Getting Started with the EasyWeb2 Hardware Platform: Digital I/O (LEDs, Buttons)

by Alex Milenkovich, milenkovic@computer.org

Objectives: This tutorial will help you get started with the MSP30 IAR C program development on the hardware platform EasyWeb2. You will learn the following topics:

- Basic clock system (clock configuration)
- Interfacing buttons.
- Program downloading on a real platform (Olimex EasyWeb2).

Note: Required are all previous tutorials.

1. Blink a LED Using Software Delay

This section defines the problem that will be solved by the "Blink a LED" application.
Your task is to write a C program that will repeatedly blink the status LED on the EasyWeb2 board every 0.5 second (toggle rate is 2 Hz), i.e., the LED6 will be on and off for about 0.25 seconds each.

Step 1: Analyze the assignment.
In order to better understand the assignment we will first study schematics of the board. This board includes a TI's MSP430 microcontroller (MSP430F149), Ethernet controller (CS8900A), serial RS232 port, a status led (LED6), a buzzer, four buttons (B1-B4), and several extension slots that allow an easy access to all microcontroller ports. Detailed schematic of the board is provided in the following document:
http://www.ece.uah.edu/~milenka/cpe323-08F/docs/easyweb2-sch.gif.
Open the easyweb2-sch.gif file, zoom in and study the schematic. Please locate the status LED output port. It is connected to port P2.1. Please locate the buttons B1-B4. Which port pins are used to interface the buttons?

Step 2. Develop a plan.
From the schematic, it is clear that if we want LED6 on we should provide a logical '0' at the output port of the microcontroller (port P2.1), and a logical '1' if we want LED6 to be off. Why? We could take several approaches to solving this problem. The simplest one is to toggle the port P2.1 and have 0.25 seconds delay in software.

After initializing the microcontroller, our program will spend all its time in an infinite loop (LED6 should be repeatedly blinked). Inside a loop we will toggle the port P2.1 and then wait for approximately 0.25 seconds.

2. Blink a LED Project: C Code

1. Figure 1 shows a C program implementing blinking the status LED using software delay.

2. Notes: Upon power-on reset, the MSP430F149 clocks MCLK and SMCLK are supplied internally from the DCO. The default clock frequency is approximately 800 KHz on both.

```
//******************************************************************************
//  MSP-FET430P140 Demo - Software Toggle P2.1
//
//  Description; Toggle P2.1 by xor'ing P2.1 inside of a software loop.
//  ACLK = n/a, MCLK = SMCLK = default DCO ~ 800k
//
//                MSP430F149
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |             P2.1|-->LED
//
//  M. Buccini
//  Texas Instruments, Inc
//  January 2002
//  Built with IAR Embedded Workbench Version: 1.25A
//
//  @Alex Milenkovich, milenkovic@computer.org
//  The University of Alabama in Huntsville
//  February 2005
//  Modified for easyWeb2 board to blink the Status led (port P2.1)
//******************************************************************************

#include  <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  P2DIR |= 0x02;                        // Set P2.1 to output direction (0000_0010)

  for (;;)
  {
   unsigned int i;

    P2OUT ^= 0x02;                      // Toggle P2.1 using exclusive-OR

    i = 50000;                          // Delay
    do (i--);
    while (i != 0);
  }
}
```

Figure 1. C program for Blinking the Status LED using Software Delay.

Let us first estimate the software delay in the do-while loop. The corresponding assembly code for the do-while loop is shown below (marked in red).

```
     40               do (i--);
   \                   ??main_0:
   \   000012   3F53          ADD.W   #0xffff, R15
     41               while (i != 0);
   \   000014   0F93          CMP.W   #0x0, R15
   \   000016   FD23          JNE     ??main_0
   \   000018   F83F          JMP     ??main_1
```

One iteration of the do-while loop requires 4 processor clock cycle (use software emulation mode and the debugger's CYCLECOUNTER to determine the number of clock cycles; in general, it depends on C statements you use and compiler optimization switches). The total delay is thus equal to the number of iterations multiplied by the number of clock cycles per iteration and multiplied by the clock cycle time (or divided by the clock frequency): 50,000*4/800,000 = 0.25 seconds. To make 1 second software delay we can repeat the do-while loop 4 times in a sequence. Even better, we can write a subroutine that ensures known software delay (e.g., 0.25 second) and invoke it as many times as needed in the main program.

3. Assignments

Assignment #1: **Blinking the Status LED.**

Write a C program that toggles the status LED on the EasyWeb2 development board with frequency of 0.5 Hz (approx. 1 second is on and 1 second is off). Use the high frequency crystal for MCLK (8 MHz) and ACLK (1 MHz), and watchdog timer in the interval-timer mode. Configure the watchdog timer to be in the *interval timer* mode, clocked by ACLK. Change parameters to increase the toggle frequency to 4 Hz and to decrease it to 0.25 Hz.

Notes:

(1) What are initial conditions of the basic clock module? What are default values in the control registers? To configure the basic clock module to source MCLK from an external crystal from LFXT1 (8 MHz) and to ensure ACLK to have 1 MHz (8MHz/8) use the following code snippet. For this device you should use this sequence of steps whenever you want to source MCLK from external crystals.

```
  BCSCTL1 |= XTS;    // ACLK = LFXT1 = HF XTAL
  BCSCTL1 |= DIVA0;  // ACLK = XT1 / 8 (1 MHz)
  BCSCTL1 != DIVA1;
  do                 // wait for oscillator
  {
    IFG1 &= ~OFIFG;                   // Clear OFIFG (Oscillator Fault) flag
    for (i = 0xFF; i > 0; i--);       // Delay, time for flag to set
  }
  while ((IFG1 & OFIFG));             // OFIFG flag still set?

  BCSCTL2 |= SELM_3;                  // MCLK = LFXT1 (safe)
```

(2) Let us consider the number of iterations in a do-while loop we would need to ensure 1 second software delay: N*4/8MHz = 1 second => N = 8,000,000/4 = 2,000,000. We cannot have an integer variable initialized to 2 million (the maximum unsigned integer is 2^16-1 = 65,535). You could use a long integer variable, but the do-while loop will now take more clock cycles. Experiment on your own to determine how to ensure the required delay in software.

(3) An alternative option is to use the watchdog timer (WDT) as an interval timer that will generate an interrupt request periodically. For example, we can use ACLK as the clock source for the WDT and make it raise an interrupt request every 2^15 ACLK clock cycles (32,768*1 µs = 32.768 ms ~ 32 ms). The WDT ISR can count the number of 32 ms periods (jiffy clocks) and take an action when a desired number of jiffy clocks has been reached. See below for the WDT ISR code.

```
#define WDT_ADLY_32ms  (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL) // fACLK*2**15 = 32ms
#pragma vector=WDT_VECTOR
```

```
__interrupt void watchdog_timer(void){
  static int i = 0;
  i++;
  if (i == 33) {    // 1 second (33*32.7 ms = 1 sec)
    P2OUT ^= 0x02;  // Toggle P2.1 using exclusive-OR
    i = 0;          // reset the counter
  }
```

Assignment #2: Interfacing buttons on the MSP430 EasyWeb2

Write a C program that scans the buttons of the EasyWeb2 board and responds as described below. The buttons B1, B2, B3, and B4 are connected to P4.4 – P4.7.

- If the yellow button (B1) is pressed (port P4.4), blinks the status LED (port P2.1) with frequency of 4 Hz.
- If the red button (B2) is pressed (port P4.5), toggle Relay 2 (connected to the port P1.6).
- If the white button (B3) is pressed (port P4.6), toggle Relay 1 every 5 seconds. Relay 1 is connected to the port P1.5.

Notes:

(1) To interface a button we need to detect that it has been pressed, to debounce it (apply software delay to ensure that is indeed pressed, rather than a faulty detection caused by noise), and to detect that is has been released. E.g., see below these steps for button 1 (B1).

```
#define B1  BIT4&P4IN         // B1 - P4.4
...
     if ((B1) == 0) {
          for(i = DEBOUNCE; i > 0; i--); // debounce
          if((B1) == 0) bp = 1; // set B1 flag to indicate it has been pressed
          while(B1==0) {};      // wait for release
          // do something if needed
      }
```

(2) Your main program can be an infinite loop with the code to interface various buttons and perform one-time tasks (e.g., toggle a relay), while the periodic tasks can be handled in the WDT ISR.