EENG 3910: Project V – Digital Signal Processing System Design
Assignment 2 Part 2
Due 15 February 2016
Nathan Ruprecht
UNT – Electrical Engineering

Introduction

The purpose of part 2 of assignment 2 is to get us involved with using interrupts and coding to show results on the board as well as putty. Part 1 was analyzing what was happening, whereas part 2 is making changes and seeing the differences.

Results and Discussion

Problem 5: Programming onboard pushbuttons

Including the libraries and header files to be used. Defining variables to make coding more user friendly. For example, typing "RED_LED" is easier for most people to understand as opposed to "GPIO_PIN_1." Defining functions and constants.

In main, it initializes local variables, sets the clock at 40 MHz, and initializing functions such as setting up the LEDs, push buttons, UART, and timer. Sets up the interrupt and timers then displays the original text to putty screen. If there is a user input (keyboard input), then it goes into the switch statement to change the LED color. If there is a button press, it outputs a statement to the screen of the button being pushed, and then another when it is released.

I think Dr. Li explained bouncing best in class. When the button is pushed, the signal actually bounces before being steady. So you would see it go from off, multiple spikes/bounces between on and off, and then levelling out and staying on. If the program runs correctly just off of that, the interrupt flag is set every spike. So in changing LED colors, it would change colors during every spike. So to fix it on the software side, you make another variable with a timer. If the button is pushed, the system sees that first spike and starts the clock for that variable. If the signal is still high (on) after that time, the signal has levelled out to be "on" and the interrupt runs just once to change the LED color. Since the spikes happen so fast and over a very small amount of time, the timer is a short amount of time and the naked eye doesn't see the delay. If there is noise and the signal spikes, the timer starts, but since it doesn't level out, the interrupt flag is reset, stays at 0, and the LED won't change.

Problem 6: Change LED color using onboard pushbuttons

I added 3 simple lines of code to what was already given. I assigned the current LED to what was mandated by the instructions. So when the left button was pushed, the LED changed to red. When the right button was pushed, LED became green. And blue when both were pushed.

Summary and Conclusion

This assignment seemed simple enough to change the current LED color to one of the three pins. A problem I have in understanding is how the board knows what cur_LED is. Just because I change the global variable cur_LED to RED_LED, how does that translate to the output toggling from another pin to pin 1 on the board? I think this assignment sets us up for future projects to refer back on syntax and structure of this code.

Modified Source Code:

```
// Check button states.
cur_button_states = button_states;
if(saved_button_states != cur_button_states){
	if((~saved_button_states & LEFT_BUTTON) && (cur_button_states & LEFT_BUTTON))
	{
		UARTprintf("Left button pushed down.\n> ");
		cur_LED = RED_LED;
	}
	if((saved_button_states & LEFT_BUTTON) && (~cur_button_states & LEFT_BUTTON))
	{
		UARTprintf("Left button released.\n> ");
	}
	if((~saved_button_states & RIGHT_BUTTON) && (cur_button_states& RIGHT_BUTTON))
	{
		UARTprintf("Right button pushed down.\n> ");
		cur_LED = GREEN_LED;
	}
	if((saved_button_states & RIGHT_BUTTON) && (~cur_button_states& RIGHT_BUTTON))
	{
		UARTprintf("Right button released.\n> ");
	}
	if(cur_button_states == (LEFT_BUTTON | RIGHT_BUTTON)) {
		UARTprintf("Both buttons held down.\n> ");
		cur_LED = BLUE_LED;
	}
```