Lab Report 4

This lab provides exercises towards the use of input capture and output compare along with timers on PIC 24 MCU. Given that the PWM is able to determine the brightness of the LED being tested, a double click detector can be implemented and observed by the brightness of the LED.

```
void initServo(void){
   __builtin_write_OSCCONL(OSCCON & 0xbf); // unlock PPS
   RPINR7bits.IC1R = 8;  // Use Pin RP8 = "8", for Input Capture 1 (Table 10-2)//////////
   RPOR3bits.RP6R = 18;  // Use Pin RP6 for Output Compare 1 = "18" (Table 10-3)
   __builtin_write_OSCCONL(OSCCON | 0x40); // lock   PPS

   T3CON=0x0010;
   IFS0bits.T3IF=0;
   TMR3=0;
   PR3=39999;//   20ms/62.5ns/8-1
   T3CONbits.TON = 1;

   OC1CON = 0;    // turn off OC1 for now
   OC1R = 3999;   // servo start position. We won?t touch OC1R again
   OC1RS = 3999;  // We will only change this once PWM is turned on
   OC1CONbits.OCTSEL = 1; // Use Timer 3 for compare source
   OC1CONbits.OCM = 0b110; // Output compare PWM w/o faults
}
```

This function first uses the Peripheral pin select to set RB8 as input capture pin and RB6 as output compare pin. It initializes Timer 3 for the use of output compare. The prescaler for Timer 3 was selected to be 8, so the period can be calculated under the given 16MHz frequency which is 40000. With the use of pull-up resistor, particularly the OC1RS register determines the low PWM since the idle state is high. To set the LED brighter, the method is to set OC1RS higher but within the PR3 range which in this lab was set to 36000 as 90% brightness and 4000 as 10% brightness. For the purpose of observation the LED was designed to be 10% bright in idle state and whenever it detects double click of button it goes up to 90% and stays for 2 seconds.

```
void setServo(int Val){
   OC1RS=Val;
}
```

This function simply takes one parameter and sets it as OC1RS for the purpose of low pulse width that determines the brightness of the LED. The reason is that whenever the timer starts counting from reset, the pulse would go low and stay low until the clock count gets to the parameter value, then the signal would go high until the clock resets. Higher value provides longer pulse width.

```
void initPushButton(void){
   T2CON=0b0000000000110000;//1:256
   PR2=62499;
   TMR2=0;
   _T2IF=0;
   _CN22PUE = 1;
   _T2IE=1;
   T2CONbits.TON=1;

   IC1CON = 0; // Turn off and reset internal state of IC1
   IC1CONbits.ICTMR = 1; // Use Timer 2 for capture source
   IC1CONbits.ICM = 0b010; // Turn on and capture every falling edge
   IC1CON = 1;
   _IC1IF = 0;
   _IC1IE = 1;
}
```

This function initializes Timer 2 along with the use of input capture. It also utilizes Timer 2 for the use of 2 seconds holding inside the main function which will be explained later in this report. Below is the IC1 interrupt function that gets called whenever a falling edge is detected, as well as the use of Timer 2.

```
volatile int  rollover = 0;
void __attribute__((interrupt, auto_psv)) _T2Interrupt(){
   rollover++;
   IFS0bits.T2IF=0;
}

volatile unsigned long start,diff,end, r;
void __attribute__((__interrupt__,__auto_psv__)) _IC1Interrupt(void) {
   _IC1IF = 0;
   end=IC1BUF;
   diff=end+62500*rollover-start-r*62500;
   if(diff>1250){
      if(diff>15625 && diff<53125){
         flag=1;
      }
```

```
        start=end;
        r=rollover;
    }
}
```
Inside this function, the difference between two detected falling edges is always calculated and tested whether actual pressing or noise bouncing. If it is tested not bouncing, it would then be tested if the two falling edges are operated within 0.25-0.75 seconds. An actual double click would raise the ‘flag’ variable which later can notify the main function to set the LED brighter.

The first part of the lab was to use the method of polling which had no use of the input capture. It detects falling edges and stores the current timer value for future calculation that determines whether an actual press or bouncing. Below is the main function,
```
setup();
initServo();
initPushButton();
int prevState,curState=1;
unsigned long long int t1,t2;
 double diff;
    while(1){
        curState=PORTBbits.RB8;
        if(curState==0 && prevState == 1){
            xie_wait_1ms();
            xie_wait_1ms();
            curState=PORTBbits.RB8;
            if(curState==0 && prevState==1){
                t2=TMR2+rollover*62500;
                diff=(t2-t1)*256*0.0000000625;
                if(diff<0.25){
                    setServo(35999);
                    int i;
                    for(i=0;i<2000;i++){
                        xie_wait_1ms();
                    }
                    setServo(3999);
                }else{
                    setServo(3999);
                }
                t1=t2;
            }
        }
        prevState=curState;
        xie_wait_1ms();
        xie_wait_1ms();
    }
```

The second part of the lab was to utilize the input capture function that does the same. Below is the main function,

```
while(1){
        if(flag==1){
            setServo(35999);
            flag=0;
            TMR2=0;
            rollover=0;
            }
        if(rollover==2){
            setServo(3999);
            }
        }
```

Whenever it is in idle state, the LED would have 4000 low pulse width and be kept as initialized at the beginning. Whenever a falling edge is detected the IC1 interrupt will be called and if inside the interrupt function the ‘flag’ variable is raised, meaning a detection of double press is official, the OC1RS will be set 36000 high to make LED brighter for the observation purpose. As OC1RS is set high, Timer 2 and the rollover that counts it will be reset right after it so that when the rollover gets to 2 again, OC1RS will be set back to low. This process takes roughly 2 seconds. This achieves the goal of the demonstration of double click. Since the use of interrupts does not affect the normal CPU running when the interrupt function is not called, the CPU does not need to be set to always looking to serve as it does in polling. This would provide more freedom inside the CPU and allows it to run other operations and won’t be halted. This feature has better technique in use than in the first part of the lab where when the wait_1ms() function is called, everything in the CPU has to halt and wait for it to finish processing.

As in the steps of finishing the lab work, some struggles were met and were hard to solve. These include the debouncing process and the contact signal perfection of the button being used. Since the value of the ‘diff’ variable inside the IC1 interrupt would differ from every button due to the physical design difference of each button, the one being used in this lab was hard to find the perfect value that could demo the work. It is considered that the detections of debouncing and actual pressing are occasionally messed up which cause the demo not well performed. If a button with higher configuration was used, it is believed that the double click would be easier to detect.