.
  - If the board is aiming below the point, target appears on top half of the strip.
  - If the board is aiming above the point, target appears on bottom half of the strip.
  - If the board is aiming at the correct height, turn on all LEDs to indicate on target.
  - The following table shows what the LEDs should look like
    - `o` means on (but not super blight)
    - `X` means on and bright
    - `-` means off

Aiming way below the point					Aiming directly at the point's height					Aiming way above the point
o	X	o	-	-	X	-	-	-	-	-
-	o	X	o	-	X	-	-	-	-	-
-	-	o	X	o	X	-	-	-	-	-
-	-	-	o	X	X	o	-	-	-	-
-	-	-	-	o	X	X	o	-	-	-
-	-	-	-	-	X	o	X	o	-	-
-	-	-	-	-	X	-	o	X	o	-
-	-	-	-	-	X	-	-	o	X	o

  - See the video explaining how the LEDs must behave; it's easier to show that describe!
- Program must read the accelerometer, recompute the LED values, and display it on the NeoPixel LED strip at least 10 times per second (much more often is also fine!).
- Consider the board to be pointing at point if it is within +/- 0.1 on our scale here.

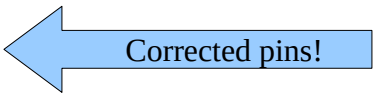
Tips

- When tilted all the way one direction, it will correspond to -1 / +1 on one axis of the accelerometer.
- Experiment with tilting the board to determine which of the X/Y/Z readings change as you rotate the board. These axes name and its orientation may not match your initial expectations. As long as the game plays smoothly on your hardware, that's what we are looking for!
- You can likely ignore the Z axis's reading.
- Have the Linux app compute which LEDs to turn on, and what colours they should be, and then transmit the full "frame" of LED colours to the PRU for it to display.
- You can pick the specific shades of each colour to use.

1. Colours relate to starboard (left) is red and port (right) is green.

1.3 Joystick

- **The PRU mus**t continuously read the Zen cape's joystick DOWN and RIGHT pins.
- It must report these values to the Linux app for it to make decisions.[1]
- Zen cape pins for joystick:
  - Down: P8_16 (used for "fire")
  - Right: P8_15 (used to exit app)



- If the user presses down on the joystick (fire) while the board is aiming at the point, then the point is considered "hit".
  - The Linux application should then generate a new point for the user to fire at.
- For fun (no marks) you may make firing, or hitting the point, trigger a brief animation on either the 14-seg display or the NeoPixel. (Only play with this once you have completed the rest of the assignment!)

1.4 14-Seg Display

- Show the number of times the user has hit the dot on the 14-seg display.

1.5 PWM Buzzer

- When the user fires (joystick down), play sequence of sounds on the PWM buzzer.
  - If the user hits the target, the sound played should reasonably indicate a hit.
  - If the user misses the target, the sound played should reasonably indicate a miss.
- There is a lot of flexibility around what is reasonable, but at a minimum:
  - Each sound played should consist of numerous notes or frequencies.
  - The sounds should be quite different.
  - The sounds should last between 0.5s and 3s.
  - The application must continue to operate while playing back sounds (background thread).
  - OK for a new sound to stop a previous sound (need not have multiple sounds in playback at once).
  - User must be able to reasonably fire a miss or a hit and trigger a single sound.
  - User must be able to hold the joystick down to trigger the sounds reasonably quickly, but still sound good.
- It is expected to run the app using `sudo` to give it permission to access PWM files.

Tips

- If your app crashes and leaves the buzzer buzzing, you can temporarily silence it by removing the black jumper on the Zen cape between the round buzzer and the P9 header.
- To playback a sound, have a thread dedicated to telling the buzzer to play different frequencies over time.

1.6 Startup / Shutdown

- At startup, the Linux application must run all necessary commands to configure the pins for the application's needs. Specifically, it likely needs to:
  - Map the NeoPixel and Joystick Right/Down pins to the PRU
  - Map the 14-seg display control pins to Linux GPIO (this is done when the board reboots; however, some PRU sample code needs the pin mapped to the PRU, so you'll save yourself some frustrating debugging by having the app setup these pins too!)

1. *Yes, it would be easier to just have Linux read these pins; however, this demonstrates that the PRU can send data to the Linux app.*

    - Setup PWM pins
- When the user presses RIGHT on the joystick, the application must shutdown promptly. At shutdown, the app must:
    - free all allocated memory (no lost, none still accessible)
    - Stop any sound on the buzzer[1]
    - Turn all NeoPixel LEDs off
    - Display nothing on the 14-seg display.

1.7 Troubleshooting Tips

All of these are optional!

- Use the console output any way you like! I suggest you frequently display things like:
    - What X/Y position the board is pointing to.
    - The X/Y position of your point; possibly the distance between where it is pointing and the target.
    - Which LED is the "centre"
    - State of the joystick right/down pins read by PRU

- Test that shutdown works correctly.
    - Try pressing Right immediately after pressing down (should trigger a sound):
    - App should turn off displays and stop sound.

- **Unable to successfully get PRU working?**
    - Run sample code to prove hardware is wired correctly.
    - Check your application correctly runs the necessary `config-pin` commands.
    - Add a new int value to the start and end of your shared memory. Have PRU set some interesting value in these, such as a counter of how many times it has run through it's main loop. Display these values in the Linux terminal every second. This will prove you can communicate from the PRU to Linux, and thhe PRU code is still running.

- **Buzzer not working?**
    - Ensure the jumper is connecting the buzzer pins (between the round black buzzer and the P9 header).
    - Follow the PWM guide to test the process on the command-line first.

- **14-seg display not working?**
    - Ensure you have run the correct `config-pin` commands to map the digit-enable lines to be GPIO pins and not map them to the PRU
    - Ensure you are running code for the correct Zen colour (hardware is different).

- **Accelerometer not working?**
    - Ensure you are running code for the correct Zen colour (hardware is different).
    - Reboot the board because incorrect commands can cause it to be in a funny state.

- **Game not working?**

1. *This is a critical requirement, for everyone's peace of mind!*

- Ensure you have all the basic modules working (reading joystick, accelerometer, ...).
- Use the screen to print status information that can help you debug.

2. Deliverables

Submit the items listed below to the CourSys: https://courses.cs.sfu.ca/

1. `as4-finddot.tar.gz`
   Compressed copy of source code and CMake build script. This folder must include your code to both the Linux app and the PRU code.

   Hint: Compress the `as4/` directory with the command
   ```
   $ tar cvzf as4-finddot.tar.gz as4
   ```

To submit, you'll need to create a group. If you worked individually, it will be a group of 1.
Remember that all submissions will automatically be compared for unexplainable similarities.

2.1 Informal Milestones

- How to start
  - If you want to work with a partner, start looking for one!
  - Follow the guides to get started:
    - PRU Guide
    - PWM Guide
  - Read all sections of the assignment. Design HAL modules. Think about what modules your application will have. Should you have a HAL module to communicate with the PRU?
  - Think about the application design. What data needs to be sent to the PRU? What data needs to be read from the PRU? How will you handle playing notes on the PWM?
  - Think about how you will shutdown the application correctly.
  - Get the low-level PWM and PRU code routines working smoothly.
  - Get your Linux app being able to reliably send colours to the PRU to display.
- **Half done**
  - Linux can display pattern on the NeoPixel strip.
  - PWM code able to play a little tune (sequence of notes).
- **Final checks**
  - Ensure correct behaviour when triggering 2 sounds quickly.
  - Ensure 14-seg displays score smoothly.
  - Double check your final ZIP file for correctness! No last minute refactoring bugs! Ensure both your Linux and PRU code are in the archive.

