CPE 325: Embedded Systems Laboratory
Laboratory Tutorial #6:
MSP430 Interrupt ad Clock Subsystem

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

Objective:

This tutorial will introduce the clock module of the MSP430FG4618 device (FLL+), the oscillator sources, and interrupts in C and assembly language. You will learn the following topics:

Using interrupts in C/assembly
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 TI experimenter's 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 MSP430 Experimenter Board hardware, specifically LEDs and switches, using C language. We will redo the same examples using 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} ~ 1s. 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.

```
1  ;-------------------------------------------------------------------------------
2  ;   File:        Lab6_D1.asm
3  ;   Description: The program toggles LEDs periodically.
4  ;                The LEDs are initialized off. An endless loop is entered.
5  ;                A SWDelay1 loop creates 1s delay before toggling the LEDs.
6  ;                LEDs will toggle: off for 1s and on for 1s.
7  ;
8  ;   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
9  ;   Platform:    TI Experimenter's Board
10 ;
11 ;                 MSP430xG461x
12 ;             -----------------
13 ;         /|\|                 |
14 ;          | |                 |
15 ;          --|RST              |
16 ;            |             P2.2|-->LED1(GREEN)
17 ;            |             P2.1|-->LED2(YELLOW)
18 ;
19 ;   Author:  Aleksandar Milenkovic, milenkovic@computer.org
20 ;   Date:    September 14, 2018
```

```
21  ;-------------------------------------------------------------------------------
22              .cdecls C,LIST,"msp430.h"       ; Include device header file
23
24  ;-------------------------------------------------------------------------------
25              .def    RESET                   ; Export program entry-point to
26                                              ; make it known to linker.
27  ;-------------------------------------------------------------------------------
28              .text                           ; Assemble into program memory.
29              .retain                         ; Override ELF conditional linking
30                                              ; and retain current section.
31              .retainrefs                     ; And retain any sections that have
32                                              ; references to current section.
33
34  ;-------------------------------------------------------------------------------
35  RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer
36  StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
37  Setup:      bis.b   #0x06,&P2DIR            ; Set P2.2 and P2.1 to output
38                                              ; direction (0000_0110)
39              bic.b   #0x04,&P2OUT            ; Set P2OUT to 0x0000_0100 (LEDS off)
40  InfLoop:    mov.w   #0xFFFF, R5             ; Software delay (65,535*16cc/2^20 ~ 1s)
41  SWDelay1:   nop                             ; 1cc (total delay is 16 cc)
42              nop
43              nop
44              nop
45              nop
46              nop
47              nop
48              nop
49              nop
50              nop
51              nop
52              nop
53              nop
54              dec.w  R5                       ; 1cc
55              jnz    SWDelay1                 ; 2cc
56              xor.b  #0x06, P2OUT             ; toggle LEDs
57              jmp    InfLoop                  ; goto InfLoop
58
59  ;-------------------------------------------------------------------------------
60  ; Stack Pointer definition
61  ;-------------------------------------------------------------------------------
62              .global __STACK_END
63              .sect   .stack
64
65  ;-------------------------------------------------------------------------------
66  ; Interrupt Vectors
67  ;-------------------------------------------------------------------------------
68              .sect   ".reset"                ; MSP430 RESET Vector
69              .short  RESET
70              .end
```

Figure 1. Toggling the LEDs in Assembly Language

1.2 Interfacing Switches in Assembly Language (Polling)

Figure 2 shows assembly program that interfaces SW1 and LED1. SW1 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.

```
1   ;-------------------------------------------------------------------------------
2   ;   File:        Lab6_D2.asm
3   ;   Description: The program demonstrates Press/Release using SW1 and LED1.
4   ;                LED1 is initialized off.
5   ;                When SW1 press is detected, a software delay of 20 ms
6   ;                is used to implement debouncing. The switch is checked
7   ;                again, and if on, LED1 is turned on until SW1 is released.
8   ;
9   ;   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
10  ;   Platform:    TI Experimenter's Board
11  ;
12  ;                 MSP430xG461x
13  ;              -----------------
14  ;          /|\|                 |
15  ;           | |                 |
16  ;           --|RST              |
17  ;             |             P2.2|-->LED1(GREEN)
18  ;             |             P1.0|<--SW1
19  ;
20  ;   Author:  Aleksandar Milenkovic, milenkovic@computer.org
21  ;   Date:    September 14, 2018
22  ;-------------------------------------------------------------------------------
23
24              .cdecls C,LIST,"msp430.h"       ; Include device header file
25
26  ;-------------------------------------------------------------------------------
27              .def    RESET                   ; Export program entry-point to
28                                              ; make it known to linker.
29  ;-------------------------------------------------------------------------------
30              .text                           ; Assemble into program memory.
31              .retain                         ; Override ELF conditional linking
32                                              ; and retain current section.
33              .retainrefs                     ; And retain any sections that have
34                                              ; references to current section.
35
36  ;-------------------------------------------------------------------------------
37  RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer
38  StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
39  ;-------------------------------------------------------------------------------
40  SetupP2:
41              bis.b   #004h, &P2DIR           ; Set P2.2 to output
42                                              ; direction (0000_0100)
43              bic.b   #004h, &P2OUT           ; Set P2OUT to 0x0000_0100 (ensure
```

```
44                                                  ; LED1 is off)
45  ChkSW1:     bit.b   #01h, &P1IN                 ; Check if SW1 is pressed
46                                                  ; (0000_0001 on P1IN)
47              jnz     ChkSW1                      ; If not zero, SW is not pressed
48                                                  ; loop and check again
49  Debounce:
50              mov.w   #2000, R15                  ; Set to (2000 * 10 cc = 20,000 cc)
51  SWD20ms:    dec.w   R15                         ; Decrement R15
52              nop
53              nop
54              nop
55              nop
56              nop
57              nop
58              nop
59              jnz     SWD20ms                     ; Delay over?
60              bit.b   #00000001b, &P1IN           ; Verify SW1 is still pressed
61              jnz     ChkSW1                      ; If not, wait for SW1 press
62
63  LEDon:      bis.b   #004h, &P2OUT               ; Turn on LED1
64  SW1wait:    bit.b   #001h, &P1IN                ; Test SW1
65              jz      SW1wait                     ; Wait until SW1 is released
66              bic.b   #004, &P2OUT                ; Turn off LED1
67              jmp     ChkSW1                      ; Loop to beginning
68
69  ;-------------------------------------------------------------------------------
70  ; Stack Pointer definition
71  ;-------------------------------------------------------------------------------
72              .global __STACK_END
73              .sect   .stack
74
75  ;-------------------------------------------------------------------------------
76  ; Interrupt Vectors
77  ;-------------------------------------------------------------------------------
78              .sect   ".reset"                    ; MSP430 RESET Vector
79              .short  RESET
80              .end
```

Figure 2. Turn on LED1 when SW1 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 instruction 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 transfer 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 bits on the desired port
- Specify whether the interrupt is called 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 BIT0 of Port1 (P1IE=0x0000_0001b). 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.BIT0. The interrupt service routine starts at label SW1_ISR. The state of the input is checked; if P1IN.BIT0 is not a logic 0 we exit the ISR; otherwise, de-bouncing is performed. If SW1 is still pressed after 20 ms, LED1 is turned on. The program then waits for SW1 to be released. Note lines 88 and 89 that initialize the IVT entry 20 reserved for Port 1.

```
1  ;-------------------------------------------------------------------------------
2  ;   File:        Lab6_D3.asm
3  ;   Description: The program demonstrates Press/Release using SW1 and LED1.
4  ;                LED1 is initialized off. The main program enables interrupts
5  ;                from P1.BIT0 (SW1) and remains in an infinite loop doing nothing.
6  ;                P1_ISR implements debouncing and waits for a SW1 to be released.
7  ;
8  ;   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO = 2^20=1,048,576 Hz
9  ;   Platform:    TI Experimenter's Board
10 ;
11 ;                 MSP430xG461x
12 ;              -----------------
13 ;          /|\|                 |
14 ;           | |                 |
15 ;           --|RST              |
16 ;             |                 |
17 ;             |                 |
18 ;
19 ;   Author:  Aleksandar Milenkovic, milenkovic@computer.org
20 ;   Date:    September 14, 2018
21 ;-------------------------------------------------------------------------------
22             .cdecls C,LIST,"msp430.h"       ; Include device header file
23
24 ;-------------------------------------------------------------------------------
25             .def    RESET                   ; Export program entry-point to
26                                             ; make it known to linker.
27             .def    SW1_ISR
28 ;-------------------------------------------------------------------------------
29             .text                           ; Assemble into program memory.
30             .retain                         ; Override ELF conditional linking
31                                             ; and retain current section.
32             .retainrefs                     ; And retain any sections that have
33                                             ; references to current section.
34
35 ;-------------------------------------------------------------------------------
36 RESET:      mov.w   #__STACK_END, SP        ; Initialize stack pointer
37 StopWDT:    mov.w   #WDTPW|WDTHOLD, &WDTCTL ; Stop watchdog timer
```

```
38  ;-------------------------------------------------------------------------------
39  Setup:
40              bis.b   #004h, &P2DIR           ; Set P2.2 to output
41                                              ; direction (0000_0100)
42              bic.b   #004h, &P2OUT           ; Set P2OUT to 0x0000_0100
43                                              ; (ensure LED1 is off)
44              bis.w   #GIE, SR                ; Enable Global Interrupts
45              bis.b   #001h, &P1IE            ; Enable Port 1 interrupt from bit 0
46              bis.b   #001h, &P1IES           ; Set interrupt to call from hi to low
47              bic.b   #001h, &P1IFG           ; Clear interrupt flag
48  InfLoop:
49              jmp     $                       ; Loop here until interrupt
50
51  ;-------------------------------------------------------------------------------
52  ; P1_0 (SW1) interrupt service routine (ISR)
53  ;-------------------------------------------------------------------------------
54  SW1_ISR:
55              bic.b   #001h, &P1IFG           ; Clear interrupt flag
56  ChkSw1:     bit.b   #01h, &P1IN             ; Check if SW1 is pressed
57                                              ; (0000_0001 on P1IN)
58              jnz     LExit                   ; If not zero, SW is not pressed
59                                              ; loop and check again
60  Debounce:   mov.w   #2000, R15              ; Set to (2000 * 10 cc )
61  SWD20ms:    dec.w   R15                     ; Decrement R15
62              nop
63              nop
64              nop
65              nop
66              nop
67              nop
68              nop
69              jnz     SWD20ms                 ; Delay over?
70              bit.b   #00000001b,&P1IN        ; Verify SW1 is still pressed
71              jnz     LExit                   ; If not, wait for SW1 press
72  LEDon:      bis.b   #004h,&P2OUT            ; Turn on LED1
73  SW1wait:    bit.b   #001h,&P1IN             ; Test SW1
74              jz      SW1wait                 ; Wait until SW1 is released
75              bic.b   #004,&P2OUT             ; Turn off LED1
76  LExit:      reti                            ; Return from interrupt
77  ;-------------------------------------------------------------------------------
78  ; Stack Pointer definition
79  ;-------------------------------------------------------------------------------
80              .global __STACK_END
81              .sect   .stack
82
83  ;-------------------------------------------------------------------------------
84  ; Interrupt Vectors
85  ;-------------------------------------------------------------------------------
86              .sect   ".reset"                ; MSP430 RESET Vector
87              .short  RESET
88              .sect   ".int20"                ;P1.x Vector
89              .short  SW1_ISR
90              .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 SW1 is pressed and turns LED1 off when SW1 is released. The main configures and initializes ports, configures interrupts, and enters an infinite loop where the program waits for SW1 to be released to turn off LED1. P1_ISR is entered upon detection of the switch press; the code inside clears P1.IFG0 and turns on LED1. Please not C convention to indicate that Port1_ISR corresponds to PORT1_VECTOR in the interrupt vector table.

```
1  /*******************************************************************************
2   *   File:        Lab6_D4.c
3   *   Description: The program detects when SW1 is pressed and turns on LED1.
4   *                LED1 is kept on as long as SW1 is pressed.
5   *                P1_ISR is used to detect when SW1 is pressed.
6   *                Main program polls SW1 and turns off when a release is detected.
7   *   Board:       MSP430FG461x/F20xx Experimenter Board
8   *   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
9   *
10  *                 MSP430FG461x
11  *            +-----------------+
12  *            |                 |
13  *            |                 |
14  *            |                 |
15  *            |                 |
16  *            |            P2.2|--> LED1
17  *            |            P1.0|<-- SW1
18  *
19  *   Author: Aleksandar Milenkovic, milenkovic@computer.org
20  *   Date:   September 2010
21  *******************************************************************************/
22  #include  <msp430.h>
23
24  #define   SW1 BIT0&P1IN             // SW1 is P1IN&BIT0
25
26  void main(void)
27  {
28      WDTCTL = WDTPW+WDTHOLD;         // Stop WDT
29      P2DIR |= BIT2;                  // Set LED1 as output
30      P2OUT = 0x00;                   // clear LED1 status
31      _EINT();                        // enable interrupts
32      P1IE |= BIT0;                   // P1.0 interrupt enabled
33      P1IES |= BIT0;                  // P1.0 hi/low edge
34      P1IFG &= ~BIT0;                 // P1.0 IFG cleared
35      for(;;) {
36          while((SW1) == 0);          // Wait until SW1 is released
37          P2OUT &= ~BIT2;             // LED1 is turned off
38      }
39  }
40  // Port 1 interrupt service routine
41  #pragma vector = PORT1_VECTOR
42  __interrupt void Port1_ISR (void)
43  {
44      P2OUT |= BIT2;                  // LED1 is turned ON
```

```
45      P1IFG &= ~BIT0;                 // P1.0 IFG cleared
46  }
```

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 SW1pressed 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.BIT0 (SW1 is pressed). In that case, the ISR turns on LED1, sets the SW1pressed and configures P1IES to trigger an interrupt when a rising edge is detected on P1IN.BIT0. 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.

```
1   /*****************************************************************************
2    *   File:        Lab6_D5.c
3    *   Description: The program detects when SW1 is pressed and turns on LED1.
4    *                LED1 is kept on as long as SW1 is pressed.
5    *                P1_ISR is used to detect both SW1 presses and releases.
6    *   Board:       MSP430FG461x/F20xx Experimenter Board
7    *   Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
8    *
9    *                 MSP430FG461x
10   *             +-----------------+
11   *             |                 |
12   *             |                 |
13   *             |                 |
14   *             |                 |
15   *             |           P2.2|--> LED1
16   *             |           P1.0|<-- SW1
17   *
18   *   Author: Aleksandar Milenkovic, milenkovic@computer.org
19   *   Date:   September 2010
20  *****************************************************************************/
21  #include  <msp430.h>
22
23  unsigned char SW1pressed = 0;       // SW1 status (0 not pressed, 1 pressed)
24
25  void main(void) {
26      WDTCTL = WDTPW+WDTHOLD;         // Stop WDT
27      P2DIR |= BIT2;                  // Set LED1 as output
28      P2OUT = 0x00;                   // Clear LED1 status
29      SW1pressed = 0;
30      _EINT();                        // Enable interrupts
31      P1IE |= BIT0;                   // P1IE.BIT0 interrupt enabled
32      P1IES |= BIT0;                  // P1IES.BIT0 hi/low edge
33      P1IFG &= ~BIT0;                 // P1IFG.BIT0 is cleared
34      _BIS_SR(LPM0_bits + GIE);       // Enter LPM0(CPU is off); Enable interrupts
35  }
36
```

```
37  // Port 1 interrupt service routine
38  #pragma vector = PORT1_VECTOR
39  __interrupt void Port1_ISR (void) {
40      if (SW1pressed == 0) {
41          SW1pressed = 1;
42          P2OUT |= BIT2;                // LED1 is turned ON
43          P1IFG &= ~BIT0;               // P1IFG.BIT0 is cleared
44          P1IES &= ~BIT0;               // P1IES.BIT0 low/high edge
45      } else if (SW1pressed == 1) {
46          SW1pressed = 0;
47          P2OUT &= ~BIT2;                // LED1 is turned ON
48          P1IFG &= ~BIT0;                // P1IFG.BIT0 is cleared
49          P1IES |= BIT0;                 // P1IES.BIT0 hi/low edge
50      }
51  }
```

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 FLL+ clock module used in the MSP430FG4618 device.

3.1 FLL+

The more recent MSP430 devices use an on-chip system clock called the FLL+ (frequency locked loop). This module can be programmed to provide a range of core clock frequencies, which are frequency-locked to an external crystal (usually a 32,768 Hz wrist-watch type crystal which has good stability). A frequency-lock, or **frequency-locked loop** (FLL), is an electronic control system that generates a signal that is locked to the frequency of an input or "reference" signal. This circuit compares the frequency of a controlled oscillator (e.g., from an on-chip digitally-controlled oscillator) to the reference (e.g., external crystal), automatically raising or lowering the frequency of the oscillator until its frequency (but not necessarily its phase) is matched to that of the reference. Figure 6 shows the block diagram of the FLL+ clock module. The module supports two or three clock sources as follows.

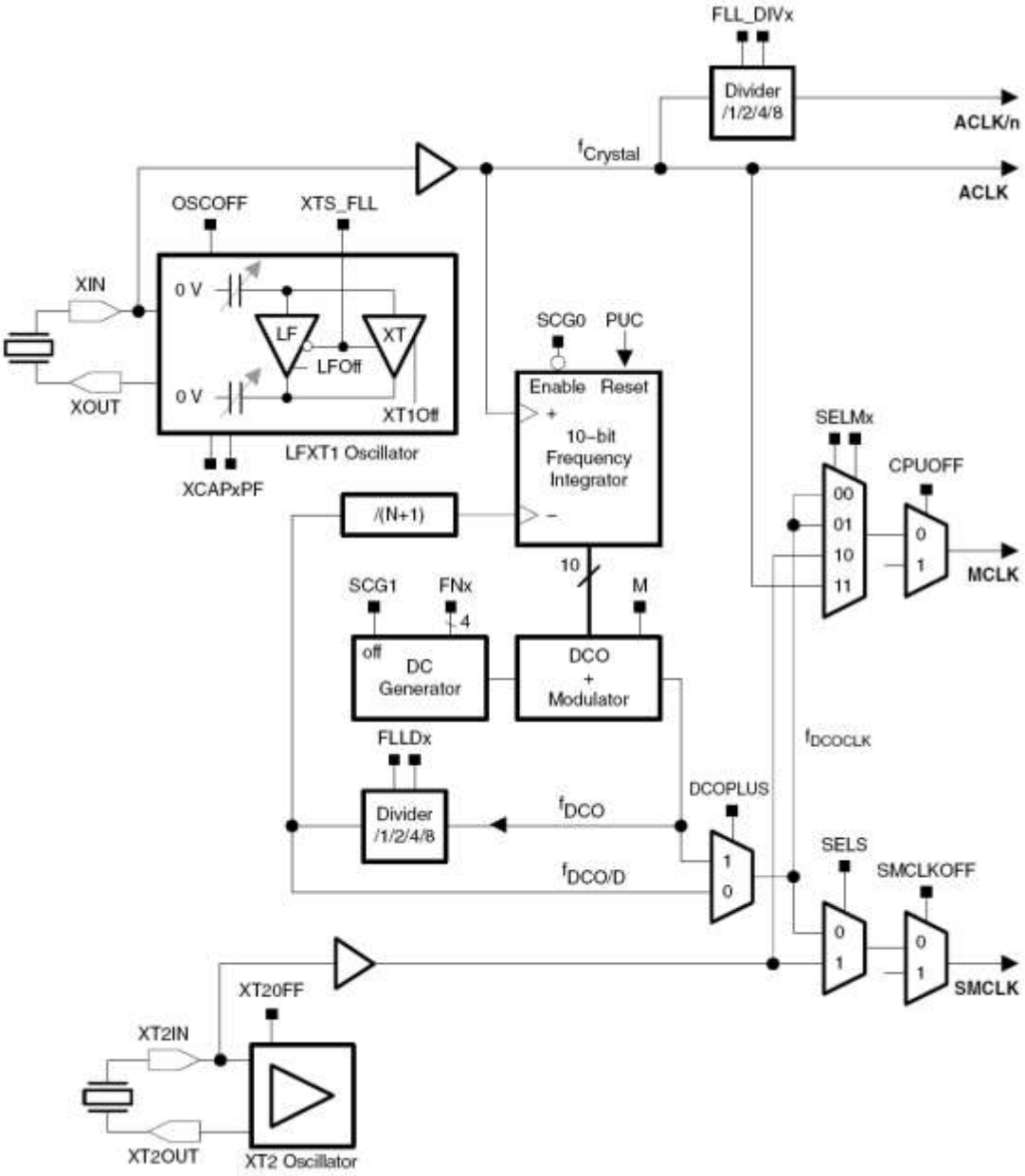


Figure 6. Block diagram of FLL+ module in MSP430FG461x devices

LFXT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals or standard crystals or resonators in the 450-kHz to 8-MHz range. The LFXT1 oscillator supports ultra-low current consumption using a 32,768-Hz watch crystal in LF mode (control bit XTS_FLL is cleared, i.e., XTS_FLL = 0). A watch crystal connects to XIN and XOUT without any external components. The LFXT1 oscillator also supports high-speed crystals or resonators when in HF mode (XTS_FLL = 1). The high-speed crystal or resonator connects to XIN and XOUT.

XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range. XT2 sources XT2CLK and its

characteristics are identical to LFXT1 in HF mode, except XT2 does not have internal load capacitors. The required load capacitance for the high-frequency crystal or resonator must be provided externally. The XT2OFF bit disables the XT2 oscillator if XT2CLK is unused for MCLK (SELMx ≠ 2 or CPUOFF = 1) and SMCLK (SELS = 0 or SMCLKOFF = 1).

DCOCLK: Internal digitally controlled oscillator (DCO) with RC-type characteristics, stabilized by the FLL. The DCO is an integrated ring oscillator with RC-type characteristics. The DCO frequency is stabilized by the FLL to a multiple of ACLK as defined by N, the lowest 7 bits of the SCFQCTL register. The DCOPLUS bit sets the f_{DCOCLK} frequency to f_{DCO} or $f_{DCO/D}$. The FLLDx bits configure the divider, D, to 1, 2, 4, or 8. By default, DCOPLUS = 0 and D = 2, providing a clock frequency of $f_{DCO/2}$ on f_{DCOCLK}. The multiplier (N+1) and D set the frequency of DCOCLK.
DCOPLUS = 0: $f_{DCOCLK} = (N + 1) \times f_{ACLK}$
DCOPLUS = 1: $f_{DCOCLK} = D \times (N + 1) \times f_{ACLK}$
Four clock signals are available from the FLL+ module, as follows.

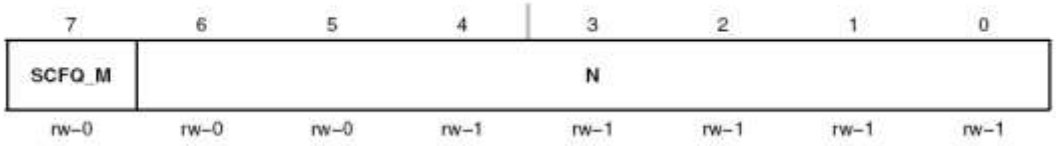
- ACLK: Auxiliary clock. The ACLK is software selectable as LFXT1CLK or VLOCLK as clock source. ACLK is software selectable for individual peripheral modules.
- ACLK/n: Buffered output of the ACLK. The ACLK/n is ACLK divided by 1,2,4, or 8 and used externally only.
- MCLK: Master clock. MCLK is software selectable as LFXT1CLK, VLOCLK, XT2CLK (if available), or DCOCLK. MCLK can be divided by 1, 2, 4, or 8 within the FLL block. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. SMCLK is software selectable as XT2CLK (if available) or DCOCLK. SMCLK is software selectable for individual peripheral modules.

The FLL+ clock module registers are described below. The SCFQCTL, SCFI0/1 and FLL_CTL0/1 registers govern the FLL+ clock module operation and they can be reconfigured by software at any time during program execution.

Register	**Short Form**	**Register Type**	**Address**	**Initial State**
System clock control	SCFQCTL	Read/write	052h	01Fh with PUC
System clock frequency integrator 0	SCFI0	Read/write	050h	040h with PUC
System clock frequency integrator 1	SCIF1	Read/write	051h	Reset with PUC
FLL+ control register 0	FLL_CTL0	Read/write	053h	003h with PUC
FLL+ control register 1	FLL_CTL1	Read/write	054h	Reset with PUC

The format of the SCFQCTL register is given in Figure 7. Its initial value is 1Fh, which means that modulation is enabled (SCFQ_M=0, and N=001_1111 = 31).

SCFQCTL, System Clock Control Register



SCFQ_M	Bit 7	Modulation. This enables or disables modulation 0 Modulation enabled 1 Modulation disabled
N	Bits 6-0	Multiplier. These bits set the multiplier value for the DCO. N must be > 0 or unpredictable operation will result. When DCOPLUS=0: $f_{DCOCLK} = (N + 1) \cdot f_{crystal}$ When DCOPLUS=1: $f_{DCOCLK} = D \times (N + 1) \cdot f_{crystal}$

Figure 7. Format of the SCFQCTL register

The format of the SCFI0 and SCFI1 registers is given in Figure 8 and Figure 9, respectively. The SCFI0 initial value is 0x40, which means FLLDx=00, FN_x=1000, MODx(LSB)=00. The SCFI1 initial value is 0x00, meaning that DCOx=00000, and MODx(MSB)=000. Similarly, Figure 10 and Figure 11 show the formats of the registers FLL_CTL0 and FLL_CTL1. Based on the registers' initial values we can determine the clock conditions after the PUC signal. Analyze the schematic of the TI experimenter's board. Locate the input pins XIN and XOUT. What is connected to these pins? Analyze the block diagram in Figure 4 and determine configuration of each resource. List the different sources of inputs for ACLK, SMCLK, and MCLK clocks. How can we choose them? What is the default clock frequency on ACLK, DCOCLK, MCLK, and SMCLK? From Figure 6, analyze what is the maximum possible value of N and DCOCLK clock frequency? Show your work.

SCFI0, System Clock Frequency Integrator Register 0

7	6	5	4	3	2	1	0
FLLDx		FN_x				MODx (LSBs)	
rw–0	rw–1	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

FLLDx	Bits 7-6	FLL+ loop divider. These bits divide f_{DCOCLK} in the FLL+ feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits. 00 /1 01 /2 10 /4 11 /8
FN_x	Bits 5-2	DCO Range Control. These bits select the f_{DCO} operating range. 0000 0.65 - 6.1 MHz 0001 1.3 - 12.1 MHz 001x 2 - 17.9 MHz 01xx 2.8 - 26.6 MHz 1xxx 4.2 - 46 MHz
MODx	Bits 1–0	Least significant modulator bits. Bit 0 is the modulator LSB. These bits affect the modulator pattern. All MODx bits are modified automatically by the FLL+.

Figure 8. Format of the SCFI0 Register

SCFI1, System Clock Frequency Integrator Register 1

7	6	5	4	3	2	1	0
DCOx					MODx (MSBs)		
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

DCOx	Bits 7-3	These bits select the DCO tap and are modified automatically by the FLL+.
MODx	Bit 2	Most significant modulator bits. Bit 2 is the modulator MSB. These bits affect the modulator pattern. All MODx bits are modified automatically by the FLL+.

Figure 9. Format of the SCFI1 Register

FLL_CTL0, FLL+ Control Register 0

7	6	5	4	3	2	1	0
DCOPLUS	XTS_FLL	XCAPxPF		XT2OF†	XT1OF	LFOF	DCOF
rw–0	rw–0	rw–0	rw–0	r–0	r–0	r–(1)	r–1

† Not present in MSP430x41x, MSP430x42x devices

DCOPLUS Bit 7 DCO output pre-divider. This bit selects if the DCO output is pre-divided before sourcing MCLK or SMCLK. The division rate is selected with the FLL_DIV bits
0 DCO output is divided
1 DCO output is not divided

XTS_FLL Bit 6 LFTX1 mode select
0 Low frequency mode
1 High frequency mode

XCAPxPF Bits 5–4 Oscillator capacitor selection. These bits select the effective capacitance seen by the LFXT1 crystal or resonator. Should be set to 00 if the high frequency mode is selected for LFXT1 with XTS_FLL = 1.
00 ~1 pF
01 ~6 pF
10 ~8 pF
11 ~10 pF

XT2OF Bit 3 XT2 oscillator fault. Not present in MSP430x41x, MSP430x42x devices.
0 No fault condition present
1 Fault condition present

XT1OF Bit 2 LFXT1 high frequency oscillator fault
0 No fault condition present
1 Fault condition present

LFOF Bit 1 LFXT1 low frequency oscillator fault
0 No fault condition present
1 Fault condition present

DCOF Bit 0 DCO oscillator fault
0 No fault condition present
1 Fault condition present

Figure 10. Format of the FLL_CTL0 register. The initial value is 0x03=> DCOPLUS=0, XT1OFF=0, LFOF=1, and DCOF=1.

FLL_CTL1, FLL+ Control Register 1

7	6	5	4	3	2	1	0
LFXT1DIG‡	SMCLK OFF†	XT2OFF†	SELMx†		SELS†	FLL_DIVx	
rw–0	rw–0	rw–(1)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

† Not present in MSP430x41x, MSP430x42x devices.
‡ Only supported by MSP430xG46x and MSP430x47x devices. Otherwise unused.

LFXT1DIG	Bit 7	Select digital external clock source. This bit enables the input of an external digital clock signal on XIN in low frequency mode (XTS_FLL = 0). Only supported in MSP430xG46x and MSP430x47x devices. 0 Crystal input selected. 1 Digital clock input selected.
SMCLKOFF	Bit 6	SMCLK off. This bit turns off SMCLK. Not present in MSP430x41x, MSPx42x devices. 0 SMCLK is on 1 SMCLK is off
XT2OFF	Bit 5	XT2 off. This bit turns off the XT2 oscillator. Not present in MSP430x41x, MSPx42x devices. 0 XT2 is on 1 XT2 is off if it is not used for MCLK or SMCLK.
SELMx	Bits 4–3	Select MCLK. These bits select the MCLK source. Not present in MSP430x41x, MSP430x42x devices. 00 DCOCLK 01 DCOCLK 10 XT2CLK 11 LFXT1CLK
SELS	Bit 2	Select SMCLK. This bit selects the SMCLK source. Not present in MSP430x41x, MSP430x42x devices. 0 DCOCLK 1 XT2CLK
FLL_DIVx	Bits 1–0	ACLK divider 00 /1 01 /2 10 /4 11 /8

Figure 11. Format of the FLL_CTL1 register.

3.2 Programming FLL+ Clocks: Examples

The following examples illustrate (Figure 12 and Figure 13) 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: MSP430xG46x Demo - FLL+, Runs Internal DCO at 2.45MHz
 *               This program demonstrates setting the internal DCO to run at
 *               2.45MHz with auto-calibration by the FLL+ circuitry.
 *               ACLK = LFXT1 = 32768Hz,
 *               MCLK = SMCLK = DCO = (74+1) x ACLK = 2457600Hz
 *               An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                MSP430xG461x
 *            -----------------
 *        /|\|              XIN|-
 *         | |                 | 32kHz
 *         --|RST          XOUT|-
 *           |                 |
 *           |             P1.1|--> MCLK = 2.45MHz
 *           |                 |
 *           |             P1.4|--> SMCLK = 2.45MHz
 *           |             P1.5|--> ACLK = 32kHz
 *           |                 |
 *
 *  Author: Aleksandar Milenkovic, milenkovic@computer.og
 *  Date:   September 2010
 *******************************************************************************/
#include  <msp430.h>

void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;      // Stop watchdog timer
   FLL_CTL0 |= XCAP18PF;          // Set load capacitance for xtal
   SCFI0 |= FN_2;                 // DCO range control
   SCFQCTL = 74;                  // (74+1) x 32768 = 2.45MHz
   P1DIR |= 0x32;                 // P1.1, P1.4 & P1.5 to output direction
   P1SEL |= 0x32;                 // P1.1, P1.4 & P1.5 to output MCLK, SMCLK & ACLK

   while(1);                      // Loop in place
}

```

Figure 12. Changing DCO to Run at 2.45 MHz using FLL+ Module (Lab6_D6.c)

```
/*******************************************************************************
 *  File:        Lab6_D7.c
 *  Description: MSP430xG46x Demo - FLL+, Runs Internal DCO at 8MHz
 *               This program demonstrates setting the internal DCO to run at
 *               8MHz with auto-calibration by the FLL+ circuitry.
 *  Clocks:      ACLK = LFXT1 = 32768Hz,
 *               MCLK = SMCLK = DCO = (121+1) x 2 x ACLK = 7995392Hz
 *               An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                MSP430xG461x
 *            -----------------
 *        /|\|              XIN|-
 *         | |                 | 32kHz
```

```
 *         --|RST          XOUT|-
 *           |                 |
 *           |             P1.1|--> MCLK = 8MHz
 *           |                 |
 *           |             P1.4|--> SMCLK = 8MHz
 *           |             P1.5|--> ACLK = 32kHz
 *           |                 |
 *
 *  Author: Aleksandar Milenkovic, milenkovic@computer.og
 *  Date:   September 2010
 ******************************************************************************/

#include  <msp430.h>

void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
   FLL_CTL0 |= DCOPLUS + XCAP18PF; // DCO+ set, freq = xtal x D x N+1
   SCFI0 |= FN_4 + FLLD_2;         // DCO range control
   SCFQCTL = 121;                  // (121+1) x 32768 x 2 = 7.99 MHz
   P1DIR |= 0x32;                  // P1.1, P1.4 & P1.5 to output direction
   P1SEL |= 0x32;                  // P1.1, P1.4 & P1.5 to output MCLK, SMCLK & ACLK
   while(1);                       // Loop in place
}
```

Figure 13. Changing DCO to Run at 8 MHz using FLL+ Module (Lab6_D7.c)

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 MSP430 Experimenter's Board hardware schematic
- Chapter 11 in the MSP430FG4618 user's guide (pages 407-414)
- Chapter 7 in the John H. Davies' *MSP430 Microcontroller Basics*