Calculation The Interval Between Two External Signals

Team Members:

1. Islam Atef Mohamed
2. Ahmed Galal Mahmoud
3. Ahmed Mohamed Abo Bakr
4. Ahmed Ezzat
5. Ahmed Heshmat
6. Esraa Rizk
7. Aya Gamal
8. Asmaa Ahmed Mohamed

Idea:

Measuring the time in terms of milliseconds between two predefined external input signals that come from a bush button.

The External input Signals Mechanism:

By using the External Interrupt (INT0) that exits on pin(2), port(D), Defining the I/O operation mode of this pin as {INPUT PULL-UP}: CLR_BIT(DDRD, DDD2);
SET_BIT(PORTD, PORTD2);

, defining the sense control of the (INT0) as The falling edge generates an interrupt request: SET_BIT(MCUCR, ISC01);

, and Finally enable the Interrupt itself: SET_BIT(GICR, INT0);

The Timer/counter0 configurations:

This timer is a (8-bit) timer, so its maximum capacity is (255) as it start from (0).

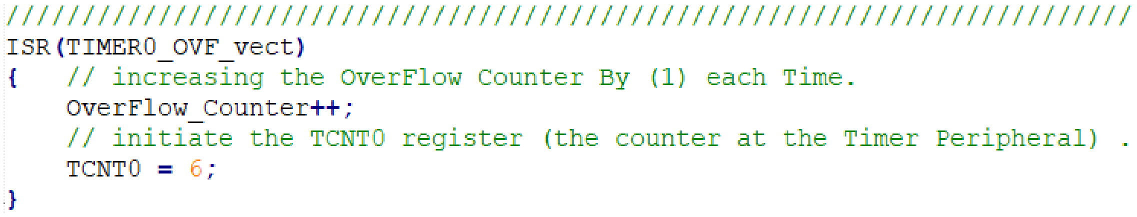
As we operate the Micro-controller at frequency equals (16 MHz) comes from an external Crystal Oscillator, defining that through the (FUSE BYTES) configurations.

By defining the TIMER to operate at (Normal Mode), which is the default mode, and with a prescaler of ($Clk_{i/o}$/64), so the needed value that the counter will start counting from is (6) according to the equation:

$$\text{initial\_value} = (255 - ((16*10^{6})*(1*10^{-3}) / 64)+1) = 6$$

with this initial value of the timer, the period if each overflow will be (1 ms).

Finally, Enable the overflow interrupt and write its ISR as:

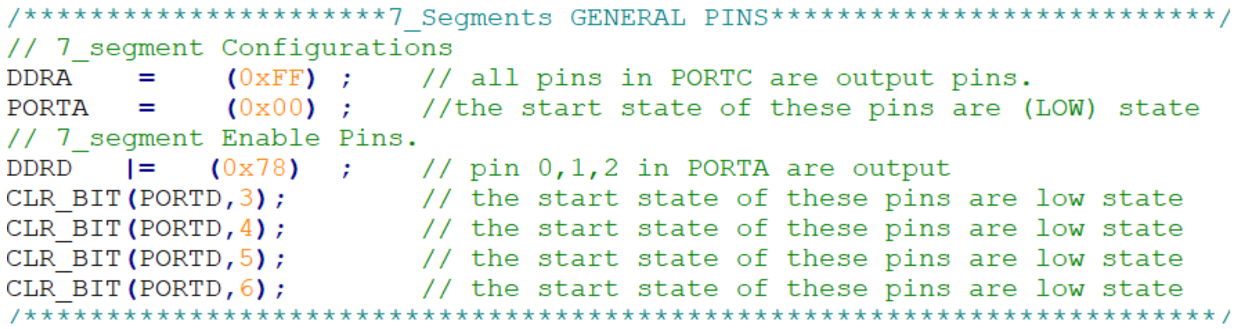


```
////////////////////////////////////////////////////////////////////////////
ISR(TIMER0_OVF_vect)
{   // increasing the OverFlow Counter By (1) each Time.
    OverFlow_Counter++;
    // initiate the TCNT0 register (the counter at the Timer Peripheral) .
    TCNT0 = 6;
}
```

7-Segment mechanism:

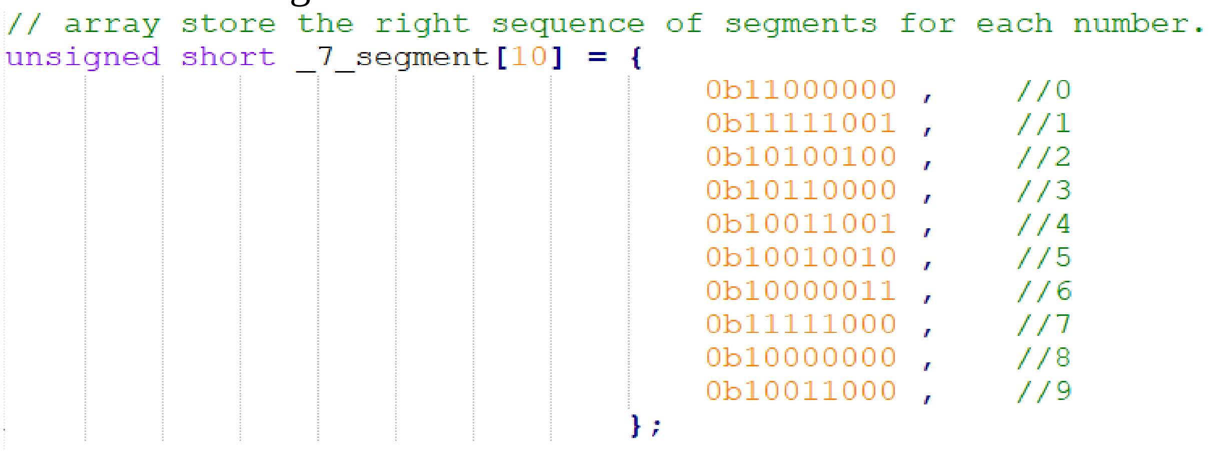
7-segment is a 7-LEDs share the same (Vcc) in the common-anode case, thin (Vcc) terminal is known as the "Enable-Pin", and in this project, we have a 7-segment matrix consists of four 7-segments.

First, defining all the terminals as "OUTPUT", and define their initial states:



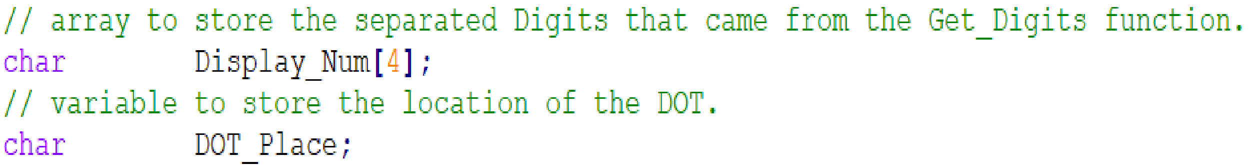
```
/**********************7_Segments GENERAL PINS***************************/
// 7_segment Configurations
DDRA    =    (0xFF) ;    // all pins in PORTC are output pins.
PORTA   =    (0x00) ;    //the start state of these pins are (LOW) state
// 7_segment Enable Pins.
DDRD    |=   (0x78)  ;   // pin 0,1,2 in PORTA are output
CLR_BIT(PORTD,3);        // the start state of these pins are low state
CLR_BIT(PORTD,4);        // the start state of these pins are low state
CLR_BIT(PORTD,5);        // the start state of these pins are low state
CLR_BIT(PORTD,6);        // the start state of these pins are low state
/***********************************************************************/
```

Second, creating an array that contains the state of each segment for each Digit:



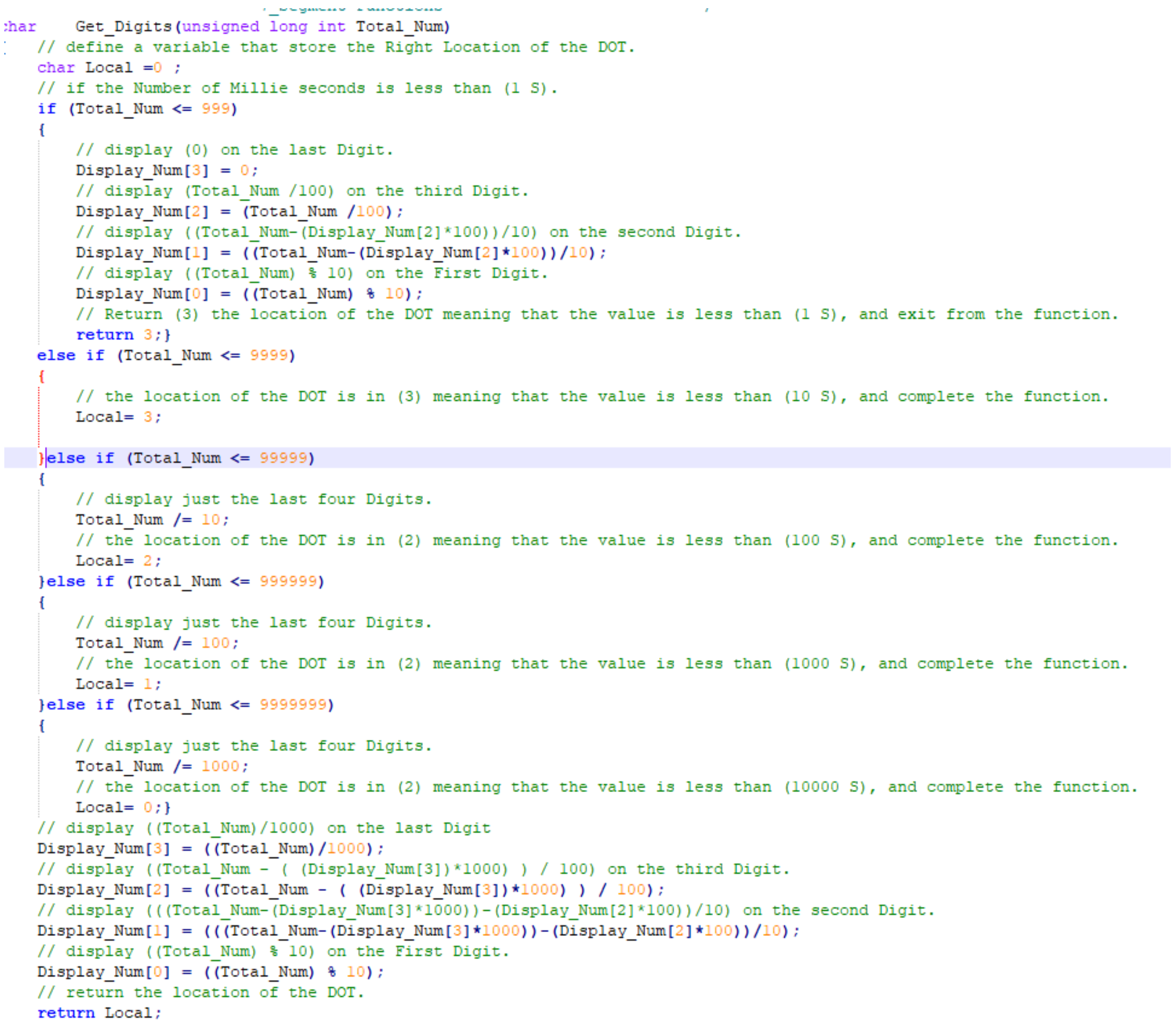
```
// array store the right sequence of segments for each number.
unsigned short _7_segment[10] = {
                                    0b11000000 ,    //0
                                    0b11111001 ,    //1
                                    0b10100100 ,    //2
                                    0b10110000 ,    //3
                                    0b10011001 ,    //4
                                    0b10010010 ,    //5
                                    0b10000011 ,    //6
                                    0b11111000 ,    //7
                                    0b10000000 ,    //8
                                    0b10011000 ,    //9
                                };
```

Third, creating an array of four elements, each element store the number that will be displayed on its 7-segment digit, and a variable that will store the right location of the DOT:



```
// array to store the separated Digits that came from the Get_Digits function.
char        Display_Num[4];
// variable to store the location of the DOT.
char        DOT_Place;
```

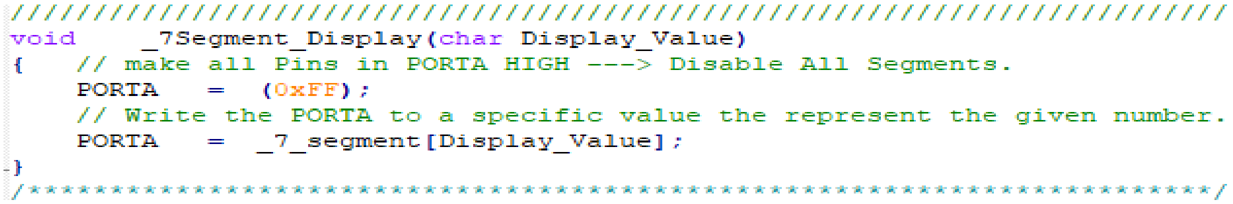
Fourth, creating a function that will take any number to separate its digits and store them into the “Display_Num” array, and detecting the right location of the DOT according to the number size:



```
:har    Get_Digits(unsigned long int Total_Num)
    // define a variable that store the Right Location of the DOT.
    char Local =0 ;
    // if the Number of Millie seconds is less than (1 S).
    if (Total_Num <= 999)
    {
        // display (0) on the last Digit.
        Display_Num[3] = 0;
        // display (Total_Num /100) on the third Digit.
        Display_Num[2] = (Total_Num /100);
        // display ((Total_Num-(Display_Num[2]*100))/10) on the second Digit.
        Display_Num[1] = ((Total_Num-(Display_Num[2]*100))/10);
        // display ((Total_Num) % 10) on the First Digit.
        Display_Num[0] = ((Total_Num) % 10);
        // Return (3) the location of the DOT meaning that the value is less than (1 S), and exit from the function.
        return 3;}
    else if (Total_Num <= 9999)
    {
        // the location of the DOT is in (3) meaning that the value is less than (10 S), and complete the function.
        Local= 3;

    }else if (Total_Num <= 99999)
    {
        // display just the last four Digits.
        Total_Num /= 10;
        // the location of the DOT is in (2) meaning that the value is less than (100 S), and complete the function.
        Local= 2;
    }else if (Total_Num <= 999999)
    {
        // display just the last four Digits.
        Total_Num /= 100;
        // the location of the DOT is in (2) meaning that the value is less than (1000 S), and complete the function.
        Local= 1;
    }else if (Total_Num <= 9999999)
    {
        // display just the last four Digits.
        Total_Num /= 1000;
        // the location of the DOT is in (2) meaning that the value is less than (10000 S), and complete the function.
        Local= 0;}
    // display ((Total_Num)/1000) on the last Digit
    Display_Num[3] = ((Total_Num)/1000);
    // display ((Total_Num - ( (Display_Num[3])*1000) ) / 100) on the third Digit.
    Display_Num[2] = ((Total_Num - ( (Display_Num[3])*1000) ) / 100);
    // display (((Total_Num-(Display_Num[3]*1000))-(Display_Num[2]*100))/10) on the second Digit.
    Display_Num[1] = (((Total_Num-(Display_Num[3]*1000))-(Display_Num[2]*100))/10);
    // display ((Total_Num) % 10) on the First Digit.
    Display_Num[0] = ((Total_Num) % 10);
    // return the location of the DOT.
    return Local;
```

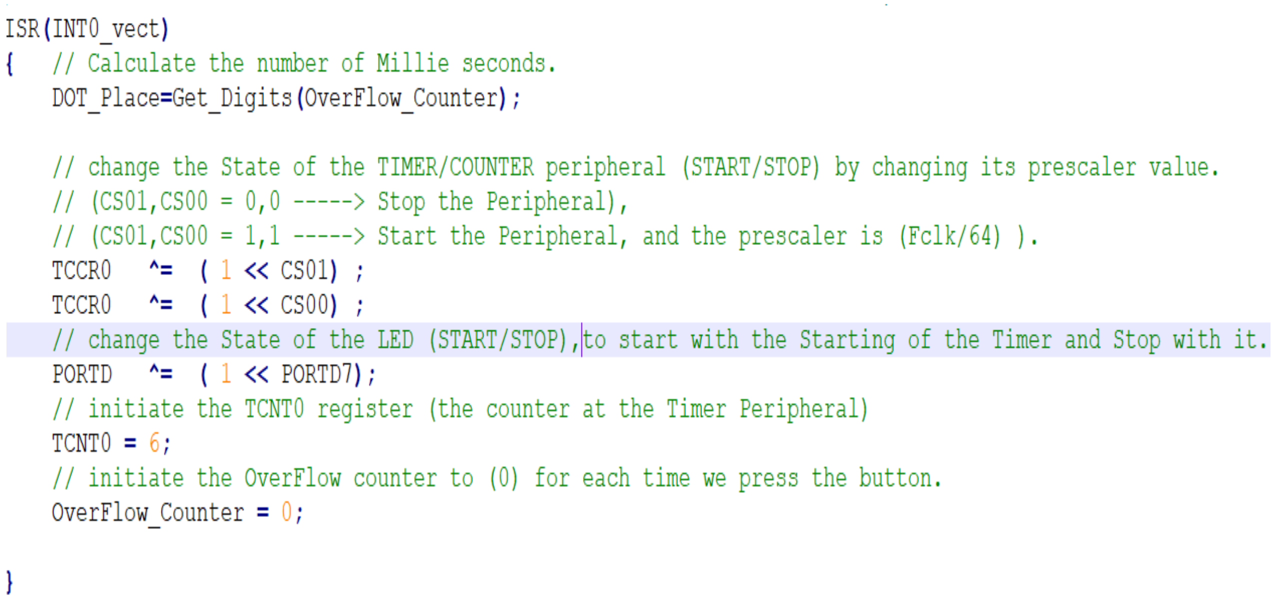
Finally, creating a function responsible of displaying the Digits on the 7-segment:



```
////////////////////////////////////////////////////////////////////////////
void    _7Segment_Display(char Display_Value)
{   // make all Pins in PORTA HIGH ---> Disable All Segments.
    PORTA    =   (0xFF);
    // Write the PORTA to a specific value the represent the given number.
    PORTA    =   _7_segment[Display_Value];
}
/**************************************************************************/
```

The Operation Mechanism:

Most of operations are done inside the (ISR) of the external interrupt, and this operations are calculating the number of milliseconds, stop or start the timer according to its previous state, open or close an indicating LED according its previous state, this LED shows the state of the timer, initiate the timer counter to the value that calculated to make the Overflow period equals (1 mS), and finally change the value of the variable that store the number of overflows to (0).

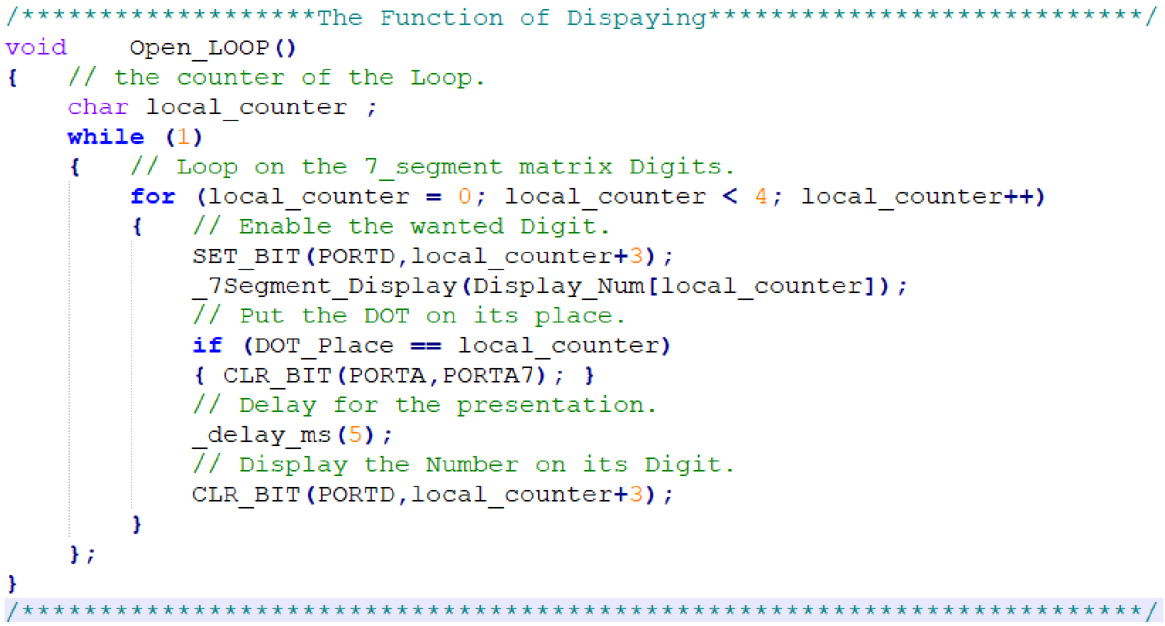


```
ISR(INT0_vect)
{   // Calculate the number of Millie seconds.
    DOT_Place=Get_Digits(OverFlow_Counter);

    // change the State of the TIMER/COUNTER peripheral (START/STOP) by changing its prescaler value.
    // (CS01,CS00 = 0,0 -----> Stop the Peripheral),
    // (CS01,CS00 = 1,1 -----> Start the Peripheral, and the prescaler is (Fclk/64) ).
    TCCR0   ^=  ( 1 << CS01) ;
    TCCR0   ^=  ( 1 << CS00) ;
    // change the State of the LED (START/STOP),to start with the Starting of the Timer and Stop with it.
    PORTD   ^=  ( 1 << PORTD7);
    // initiate the TCNT0 register (the counter at the Timer Peripheral)
    TCNT0 = 6;
    // initiate the OverFlow counter to (0) for each time we press the button.
    OverFlow_Counter = 0;

}
```

The last part of the operations is what happened inside the open loop, and in this part we display each digit for a specific amount of time to apply the persistence of the vision (POV) concept.



```
/*******************The Function of Dispaying****************************/
void    Open_LOOP()
{   // the counter of the Loop.
    char local_counter ;
    while (1)
    {   // Loop on the 7_segment matrix Digits.
        for (local_counter = 0; local_counter < 4; local_counter++)
        {   // Enable the wanted Digit.
            SET_BIT(PORTD,local_counter+3);
            _7Segment_Display(Display_Num[local_counter]);
            // Put the DOT on its place.
            if (DOT_Place == local_counter)
            { CLR_BIT(PORTA,PORTA7); }
            // Delay for the presentation.
            _delay_ms(5);
            // Display the Number on its Digit.
            CLR_BIT(PORTD,local_counter+3);
        }
    };
}
/************************************************************************/
```