CPE 325: Embedded Systems Laboratory
Laboratory Tutorial #6:
MSP430 Interrupts and Universal Clock Subsystem

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

Objective:

This tutorial will teach you how to write interrupt service routines in assembly and C/C++ programming languages. In addition, it will describe the universal clock subsystem of the MSP430F5529 device that is responsible for generating internal clocks. You will learn the following topics:

Using interrupts in C/assembly (specifically working with port interrupts)
The clock subsystem and clock configuration
Working with the TI experimenter's board

Notes:

All previous tutorials are required for successful completion of this lab, especially, the tutorials introducing the MSP-EXP430F5529LP board and the Code Composer Studio software development environment.

Contents

1 Interfacing Switches and LEDs in Assembly (Polling and Interrupts)

In the handout for Laboratory #3 we learned how to interface with the MSP-EX430F5529LP hardware, specifically LEDs and switches, using C language. We will redo the same examples using the MSP430 assembly language.

1.1 Toggling LEDs in Assembly Language

Figure 1 shows the assembly code of the blink application (Lab6_D1.asm). Here is a brief description of the assembly code for this application. In addition to the portions of the code that were discussed in the previous labs we can discuss some new additions. The .text is a segment control assembler directive that controls how code and data are located in memory. .text is used to mark the beginning of a relocatable code. The linker can recognize any other type of segment (e.g., __STACK_END for code stack). Our main loop that flashes the LEDs starts at the InfLoop label. The code starting at the label SWDelay1 implements the software delay to make sure the LEDs blink at the appropriate interval. To exactly calculate the software delay we need to know the instruction execution time and the clock cycle time. The register R15 is loaded with 65,535 (the maximum unsigned integer that can fit in a 16-bit register). The dec.w instruction takes 1 clock cycle to execute, and jnz L1 takes 2 clock cycles to execute (note: this can be determined by enabling and reading the value of the clock in CCS). The nop instruction takes 1 clock cycle. The number of nop instructions in the loop is determined so that the total number of clocks in the SWDelay1 loop is 16. Determining clock cycle time requires in-depth understanding of the FLL-Clock module of the MSP430 which is discussed later in this tutorial. We note that the processor clock frequency is 1,048,576 Hz (2^{20} Hz) for the default configuration. The total delay is thus $65{,}535*16/2^{20} \sim 1$s. Note: nop instructions are often used in creating software delays because they do not affect the state of the registers and take exactly one clock cycle to execute.

```
;-------------------------------------------------------------------------------
;   File:        Lab6_D1.asm
;   Description: The program toggles LEDs periodically.
;                LED1 is initialized off, LED2 is initalized on.
;                Main program loop:
;                   the SWDelay1 loop creates 1s delay before
;                   toggling the LEDs (ON/OFF).
;
;   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
;   Platform:    TI EXP430F5529LP Launchpad
;
;                 MSP430xF5529
;              -----------------
;          /|\|                 |
;           | |                 |
;           --|RST              |
;             |             P1.0|-->LED1(RED)
;             |             P4.7|-->LED2(GREEN)
;
;   Author:     Aleksandar Milenkovic, milenkovic@computer.org
```

```
;   Date:       September 14, 2018
;   Modified:   Prawar Poudel, August 08, 2019
;-------------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

SETUP:
            bis.b   #0x01,&P1DIR            ; set P1.0 as output, 0'b0000 0001
            bis.b   #0x80,&P4DIR            ; set P4.7 as output, 0'b1000 0000

            bic.b   #0x01,&P1OUT            ; turn P1.0 OFF
            bis.b   #0x80,&P4OUT            ; turn P4.7 ON
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
InfLoop:
            mov.w #0xFFFF,R5    ; move 0xFFFF to R5 which will be out counter
SWDelay1:
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop                 ; 13 NOPs + extra 3cc is a delay of 16cc
            nop                 ; so the total delay is 65535*16cc/2^20 ~ 1s
            nop
            nop
            nop
            dec.w R5                        ; 1cc
            jnz SWDelay1                    ; 2cc
            xor.b #0x01,&P1OUT              ; toggle 1.0
            xor.b #0x80,&P4OUT              ; toggle 4.7
            jmp InfLoop                     ; go to InfLoop
            nop
```

```

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET

```

Figure 1. Blinking the LEDs in Assembly Language

1.2 Interfacing Switches in Assembly Language (Polling)

Figure 2 shows assembly program that interfaces S1 and LED1. S1 is connected to P1.BIT0 (ports are configured by default as input) and LED1 is connected to P2.BIT2 (should be configured as a digital output). BIT0 of P1 is checked. If pressed a logic 0 should be detected in P1IN.BIT0; otherwise it should read as a logic 1. When a press is detected, a software delay of 20 ms is implemented to support de-bouncing of the switch. If the switch is still pressed, the program turns on LED1. The program continually checks whether the switch is still pressed. If a release (depress) is detected, LED1 is turned off.

```
;-------------------------------------------------------------------------------
;   File:         Lab6_D2.asm
;   Description:  The program demonstrates Press/Release using S1 and LED1.
;                 LED1 is initialized off.
;                 When an S1 press is detected, a software delay of 20 ms
;                 is used to implement debouncing. The switch is checked
;                 again, and if it's on, LED1 is turned on until S1 is released.
;
;   Clocks:       ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
;   Platform:     TI EXP430F5529LP Launchpad
;
;                  MSP430F5529
;              -----------------
;          /|\|                 |
;           | |                 |
;           --|RST              |
;             |             P1.0|-->LED1(REd)
;             |             P2.1|<--SW1
;
;   Author:     Aleksandar Milenkovic, milenkovic@computer.org
;   Date:       September 14, 2018
;   Modified:   Prawar Poudel, August 08, 2019
;-------------------------------------------------------------------------------

            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
```

```
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------------
RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer
StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
SetupP2:
            bis.b   #001h, &P1DIR           ; Set P1.0 to output
                                            ; direction (0000_0001)
            bic.b   #001h, &P1OUT           ; Set P1OUT to 0x0000_0001 (ensure
                                            ; LED1 is off)

            bic.b   #002h, &P2DIR           ; SET P2.1 as input for SW1
            bis.b   #002h, &P2REN           ; Enable Pull-Up resister at P2.1
            bis.b   #002h, &P2OUT           ; required for proper IO set up

ChkSW1:     bic.b   #001h, &P1OUT
            bit.b   #002h, &P2IN             ; Check if SW1 is pressed
                                            ; (0000_0010 on P1IN)
            jnz     ChkSW1                  ; If not zero, SW1 is not pressed
                                            ; loop and check again
Debounce:
            mov.w   #2000, R15              ; Set to (2000 * 10 cc = 20,000 cc)
SWD20ms:    dec.w   R15                     ; Decrement R15
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            jnz     SWD20ms                 ; Delay over?
            bit.b   #00000010b, &P2IN       ; Verify SW1 is still pressed
            jnz     ChkSW1                  ; If not, wait for SW1 press

LEDon:      bis.b   #001h, &P1OUT           ; Turn on LED1
SW1wait:    bit.b   #002h, &P2IN            ; Test SW1
            jz      SW1wait                 ; Wait until SW1 is released
            bic.b   #001h, &P1OUT            ; Turn off LED1
            jmp     ChkSW1                  ; Loop to beginning
            nop

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

```

```
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
            .end


```

Figure 2. Turn on LED1 when S1 is Pressed (Lab6_D2.asm)

1.3 Interfacing Switches in Assembly Language (Interrupt Service Routine)

With microcontrollers, it is often useful to be able to use interrupts in our programs. An interrupt allows an automatic break from the current program flow based on a set of conditions. Some of the I/O ports on the MSP430 have an interrupt capability that you can configure. When the interrupt conditions are met, the program execution departs into a service routine that handles the interrupt event. Once service routine is completed the control transfers back to the main program where it left off using a RETI (return from interrupt) instruction. We will learn more about interrupts in a subsequent lab, but you should understand how interrupt vectors are used and what interrupts do. To set up an interrupt for an I/O port, we have to perform a few tasks:

- Enable global interrupts in the status register
- Enable interrupts to occur for the particular events by setting control bits in corresponding registers associated with ports P1 or P2
- Specify whether the interrupt is triggered on a falling edge or rising edge
- Initialize the interrupt flag by clearing it

An example of using interrupts to interface the switches of the MSP430 experimenter board is shown in Figure 3. The main program configures ports, enables the global interrupts (GIE bit is SR is set), enables interrupt from BIT1 of Port1 (P1IE=0x0000_0010b). As pressing a switch corresponds to having input signal from a logic ‘1’ to a logic ‘0’, the interrupt arises when a falling edge is detected at P1IN.BIT1. The interrupt service routine starts at label SW2_ISR. The state of the input is checked; if P1IN.BIT1 is not a logic 0 we exit the ISR; otherwise, de-bouncing is performed. If SW2 is still pressed after 20 ms, LED1 is turned on. The program then waits for SW2 to be released. Note lines 94 and 95 that initialize the IVT entry 47 reserved for Port 1.

```
1  ;-------------------------------------------------------------------------------
2  ;   File:        Lab6_D3.asm
3  ;   Description: The program demonstrates Press/Release using S2 and LED1.
4  ;                LED1 is initialized off. The main program enables interrupts
5  ;                from P1.BIT1 (S2) and remains in an infinite loop doing nothing.
6  ;                P1_ISR implements debouncing and waits for a S2 to be released.
7  ;
8  ;   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
9  ;   Platform:    TI EXP430F5529LP Launchpad
10 ;
11 ;                 MSP430F5529
```

```
;              -----------------
;          /|\|                 |
;           | |                 |
;           --|RST              |
;             |             P1.0|-->LED1(RED)
;             |             P1.1|<--S2
;
;   Author:     Aleksandar Milenkovic, milenkovic@computer.org
;   Date:       September 14, 2018
;   Modified:   Prawar Poudel, Auguest 8, 2019
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .def    S2_ISR
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------------
RESET:      mov.w   #__STACK_END, SP        ; Initialize stack pointer
StopWDT:    mov.w   #WDTPW|WDTHOLD, &WDTCTL ; Stop watchdog timer
;-------------------------------------------------------------------------------
Setup:
            bis.b   #001h, &P1DIR           ; Set P1.0 to output
                                            ; direction (0000_0001)
            bic.b   #001h, &P1OUT           ; Set P1OUT to 0x0000_0001

            bic.b   #002h, &P1DIR           ; SET P1.1 as input for S2
            bis.b   #002h, &P1REN           ; Enable Pull-Up resister at P1.1
            bis.b   #002h, &P1OUT           ; required for proper IO set up


            bis.w   #GIE, SR                ; Enable Global Interrupts
            bis.b   #002h, &P1IE            ; Enable Port 1 interrupt from bit 1
            bis.b   #002h, &P1IES           ; Set interrupt to call from hi to low
            bic.b   #002h, &P1IFG           ; Clear interrupt flag
InfLoop:
            jmp     $                       ; Loop here until interrupt

;----------------------------------------------------------------------------
; P1_0 (S2) interrupt service routine (ISR)
;----------------------------------------------------------------------------
S2_ISR:
            bic.b   #002h, &P1IFG           ; Clear interrupt flag
ChkSW2:     bit.b   #02h, &P1IN             ; Check if S2 is pressed
                                            ; (0000_0010 on P1IN)
            jnz     LExit                   ; If not zero, SW is not pressed
                                            ; loop and check again
Debounce:   mov.w   #2000, R15              ; Set to (2000 * 10 cc )
```

```
SWD20ms:    dec.w   R15                     ; Decrement R15
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            jnz     SWD20ms                 ; Delay over?
            bit.b   #00000010b,&P1IN        ; Verify S2 is still pressed
            jnz     LExit                   ; If not, wait for S2 press
LEDon:      bis.b   #001h,&P1OUT            ; Turn on LED1
SW2wait:    bit.b   #002h,&P1IN             ; Test S2
            jz      SW2wait                 ; Wait until S2 is released
            bic.b   #001,&P1OUT             ; Turn off LED1
LExit:      reti                            ; Return from interrupt
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"        ; MSP430 RESET Vector
            .short  RESET
            .sect   ".int47"        ; PORT1_VECTOR,
            .short  S2_ISR         ; please check the MSP430F5529.h header file
            .end

```

Figure 3. Press/Release Using Port 1 ISR (Lab6_D3.asm)

2 Interfacing Switches and LEDs Using Interrupts in C

Figure 4 shows a C program that turns LED1 on when S2 is pressed and turns LED1 off when S2 is released. The main program configures and initializes ports, configures interrupts, and enters an infinite loop where the program waits for S2 to be released to turn off LED1. P1_ISR is entered upon detection of the switch press; the code inside clears P1.IFG1 and turns on LED1. Please note C convention to indicate that Port1_ISR corresponds to PORT1_VECTOR in the interrupt vector table.

```
/*******************************************************************************
 *   File:        Lab6_D4.c
 *   Description: The program detects when S2 is pressed and turns on LED1.
 *                LED1 is kept on as long as S2 is pressed.
 *                P1_ISR is used to detect when S2 is pressed.
 *                Main program polls S2 and turns off when a release is detected.
 *   Board:       MSP-EXP430F5529LP Launchpad
 *   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
 *
 *                 MSP430F5529
```

```
 *            +-----------------+
 *            |                 |
 *            |                 |
 *            |                 |
 *            |                 |
 *            |             P1.0|--> LED1
 *            |             P1.1|<-- S2
 *
 *   Author:    Aleksandar Milenkovic, milenkovic@computer.org
 *   Date:      September 2010
 *   Modified:  Prawar Poudel, August 08, 2019
*******************************************************************************/
#include  <msp430.h>
#define   S2 BIT1&P1IN              // S2 is P1IN&BIT1

void main(void) {
    WDTCTL = WDTPW+WDTHOLD;         // Stop WDT

    P1DIR |= BIT0;                  // Set LED1 as output
    P1OUT = 0x00;                   // Clear LED1

    P1DIR &= ~BIT1;                 // Set the direction at S2 as input
    P1REN |= BIT1;                  // Enable Pull-up resistor
    P1OUT |= BIT1;                  // Required for proper IO

    _EINT();                        // Enable interrupts

    P1IE |= BIT1;                   // P1.1 interrupt enabled
    P1IES |= BIT1;                  // P1.1 hi/low edge
    P1IFG &= ~BIT1;                 // P1.1 IFG cleared

    for(;;) {
        while((S2) == 0);           // Wait until S2 is released
        P1OUT &= ~BIT0;             // LED1 is turned off
    }
}

// Port 1 interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port1_ISR (void) {
    P1OUT |= BIT0;                // LED1 is turned ON
    P1IFG &= ~BIT1;               // P1.0 IFG cleared
}
```

Figure 4. Press/Release Using Port 1 ISR (Lab6_D4.c)

Looking at the program in Figure 4 we can see that release is detected in the main program. A better implementation would delegate both press and release activities into the P1 ISR as shown in Figure 5. To implement this, we need to establish a global variable called S2pressed that keeps the current state of the switch (0 – released, 1 – pressed). At the beginning we expect a press event, so Port 1 is configured to wait for a falling edge on P1IN.BIT1 (SW2 is pressed). In that case, the ISR turns on LED1, sets the S2pressed and configures P1IES to trigger an interrupt when a rising edge is detected on P1IN.BIT1. When the switch is pressed and we the ISR is entered, the steps are taken to turn LED1 off and configure P1IES so that a new press event can be detected. This way, all work is done inside the P1 ISR and main program can put the processor into sleep state.

```
/*------------------------------------------------------------------------------
 *   File:        Lab6_D5.c
 *   Description: The program detects when S2 is pressed and turns on LED1.
 *                LED1 is kept on as long as S2 is pressed.
 *                P1_ISR is used to detect both S2 presses and releases.
 *   Board:       EXP430F5529LP Launchpad
 *   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
 *
 *                 MSP430F5529
 *             +-----------------+
 *             |                 |
 *             |                 |
 *             |                 |
 *             |                 |
 *             |             P1.0|--> LED1
 *             |             P1.1|<-- S2
 *
 *   Author: Aleksandar Milenkovic, milenkovic@computer.org
 *   Date:   September 2010
 *------------------------------------------------------------------------------*/
#include  <msp430.h>

unsigned char S2pressed = 0;        // S2 status (0 not pressed, 1 pressed)

void main(void) {
    WDTCTL = WDTPW+WDTHOLD;         // Stop WDT
    P1DIR |= BIT0;                  // Set LED1 as output
    P1OUT = 0x00;                   // Clear LED1 status

    S2pressed = 0;

    P1DIR &= ~BIT1;                 // Set the direction at S2 as input
    P1REN |= BIT1;                  // Enable Pull-up resistor
    P1OUT |= BIT1;                  // Required for proper IO

    _EINT();                        // Enable interrupts
    P1IE |= BIT1;                   // P1IE.BIT1 interrupt enabled
    P1IES |= BIT1;                  // P1IES.BIT1 hi/low edge
    P1IFG &= ~BIT1;                 // P1IFG.BIT1 is cleared

    _BIS_SR(LPM0_bits + GIE);       // Enter LPM0(CPU is off); Enable interrupts
}

// Port 2 interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port1_ISR (void) {
    if (S2pressed == 0) {
        S2pressed = 1;
        P1OUT |= BIT0;                // LED1 is turned ON
```

```
        P1IFG &= ~BIT1;               // P1IFG.BIT0 is cleared
        P1IES &= ~BIT1;               // P1IES.BIT0 low/high edge
    } else if (S2pressed == 1) {
        S2pressed = 0;
        P1OUT &= ~BIT0;                // LED1 is turned ON
        P1IFG &= ~BIT1;                // P1IFG.BIT0 is cleared
        P1IES |= BIT1;                 // P1IES.BIT0 hi/low edge
    }
}

```

Figure 5. Press/release Using Port 1 ISR – An Improved Implementation (Lab6_D5.c)

3 Clock Module

In the previous examples we have learned how to write a program that toggles the LEDs connected to the MSP430's output ports. We have also learned how write code to generate software delays. In our example, we assumed that the processor clock is around 1 µs (i.e., the clock frequency is approximately 1 MHz). The MSP430 family supports several clock modules and a user has a full control over these modules. By changing the content of relevant clock module control registers, one can change the processor clock frequency, as well as the frequency of other clock signals that are used for peripheral devices. In the next section, we will discuss the organization of the Unified Clock System (UCS) used in the MSP430F5529 device.

3.1 Unified Clock System (UCS)

MSP430x5xx family of microcontrollers have Unified Clock System (UCS) that provides various clocks for the MSP430 modules. The UCS module includes up to five clock sources and provide three clock signals.

The five clock sources included in the UCS module are as follows:

- **XT1CLK**: Low-frequency or high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, standard crystals, resonators, or external clock sources in the 4 MHz to 32 MHz range. XT1CLK can be used as a clock reference into the FLL. Some devices only support the low frequency oscillator for XT1CLK.
- **VLOCLK**: Internal very low power, low-frequency oscillator with 10-kHz typical frequency.
- **REFOCLK**: Internal trimmed low-frequency oscillator with 32768-Hz typical frequency, can be used as a clock reference into the FLL.
- **DCOCLK**: Internal digitally controlled oscillator (DCO) that can be stabilized by the FLL.
- **XT2CLK**: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4 MHz to 32 MHz range. XT2CLK can be used as a clock reference into the FLL.

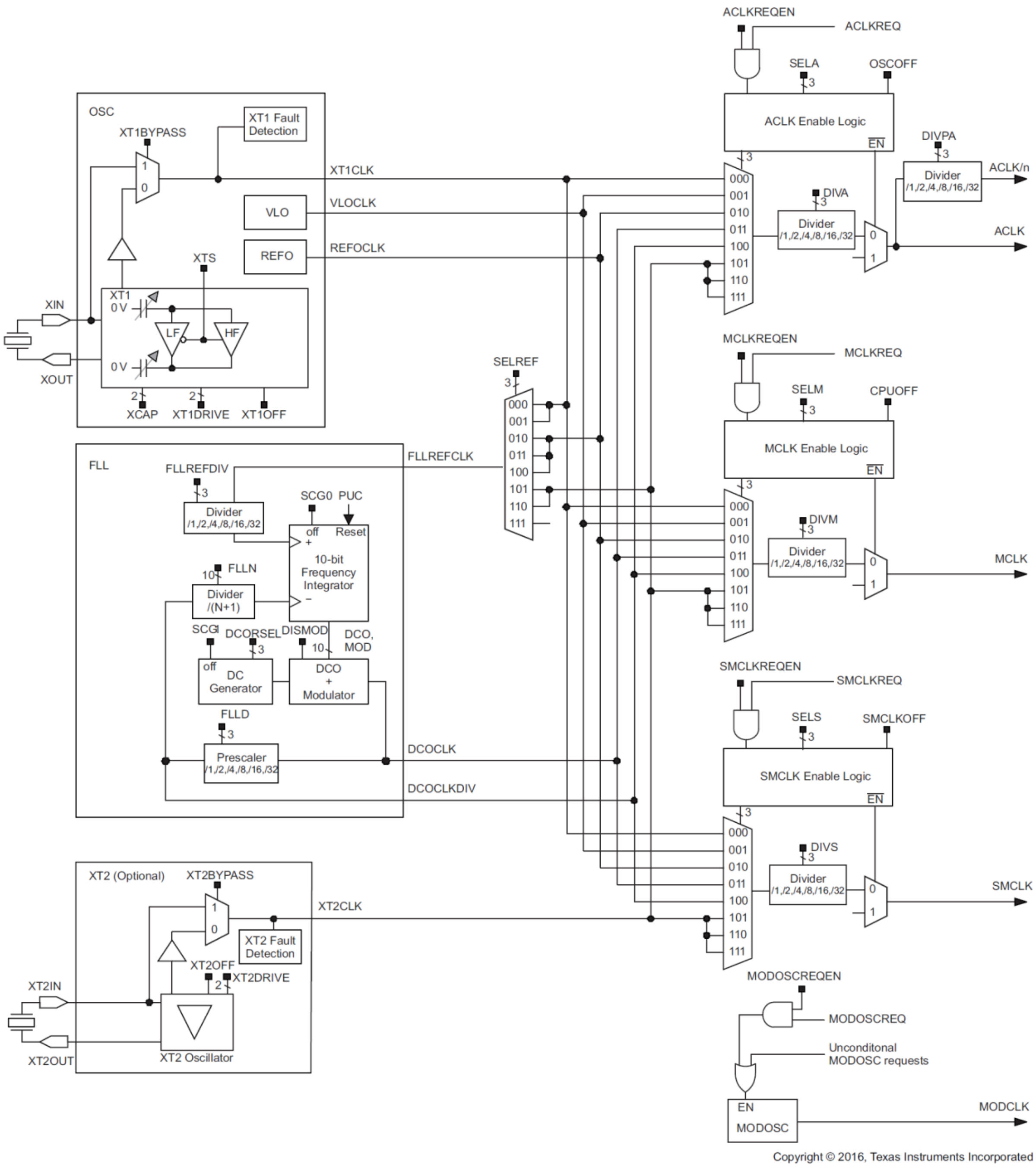

Figure 6. Block diagram of UCS module in MSP430x5xx devices

Following are the clock signals provided by the USC module:

- **ACLK**: Auxiliary clock. The ACLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK/n is ACLK divided by 1, 2, 4, 8, 16, or 32 and is available externally at a pin. ACLK is software selectable by individual peripheral modules.

- **MCLK**: Master clock. MCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- **SMCLK**: Subsystem master clock. SMCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. SMCLK can be divided by 1, 2, 4, 8, 16, or 32. SMCLK is software selectable by individual peripheral modules.

On a PUC, the configuration of UCS is as follows:

- XT1 in LF mode is selected as source for XT1CLK which is selected for ACLK.
- DCOCLKDIV is selected for MCLK and SMCLK.
- FLL operation is enabled and XT1CLK is selected as reference clock for FLL.
- If the 32768 Hz crystal is used for XT1CLK, fault control logic sources ACLK with 32768 Hz REFOCLK because XT1 takes some time to stabilize.
- When the crystal start-up is obtained, FLL stabilizes MCLK and SMCLK at 1048576 Hz (2^20 Hz). f_{DCO} = 2097152 Hz

The operating modes of UCS is controlled using the following registers: **SCG0**, **SCG1**, **OSCOFF** and **CPUOFF**.

The UCS module can be configured using registers from **UCSCTL0** through **UCSCTL8**.

Digital Controlled Oscillator (DCO):

- The frequency of DCO can be adjusted by software using DCORSEL (in UCSCTL1 register), DCO and MOD bits (in UCSCTL0).
- DCO can also be stabilized using FLL to a multiple of FLL reference clock (i.e. multiple frequency of FLLREFCLK/n). FLL can accept different reference sources selectable by SELREF bits (UCSCTL3). The reference sources can be XT1CLK, REFOCLK or XT2CLK if available.
- The value of *n* can be defined in FLLREFDIV bits in UCSCTL3 register (n = 1,2,4,8,12 or 16). Default value is *n*=1.
- FLLD bits (in UCSCTL2 register) can be configured for FLL pre-scalar divider value D of 1,2,4,8,16 or 32. The default of D = 2 and MCLK and SMCLK are sourced from DCOCLKDIV thus providing clock frequency of DCOCLK/2.
- The divider (N+1) and divider D define DCOCLK and DCOCLKDIV frequencies. Value N+1 can be set from FLLN bits (in UCSCTL2 register) where N>0. Setting FLLN to 0 will cause FLLN to be 1 to prevent unintentional write.
- The final frequency of DCOCLK and DCOCLKDIV are as follows

$$f_{DCOCLK} = D \times (N + 1) \times (f_{FLLREFCLK} \div n)$$
$$f_{DCOCLKDIV} = (N + 1) \times (f_{FLLREFCLK} \div n)$$

Figure 7 Formula to compute DCOCLK and DCOCLKDIV frequencies

Frequency Locked Loop (FLL):

- The FLL continuously counts up or down a frequency integrator.
- The count is adjusted +1 with the frequency $f_{FLLREFCLK}/n$ (n=1,2,4,8,12 Or 16) or -1 with the frequency $f_{DCOCLK}/[D*(N+1)]$
- Five of the integrator bits (UCSCTL0bits 12 to 8) set the DCO frequency tap. Thirty-two taps are implemented for the DCO, and each is approximately8% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps. For a given DCO bias range setting, time must be allowed for DCO to settle on the proper tap operation. $(n*32)f_{FLLREFCLK}$ cycles are required between taps requiring worst case of $(n*32*32)f_{FLLREFCLK}$ cycles for DCO to settle.

3.2 Changing Processor Clocks: Examples

The following examples illustrate (Figure 8 and Figure 9) how you can change the processor clock frequency by modifying individual bits in the control registers. Please note that these examples only change the clocks and make them visible on external ports (some digital I/O ports have a special function to pass the clocks to the output, so we can observe them from the outside by connecting to oscilloscope). For learning how internal digitally controlled oscillator works read the corresponding user manual.

```
/*******************************************************************************
 *  File:          Lab6_D6.c
 *  Description: MSP430F5529 Demo - FLL, Runs Internal DCO at 2.45MHz
 *               This program demonstrates setting the internal DCO to run at
 *               2.45MHz.
 *  Clocks:      ACLK = 32768Hz,
 *               MCLK = SMCLK = DCO = (74+1) x ACLK = 2457600Hz
 *
 *                MSP430F5529
 *             -----------------
 *         /|\|              XIN|-
 *          | |                 | 32kHz
 *          --|RST          XOUT|-
 *            |                 |
 *            |             P7.7|--> MCLK = 2.45MHz
 *            |                 |
 *            |             P2.2|--> SMCLK = 2.45MHz
 *            |             P1.0|--> ACLK = 32kHz
 *            |                 |
 *
 *  Author:     Aleksandar Milenkovic, milenkovic@computer.og
 *  Date:       September 2010
 *  Modified:   Prawar Poudel, August 2020
 ******************************************************************************/
#include  <msp430.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
  P1DIR |= BIT0;                            // ACLK set out to pins
```

```
    P1SEL |= BIT0;
    P2DIR |= BIT2;                          // SMCLK set out to pins
    P2SEL |= BIT2;
    P7DIR |= BIT7;                          // MCLK set out to pins
    P7SEL |= BIT7;

    UCSCTL3 = SELREF_2;                     // Set DCO FLL reference = REFO
    UCSCTL4 |= SELA_2;                      // Set ACLK = REFO
    UCSCTL0 = 0x0000;                       // Set lowest possible DCOx, MODx

    // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
    do
    {
      UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);
                                            // Clear XT2,XT1,DCO fault flags
      SFRIFG1 &= ~OFIFG;                    // Clear fault flags
    } while (SFRIFG1&OFIFG);                // Test oscillator fault flag

    __bis_SR_register(SCG0);                // Disable the FLL control loop
    UCSCTL1 = DCORSEL_4;                    // Select DCO range for operation
    UCSCTL2 |= 74;                          // Set DCO Multiplier for 2.45MHz
                                            // (N + 1) * FLLRef = Fdco
                                            // (74 + 1) * 32768 = 2.45MHz
    __bic_SR_register(SCG0);                // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // UG for optimization.
    // 32 x 32 x 2.45 MHz / 32,768 Hz = 76600 = MCLK cycles for DCO to settle
    __delay_cycles(76000);

    while(1);
  }

```

Figure 8. Changing DCO to Run at 2.45 MHz using UCS Module (Lab6_D6.c)

```
/*******************************************************************************
 *  File:         Lab6_D7.c
 *  Description: MSP430F5529 Demo - FLL, Runs Internal DCO at 8MHz
 *                This program demonstrates setting the internal DCO to run at
 *                8MHz.
 *  Clocks:       ACLK = 32768Hz,
 *                MCLK = SMCLK = DCO = (121+1) x 2 x ACLK = 7995392Hz
 *
 *                 MSP430F5529
 *             -----------------
 *         /|\|              XIN|-
 *          | |                 | 32kHz
 *          --|RST          XOUT|-
 *            |                 |
 *            |             P7.7|--> MCLK = 8MHz
 *            |                 |
 *            |             P2.2|--> SMCLK = 8MHz
```

```
 *           |              P1.0|--> ACLK = 32kHz
 *           |                  |
 *
 *  Author:     Aleksandar Milenkovic, milenkovic@computer.og
 *  Date:       September 2010
 *  Modified:   Prawar Poudel
 *  Date:       August 2020
 ******************************************************************************/

#include  <msp430.h>

void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;            // Stop watchdog timer

   P1DIR |= BIT0;                       // ACLK set out to pins
   P1SEL |= BIT0;
   P2DIR |= BIT2;                       // SMCLK set out to pins
   P2SEL |= BIT2;
   P7DIR |= BIT7;                       // MCLK set out to pins
   P7SEL |= BIT7;

   UCSCTL3 = SELREF_2;                  // Set DCO FLL reference = REFO
   UCSCTL4 |= SELA_2;                   // Set ACLK = REFO
   UCSCTL0 = 0x0000;                    // Set lowest possible DCOx, MODx

   // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
   do
   {
     UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);
                                             // Clear XT2,XT1,DCO fault flags
     SFRIFG1 &= ~OFIFG;                      // Clear fault flags
   } while (SFRIFG1&OFIFG);                  // Test oscillator fault flag

   __bis_SR_register(SCG0);                  // Disable the FLL control loop
   UCSCTL1 = DCORSEL_5;                      // Select DCO range 8MHz operation
   UCSCTL2 |= 249;                           // Set DCO Multiplier for 8MHz
                                             // (N + 1) * FLLRef = Fdco
                                             // (249 + 1) * 32768 = 8MHz
   __bic_SR_register(SCG0);                  // Enable the FLL control loop

   // Worst-case settling time for the DCO when the DCO range bits have been
   // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
   // UG for optimization.
   // 32 x 32 x 8 MHz / 32,768 Hz = 250000 = MCLK cycles for DCO to settle
   __delay_cycles(250000);
   while(1);                            // Loop in place
}
```

Figure 9. Changing DCO to Run at 8 MHz using UCS Module (Lab6_D7.c)

```
/*******************************************************************************
 *  File:        Lab6_D8.c
 *  Description: MSP430F5529 Demo - FLL, Runs Internal DCO at 1MHz
 *               This program demonstrates setting the internal DCO to run at
 *               8MHz when SW1 is pressed.
 *
 *               A LED will keep blinking at 1Hz (0.5s ON and 0.5s OFF) at
 *               clock running at default frequency of 1Mhz. When SW1 is pressed,
 *               the frequency is changed to ~ 8 Mhz. When SW1 is pressed again,
 *               the frequency is restored to 1Mhz.
 *
 *  Clocks:      ACLK = 32768Hz,
 *               MCLK = SMCLK = DCO = (249+1) x ACLK = 8192000Hz
 *
 * Input:       SW1 (P2.1)
 * Output:      LED2 (P4.7)
 *
 *                MSP430F5529
 *            -----------------
 *        /|\|              XIN|-
 *         | |                 | 32kHz
 *         --|RST          XOUT|-
 *           |                 |
 *     SW1-->|P2.1         P7.7|--> MCLK = 1 or 8MHz
 *    LED2<--|P4.7             |
 *           |             P2.2|--> SMCLK = 1 or 8MHz
 *           |             P1.0|--> ACLK = 32kHz
 *           |                 |
 *
 *  Modified:   Prawar Poudel
 *  Date:       August 2020
 ******************************************************************************/

// mandatory include statement
#include  <msp430.h>

// this function configures the clock sources as follows
// .. use internal REFOCLK for FLL reference clock (UCSCTL3 = SELREF_2)
// .. ACLK is sourced with REFOCLK (UCSCTL4 |= SELA_2)
// .. sets DCO tap to 0 (UCSCTL0 = 0)
// .. sets the modulation bit counter value to 0 (UCSCTL0 = 0)
void configure_clock_sources();
// this function changes the frequency of clock to 8 MHZ
inline void change_clock_freq_8Mhz();
// this function changes the frequency of clock to 8 MHZ
inline void change_clock_freq_1Mhz();


char is8Mhz = 0;

void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;              // Stop watchdog timer

   P1DIR |= BIT0;                         // ACLK set out to pins
```

```
    P1SEL |= BIT0;

    P2DIR |= BIT2;                    // SMCLK set out to pins
    P2SEL |= BIT2;

    P7DIR |= BIT7;                    // MCLK set out to pins
    P7SEL |= BIT7;

    _EINT();                          // enable interrupts
    P2DIR &= ~BIT1;                   // set P2.1 as input (SW1)
    P2REN |= BIT1;                    // enable pull-up resistor
    P2OUT |= BIT1;
    P2IE |= BIT1;                     // enable interrupt at P2.1
    P2IES |= BIT1;                    // enable hi->lo edge for interrupt
    P2IFG &= ~BIT1;                   // clear any errornous interrupt flag

    P4DIR |= BIT7;                    // set P4.7 as output (LED2)

    configure_clock_sources();        // configure the clock sources

    while(1)                          // Loop in place (infinite)
    {
        P4OUT ^= BIT7;                // toggle LED2
        __delay_cycles(500000);       // arbitrary delay of 500ms
    }
}

// this ISR handles the SW1 key press
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
    // let us clear the flag
    P2IFG &= ~BIT1;

    //debouncing section
    __delay_cycles(25000);

    // if SW1 is not pressed, return
    if((P2IN&BIT1)!=0x00)
        return;


    if(is8Mhz==0)
    {
        // if not at 8Mhz, let us change to 8Mhz
        change_clock_freq_8Mhz();
        is8Mhz = 1;
    }else
    {
        // if already in 8Mhz, let us take back to 1Mhz
        change_clock_freq_1Mhz();
        is8Mhz = 0;
    }
}
```

```c
// this function changes the frequency of clock to 8 MHZ
void change_clock_freq_8Mhz()
{
    __bis_SR_register(SCG0);                    // Disable the FLL control loop
    UCSCTL1 = DCORSEL_5;                        // Select DCO range 8MHz operation
    UCSCTL2 = 249;                              // Set DCO Multiplier for 8MHz
                                                // (N + 1) * FLLRef = Fdco
                                                // (249 + 1) * 32768 = 8MHz
    __bic_SR_register(SCG0);                    // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // UG for optimization.
    // 32 x 32 x 8 MHz / 32,768 Hz = 250000 = MCLK cycles for DCO to settle
    __delay_cycles(250000);
}

// this function changes the frequency of clock to 1 MHZ
void change_clock_freq_1Mhz()
{
    __bis_SR_register(SCG0);                    // Disable the FLL control loop
    UCSCTL1 = DCORSEL_3;                        // Select DCO range 1MHz operation
    UCSCTL2 = 32;                               // Set DCO Multiplier for 1MHz
                                                // (N + 1) * FLLRef = Fdco
                                                // (32 + 1) * 32768 = 1MHz
    __bic_SR_register(SCG0);                    // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // UG for optimization.
    // 32 x 32 x 1 MHz / 32,768 Hz = 33792 = MCLK cycles for DCO to settle
    __delay_cycles(33792);
}


// this function configures the clock sources as follows
// .. use internal REFOCLK for FLL reference clock (UCSCTL3 = SELREF_2)
// .. ACLK is sourced with REFOCLK (UCSCTL4 |= SELA_2)
// .. sets DCO tap to 0 (UCSCTL0 = 0)
// .. sets the modulation bit counter value to 0 (UCSCTL0 = 0)
void configure_clock_sources()
{
    UCSCTL3 = SELREF_2;                     // Set DCO FLL reference = REFO
    UCSCTL4 |= SELA_2;                      // Set ACLK = REFO
    UCSCTL0 = 0x0000;                       // Set lowest possible DCOx, MODx

    // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
    do
    {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);   // Clear XT2,XT1,DCO fault flags
    SFRIFG1 &= ~OFIFG;                            // Clear fault flags
    } while (SFRIFG1&OFIFG);                      // Test oscillator fault flag
}
```

Figure 10 Changing Clock Frequency to Observe Change in LED Blinking

4 References

It is crucial that you become familiar with the basics of how digital ports work – how to set their output direction, read from or write to the ports, set interrupts, and set up their special functions. We will be using these features to control hardware and communication between devices throughout this class. Please reference the following material to gain more insight on the device:

- The MSP-EXP430F5529LP board's user guide
- Chapter 5 in the MSP430F5529 user's guide (pages 158-185)
- Chapter 7 in the John H. Davies' *MSP430 Microcontroller Basics*