EZ-USB Timer Interrupts in C

Abstract

This application note is aimed at helping EZ-USB firmware developers use timer interrupts in their own applications, by providing a frameworks based timer interrupt program written in C. The assumption is made that one has a general understanding of how interrupts work within the 8051 concept.

When this program is run, users will be able to light the seven segment LED on the EZ-USB Development Board with a 0-9 count, and control the step rate (1s - 5s) using BULK OUT endpoint transfers from the EZ-USB Control Panel.

Introduction

The 8051 core includes three timers/counters (Timer 0, Timer 1, and Timer 2). Each timer/counter can operate as either a timer with a clock rate based on the internal 24 MHz clock, or as an event counter clocked by the T0 pin (Timer 0), T1 pin (Timer 1), or the T2 pin (Timer 2). Each timer/counter consists of a 16-bit register that is accesible to software as two special function registers (SFRs):

- **Timer 0 - TL0 and TH0**
- **Timer 1 - TL1 and TH1**
- **Timer 2 - TL2 and TH2**

Timer 0 and Timer 1 are mostly used to generate timer tick interrupts, and thus we will be concerned with the details regarding these two timers. Timer 2 is mostly used as a baud rate generator.

The two SFRs that control the modes of Timer 0 and Timer 1 are **TMOD** and **TCON**, shown in *Figure 1* and *Figure 2*.

The **TMOD SFR** configures the timers to be in timer mode or counter mode. In *timer* mode, the timer is counting the internal clock. In the *counter* mode, the timer counts transitions on a designated input pin of the 8051, in this case the T0 or T1 pins. Bits 7–4 are associated with Timer 1, and bits 3–0 are associated with Timer 0.

The **TCON SFR** turns on the counting or timing, and TH1/TH0 and TL1/TL0 determine the initial values for each timer.

The default timer clock scheme for the 8051 timers is 12 CLK24 cycles per increment, the same as in the standard 8051. However, in the EZ-USB 8051 the instruction cycle is 4 CLK24 cycles. Another SFR called **CKCON** allows the EZ-USB 8051 to use the default rate (12 clocks per timer increment), keeping compability with existing application code. However, applications that require fast timing can set the timers to increment every 4 CLK24 cycles by setting bits in the CKCON register.

Figure 3 shows the details associated with the **CKCON** register. For further details, please refer to **Appendix C of the EZ-USB TRM v1.8.**

Figure 1. TMOD Register

GATE = 0 — timer runs when TR0 (TR1) is set
GATE = 1 —timer runs only when INT0 (INT1) is high along with TR0 (TR1)
C/T = 0 — input from system clock
C/T = 1 — input from T0 (T1) pin.

M1 M0
Mode 00 — 13-bit counter, lower 5 bits of TL0 (TL1) and all 8 bits of TH0 (TH1)
Mode 10 — 8-bit auto-reload. The value in TH0 (TH1) will be stored in TL0 (TL1) when the latter overflows.
Mode 01 —16-bit counter (which is what we want!)
Mode 11
For Timer 0 — Two 8-bit counters
For Timer 1 — Timer 1 inactive

Figure 2. TCON Register

TF0, TF1—Overflow flag - set by hardware when timer overflows. Cleared by hardware when CPU vectors to ISR.

TR0, TR1 — Turns timer on when set, turns timer off when cleared.

IE0, IE1 — Interrupt edge flag - set by the hardware when external interrupt falling edge or low level is detected. This flag is cleared when the interrupt is processed.

IT0, IT1 — Interrupt type set by software. Set means a falling edge triggered interrupt, and cleared means a low-level triggered interrupt.

(MSB) (LSB)

Rsrvd	Rsrvd	T2M	T1M	T0M	MD2	MD1	MD0

Figure 3. CKCON Register

Rsvrd — Reserved

T2M — Timer 2 clock select. When T2M = 0, Timer 2 uses CLK24/12 (for compatibility with 80C32); when T2M = 1, Timer 2 uses CLK24/4. This bit has no effect when Timer 2 is configured for baud rate generation.
T1M —Timer 1 clock select. When T1M = 0, Timer 1 uses CLK24/12 (for compatibility with 80C32); when T1M = 1, Timer 1 uses CLK24/4.
T0M —Timer 0 clock select. When T0M = 0, Timer 0 uses CLK24/12 (for compatibility with 80C32); when T0M = 1, Timer 0 uses CLK24/4.
MD2, MD1, MD0 -- Controls the number of cycles to be used for external MOVX instructions, or stretch. These bits are not used for configuring timers.

In our application, we are interested in using Timer 0 as a 16-bit counter. Therefore, when the timer count reaches past 65535, a timer overflow interrupt will occur. When this happens, the 8051 looks for the Timer 0 ISR at address 000Bh.

Time Calculation

Let's say that we are interested in triggering a timer overflow interrupt every **10 ms, or 100 Hz**. The enhanced 8051 in the EZ-USB chip runs at **24 MHz**, and if we configure it with a **12-clock bus cycle**,in order to cause a timer overflow every 10 ms we need to set **TH0/TL0 to:**

10000h - (24,000,000 / 12 * FREQ) (where FREQ = 100Hz)
= B1E0
= TIMER0_COUNT

Timer Setup

The following code segment initializes the timer, and allows for a timer overflow interrupt to occur every 10 ms. Put this code in timer.c and call it from whereever you initialize your target hardware. In this case, TD_Init().

```
void timer0_init (void);              // function prototype

// this function enables Timer 0. Timer 0 generates
a synchronous interrupt once every 10ms or 100Hz

void timer0_init (void)
{
        EA = 0;                       // disables all interrupts
        timer0_tick = 0;

        TR0 = 0;                      // stops Timer 0

        CKCON = 0x03;                 // Timer 0 using CLK 24/12
        TMOD &= ~0x0F;                // clear Timer 0 mode bits
        TMOD |=0x01;                  // setup Timer 0 as a 16-bit timer

        TL0 = (TIMER0_COUNT & 0x00FF); // loads the timer count
        TH0 = (TIMER0_COUNT >> 8);

        PTO = 0;                      // sets the Timer 0 interrupt to low priority
        ET0 = 1;                      // enables Timer 0 interrupt
        TR0 = 1;                      // starts Timer 0
        EA = 1;                       // enables all interrupts
}
```

Once the timer is initialized, the target hardware has a 10ms timer.

The Timer 0 ISR

When Timer 0 overflows the ISR shown below is invoked, and the timer tick is incremented.

```
// This function is an interrupt service rountine for Timer 0. It should
// never be called by a C or assembly function. It will be executed auto-
// matically when Timer 0 overflows.

void timer0 (void) interrupt 1 using 1
{
        // "interrupt 1" tells the compiler to look for the ISR at address
        // 000Bh "using 1" tells the compiler to use register bank 1

        // Stop timer 0, adjust the timer 0 counter so that we get another
        // in 10ms and restart the timer

        TR0 = 0;                      // stop timer

        TL0 = TL0 + (TIMER0_COUNT & 0x00FF);
        TH0 = TH0 + (TIMER0_COUNT >> 8);

        TR0 = 1;                      // start timer 0

        // Increment the timer tick. This interrupt should occur
        // approximately every 10ms. So, the resolution of the timer will
        // be 100Hz not including interrupt latency.

        timer0_tick++;
}
```

The next two functions are very useful as ***timer0_count*** is used to get the current tick, and ***timer0_delay*** can be used to wait for an input, or flash an LED every second. If you invoke this routine using timer0_delay(100), the delay will be approximately 1 second.

timer0_count:

```
unsigned timer0_count (void);

// This function returns the current Timer 0 tick count.
// unsigned timer0_count (void)
{
        unsigned t;
        EA = 0;
        t = timer0_tick;
        EA = 1;
        return (t);
}
```

timer0_delay:

```
void timer0_delay (unsigned count);

// This function waits for 'count' timer ticks to pass.
void timer0_delay (unsigned count)
{
        unsigned start_count;
        start_count = timer0_count();// get the starting count

        while ((timer0_count() - start_count) <= count)
        // wait for timer0_tick to reach count
        {
        }

}
```

Endpoint Control Code

The code that controls how fast the 0-9 count steps is in **TD_Poll()**. Basically, you can write a single byte to **EP2 (ranging from 0x00 to 0xFF)**, and that value will be passed into the timer0_delay function.

For example, if you did a **BULK OUT of 01h to EP2**, the LED would flash every second. **Table 1** shows the hex values that correspond to delays ranging from 1 second to 5 seconds.

Note: The number of bytes you write to EP2 will determine how many loops the 0-9 count goes through. Performing an IN transfer afterwards will give you the timer tick count in the first byte. Code has also been placed into TD_Init() to flash the LED every 0.4 seconds after the hex file has finished downloading.

```
void TD_Poll(void)      // Called repeatedly while the device is idle
{

        BYTE count, i, j, temp;
        BYTE factor = 0x64;
        WORD new;

        // Is there something in the OUT2BUF buffer,
        // Is the IN2BUF available,

        if (!(EPIO[OUT2BUF_ID].cntrl & bmEPBUSY))
            if(!(EPIO[IN2BUF_ID].cntrl & bmEPBUSY))
            {
                count = OUT2BC;// then loopback the data
                for(i=0;i<count;i++)
                {
                // stores control byte in temp variable and multiplies this
                // value of 0x00 - 0xFF by 0x64 to give 1s - 5s range

                 IN2BUF[i] = OUT2BUF[i];
                 temp = OUT2BUF[i];
                 new = temp*factor;
                 IN2BUF[0] = timer0_count();

                // Loop counts from 0-9 and writes to LED with Digit
                // Waits for write and delays digit update by byte written
                // to EP2
                    for(j=0;j<10;j++)
                    {
                 EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[j]));
                 EZUSB_WaitForEEPROMWrite(LED_ADDR);
                 timer0_delay(new);
                }
                 }
                IN2BC = i;                  // Arm the IN endpoint
                OUT2BC = 0; // Arm the OUT so it can
                            // receive the next packet
            }
}
```

Table 1.

Hex Value	Seconds
01	1
02	2
03	3
04	4
05	5

To build and run this project:
Insert the code segments shown into ep_pair.c, and built it using the Keil tools. You can download the hex file using the Control Panel and perform bulk transfers to EP2 to see its functionality.

Code Listing: Timer.c

```
//-----------------------------------------------------------------------------
//              File:           timer.c
//              Contents:       Programmable timer interrupt that controls step
//                              interval (ranging from 1-5 s) of LED display from 0-9,
//                              using endpoint control from the Control Panel.
//              Length Field:Controls how many times the 0-9 count will loop.
//              Hex Bytes Field: Controls how fast the count steps
//              1) 01h -> 1s
//              2) 02h -> 2s
//              3) 03h -> 3s
//              4) 04h -> 4s
//              5) 05h -> 5s
//
//              Author:                 Zin Thein Kyaw
//
//              Copyright (c) 2000 Cypress Semiconductor. All rights reserved
//-----------------------------------------------------------------------------

#include "ezusb.h"
#include "ezregs.h"

// 10ms interrupt

#define TIMER0_COUNT 0xB1E0 // 10000h - ((24,000,000 Hz / (12 * 100))
                            // EZ-USB 8051 runs on either a 4-clock
                            // bus cycle or the traditional 12-clock bus
                            // cycle
static unsigned timer0_tick;        // timer tick variable

//-----------------------------------------------------------------------------
// Timer Interrupt
// This function is an interrupt service routine for Timer 0. It should never
// be called by a C or assembly function. It will be executed automatically
// when Timer 0 overflows.

// "interrupt 1" tells the compiler to look for this ISR at address 000Bh
// "using 1" tells the compiler to use register bank 1

void timer0 (void) interrupt 1 using 1
{
        // Stop Timer 0, adjust the Timer 0 counter so that we get another
        // in 10ms, and restart the timer.

        TR0 = 0;                // stop timer

        TL0 = TL0 + (TIMER0_COUNT & 0x00FF);
        TH0 = TH0 + (TIMER0_COUNT >> 8);

        TR0 = 1;                // start Timer 0

        // Increment the timer tick. This interrupt should occur approxi-
        // mately every 10ms. So, the resolution of the timer will be 100Hz
        // not including interrupt latency.

        timer0_tick++;
}




void timer0_init (void);

// This function enables Timer 0. Timer 0 generates a synchronous interrupt
// once every 100Hz or 10 ms.

void timer0_init (void)
{
        EA = 0;                 // disables all interrupts
        timer0_tick = 0;

        TR0 = 0;                // stops Timer 0

        CKCON = 0x03;           // Timer 0 using CLK24/12

        TMOD &= ~0x0F;          // clear Timer 0 mode bits
        TMOD |= 0x01;           // setup Timer 0 as a 16-bit timer

        TL0 = (TIMER0_COUNT & 0x00FF);// loads the timer counts
        TH0 = (TIMER0_COUNT >> 8);

        PT0 = 0;                // sets the Timer 0 interrupt to low priority
        ET0 = 1;                // enables Timer 0 interrupt
        TR0 = 1;                // starts Timer 0
        EA =  1;                // enables all interrupts
}


// This function returns the current Timer 0 tick count.
unsigned timer0_count (void)
{
        unsigned t;
        EA = 0;
        t = timer0_tick;
        EA = 1;
        return (t);
}

// This function waits for 'count' timer ticks to pass.

void timer0_delay (unsigned count)
{
        unsigned start_count;

        start_count = timer0_count();       // get the starting count

        // wait for timer0_tick to reach count
        while ((timer0_count() - start_count) <= count)
        {
        }

}
```

Code Listing: Periph.c

```
#pragma NOIV                            // Do not generate interrupt
                                        // vectors
//-----------------------------------------------------------------------------
//              File:           periph.c
//              Contents:   Hooks required to perform USB peripheral function.
//              Copyright (c) 2000 Cypress Semiconductor, Inc. All rights reserved
//-----------------------------------------------------------------------------

#include "ezusb.h"
#include "ezregs.h"

// LED address
#define LED_ADDR 0x021

extern BOOL GotSUD;                     // Received setup data flag
extern BOOL Sleep;
extern BOOL Rwuen;
extern BOOL Selfpwr;

BYTE        Configuration;              // Current configuration
BYTE        AlternateSetting;           // Alternate settings

// Seven-segment digits in hex
BYTE xdata Digit[] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
0x80, 0x98, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e };

void timer0_init (void);
unsigned timer0_count (void);
void timer0_delay (unsigned count);
void TD_Poll (void);

//-----------------------------------------------------------------------------
// Task Dispatcher hooks
//              The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------------

void TD_Init(void)                      // Called once at startup
{
        int j;
        EZUSB_InitI2C();                // Initialize I2C Bus
        timer0_init();                  // Initialize Timer 0

        // Enable endpoint 2 in, and endpoint 2 out
        IN07VAL = bmEP2;                // Validate all EP's
        OUT07VAL = bmEP2;

        // Enable double buffering on endpoint 2 in, and endpoint 2 out
        USBPAIR = 0x09;

        // Arm Endpoint 2 out to recieve data
        EPIO[OUT2BUF_ID].bytes = 0;

        // Setup breakpoint to trigger on TD_Poll()
        BPADDR = (WORD)TD_Poll;
        USBBAV |= bmBPEN;               // Enable the breakpoint
        USBBAV &= ~bmBPPULSE;

        // this loop flashes the LED with a 0-9 count at a
        // default rate of 0.4 secs immediately after download

        for(j=0;j<10;j++)
        {
                EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[j]));
                EZUSB_WaitForEEPROMWrite(LED_ADDR);
                timer0_delay(40);
        }

        Rwuen = TRUE;
}
```

```
void TD_Poll(void)      // Called repeatedly while the device is idle
{

    BYTE count, i, j, temp;
    BYTE factor = 0x64;
    WORD new;

    // Is there something in the OUT2BUF buffer,
    // Is the IN2BUF available,

    if (!(EPIO[OUT2BUF_ID].cntrl & bmEPBUSY) )
       if(!(EPIO[IN2BUF_ID].cntrl & bmEPBUSY) )
       {
          count = OUT2BC;                   // then loopback the data
          for(i=0;i<count;i++)
          {
          // stores control byte in temp variable and multiplies this
          // value of 0x00 - 0xFF by 0x64 to give 1s - 5s range

           IN2BUF[i] = OUT2BUF[i];
           temp = OUT2BUF[i];
              new = temp*factor;
              IN2BUF[0] = timer0_count();

          // Loop counts from 0-9 and writes to LED with Digit
          // Waits for write and delays digit update by byte written
          // to EP2

          for(j=0;j<10;j++)
          {
          EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[j]));
          EZUSB_WaitForEEPROMWrite(LED_ADDR);
          timer0_delay(new);
          }
          }
          IN2BC = i;                        // Arm the IN endpoint
          OUT2BC = 0;                       // Arm the OUT so it can receive
                                            / /the next packet
       }
}
```

..... the rest is the same as periph.c