Project 4
Keerthi Radhakrishnan
kr6eg

PWM

- **The goal was to design a PWM system that linearly varied the duty cycle of a square wave generated.**
- **With a slow duty cycle, the “ON” time is maximized, and the intensity of the light is thus maximized (and vice versa).**
- **The goal was to write C code that handled the linear adjustment of the duty cycle via interrupt vectors.**

ISR (Timer A0 Vector)

- Represents the ISR for the CCIFG0 flag. This flag is thrown every time the TAR is about to reset (right before the TAIFG flag is thrown).
- This ISR is responsible for the actual modification of the duty cycle.

Code (Timer A0 Vector)

```c
#pragma vector = TIMER_A0_VECTOR

__interrupt void mainClock(void) {
    if (pwmOn) {
        if (TA0CCR1 >= TA0CCR0) {
            isRising = 0;
        } else if (TA0CCR1 <= 0) {
            isRising = 1;
        }

        if (isRising) {
            TA0CCR1 += 0x1;
        } else {
            TA0CCR1 -= 0x1;
        }
    }
}
```

- **pwmOn represents the button variable for the second section.**
- **isRising is a char variable that represents the rising/falling state of the duty cycle.**

ISR (Timer A1 Vector)

- Represents the ISR for the CCIFG1 flag. This flag is supposed to be thrown at any time between 0% duty cycle and 100% duty cycle.
- This flag is responsible for actually toggling the LEDs with the frequencies specified by the duty cycle.

Code (Timer A1 Vector)

```
#pragma vector = TIMER_A1_VECTOR

__interrupt void changeDutyCycle(void) {
    switch (TAIV) {
    case TA0IV_TACCR1:
        P1OUT &= ~(BIT0 | BIT6);
        break;
    case TA0IV_TAIFG:
        if (TA0CCR1 > 0) {
            P1OUT |= (BIT0 | BIT6);
        }
        TA0CTL &= ~BIT0;
    }
}
```

RC Filter

- After making sure that this worked at all, we then designed an RC filter that would return a triangle wave (approximately) from the waveform described by this varying duty cycle square wave.
- In order to minimize noise, the response had to be close to a step response from a first order RC circuit.
- As a result, we chose to filter virtually every frequency aside from the fundamental, which was governed by CCR0.
- We chose a resistance of 1 million ohms, and a capacitance of 0.1 microfarads.

RC Filter Output & PWM signal

- **We can see the filter producing a triangle wave from the alternating high and low frequencies of the PWM signal.**

PWM Control

- The second part of this project involved using a button and its corresponding ISR to pause/unpause the PWM.
- Essentially the button functioned as a toggle here, and if it was set to the “on” state, then the PWM would continue.

Code Summary

- Virtually identical to the previous section's code, save for this extra ISR and the initialization code for the button itself.
- We caught a glimpse of the variable pwmOn as well.
- Essentially, within the routine for the button, pwmOn is a toggled variable.
- The other ISRs check for the status of this variable.

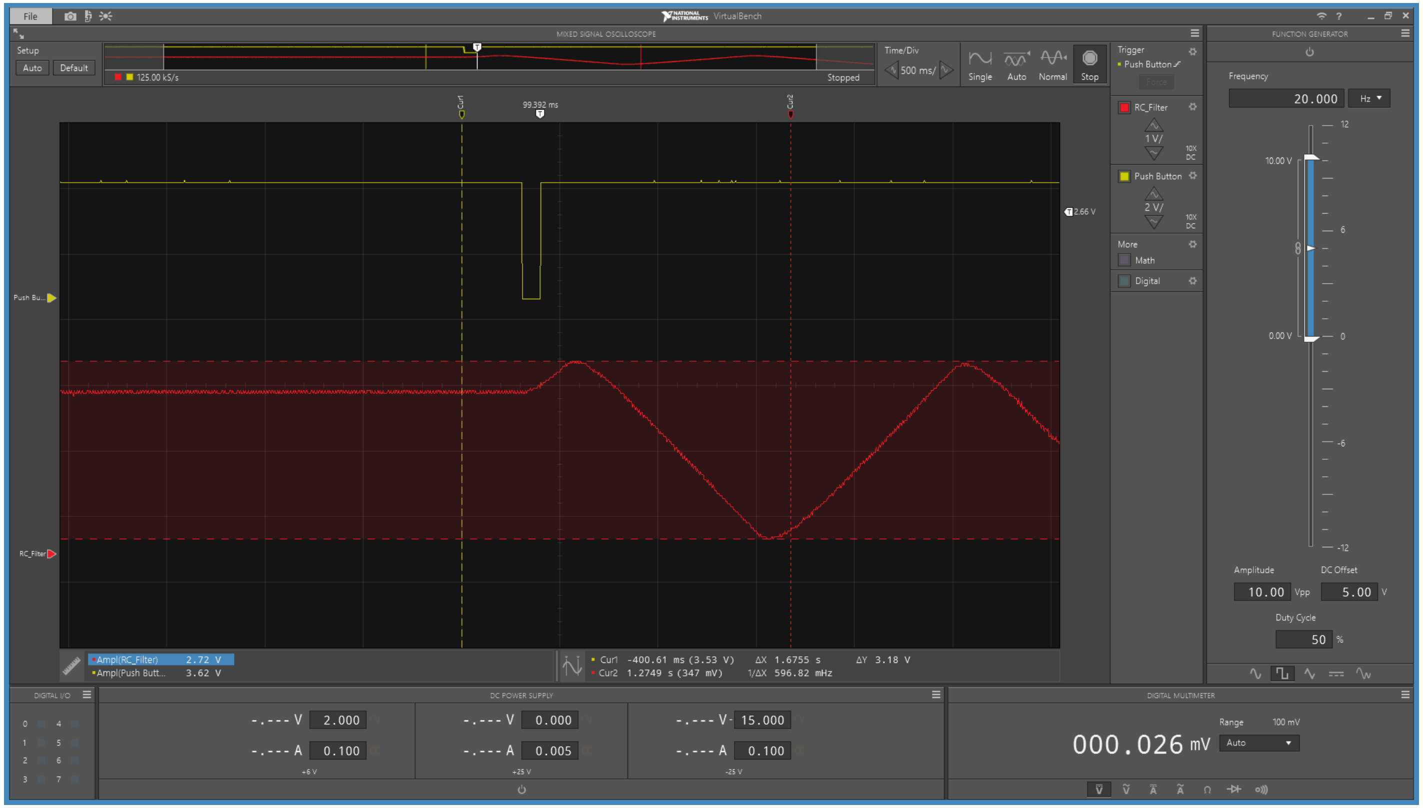
Code (Port 1 Vector)

```
#pragma vector = PORT1_VECTOR

__interrupt void onButtonPush(void) {
    pwmOn ^= 1;
    P1IFG &= ~BIT3;
}
```

- **Toggles the “pause” state of the PWM and reset the IFG for the button manually.**

RC Filter - Part 2



- **This is the same RC filter. All we did was pause the PWM operation, and then unpause it before taking the screenshot.**

Fin