Using Timer Interrupt in Cypress EZ-USB® FX2LP™ Based Applications

Author: Gayathri Vasudevan

Associated Project: Yes
Associated Part Family: CY7C68013/14/15/16
Related Application Notes: None

Abstract

AN1193 is aimed at helping EZ-USB® FX2LP™ based firmware developers use timer interrupts in their applications, by providing a framework 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, you should be able to light the seven-segment LED on the FX2LP Development Board (CY3684) with a 0-9 count, and control the step rate 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 accessible 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 compatibility 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 see the Appendix C of the EZ-USB TRM v1.8.

Figure 1. Register TMOD

(MSB)						(LSB)	
Gate	C/T	M1	M0	Gate	C/T	M1	M0
Timer 1				Timer 0			

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.

Modes 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. Register TCON

(MSB)						(LSB)	
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF0, TF1 — Overflow flag - set by hardware when timer overflows. Cleared by hardware when the 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.

Figure 3. Register CKCON

(MSB)						(LSB)	
Rsrvd	Rsrvd	T2M	T1M	T0M	MD2	MD1	MD0

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 hap-pens, the 8051 looks for the Timer 0 ISR at address **000Bh**.

Time Calculation

Let us 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 **48 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 - (48,000,000 / 12 * FREQ) (where FREQ = 100Hz)

= 63C0h

= 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 wherever you initialize your target hardware. In this case, TD_Init().

```
// 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
CLKOUT/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
```

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

Timer 0 ISR

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

```
// 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 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

```
// 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

This function takes count as input parameter, which is the number of 10 ms delays to be introduced, i.e., timer0_delay(count) will introduce a delay of (count x 10) ms delay. It will check timer0_tick value being returned by the timer0_count() and waits until timer0 interrupt occurs 'count' number of times. Timer0 interrupt will occur after every 10 ms, since we have configured the timer0 to be a 10 ms timer.

```
// 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){}
}
```

Endpoint Control Code

The code that controls how fast the 0-9 count steps is in TD_Poll(). Basically, you can write bytes to EP2 (ranging from 0x00 to 0xFF), and each 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. Please note, delays can range from 1s to 255s depending on the byte you transfer (01H – FFH).

Note The number of bytes you write to EP2 will determine how many loops the 0-9 count goes through. Performing an IN transfers afterwards will give you the timer tick counts corresponding to the first two transfers to EP2. 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;

    if(!(EP2468STAT & bmEP2EMPTY))
    {
        // check EP2 EMPTY(busy) bit in
EP2468STAT (SFR), core set's this bit
        //when FIFO is empty

        count = (EP2BCH << 8) + EP2BCL;
        // loop EP2OUT buffer data to EP6IN
        for( i = 0x0000; i < count; i++ )
            {
                temp = EP2FIFOBUF[i];
                // stores control byte in
temp variable
                new = temp*factor;
                // multiplies value of
0x00-0xFF by 64 to give 1s-5s range
                if(!(EP2468STAT &
bmEP6FULL))
                {
                // check EP6 FULL(busy) bit
in EP2468STAT (SFR), core set's this
                //bit when FIFO is full

                    EP6FIFOBUF[i] =
timer0_count();
                // if you'd like to see the
value of timer0_tick
                    }
                for(j=0;j<10;j++) // loop
counts from 0-9
            {
            // Writes to LED with Digit
```

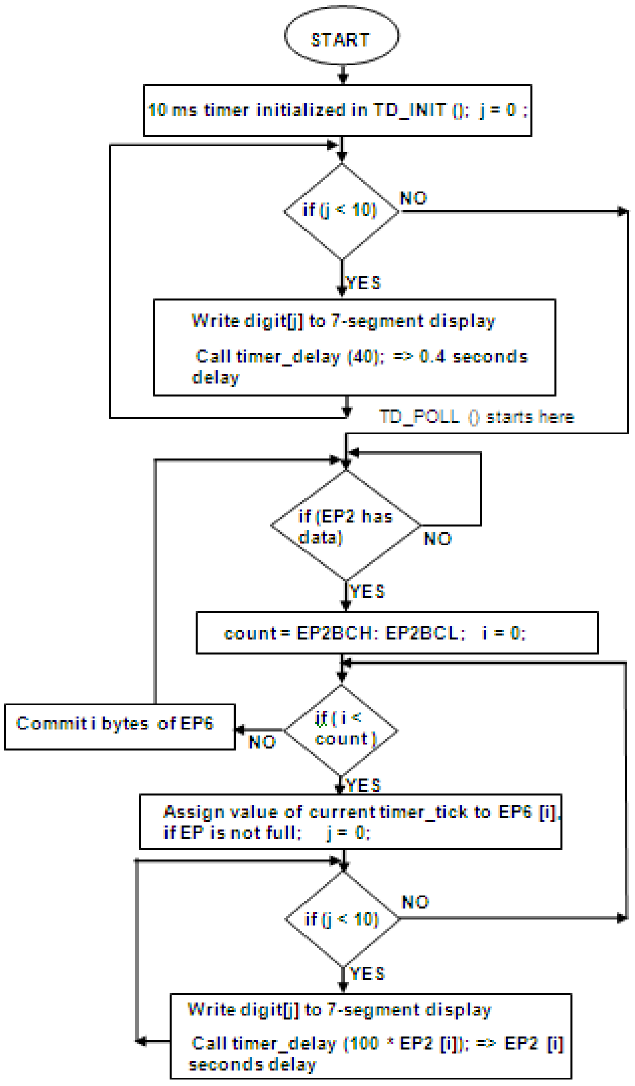
```
                EZUSB_WriteI2C(LED_ADDR,
0x01, &(Digit[j]));
                // Waits for write

EZUSB_WaitForEEPROMWrite(LED_ADDR);
                 // Delays digit update by
byte written to EP2
                 timer0_delay(new);
            }
        }
        EP6BCL = i;
        EP2BCL = 0x80;

    }
}
```

Table 1. Hex Values and Corresponding Delays

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

Flowchart

Flowchart in this section gives the overall flow of the firmware, summing up what was explained in the previous section.


START
10 ms timer initialized in TD_INIT (); j = 0 ;
if (j < 10)
NO
YES
Write digit[j] to 7-segment display
Call timer_delay (40); => 0.4 seconds delay
TD_POLL () starts here
if (EP2 has data)
NO
YES
count = EP2BCH: EP2BCL; i = 0;
Commit i bytes of EP6
NO
if (i < count)
YES
Assign value of current timer_tick to EP6 [i], if EP is not full; j = 0;
if (j < 10)
NO
YES
Write digit[j] to 7-segment display
Call timer_delay (100 * EP2 [i]); => EP2 [i] seconds delay

Running the Project

1. Download and install Cypress SuiteUSB 3.4. This installs a utility named Control Center.
2. Connect CY3684 board to the PC with EEPROM enable switch in the No EEPROM position. They will enumerate with the default internal descriptor. Use the *CyUSB.inf* file in the Drivers folder to bind with the device. For help with binding the driver, see *MatchingDriverToUSBDevice.htm* in the Drivers folder.
3. Download the *timer0.hex* file, which can be found in the attachment along with this application note at the location “\Timer in FX2LP\firmware\timer0.hex”, using the Control Centre and perform bulk transfers to EP2 to see its functionality.

Depending on the values you send to EP2 OUT Endpoint, the step rate (i.e. the inter-digit delay) of the counts displayed in the seven segment display gets modified. Table 1 shows the hex value that can be sent to EP2 and the corresponding delay. Hex values that can be sent can range from 0x01 to 0xFF. In this example, you can also see the value of timer_tick that is being assigned to EP6 and committed in TD_POLL() function, that corresponds to first two transfers to EP2 OUT. This example is just to demonstrate how the timers in FX2LP can be used. Users can customize this accordingly.

Document History

Document Title: Using Timer Interrupt in Cypress EZ-USB® FX2LP™ Based Applications – AN1193

Document Number: 001-16828

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1345183	YIS	08/03/2007	Recataloged application note
*A	3176743	SSJO	02/18/2011	Updated for related projects. Updated Title and template
*B	3356912	GAYA	08/29/2011	Updated Source and Endpoint codes. Removed Appendix section Updated ‘Running the Project’ section
*C	3446128	GAYA	11/23/2011	Updated Source and Endpoint codes and ‘Running the Project’ section Added flowchart Updated template

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

EZ-USB is a registered trademark of Cypress Semiconductor Corporation. All product and company names mentioned in this document are the trademarks of their respective holders.

Cypress Semiconductor	Phone	: 408-943-2600
198 Champion Court	Fax	: 408-943-4730
San Jose, CA 95134-1709	Website	: www.cypress.com