Lecture 5: Project 0 discussion

- Coding: set input/output ports
- Coding: input monitoring
- Implementation strategy

Set input/output ports

- Each individual pin of a port is tri-state.
- We can change the direction of each pin through programming the *TRISx* special-function register.
- The rule is simple to remember:
  - Clear a bit to **0** for an Output pin.
  - Set a bit to **1** for an Input pin.
- Example:

```
// set PORTA as inputs
TRISA = 0xFF;
```


Set input/output ports

- What about setting PORTB as outputs?

```
// set PORTB as outputs
TRISB = 0;
```

- This is not enough! PORTB pins can also be use for analog input function.
- Therefore, we have to first set PORTB pins as digital. This can be done by specifying each bit of AD1PCFG to 1 (see Figure 1.7 of your textbook).

```
// set PORTB as outputs
AD1PCFG=0xffff;    // all PORTB as digital
TRISB=0,           // all PORTB as output
```


Set input/output ports

- Buttons 1 and 2 correspond to pins 6 and 7 of PortA
- LEDs 1~4 correspond to pins 10~13 of PortB
- How do we figure these out?
  - These pins are defined in the "Cerebot 32MX4_sch_D.pdf" (page 4)
- How to program only pins 6~7 of PortA and pins 10~13 of PortB?
  - Use bitwise AND and OR operations!

```
TRISA |= 0xC0;     // set bits 6~7 to 1
TRISB &= 0xC3FF;   // set bits 10~13 to 0
```

Input monitoring

A straightforward extension of template code looks like this:

```
main() {
    // 1. configure ports
    // 2. initialize internal variables
    while(1)  {
        if (Btn1) {
            // 3. debouncing
            // 4. set next states and outputs
            //    based on current states
        }
        else if (Btn2) {
            // 5. debouncing
            // 6. set next states and outputs
            //    based on current states
        }
        // 7. loop for controlling rotation frequency
    }
}
```

Does this code work?

Input monitoring

A straightforward extension of template code looks like this:

```
main() {
    // 1. configure ports
    // 2. initialize internal variables
    while(1)  {   <-- no way to detect if a button is released!
        if (Btn1) {
            // 3. debouncing
            // 4. set next states and outputs
            //    based on current states
        }
        else if (Btn2) {
            // 5. debouncing
            // 6. set next states and outputs
            //    based on current states
        }
        // 7. loop for controlling rotation frequency
    }
}
```

no way to detect if a button is pressed/released!

Does this code work? NO

Input monitoring

- How to detect button press/release?
  - Use a variable to "lock" state transitions when a button is pressed
  - Only perform state transitions when the variable is unlocked

```
if (XXX && !locked) {
    …
    …
    locked = 1;
}
```

  - "Unlock" state transitions when both buttons are released

```
if (YYY && ZZZ) {
    …
    …
    locked = 0;
}
```


Implementation Strategy

- Two possible strategies:
  - Start coding the entire project and test it when we are done.
  - Devise a baby step methodology in which only small pieces of code are tested at a time.
- Which one do you choose?

> ➢ Our recommendation: since there is a lot of room for human mistakes, the second one is a much safer approach.
> ➢ Our goal: partition the project in a way that the smaller pieces are easier to code & debug

Implementation Strategy

4 recommended steps:

- LED rotation only
  - Be able to set LED 1 as always-on and rotate LEDs on the Pmods at a frequency of about 1Hz
- Initial/Reset LED rotation
  - Be able to use Btn1 to reset all LEDs to 0
- Play/Pause LED rotation
  - Be able to use Btn2 to pause LED rotation
- Integration and testing
  - Test corner cases (e.g., press both buttons simultaneously, press one button before releasing the other…)