LAB 3: Initialization and GPIO

Chương trình sẽ tắt mở lần lượt từng LED red->LED Blue->Led Green và sau đó quay lại mở LED red. Trong mỗi lần mở một LED thì chương trình sẽ delay một it thời gian để người dùng có thể nhận biết.

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
int PinData=2;
int main(void)
{
	// This setting used PLL clock from 200Mhz/5 = 40Mhz
	// Use external clock 16Mhz is a source of PLL
	SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
	// Enable Clock GPIO port F
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
	// Enable 3 GPIO PF.1, PF.2 and PF.3 is Output
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

	while(1)
	{
		// If PinData = 2 => PF.1 = 1 => Led Red is ON
		// If PinData = 4 => PF.2 = 1 => Led Blue is ON
		// If PinData = 8 => PF.3 = 1 => Led Green is ON
		// While one LED is on, the other leds are off
		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, PinData);
		// Delay some time that Led ON is able to recogized by user
		SysCtlDelay(2000000);
		// Turn off 3 Leds by setting PF.1 = 0, PF.2 = 0; PF.3 = 0
		GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);
		// Delay some time to switch off all led
		SysCtlDelay(2000000);
		//  Change sequence that Led is turned ON
		// ON LED RED -> ON LED BLUE -> ON LED GREEN and then ON LED RED
```

```
            if(PinData==8) {PinData=2;} else {PinData=PinData*2;}
        }
}
```

LAB 3: RESULT

Kết quả là 3 led RED, GREEN, BLUE xen kẻ nhau sáng. Như hình bên dưới.

LAB 4: Interrupts and the Timer

Chương tình sẽ khởi động một Timer hoạt động ở tần số 20 Hz để nhấp nháy led BLUE ở tần số 10HZ. Chường trình sẽ dùng timer A mốt time out để tạo ra interrupt ở tần so 20Hz gấp 2 lần tần số của LED blue. Mỗi khi vào interrupt của TIMER A, chương trình sẽ kiểm tra trước đó Blue LED có được On không nếu ON thì chuyển trạng thái Blue LED san OFF và làm ngược lại cho lần interrupt tiếp theo. Hàm main sẽ làm hàm lặp vô tận và không làm gì để đợi interrupt xảy ra.

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

int main(void)
{
        unsigned long ulPeriod;

        // Set clock of system = 40Mhz, Use external clock 16Mhz is a source for PLL
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

        // Enable clock for PORT F
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        // Enable 3 GPIO Port PF.1, PF.2 and PF.3 is output
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

        // Enable clock and peripheral that relate to ITMER 0
```

```
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
	// Timer 0 is configured to work in 32 bits timer
	// It combines 2 timers TIMER 0A and TIMER 0B which every timer contains 16 bits timer
	TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);

	// Calculate number of clock circle for 10Hz
	// We need to toggle LED at 10 Hz but we need to turn ON and then OFF led so
	// Period of interrupt = 1/2 periods of 10Hz.
	ulPeriod = (SysCtlClockGet() / 10) / 2;
	// Load period to timer A
	TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);

	// Enable timer A interrupt
	IntEnable(INT_TIMER0A);
	// Enable timer interrupt source for TIMER AR is using mode TIMEOUT
	TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
	// Enable global interrupt for device
	IntMasterEnable();

	// Start Timer A to process time out of Timer A and its interrupt
	TimerEnable(TIMER0_BASE, TIMER_A);

	while(1)
	{
	}
}

// Interrupt TIMER A Handler
void Timer0IntHandler(void)
{
	// Clear the timer interrupt flag that interrupt happened
	TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

	// Read the current state of the GPIO pin and
	// write back the opposite state
	// Read LED BLUE is ON or not
```

```
        // If LED Blue is ON, then turn off 3 leds (RED, BLUE, GREEN)
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
        {
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
        }
        else
        {
                // IF Blue LED is Off before, and then it is ON
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
        }
}
```

LAB 4: RESULT

LAB 5: ADC12

Bài ADC từ temperature sensor bên trong chip. ADC được cấu hình dùng sequencer 1 có FIFO bằng 4 phần tử. Giá trị ADC đo được quy đổi ra thành giá tị độ C và F. Trong chương trình hỗ trợ thêm UART printf để in giá trị nhiệt độ đo được ra ngoài.

Chương trình sẽ dùng hàm prinft để xuất giá trị đo được ra màn hình để quan sát. Trước khi vào hàm main khi khởi động printf ta cho xuất chuỗi. Trong đó có ký tự \n là ký tự xuống dòng báo cho terminal xuống dòng trước khi kết thúc chuỗi.

```
“ADC -> \n "
 “Type: Internal Temperature Sensor\n "
 “Samples: One\n"
 "  Update Rate: 250ms\n"
  "  Input Pin: Internal temperature sensor\n \n "
```

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "utils/uartstdio.h"
```

```
//*****************************************************************************
//
// This function sets up UART0 to be used for a console to display information
// as the example is running.
//
//*****************************************************************************
void
InitConsole(void)
{
  //
  // Enable GPIO port A which is used for UART0 pins.
  // TODO: change this to whichever GPIO port you are using.
  //
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

  //
  // Configure the pin muxing for UART0 functions on port A0 and A1.
  // This step is not necessary if your part does not support pin muxing.
  // TODO: change this to select the port/pin you are using.
  //
  GPIOPinConfigure(GPIO_PA0_U0RX);
  GPIOPinConfigure(GPIO_PA1_U0TX);

  //
  // Select the alternate (UART) function for these pins.
  // TODO: change this to select the port/pin you are using.
  //
  GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

  //
  // Initialize the UART for console I/O.
  //
  UARTStdioInit(0);
}

int main(void)
```

```
{
        // Create buffer 4 items 32 bits to for FIFO because
        // system use sequencer 1
        unsigned long ulADC0Value[4];
        // Store value avarage of temperature
        volatile unsigned long ulTempAvg;
        // Store temperature in Celsius
        volatile unsigned long ulTempValueC;
        // Store temperature in Fahrenheit
        volatile unsigned long ulTempValueF;

        // Setting clock of system = 40Mhz
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

        InitConsole();

        // Display the setup on the console.

         UARTprintf("ADC ->\n");
         UARTprintf("  Type: Internal Temperature Sensor\n");
         UARTprintf("  Samples: One\n");
        UARTprintf("  Update Rate: 250ms\n");
        UARTprintf("  Input Pin: Internal temperature sensor\n\n");

        // Setting clock and peripheral for ADC0
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        // Speed of ADC samples = 250,000 samples per second
        SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
        //ADCHardwareOversampleConfigure(ADC0_BASE, 64);
        // Temporarily, we disable the ADC 0 sequence 1 to confure the other parameter
        ADCSequenceDisable(ADC0_BASE, 1);

        // Configure ADC0 with mode trigger 1 and sequencer 1 with highest prioprity
        ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

        // Declear the 4 samples termperature sensors to store in sequencer 1 and calculate average
```

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
// Enable temperature sensor select, Interrupt enable and Sequence end select to ADC mode
ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
// Enable ADC sequence 1 and start to measure data
ADCSequenceEnable(ADC0_BASE, 1);

while(1)
{
        // Clear ADC conversion flag after measurement is completed
        ADCIntClear(ADC0_BASE, 1);

        // Trigger ADC Converstion to start mesurement again
        ADCProcessorTrigger(ADC0_BASE, 1);

        // Wait until completing ADC conversion
        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {
        }

        // Read Raw data of temperature sequence 1 and store in buffer
        ADCSequenceDataGet(ADC0_BASE, 1, ulADC0Value);
        ulTempAvg = (ulADC0Value[0] + ulADC0Value[1] + ulADC0Value[2] + ulADC0Value[3] +
2)/4;
        ulTempValueC = (1475 - ((2475 * ulTempAvg)) / 4096)/10;
        ulTempValueF = ((ulTempValueC * 9) + 160) / 5;

        // Display data on teminal
        UARTprintf("Celsius = %4d mV\r", ulTempValueC);
        UARTprintf("Fahrenheit = %4d mV\r", ulTempValueF);

        //
    // This function provides a means of generating a constant length
    // delay.  The function delay (in cycles) = 3 * parameter.  Delay
    // 250ms arbitrarily.
```

```
            //
            SysCtlDelay(SysCtlClockGet() / 6);    // 12 for /10
        }
}
```

LAB 5: RESULT

Kết quả xuất ra khi đo nhiệt độ của micro controller.

LAB 6: HIBERNATE

Hibernate mode se được cấu hình là được Wake up từ pin Wake (SW2). Khi người dùng bấm nút thì LED Green sẽ sáng sau đó chương trình sẽ vào hàm main và lập vô tận. Trước khi vào hibernate mode chương trình sẽ hiện ra xuất ra trên terminal với chướng trình prinft chữ “TEST”.
Khi chương trình thoát khỏi hibernate và vào hàm main thì sẽ xuất chuỗi “EXIT HIBERNATE” và có thời gian delay khoảng 1 giây để quan sát.

```
#include "utils/ustdlib.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "utils/ustdlib.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/debug.h"
#include "driverlib/hibernate.h"
#include "driverlib/gpio.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"


#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
```

```
#endif


void
InitConsole(void)
{

   // Enable GPIO port A which is used for UART0 pins.
   // TODO: change this to whichever GPIO port you are using.

   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);



   // Configure the pin muxing for UART0 functions on port A0 and A1.
   // This step is not necessary if your part does not support pin muxing.
   // TODO: change this to select the port/pin you are using.

   GPIOPinConfigure(GPIO_PA0_U0RX);
   GPIOPinConfigure(GPIO_PA1_U0TX);



   // Select the alternate (UART) function for these pins.
   // TODO: change this to select the port/pin you are using.
   GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
   // Initialize the UART for console I/O.
   UARTStdioInit(0);
}

int main(void)
{
        // Set clock = 40Mhz
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

        // Initialize for Printf
        InitConsole();
```

```
// Enable clock and peripheral for PORT F
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
// Set PF.1 PF.2 and PF.3 are GPIOs
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

// Output “Test” message on terminal
UARTCharPut(UART0_BASE, 'T');
UARTCharPut(UART0_BASE, 'E');
UARTCharPut(UART0_BASE, 'S');
UARTCharPut(UART0_BASE, 'T');

// Enable hibernate module
SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);

// Define clock support for Hibernate module
HibernateEnableExpClk(SysCtlClockGet());
// Allow GPIOs that are active during hibernate mode
HibernateGPIORetentionEnable();
// Delay
SysCtlDelay(64000000);
// Set condition to wake up PIN
HibernateWakeSet(HIBERNATE_WAKE_PIN);

// Turn off all LEDs
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, 0x00);

// Go to hibernate mode
HibernateRequest();

// Turn ON 1 LED: GREEN LED
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x08);

while(1)
{
        // Output string “EXIT HIBERNATE” on terminal
        UARTprintf("EXIT HIBERNATE\n");
```

```
            // Delay near 1 second for viewing
            SysCtlDelay(SysCtlClockGet() / 3);
        }
}
```

LAB 12

Chương trình khởi động UART 0 với các clock hệ thống là 50Mhz, khi đó chương trình cũng khởi động luôn PA0 và PA1 GPIO để cho 2 đường tính hiệu Rx và Tx của UART. UART được cấu hình với tốc độ baud là 115200, 8 bit data, 1 stop bit và không có check chẵn lẻ.
Trước khi vào hàm main thì chương trình sẽ xuất ra trên terminal dòng chữ: “Enter Text:” khi vào hàm main thì chướng trình sẽ kiểm tra liên tục trong bộ đệm RX của UART có ký tự nào nhận được không nếu có thì xuất ra terminal.
Ở kết quả bên dưới khi em gửi chuỗi “test UART” thì ở phần nhận của terminal xuất đúng ký tự “test UART” sau chuỗi “Enter Text:”

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

int main(void) {

  // Set up clock for system with 200/4 = 50Mhz
 SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
  // Enable UART 0
   SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
  // Enable peripheral for GPIOA to use  PA0 and PA1 for UART interface
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

   GPIOPinConfigure(GPIO_PA0_U0RX);
   GPIOPinConfigure(GPIO_PA1_U0TX);
   GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

   UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
```

```
    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

  UARTCharPut(UART0_BASE, 'E');
  UARTCharPut(UART0_BASE, 'n');
  UARTCharPut(UART0_BASE, 't');
  UARTCharPut(UART0_BASE, 'e');
  UARTCharPut(UART0_BASE, 'r');
  UARTCharPut(UART0_BASE, ' ');
  UARTCharPut(UART0_BASE, 'T');
  UARTCharPut(UART0_BASE, 'e');
  UARTCharPut(UART0_BASE, 'x');
  UARTCharPut(UART0_BASE, 't');
  UARTCharPut(UART0_BASE, ':');
  UARTCharPut(UART0_BASE, ' ');

  while (1)
  {
      if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
  }
}
```