Bilkent University
Computer Engineering
CS 224- Computer Organization

Preliminary Design Report

Lab 07

Section 2

Selen Uysal
21702292
İrem Seven
21704269

Lab Time: 27.12.2019

Part b)

The I/O Ports structure includes special function registers that enables us to handle data. TRISx, PORTx, LATx and ODCx are some of these special function registers (SFRs). Ports are controlled by TRISx and PORTx registers. The decision of whether the pin of the port is an output or input is determined by TRISx registers. When a bit of a TRISx register is 1, this means that those bits are for inputs. On the other hand, PORTx registers hold the value that is taken as an input or driven as an output. The data that is written to port I/O pins are held by LATx registers. The data in PORT latch can be read from LAT registers. LAT registers cannot read data from the port I/O pins whereas the port register reads the signal in the port I/O pins. A write to a port register is effectively the same as a write to a LAT register which is written to those of I/O port pins when there is a change in the output. Open-Drain control register (ODC) controls each I/O pins to set for either normal digital, when the ODC bit is 0, or open drain, when ODC bit is 1, output. It is important to know that ODC bit is only for output pins. The letter x at the registres' names indicates which port is chosen as interest.

The program in Part 2-a includes registers TRISD, TRISA, PORTD, PORTA. TRISA and PORTA are used for manipulation of data that is related with buttons "dir" and "en". TRISD and PORTD are used for the leds.

The program in Part 2-b includes registers TRISC, TRISD and related PORTD and PORTC. PORTC is used for the blinking of the leds in a very small amount of time. PORTD determines which value to display on the leds. Similarly, bits of TRISC and TRISD are set to zero to specify required outputs.

Part c)

/*In this part, using 2 pushbutton switches for en and dir inputs, send an 8-bit pattern of 10001000 to the 8 LEDs, rotating its position by 1 each 1.0 seconds. When dir =0 it rotates to the right, dir=1 makes it rotate to the left. When en=0, the pattern is displayed and rotates. When en=1, it is not displayed and its position is "frozen", so that it continues from the last position when en=1 again.*/

```
int en = 0; // Initial state of en (rotation continues)
int initial = 0b01110111; //Initial pattern. Note that 0 means on, while 1 means off.
int dir = 0; //Initial state of rotating right
int mem = 0b11111111; //We use this to store the current output. This is useful when we
//want to froze the state of the output's position
void main(){
     TRISD = 0x0; // All bits of PORTD are output. ~0 means output~
    // Three bits of PORTA are inputs but only one of them is used in this example as
    //a stop button, others are redundant. ~1 means input~
    TRISA = 0b111;
    // From PORTD, outputs will be sent to LEDs.

    // Initial pattern is sent to the LEDs through PORTD.
    PORTD = initial;
    while(1) { //This is infinite loop which waits 1 second after each iteration
        // en button is labeled as 1 on the board.
        if(PORTABits.RA1 == 0){ // If en button clicked
            en = !en; // When the button is pressed, invert the value of en
        }

        // dir button is labeled as 5 on the board.
        if(PORTABits.RA5 == 0){ // If dir button clicked
            dir = !dir; // When the button is pressed, invert value of dir
        }

        //When dir button is pressed and en is not pressed, rotate left
        if(dir && !en) {
            //Assign the current rotated value to mem before assigning it to PORTD
            PORTD = mem = (PORTD << 1) | (PORTD >> 7); //Shift the most
            //significant bit to make it the least significant bit. Then, add this value to
            //the rest of the left shifted value.
        }

        //When dir button is not pressed and en is not pressed, rotate right
        else if(!dir && !en){
            //Assign the current rotated value to mem before assigning it to PORTD
```

```
                PORTD = mem = (PORTD >> 1) | (PORTD << 7); //Shift the least
                //significant bit to make it the most significant bit. Then, add this value to
                //the rest of the right shifted value.
            }
            else {
                //Do not shift anything, that is, froze.
                PORTD = mem;
            }
            delay_ms(1000); // Wait 1 second.
        }
} // Program ends here
```

Part d)

/*In this part, function f(x)=x^3 is implemented by using the seven-segment display (SSD) on the Beti board. Only 21 numbers' cubes are displayed since we can use 4 digits. The sequence will continue from the beginning when it reaches the last element.*/

```
int sevenSegment[]={0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D, 0X7D,
                          0X07, 0X7F, 0X6F}; //These are the hexadecimal
                    //correspondence of the numbers from 0 to 9.
int elapsedTime = 0; //This used for checking the right time to display the results
int x = 0; // x is the number that we will obtain its cube
// The four integers below are the digits of the result (Since there can be 4 digits at most, we
//need to have 4 of these)
int num1;
int num2;
int num3;
int num4;
int result = 1; //This is the calculated result (cube of x)
void main(){
    TRISC=0X00;
    TRISD=0X00;
    //The lines below are used to set all the digits to zero when the seven segment display
    //first opens (we try to avoid getting weird values at the beginning)
```

```
num1 = 0;
PORTD= sevenSegment[num1];
num2 = 0;
PORTD= sevenSegment[num2];
num3 = 0;
PORTD= sevenSegment[num3];
num4 = 0;
PORTD= sevenSegment[num4];
delay_ms(10);
while(1){  // This is an infinite loop
            // This is for displaying results by our desired speed
             while (elapsedTime < 150){
                   elapsedTime = elapsedTime + 1;
                   //Fourth digit is displayed at the seven segment display
                   PORTD = sevenSegment[num1];
                   PORTCBits.RC4=1;  //Opens the led
                   delay_ms(2);
                   // Time spent between opening and closing the led is so small that we
                do not see its blink
                   PORTCBits.RC4=0;//Closes the led

                   //Third digit is displayed at the seven segment display
                   PORTD = sevenSegment[num2];
                   PORTCBits.RC3=1;
                   delay_ms(2);
                   PORTCBits.RC3=0;

                  //Second digit is displayed at the seven segment display
                   PORTD = sevenSegment[num3];
                   PORTCBits.RC2=1;
                   delay_ms(2);
                   PORTCBits.RC2=0;
                //First digit is displayed at the seven segment display
                PORTD = sevenSegment[num4];
                PORTCBits.RC1=1;
                delay_ms(2);
                PORTCBits.RC1=0;
            }
            //The first 21 cubes of the x's will be calculated, this if statement checks this.
            if( x >= 21 ){
                x = 1;
            }
            else {
                x = x + 1;
            }
            result = x * x * x;  //The cube of the x that will be displayed
            //We calculate each digits from the result by division and mod operations
            // Mod operation gets the least significant digit of the number and division by
        //10 provides us to move to the digit before
            num1 = result % 10;

            result = result / 10;
            num2 = result % 10;

            result = result / 10;
            num3 = result % 10;

            result = result / 10;
            num4 = result % 10;

            elapsedTime = 0;  //Reset the elapsedTime for the new display result
    }
}
```