Timers and Multi-Rate Process with the Cerebot MX7cK™

Revision: 28SEP2017 (JFF)

Richard W. Wall, University of Idaho, rwall@uidaho.edu

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Project 4: PIC32 Timers

Table of Contents

Project 4: PIC32 Timers 1
Table of Contents 1
Purpose 1
Minimum Knowledge and Programming Skills 2
Equipment List 2
Software Resources 2
References 2
Introduction to Timers 3
Types of Timers: 3
Peripheral Library Support for Timers: 3
Example Program: 4
Determining the Timer Clock Rate 5
Program Concepts 6
Project Tasks 7
Project Testing 8

Purpose

The purpose of this project is to understand the operation of PIC32 timers that can be used to implement a synchronized multi-rate periodic control system by polling the timer interrupt flag. Project 4 concepts leverage off the concepts explored in Projects 3 where you designed systems that controlled the stepper motor direction of shaft rotation and the step mode at a constant speed.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB IDE
3. IO pin control (See project 1)
4. Time control of computer program execution (See project 2)
5. Understanding of Finite State Machines (See Project 3)

Equipment List

1. Cerebot 32MX7cK processor board with USB cable
2. Microchip MPLAB ® X IDE
3. PmodSTEP
4. Stepper Motor (5V – 12V, 25Ω, unipolar or bi-polar)
5. Logic analyzer, or oscilloscope (Suggestion - Digilent Analog Discovery)

Software Resources

1. XC32 C/C++ Compiler Users Guide
2. PIC32 Peripheral Libraries for MBLAB C32 Compiler
3. Cerebot MX7cK Board Reference Manual
4. PIC32 Family Reference Manual Section 14: Timers
5. MPLAB ® X Integrated Development Environment (IDE)\
6. C Programming Reference

References

1. Timers Tutorial (Part 1) http://ww1.microchip.com/downloads/en/DeviceDoc/51682A.pdf
2. Timers Tutorial (Part 2) http://ww1.microchip.com/downloads/en/DeviceDoc/51702a.pdf
3. PIC32 Family Reference Manual, Section 14: http://ww1.microchip.com/downloads/en/DeviceDoc/61105E.pdf

Introduction to Timers

Types of Timers:

Section 14 of the PIC32 Family Reference Manual identifies two types of timers. The Type A timer is a 16 bit counter that can be clocked from the peripheral bus clock or a secondary oscillator that is usually 32 kHz, used for the real time clock. This timer can be used to generate an interrupt that will wake the processor up after it has been put into a sleep mode. The clock source to the timer has four software programmable prescale dividers of 1, 8, 64, or 256. The PIC32 Timer 1 is a Type A Timer.

Type B counters are similar to Type A counters in that they use 16 bit counters. However, two Type B counters can be programmed to operate together to form a 32 bit counter. The Type B counters have eight possible prescale values of 1, 2, 4, 8, 16, 32, 64, and 256.

All timers have a period register that is used to set the maximum count of the timer. The period registers are declared as PR1 for Timer 1, PR2 for Timer 2, and so on. Whenever the time count reaches the value stored in the period register, a timer interrupt flag is set and the timer register is reset to zero. Figure 1 shows the relationship of the timing in the interrupt flag to the CPU crystal frequency. Timers 2 through 5 are similar to Timer 1 with the differences being the timer prescale options and the period register.

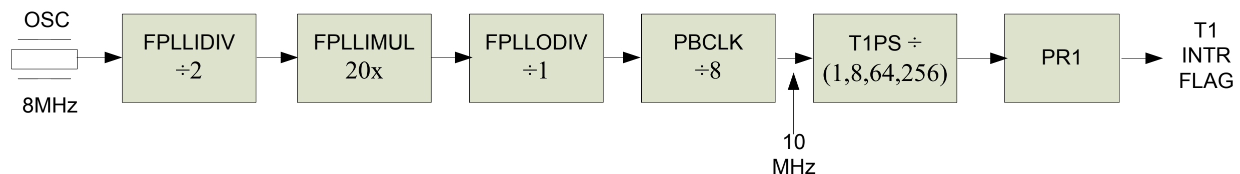

Figure 1. Divider chain from the CPU crystal (oscillator) to the Timer 1 Interrupt flag

Once the timer interrupt flag is set, it will remain set until a software instruction clears the flag. (Since the timer interrupt has not been enabled, although the timer interrupt flag is set, no interrupt will be generated. Thus, all peripherals that potentially can be used to generate an interrupt can also be used in a polled mode.) Since the timer counts from zero to the PR register value, the value set into the PR register must be one less than the desired period.

Peripheral Library Support for Timers:

Section 11 of the 32 Bit Peripheral Library Guide contains the functions and macros pertaining to the core timer and timers 1 through 5. Timers are initialized by using the *OpenTimer* macro that requires two arguments: the configuration argument and the initial PR value. For example, initializing Timer 1 to use the peripheral bus clock, a prescale value of 8 and a period of 1000 requires the following assignment.

```
OpenTimer1((T1_ON | T1_SOURCE_INT | T1_PS_1_8), 999);
```

The arguments to this macro are defined in timers.h of the peripheral library.

The timer interrupt flag is polled using the macro, *mTxGetIntFlag(),* and can be cleared using the macro, *mTxClearIntFlag().* Both macros are defined by the peripheral library.

The PR register for the timers can be changed in software at any time by simply using the assignment: *PRx = ###;* or *WritePeriodx (###);* where x is the timer number and ### is the value to be set into the PR register. The following code listing illustrates using Timer 1 to implement a delay by polling the interrupt flag.

Example Program:

```
/*********************************************************************
 * The purpose of this example code is to demonstrate the use of
 * peripheral library macros and functions supporting the PIC32MX general
 * purpose Type A Timer 1.
 *
 * Platform:     Cerebot MX7cK
 *
 * Features demonstrated:
 *      - Timer configuration
 *      - Timer Interrupt flag polling
 *      - Timer Interrupt flag resetting
 *
 *
 * Description:
 *         - This example polls the Timer 1 interrupt flag and toggles LEDA
 *
 * Oscillator Configuration Bit Settings: config_bits.h
 * Cerebot 32MXcK hardware setup: Cerebot_MX7cK.h and Cerebot_MX7cK.c
 * Notes:
 *        - Timer1 clock = FOSC/PB_DIV/PRESCALE = 80E6/8/1 = 10MHz
 *        - To generate a 1 ms delay, PR1 is loaded with (10000-1) = 9999
 *
 ********************************************************************/
#include <plib.h>
#include "config_bits.h"      // PIC32 Configuration settings
#include "Cerebot_mx7cK.h"    // Cerebot hardware

#define T1_PRESCALE          1
#define TOGGLES_PER_SEC      1000
#define T1_TICK              (FPB/T1_PRESCALE/TOGGLES_PER_SEC)

int main(void)
{
    Cerebot_mx7cK_setup();     // Common hardware setup for Cerebot MX7cK
    PORTSetPinsDigitalOut(IOPORT_B, SM_LEDS); // Set JA pins as outputs
    LATBCLR = SM_LEDS;                        // and set to zero

    // Initialize Timer 1 for 1 ms
    OpenTimer1(T1_ON | T1_PS_1_1, (T1_TICK-1));

    while(1)
```

```
    {
        Timer1_delay(1000); // toggle every second
        mPORTBToggleBits(LEDA);
    }

    return (EXIT_SUCCESS);
}

/* START FUNCTION DESCRIPTION ********************************************
Timer1_delay
SYNTAX:         void Timer1_delay(int delay);
KEYWORDS:       Millisecond, hardware delay, Timer A, multitrate timer
DESCRIPTION:    Delays the number of milliseconds specified by the passed
                argument, delay..
Parameter 1:    int - number of ms to delay
RETURN:         None
END DESCRIPTION **********************************************************/
void Timer1_delay(int delay)
{
    while(delay--)
    {
        while(!mT1GetIntFlag());    // Wait for interrupt flag to be set
        mT1ClearIntFlag();          // Clear the interrupt flag
    }
}
```

Determining the Timer Clock Rate

There are two arguments required for the "OpenTimer1" function: the configuration bits and the value to be programmed into the PR1 register. Among other options, the configuration bits turn Timer 1 on, set Timer 1 to use the peripheral clock for the input to the prescale divider, and set the prescale value.

As Figure 1 illustrates, the Timer 1 clock frequency is determined from the crystal frequency on the Cerebot MX7cK board (XTAL), the PLL input divider (FPLLIDIV), the PLL multiplier (FPLLMUL), the PLL output divider (FPLLODIV), the peripheral bus divider (FPBDIV) and the Timer prescale (TCKPS). This frequency is computed using Eq. 1. The values of XTAL, FPLLMUL, FPLLDIV, FPLLODIV, and FPBDIV are set in the common "*config_bits.h*" header file.

$$T1CLK = XTAL * FPLLMUL / (FPLLIDIV * FPLLODIV * FPBDIV * TCKPS) \quad \text{Eq. 1}$$

Timer 1 (TMR1) increments each Timer 1 clock period, as determined by Eq. 1 for T1CLK. Each Timer 1 count is compared to the value of the PR1 register. When TMR1 equals PR1, the timer has reached its terminal count, initiating two actions: (1) the timer is reset to zero, and, (2) the Timer 1 Interrupt Flag (T1IF) bit is set in the Interrupt Flag Status register (IFS0). (See the PIC32MX5XX/6XX/7XX Family Data Sheet Section 7, Table 7-1 and the p32mx795f512l.h file for additional information.) Thus, for a given Timer 1 clock frequency, the period of the Timer 1 Interrupt Flag (T1IF) is equal to the period register (PR1) plus 1. Since the Timer 1 register is

reset to zero, the value set into the PR register is one less than the total number of Timer 1 clock counts. The T1IF will be used in the next project to generate interrupts.

Program Concepts

This project introduces multi-rate processing, where individual tasks can execute at different rates using a single timer. The buttons are sampled at a fixed rate, and the motor controls are generated at a rate determined by the button values. Both the button sampling and motor stepping will use a common time base provided by the Timer 1 peripheral. The lapse of time will be detected by polling an interrupt flag rather than generating an interrupt.

The stepper motor speed specifications for this project are in revolutions per minute (RPM). However, this must be translated into steps per second that can then be implemented by executing a delay between steps. This requires an equation that converts the specified RPM to a time delay per step. Fortunately, we have selected rotational speeds that are in even multiples of one millisecond. A step frequency in steps/ms is easily computed using the appropriate conversion factors. The easiest way to determine the conversion factor is to write an expression that keeps track of the units starting with rev/min and resulting in steps/ms as shown in Eq. 2, assuming 100 steps per revolutions, i.e., full steps. Inverting this conversion factor has units of ms/step which is exactly what we need to control the speed of the stepper motor.

$$Y\, steps/ms = (X\, rev/\text{min}) \cdot (100\, steps/rev) \cdot (1\,\text{min}/60\,\text{sec}) \cdot (1\,\text{sec}/1000\, ms) \qquad \text{Eq. 2}$$

The delay needed to implement the desired rotational speed, *X*, is the inverse of *Y* computed in Eq. 2. You will be implementing a single hardware assisted software timer that polls the Timer 1 interrupt flag. The Timer 1 interrupt flag is set once each millisecond. This project uses two counter variables that are decremented each millisecond until they reach zero to implement the two different delay periods. One counter variable determines the time between steps of the stepper motor and the other determines when to poll the input buttons. The counter variables are tested inside the *main* function *while(1)* loop to determine when counter variable is zero at which time the processor completes the desired tasks and resets the counter variable to the required millisecond delay.

The state of the timer interrupt flag is be tested (polled) using the “mT1GetIntFlag()” macro. This macro returns a TRUE value to indicate that the Timer 1 has reached the terminal count that sets the Timer 1 interrupt flag bit thus indicating that a millisecond delay has occurred. The macro, “mT1ClearIntFlag()”, must be used to clear the Timer 1 interrupt flag.

The delay function that you will write is called “*Timer1_delay*” and is required to perform the tasks listed in Table 1.

Table 1. “Timer1_delay” function requirements

1. Using the macro “mT1GetIntFlag()”, wait until the Timer 1 interrupt flag is set.
2. Clear the Timer 1 interrupt flag using the macro “*mT1ClearIntFlag()*”.

3. Toggle LEDA.
4. Decrement the software variable used to count the number of milliseconds delay for the stepper motor step operation.
5. Decrement the software variable used to count the number of milliseconds delay for timing the delay between successive samples of the button status. (100 ms)
6. Return (by reference) the values of the counters to the task manager, main().

Project Tasks

Write a program the follows the outline provided below using a program format presented in Project 1. The project program will contain the following functions tha1 implement the described actions.

1. "main" – the task control function responsible for calling the support functions listed below.
    a. "system_init" – initializes all PIC32MX7 hardware as follows
        i. Sets Port G to read the state of BTN1 and BTN2
        ii. Sets Port B to output only bits 2, 3, 4, 7, 8, 9,and 10
        iii. Initialize Timer 1 to set the interrupt flag once each millisecond
    b. Implements a while(1) loop that does the following
        i. Checks if the button_delay counter variable equals zero. If it is zero, it calls the functions described in "2" and "3", toggles LEDB, and resets the counter. The buttons should be read every 100 ms.
        ii. Checks if the step_delay counter variable equals zero. If it is zero, it calls the functions described in "4" and "5", toggles LEDC, and resets the counter. The time between steps will vary, depending upon the required speed and step mode.
        iii. Implements a ms delay using "6".
2. "read_buttons" – reads the status of BTN1 and BTN2
3. "decode_buttons" – determines the values of step_direction, step_mode, and step_delay using the rules specified in Table 2.
4. "stepper_state_machine" – determines the new output code for the stepper motor
5. "output_to_stepper_motor" – sends the four bit code to the stepper motor
6. "Timer1_delay" – Implements the requirements shown in Table 1.

Table 2. Stepper motor control table

Inputs		Controls		
BTN2	BTN1	DIRECTION	MODE	SPEED
Off	Off	CW	HS	15

Off	On	CW	FS	15
On	Off	CCW	HS	10
On	On	CCW	FS	25

Project Testing

Verify that BTN1 and BTN2 implement the specified direction control. Verify that the step delay properly implements the speed specifications. HS signifies the half-step mode and FS signifies the full-step mode.

Table 3. Stepper motor control verification table

Inputs		Controls			
BTN2	BTN1	MODE	SPEED RPM	Step Delay Calculated	Step Delay Measured
Off	Off	HS	15		
Off	On	FS	15		
On	Off	HS	10		
On	On	FS	25		