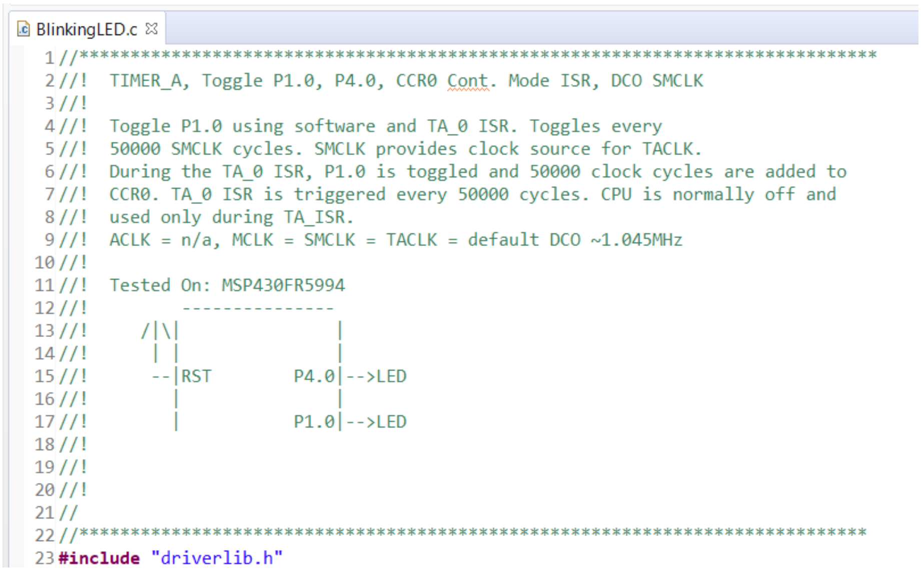
Example 1: BlinkingLED

The first example is very simple, but it will show the basics of GPIO (General Purpose Input/Output) pins and interrupt routines. The first step is to get set up in Code Composer Studio. Then, import this example project from the Github directory: [https://github.com/NCSU-Scout-IoT/SeniorDesignF2019](https://github.com/NCSU-Scout-IoT/SeniorDesignF2019). After you have the project in CCS, try connecting your Scout to your LaunchPad and flashing the example to your board.

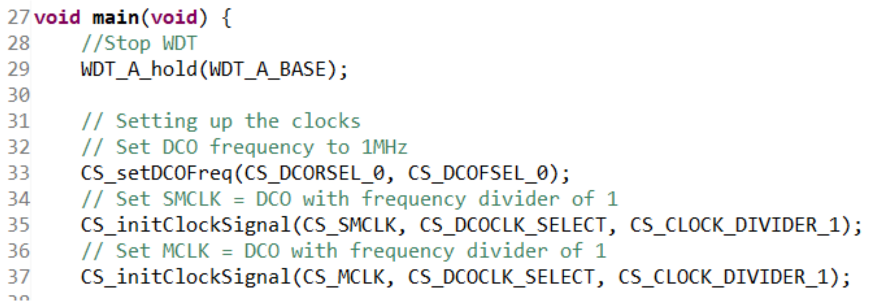
The first thing you will see is a short comment block with an explanation of what is happening in the example. In the blinking LED example, a timer is set up to operate in compare mode. In the MSP430, a timer is essentially a counter, and will increment every time its clock source ticks. When in compare mode, every time it increments, it compares its value with the value in a register called a capture/compare register (CCR). The number of CCRs to choose from depends on the timer. See the MSP430FR5994 datasheet for a complete list of CCRs and timers. For this example, we will use timer `TA1` with its first capture/compare register, `CCR0`. When the timer value is equal to the value in `CCR0`, the interrupt routine will be called. This is a special function that is accessed in our code no matter where we are when the interrupt is triggered. In this example, however, there is no other code, so the interrupt routine will only wake up from low-power mode. We will look at how exactly this interrupt operates later on.


BlinkingLED.c

```
//******************************************************************************
//!  TIMER_A, Toggle P1.0, P4.0, CCR0 Cont. Mode ISR, DCO SMCLK
//!
//!  Toggle P1.0 using software and TA_0 ISR. Toggles every
//!  50000 SMCLK cycles. SMCLK provides clock source for TACLK.
//!  During the TA_0 ISR, P1.0 is toggled and 50000 clock cycles are added to
//!  CCR0. TA_0 ISR is triggered every 50000 cycles. CPU is normally off and
//!  used only during TA_ISR.
//!  ACLK = n/a, MCLK = SMCLK = TACLK = default DCO ~1.045MHz
//!
//!  Tested On: MSP430FR5994
//!         ---------------
//!     /|\|               |
//!      | |               |
//!      --|RST        P4.0|-->LED
//!        |               |
//!        |           P1.0|-->LED
//!
//!
//!
//
//******************************************************************************
#include "driverlib.h"
```

The first line of code in every example will be the inclusion of the driver library. This library allows us to use functions that will manipulate the internal registers and controls of the MSP430 easily. The second line is the compare value definition. This acts as the period of our timer, the lower it is, the more rapidly the LEDs will blink. It will act as an unsigned 16-bit integer, so its maximum value is 65535. Values above the maximum might give unexpected behavior.

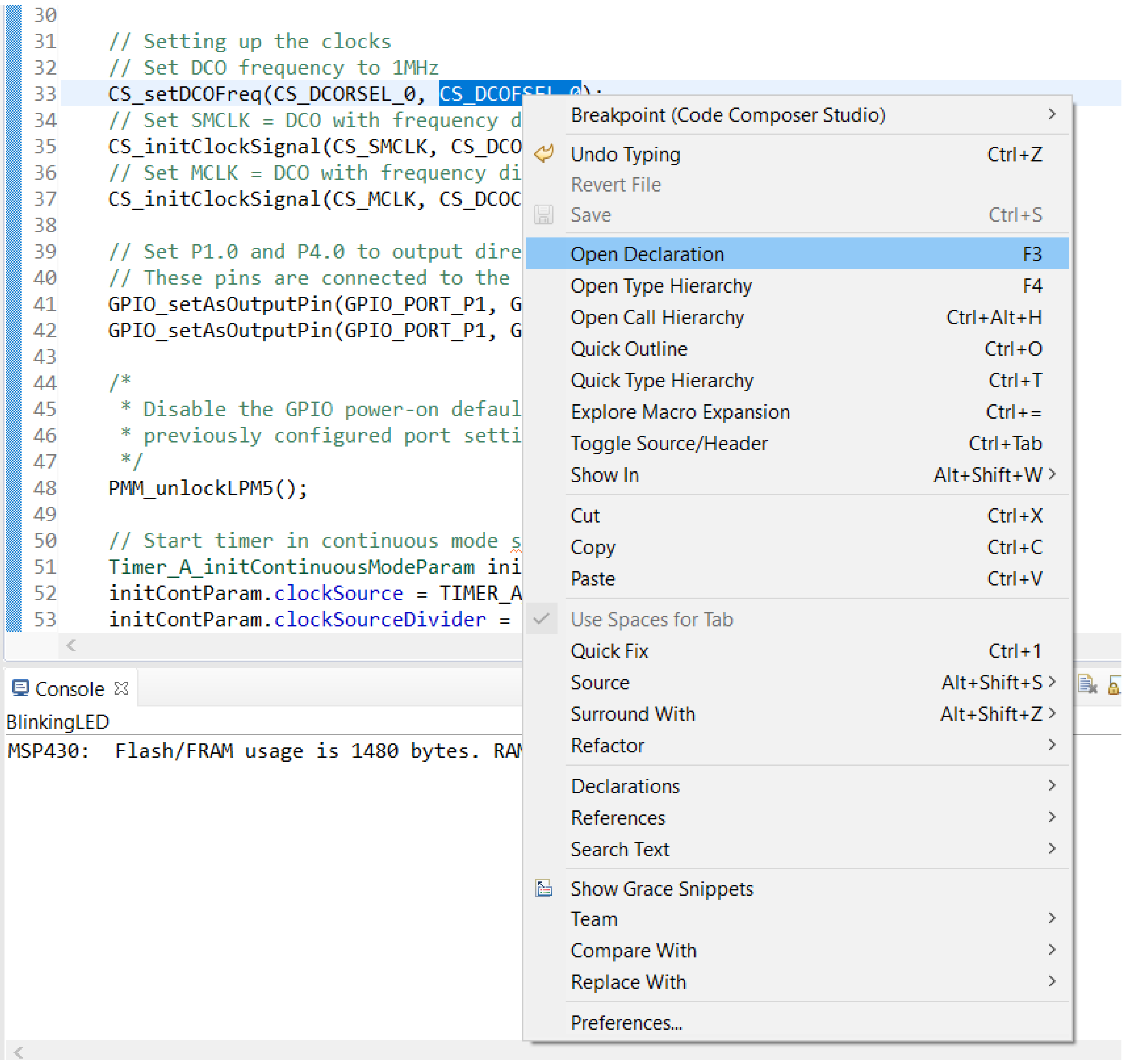
After these two lines is the beginning of the main function. The MSP430 will always begin operation executing the main function.  The first line in the main function stops the watchdog timer. We will not be using the watchdog timer in any examples. Next, the clocks are set up. This step is



```
void main(void) {
    //Stop WDT
    WDT_A_hold(WDT_A_BASE);

    // Setting up the clocks
    // Set DCO frequency to 1MHz
    CS_setDCOFreq(CS_DCORSEL_0, CS_DCOFSEL_0);
    // Set SMCLK = DCO with frequency divider of 1
    CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    // Set MCLK = DCO with frequency divider of 1
    CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
```

optional in this example because the clocks are all at their default frequencies. However, this is a good place to learn a bit about the driver library. If you select one of the arguments to the functions and right click, you can select "Open Declaration" or press F3 to navigate to the location in the driver library where the macro is defined.



```

    // Setting up the clocks
    // Set DCO frequency to 1MHz
    CS_setDCOFreq(CS_DCORSEL_0, CS_DCOFSEL_0);
    // Set SMCLK = DCO with frequency d
    CS_initClockSignal(CS_SMCLK, CS_DCO
    // Set MCLK = DCO with frequency di
    CS_initClockSignal(CS_MCLK, CS_DCOC

    // Set P1.0 and P4.0 to output dire
    // These pins are connected to the
    GPIO_setAsOutputPin(GPIO_PORT_P1, G
    GPIO_setAsOutputPin(GPIO_PORT_P1, G

    /*
     * Disable the GPIO power-on defaul
     * previously configured port setti
     */
    PMM_unlockLPM5();

    // Start timer in continuous mode s
    Timer_A_initContinuousModeParam ini
    initContParam.clockSource = TIMER_A
    initContParam.clockSourceDivider =
```

Console
BlinkingLED

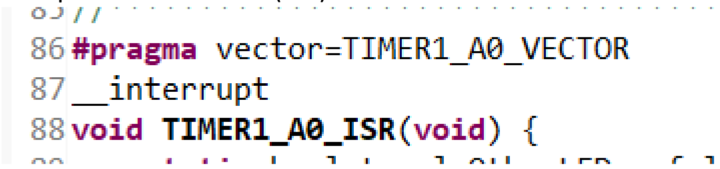
```
MSP430:  Flash/FRAM usage is 1480 bytes. RA
```

If you follow this all the way, you will find that `CS_DCOFSEL_0` is defined as `0x0000` and you can read in the comments that this translates to a frequency of 1MHz. You can change this macro in your code to one of the others defined and see what happens to your board when you re-flash it. If you ever come across a macro or function you don't understand or recognize, try following it to the driver library. After all the MCLK (master clock) and SMCLK (sub-master clock) are set to 1MHz, the GPIO pins are then set up.

To set up the ports we need to control the LEDs, we will again use functions from the driver library. You can follow these functions to `GPIO.h` and see what other options we have when setting up a port and refer to the datasheet to the capabilities of any given port. Keep in mind that the Scout uses the 64-pin PM package, so not every port in the datasheet is available. The two ports connected to the LEDs are set as simple outputs.

Lines 51-72 set up and start the timer. To start the timer, it first needs to be set up in continuous mode, meaning it simply counts up and resets at $2^{16}$-1. The function to start the timer is line 57, and the function needs an address to specify which timer to start and a struct with all the properties. You can look at all these properties and their valid values in `timer_a.h`. We set its clock source to the SMCLK with a divider of 8 which translates to a frequency of about 125kHz. We then set the timer to compare mode and point it to `CCR0`, giving it its initial value on line 69. After the timers are set up and started, the main function ends with putting the MSP430 into low-power mode.

The last part of the example is the interrupt service routine (ISR). The ISR is a function that takes no argument and returns no value and is always preceded by `#pragma vector=<VECTOR>` and `__interrupt`. This tells the compiler that this is the ISR that is triggered by the timer A1. This ISR will toggle the LED on P1.0 every time it is called (line 95) and the LED on P4.0 every other time (line 99) by toggling a boolean true and false. Also in the ISR is some math to calculate what the next value of `CCR0` should be. For example if your desired period was 1000, the timer would count to 1000, and your interrupt would trigger. If you did not change `CCR0`, it would stay as 1000, and your timer would count all the way to $2^{16}$-1 (65535) before reseting to 0 and then starting over. If you wanted a period of 1000, you would need to add 1000 to CCR0 every time you interrupt, so CCR0 would be 1000, then 2000, then 3000, …and so on. Line 91 calculates the next value of CCR0, and line 104 sets it.



```
#pragma vector=TIMER1_A0_VECTOR
__interrupt
void TIMER1_A0_ISR(void) {
```