Electrical and Electronic Engineering 2019, 9(2): 41-44
DOI: 10.5923/j.eee.20190902.03

PIC32MZ Milliseconds and Microseconds Delay Routines

Abhishek N. Patel

Research and Development Department, Horiba Instruments Inc., 20 Knightsbridge Road, Piscataway, New Jersey, United States

Abstract This paper demonstrates how to make millisecond and microsecond delay routines for PIC32MZ/PIC32 MCUs by utilizing internal clock speed of MCU. Delay routine used to suspend execution of a program for a particular time. Unfortunately, the microchip PICXC32 compiler does not give us a delay function/API to call from firmware application layer. This implementation requires PIC32MZ/PIC32 MCU based hardware and microchip MPLAB-X IDE, tool chain and Harmony framework.

Keywords C, PIC32MZ, PIC32, Firmware, Embedded System, Delay, LED Blinker

1. Introduction

An embedded system interacting with physical world, which sometime does not fast as fast a computer instruction. In addition, it can be design based on state driven or interrupt driven or based on external source state, where it needs to hold and wait or stay in same state for some time before executing next instruction. There are multiple way to accomplish this, one of them via delay. The problem I encountered was that I want to read the temperature of a chamber and if current temperature is same as requested temperature then indicate it by setting an LED. Now I cannot take it continuously, as the temperature of a chamber will not change every millisecond/microsecond. Therefore, I insert a delay in acquiring the temperature of the chamber. Delay used to take periodic inputs from the real world so that the system is practical and it save up memory in the PIC32 microcontroller/small target based system.

2. How to Make Routines for Millisecond and Microsecond Delays

The LED blinker program that is a good starting point with any 32-bits MCU like PIC32MZ as it utilizing clock speed. A sign of success, those things are working and clock speeds are right. Unfortunately, the microchip PICXC32 compiler does not give us a delay function/API to call. There is a built-in timer in every PIC32 called the Core Timer. It does not need to be set up and it is always running. It runs at half the system frequency, so if we are running at 200MHz then it will update at 100MHz.

First, let us add define for how fast we have set the fuses to. This value will be useful for many calculations we will need to do later.

```
#define SYS_FREQ      200000000 // Running at 200MHz
#define us_SCALE    (SYS_FREQ /2000000) // Microsecond scale
#define ms_SCALE    (SYS_FREQ /2000) // Millisecond scale
```

Next is, how do we use the Core Timer?

First, we need to calculate how long we need to wait. For an Example here, the core timer updates at 100MHz. Let us say we target to wait 500 microseconds. First, we needs to calculate number of clock tick requires for that.

When running 200MHz, Core Timer frequency is 100MHz

Core Timer increments every 2 SYS_CLK, Core Timer period = 10ns

```
1 ms = N x CoreTimer_period.
Where, to count 1ms, N = 100000 counts of CoreTimer
1 ms = 10 ns * 100000 = 10e6 ns = 1 ms
ms_SCALE = (GetSystemClock()/2000) @ 200 MHz = 200e6/2e3
= 100e3 = 100000
```

Let us put it to gather and make a microsecond delay function:

```
void DelayMs(unsigned long int msDelay )
{
        // Get the current core timer value and mark as a start count
for Ms.
       register unsigned int startCntms = ReadCoreTimer();
        // Convert milliseconds Ms into how many clock ticks it will
take
```

* Corresponding author:
a_n_patel@yahoo.com (Abhishek N. Patel)
Published online at http://journal.sapub.org/eee

```
        register unsigned int waitCntms = msDelay * ms_SCALE;

        // Wait until Core Timer count reaches the number we calculated earlier
        while( ReadCoreTimer() - startCntms < waitCntms );
}
```

First, we calculate how many clock ticks we need to wait. Then we read current Core Timer counter and wait until it match to the number of wait count (i.e. waitCntms) value.

```
static uint32_t ReadCoreTimer()
{
    volatile uint32_t timer;
    // get the count reg
    asm volatile("mfc0    %0, $9" : "=r"(timer));
    return(timer);

}
```

Similar to that, create a microsecond delay now.

```
void DelayUs(unsigned long int usDelay )
{
        // Get the current core timer value and mark as a start count for Usec.
        register unsigned int startCntms = ReadCoreTimer();
        // Convert microseconds Usec into how many clock ticks it will take
        register unsigned int waitCntms = usDelay * us_SCALE;
        // Wait until Core Timer count reaches the number we calculated earlier
        while( ReadCoreTimer() - startCntms < waitCntms );
}
```

NOTE: This implementation could be off by few clock cycles. As the Core Timer is a 32-bit number, which means it can only count to 4 billion before it'll wrap back around to 0. So don't use these routines for hundreds of seconds.

Now let us make that LED dance!

```
while (1)
{
    LATEINV = 1 << 0; // Toggle LAT E bit 0
    DelayMs (1000); // Delay 1 second
}
```

LATEINV is an easy way to toggle bits on Port E and requires no work on our part.

3. Millisecond Delay Routine Real World Usages for Temperature Monitoring

This section describe application of millisecond delay routine for temperature monitoring. We have define target temperature at the start of program and turn on LED once current temperature equal to targeted temperature. For this comparison, we needs to read back a current temperature after every 1 second, as it is not going to change micro/milliseconds. We are waiting for 1 second using millisecond routine and then read back the temperature via LM35 sensor. LM35 is a temperature sensor, which can measure temperature in the rage of -55℃ to 150℃. A three terminal device provides analog voltage proportional to the temperature. Higher temperature reports high output voltage and lower temperature reports low output voltage. The LED remain off till its reaches to the targeted temperature and once current temperature equal or greater than targeted temperature then it will turn on the LED.

3.1. Code for PIC32MZ and LM35 Temperature Sensor with LED

This sample code design to read a current temperature via LM35 and compare it to targeted temperature. If it equal or greater than targeted temperature then turn on the YELLOW-LED to indicates it.

```
void main()
{
        // Set up all ports to be digital and not analog except PORT-B as its reading
        // temperature via LM35.
    ANSELA = 0;
    ANSELC = 0;
    ANSELD = 0;
    ANSELE = 0;
    ANSELF = 0;
    ANSELG = 0;
    ANSELH = 0;
    ANSELJ = 0;


        float fTargetedTempCelsius = 31.0; // Set Target temperature to 31 degree Celsius.
        float fCurrentTempCelsius = 0.0;    // Set current temperature to 0 degree Celsius.
                                                                    // As its going to be updated based on the LM35.
```

```
        // Set max performance mode
        set_performance_mode();

        ADC_Init(); // Initialize ADC convertor.

    TRISECLR = 1 << 0; // Clear bit 0 of TRISE. which makes PORT E bit 0 an output for LED

        // Main loop
        while (1)
        {
                // Read the current Temperature via LM35
                // and convert it to temperature value.
                fCurrentTempCelsius = (ADC_Read(0)*4.88);
                fCurrentTempCelsius = (fCurrentTempCelsius/10.00);
                if (fCurrentTempCelsius >= fTargetedTempCelsius)
                {
                        LATEINV = 1 << 0; // Toggle LAT E bit 0 to turn on   YELLOW-LED
                }
                DelayMs(1000); // Wait for 1 second by using Millisecond Delay routine before reading current temperature.
        }
}
```

4. Millisecond Delay Routine Real World Usages for Writing to the External EEPROM via i2C Bus

This section describe application of millisecond delay routine to implement EEPROM write cycle time as per datasheet of AT24CM02 (Microchip 2MB EEPROM memory). When we try to write a data to EEPROM, it initially received into a buffer and getting transfer from received buffer to physical EEPROM memory during a write cycle time. Therefore, for write operation PIC32MZ microcontroller needs to wait in between sending multiple data packet to EEPROM. The AT24CM02 EEPROM requires 10Miliseconds between two-write operation as a write cycle time. This implemented successfully by calling a millisecond routine between two-write operations (for more detail refer AT24CM02 datasheet chapter-5.5).

4.1. Code for PIC32MZ and AT24CM02 EEPROM

This sample code implemented successfully on PIC32MZ2048EFM with AT24CM02 EEPROM, which restores first four pages of EEPROM to default "0xFF" value and wait for 10 Milliseconds by calling Millisecond delay routine(i.e. DelayMs(10)) between each page write (As per AT24CM02 datasheet recommended write cycle time).

```
void LoadDefaultEEPROM(void)
{
    WORD wEEPROMPageAdd = 0x0;   // Set the base address.
    int iTotalPageCnt = 4; // Total 4 pages require to store 1024 bytes where 256bytes/page
    int iCurPageCnt = 0;
    int iEEPROMIndex = 0;
    static BYTE bTempEEPROMBuf[256];
    // To Load default, write 0xFF to EEPROM temp. Buffer.
    memset(&bTempEEPROMBuf[iEEPROMIndex], 0xFF, sizeof(bTempEEPROMBuf));
   // Send the default data to EEPROM for write operation.
    for (iCurPageCnt = 0; iCurPageCnt < iTotalPageCnt; iCurPageCnt++)
    {// Page level loop
        // Copy the 256 byte block to temp. send buffer
        memcpy(&bTempEEPROMBuf[0], &bLoadConfgDefault[AT24CM02_EEPROM_PAGE_SIZE * iCurPageCnt], sizeof(bTempEEPROMBuf));

        // Get the EEPROM Address of current Page.
        if (iCurPageCnt == 0)
            wEEPROMPageAdd =   EEPROM_CONFIG_ADDR;
        else
            wEEPROMPageAdd =   EEPROM_CONFIG_ADDR + (AT24CM02_EEPROM_PAGE_SIZE * iCurPageCnt);

        // Send the Page write request.
        intf_Cfg_EEPROM_Write ( (wEEPROMPageAdd) , 256, &bTempEEPROMBuf[iEEPROMIndex] );


/*********************************************************************************
         * >>>>> DEV NOTES:
         * Added this 10MSec delay for each page write which allow AT24CM02
         * EEPROM to transfer the Rx Buffer to Phy. Memory. This is minimum
         * time it took to complete the write of one page of 256 bytes.

*********************************************************************************/

        DelayMs(10);
    }
    return;
}
```

5. Conclusions

These costume Delay routines can be useful for any PIC32 embedded system based project. There could be a numerous condition in real world applications where we need this microseconds/milliseconds delay routines to hold application into current state and wait for other events to complete an execution. These delay routines implemented successfully using PIC32MZ MCU hardware and MPLAB software stack that includes MPLAB X IDE, Tool chain and Harmony framework. The above example of temperature monitoring and EEPROM write cycle time demonstrated usages of millisecond delay routine.

REFERENCES

[1] "Embedded Computing and Mechatronics with the PIC32 Microcontroller 1st Edition" by Kevin Lynch (Author), Nicholas Marchuk (Author), Matthew Elwin (Author).

[2] Data Sheet of "PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Family " by Microchip.

[3] Data sheet of "LM35 Precision Centigrade Temperature Sensors" by Texas Instruments.

[4] Data sheet of "AT24CM02 2-Mbit EEPROM" by ATMEL.