1. CPE 325: Laboratory Assignment #6 Interrupts in C, MSP430 Clock Subsystem

Objectives: This tutorial will introduce the clock module of the MSP4304618 device (FLL+), the oscillator sources, and interrupts in C. You will learn the following topics:

- C programming
- Interrupts in C
- Clock subsystem and clock configuration
- Working with the TI experimenter's board

Note: All previous tutorials are required for successful completion of this lab, especially, the tutorials introducing the TI experimenter's board and the IAR software development environment.

1.1. Intro to C

Up to this point, all of the hardware programming has been performed using assembly language. Assembly language is beneficial because it allows direct control of the machine instructions for the device. Programming in assembly gives insight to how the device works, and it can also be necessary for occasions where high control of machine processes is needed. With modern compilers, however, using C to program microcontrollers can save large amounts of time, especially when creating complicated programs. These new compilers offer efficient converting of code from C to machine instructions. From this point on in the course, we will be using C for our projects.

In the previous lab handout, we learned how to interface with the MSP430 Experimenter Board hardware using assembly code. We will redo the same examples in C to demonstrate how the exact same problems would be approached using the C language. If you need to review the process for creating C projects with the IAR Workbench IDE, please refer to lab handout #1.

1.1.1. Blink an LED Using C Programming Language

In this section we repeat the previous lab's example by writing code that blinks LED1 and LED2 alternatively at a rate of 1 Hz. The code first sets up the clock (as will be discussed further in this lab), and then the P2 output is configured. At that point, an infinite loop is entered which toggles the LED and set a delay.

```
/*****************************************************************************
;   TI Experimenter board demo, blinking leds LED1 and LED2 (msp430FG4618)
;
;   Description: Toggle P2.1 and P2.2 by xoring P2.1 and P2.2 inside a loop.
;                The leds are connected to P2.1 and P2.2 and are on when
;                P2.1=1 and P2.2=1;
;                The LEDs are initialized P2.1 to be off, and P2.2 to be on;
;                ACLK = 32.768kHz, MCLK = SMCLK = default DCO
;
;                 MSP430xG461x
;             -----------------
;         /|\|                 |
;          | |                 |
;          --|RST              |
;            |             P2.2|-->LED1(GREEN)
;            |             P2.1|-->LED2(YELLOW)
;
;   Alex Milenkovich, milenkovic@computer.org
;*****************************************************************************/
#include  <msp430xG46x.h>
void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;// Stop watchdog timer
   P2DIR |= 0x06;                 // Set P2.1 and P2.2 to output direction
(0000_0110)
   P2OUT = 0x02;              // Set P2OUT to 0000_0010b (LED2 is ON, LED1 is
OFF)
     for (;;) {
     unsigned int i;
     P2OUT ^= 0x06;           // Toggle P2.1 and P2.2 using exclusive-OR
     i = 50000;             // Delay
     do (i--);
     while (i != 0);
   }
}
```

Figure 1. C program that repeatedly blinks an LED (Lab6_D1.C)

1.1.2. Switch Interfacing and Interrupts in C

In the previous lab, we created assembly code that turned on LED1 as long as SW1 was depressed. We first approached the problem by testing the P1 input bits to see if the switch was pressed, then we introduced a P1 interrupt to perform the same task. Here we repeat those same examples using C code. In figure 2 is shown the C program that repeatedly checks the switch status before lighting the LED.

```
/*****************************************************************************
;   MSP430xG46x Demo - Interfacing Switch
;   Description: This program will keep LED1 ON as long as the switch SW1 is
;   pressed and turn it OFF as soon as the switch is released. This is
;   achieved by checking conditions for the switch state (pressed/released).
;   ACLK = 32.768kHz, MCLK = SMCLK = 1 MHz
;                 MSP430xG461x
;             -----------------
;         /|\|                 |
;          | |                 |
```

```
;           --|RST          P1.0|-->SW1
;             |                 |
;             |             P2.2|-->LED1
;   Alex Milenkovich, milenkovic@computer.org
;   Max Avula, ma0004@uah.edu
;*******************************************************************************/
#include  <msp430xG46x.h>

#define SW1 BIT0&P1IN                           // B1 - P1.0 switch SW1

void main(void)
{
    int i;
    P2DIR |= 0x04;                              // Set LED1 as output
    for(;;) {
        if ((SW1) == 0)                         // if SW1 is pressed
        {
            for(i=1000; i > 0; i--);            // Make sure it is indeed pressed
            if((SW1) == 0)
            {
                P2OUT |= 0x04;                  // LED1 stays ON
            }
            while((SW1)==0);                    // Wait until SW1 is released
                P2OUT &= ~0x04;                 // LED1 stays OFF
        }
    }
}
```

Figure 2. LED1 controlled by SW1 (Lab6_D2.C).

Figure 3 shows an interrupt driven program that turns on LED1 when the switch SW1 is kept pressed. To do so, an interrupt is generated and the program now enters the interrupt service routine section of the code and turns ON LED1. When the button is released, the LED1 is turned off.

```
/*******************************************************************************
;   MSP430xG46x Demo - Interrupt driven interfacing Switch
;   Description: This program will keep LED1 ON as long as the switch SW1 is
;     pressed and turn it OFF as soon as the switch is released. This is
;     achieved by checking conditions for the switch state (pressed/released).
;     When the switch is pressed, a port 1 interrupt is generated and the
program
;     executes the code in the interrupt vector to turn ON LED1. LED1 stays on
;     until the switch is released. The release of the switch clears the
interrupt
;     vector and LED1 is turned OFF immediately.
;
;   ACLK = 32.768kHz, MCLK = SMCLK = 1 MHz
;                 MSP430xG461x
;             -----------------
;         /|\|                 |
;          | |                 |
;          --|RST          P1.0|-->SW1
;            |                 |
;            |             P2.2|-->LED1
;   Alex Milenkovich, milenkovic@computer.org
```

```
;   Max Avula, ma0004@uah.edu
;*******************************************************************************/
#include  <msp430xG46x.h>

#define SW1 BIT0&P1IN              // B1 - P1.0 switch SW1

void main(void)
{
      WDTCTL = WDTPW+WDTHOLD;      // Stop WDT
      P2DIR |= 0x04;           // Set LED1 as output
      P2OUT = 0x00;                  // clear LED1 status
      _EINT();                     // enable interrupts
      P1IE |= BIT0;                // P1.0 interrupt enabled
      P1IES |= BIT0;               // P1.0 hi/low edge
      P1IFG &= ~BIT0;              // P1.0 IFG cleared
      for(;;)
      {
          while((SW1)== 0);    // Wait until SW1 is released
          P2OUT &= ~0x04;             // LED1 stays OFF
      }
}

// Port 1 interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port1_ISR (void)
{
      P2OUT |= 0x04;       // LED1 stays ON
      P1IFG &= ~BIT0;        // P1.0 IFG cleared
}
```

Figure 3. LED1 controlled by SW1 using Port 1 interrupt service routine (Lab6_D3.C)

In the above code, the #pragma directive specifies a compiler-specific statement. Here, it is defining the ISR vector. Note that the PORT1_VECTOR is identical to the name that was used in the assembly code program. That name is specific to the port 1 interrupt vector. The function name for the ISR below that, however, is unimportant.

1.2. Clock Module

In the previous tutorial 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 MSP430F4618 device.

1.2.1. 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 KHz 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 4 shows the block diagram of the FLL+ clock module. The module supports two or three clock sources as follows.

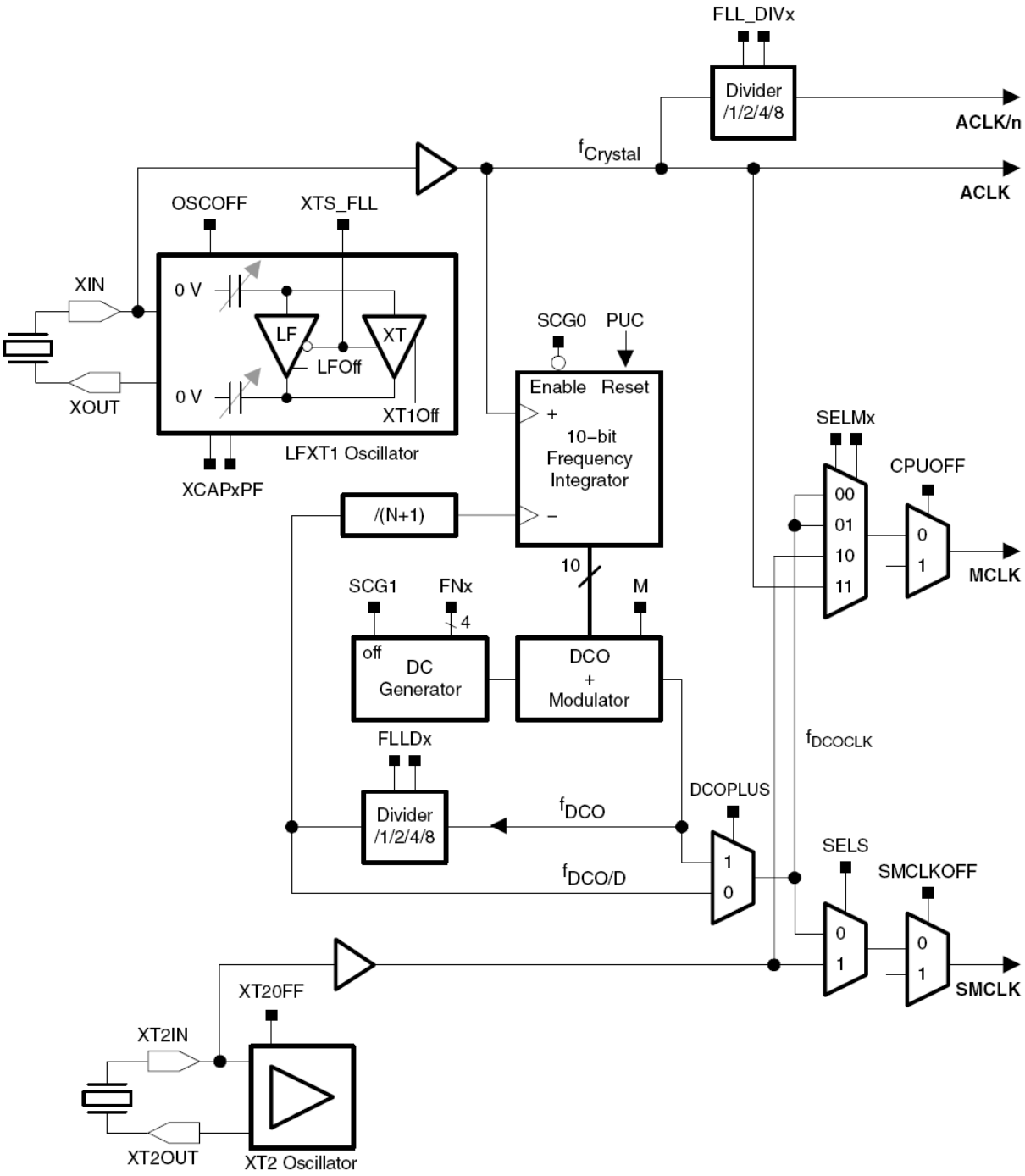


Figure 4. Block diagram of FLL+ module in MSP430FG461x devices.

- 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 HFmode, 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 5. Its initial value is 1Fh, which means that modulation is enabled (SCFQ_M=0, and N=001_1111 = 31).

SCFQCTL, System Clock Control Register

7	6	5	4	3	2	1	0
SCFQ_M	N						
rw–0	rw–0	rw–0	rw–1	rw–1	rw–1	rw–1	rw–1

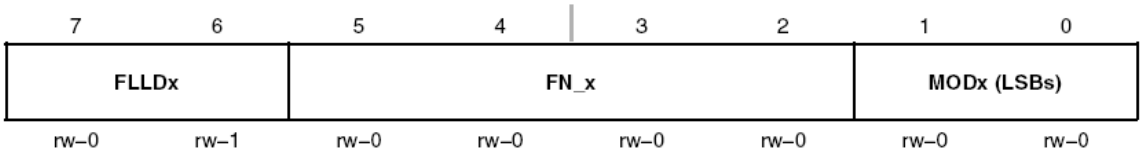
SCFQ_M	Bit 7	Modulation. This enables or disables modulation 0 Modulation enabled 1 Modulation disabled
N	Bits 6-0	Multiplier. These bits set the multiplier value for the DCO. N must be > 0 or unpredictable operation will result. When DCOPLUS=0: $f_{DCOCLK} = (N + 1) \cdot f_{crystal}$ When DCOPLUS=1: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

Figure 5. Format of the SCFQCTL register.

The format of the SCFI0 and SCFI1registers is given in Figure 6. 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 and Figure 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 on each Figure 4 and determine configuration of each resource. What is the clock frequency on ACLK, DCOCLK, MCLK, and SMCLK?

SCFI0, System Clock Frequency Integrator Register 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+.

SCFI1, System Clock Frequency Integrator Register 1

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 6. Format of the SCFI0 and SCFI1 registers.

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

Figure7. 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 8. Format of the FLL_CTL1 register.

The following examples illustrate (Figure 9 and Figure) 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). For learning how internal digitally-controlled oscillator works read the corresponding user manual.

```
1. //******************************************************************************
2. //  MSP430xG46x Demo - FLL+, Runs Internal DCO at 2.45MHz
3. //
4. //  Description: This program demonstrates setting the internal DCO to run at
5. //  2.45MHz with auto-calibration by the FLL+ circuitry.
6. //  ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = (74+1) x ACLK = 2457600Hz
7. //  //* An external watch crystal between XIN & XOUT is required for ACLK *//
8. //
9. //                MSP430xG461x
10.//             -----------------
11.//         /|\|              XIN|-
12.//          | |                 | 32kHz
13.//          --|RST          XOUT|-
14.//            |                 |
15.//            |             P1.1|--> MCLK = 2.45MHz
16.//            |                 |
17.//            |             P1.4|--> SMCLK = 2.45MHz
18.//            |             P1.5|--> ACLK = 32kHz
19.//            |                 |
20.//
21.//******************************************************************************
22.#include  <msp430xG46x.h>
23.
24.void main(void)
25.{
26.  WDTCTL = WDTPW + WDTHOLD;      // Stop watchdog timer
27.  FLL_CTL0 |= XCAP18PF;          // Set load capacitance for xtal
28.  SCFI0 |= FN_2;                 // x2 DCO, 4MHz nominal DCO
29.  SCFQCTL = 74;                  // (74+1) x 32768 = 2.45Mhz
30.
31.  P1DIR |= 0x32;                 // P1.1, P1.4 & P1.5 to output direction
32.  P1SEL |= 0x32;                 // P1.1, P1.4 & P1.5 to output MCLK, SMCLK & ACLK
33.
34.  while(1);                     // Loop in place
35.}
```

Figure 9. FLL+ clock module configuration: (Lab6_D4.C)

```
1  //*******************************************************************************
2  //  MSP430xG46x Demo - FLL+, Runs Internal DCO at 8MHz
3  //
4  //  Description: This program demonstrates setting the internal DCO to run at
5  //  8MHz with auto-calibration by the FLL+.
6  //  ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = (121+1) x 2 x ACLK = 7995392Hz
7  //  //* An external watch crystal between XIN & XOUT is required for ACLK *//
8  //
9  //                MSP430xG461x
10 //             -----------------
11 //         /|\|              XIN|-
12 //          | |                 | 32kHz
13 //          --|RST          XOUT|-
14 //            |                 |
15 //            |             P1.1|--> MCLK = 8MHz
16 //            |                 |
17 //            |             P1.4|--> SMCLK = 8MHz
18 //            |             P1.5|--> ACLK = 32kHz
19 //            |                 |
20 //
21 //*******************************************************************************
22 #include  <msp430xG46x.h>
23
24 void main(void)
25 {
26   WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
27   FLL_CTL0 |= DCOPLUS + XCAP18PF; // DCO+ set, freq = xtal x D x N+1
28   SCFI0 |= FN_4;                  // x2 DCO freq, 8MHz nominal DCO
29   SCFQCTL = 121;                  // (121+1) x 32768 x 2 = 7.99 MHz
30   P1DIR |= 0x32;                  // P1.1, P1.4 & P1.5 to output direction
31   P1SEL |= 0x32;                  // P1.1, P1.4 & P1.5 to output MCLK, SMCLK & ACLK
32
33   while(1);                       // Loop in place
34 }
```

Figure 10. FLL+ clock module configuration (Lab6_D5.C)

Consider the code shown in Figure to take into account a different processor clock cycle time (it is 0.5 us). Elaborate on the required changes?

```
/*****************************************************************************
;   MSP430xG46x Demo - Software Toggle P2.2
;   Description: Toggle P2.2 by xor'ing P2.2 inside of a software loop.
;   ACLK = 32.768kHz, MCLK = SMCLK = 2 MHz
;                MSP430xG461x
;             -----------------
;         /|\|                 |
;          | |                 |
;          --|RST              |
;            |                 |
;            |             P2.2|-->LED1
;   Alex Milenkovich, milenkovic@computer.org
;*****************************************************************************/
#include "msp430xG46x.h"

void main(void)
{
   WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
   FLL_CTL0 |= XCAP18PF;     // Set load capacitance for xtal
   SCFI0 |= FN_2;            // x2 DCO, 4MHz nominal DCO
   SCFQCTL = 60;             // (60+1) x 32768 = 1998848 Hz~2 MHz

   P2DIR |= 0x04;      // Set P2.2 to output direction (0000_0010)
   for (;;) {
     unsigned int i;
     P2OUT ^= 0x04;    // Toggle P2.2 using exclusive-OR
     i = 50000;        // Delay
     do {
       i--;
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
       asm("NOP");
     } while (i != 0);
   }
}
```

Figure11. Toggling the LED when the processor is running at 2 MHz clock (Lab6_D6.C)

Note the C statement:

```
asm("NOP");
```

In C, it is often necessary to use an assembly command. The above statement creates a NOP – a command not found in C.

1.3. Assignment

You must write a C program that first initializes the clock speed at ~2 MHz. You should then have LED1 and LED2 alternate blinking on and off at a rate of about 1 Hz (0.5 seconds on, 0.5 seconds off; if LED1 is on, LED2 is off and vice versa).

When SW1 is pressed, you must double the blinking frequency to 2 Hz by changing the clock speed. If SW1 is pressed again, the blinking frequency should reset to 1 Hz.

While SW2 is pressed, you must half the blinking frequency to 0.5 Hz by changing the clock speed. When SW2 is released, the blinking frequency should reset to the frequency before SW2 was pressed (depending on if SW1 has already changed the frequency). SW1 does not have to be responsive while SW2 is pressed.

You must use an interrupt to interface the switches.