

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HỒ CHÍ MINH
KHOA ĐIỆN ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ



BÀI TẬP 3
Tiva™ TM4C123G LaunchPad

GVHD: TS. Trương Quang Vinh
HVTH: Trần Văn Trọng - 1870262

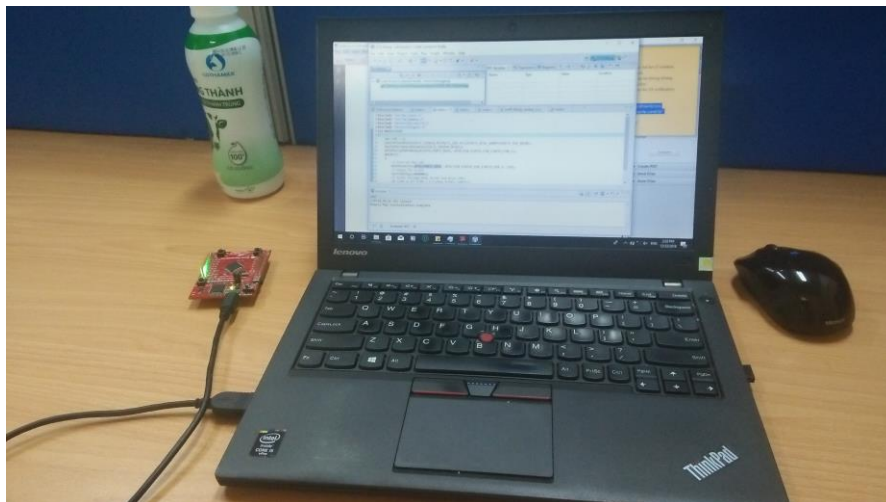
Tp.HCM, tháng 12 năm 2018

CHƯƠNG 1

Lab1: Hardware and Software Set Up

Mục tiêu của bài luyện tập lab này là tải về và cài đặt Code Composer Studio, cũng như tải các tài liệu hỗ trợ và phần mềm khác nhau được sử dụng với bài tập này. Sau đó chúng ta sẽ xem xét các nội dung của bộ kit và xác minh hoạt động với các chương trình quickstart bản demo được nạp sẵn. Các công cụ phát triển sẽ được sử dụng trong suốt các bài lab còn lại.

Dưới đây là cách nối trực tiếp bo và lap top.



CHƯƠNG 2

Lab2: Code Composer Studio

Trong bài lab này, chúng ta sẽ tạo ra một dự án có chứa hai tập tin nguồn, main.c và tm4c123gh6pm_startup_ccs.c, trong đó có chứa đoạn mã để chớp LED trên bo LaunchPad. Mục đích của thí nghiệm này là để thực hành tạo ra các dự án và tìm hiểu cái nhìn và cảm nhận của Code Composer Studio. Trong các bài lab sau đó chúng ta sẽ kiểm tra mã chi tiết hơn.

Chúng ta soạn thảo một đoạn code như bên dưới:

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

int main(void) { int LED = 2;

SysCtlClockSet(SYSCTL_SYSDIV_4/SYSCTL_USE_PLL/SYSCTL_XTAL_16MHZ/SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3);
while(1) { // Turn on the LED GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3, LED);
// Delay for a bit SysCtlDelay(2000000);
// Cycle through Red, Green and Blue LEDs if (LED == 8) {LED = 2;} else
{LED = LED*2;} }
}
```

Kết quả thí nghiệm:

Led sẽ nhấp nháy với 3 màu lam, lục và đỏ.





CHƯƠNG 3

Lab 3: Initialization and GPIO

Trong bài thí nghiệm này, chúng ta sẽ tìm hiểu làm thế nào để khởi tạo các hệ thống clock và sử dụng các thiết bị ngoại vi GPIO của kit TivaWare. Sau đó chúng ta sẽ sử dụng đầu ra GPIO nhấp nháy một đèn LED trên board.

Ý nghĩa của một số file header:

stdint.h: Các định nghĩa biến cho các tiêu chuẩn C99.

stdbool.h: Định nghĩa Boolean cho các tiêu chuẩn C99.

hw_memmap.h: Macros xác định bản đồ bộ nhớ của thiết bị Tiva C Series. Điều này bao gồm định nghĩa như ngoại vi, địa điểm, địa chỉ cơ sở như GPIO_PORTF_BASE.

hw_types.h: Xác định loại phổ biến và macro.

sysctl.h: Định nghĩa và macro cho hệ thống điều khiển API của

DriverLib: Nó bao gồm hàm API như SysCtlClockSet và SysCtlClockGet.

gpio.h: Định nghĩa và macro cho GPIO API của DriverLib bao gồm các chức năng API như GPIOPinTypePWM và GPIOPinWrite.

uint8_t ui8PinData = 2: Tạo ra một biến số nguyên được gọi là ui8PinData và khởi tạo. Điều này sẽ được sử dụng như chu kỳ giữa ba đèn LED.

Lưu ý rằng các loại C99 là một số nguyên không dấu 8-bit và tên biến phản ánh điều này.

Hàm chính:

Cấu trúc của một hàm chính

```
int main (void)
{ }
```

Nếu bạn gõ dấu “{“, trình soạn thảo sẽ tự động thêm ngoặc “}” phía sau.

Thiết lập xung clock

Cấu hình xung clock của hệ thống sử dụng một tinh thể có tần số 16MHz trên dao động chính, điều khiển các PLL 400MHz.

Các PLL 400MHz dao động ở tần số đó, nhưng có thể được điều khiển bởi các tinh thể hoặc dao động từ 5 đến 25MHz. Có một default / 2 chia trong clock và chúng ta đang xác định khác / 5, tổng số 10. hệ thống có các clock sẽ là 40MHz.

Nhập dòng mã này bên trong main ():

```
SysCtlClockSet (SYSCTL_SYSDIV_5 /
SYSCTL_USE_PLL / SYSCTL_XTAL_16MHZ /
SYSCTL_OSC_MAIN);
```

Tinh thể gắn vào các đầu vào dao động chính là 16MHz, trong khi các tinh thể gắn vào RTC đầu vào là 32,768Hz.

Trước khi gọi bất kỳ chức năng ngoại vi driverLib, chúng ta phải cung cấp clock cho ngoại vi. Nếu bạn không làm điều này, nó sẽ gây ra một lỗi ISR (lỗi địa chỉ). Đây một sai lầm phổ biến cho người dùng Tiva C Series mới. Tiếp theo ta cần chỉnh cấu

hình, ba chân GPIO kết nối với các đèn LED là kết quả đầu ra. Hình dưới đây cho thấy GPIO chân PF1, PF2 và PF3 được kết nối với các đèn LED. Để lại một dòng cho khoảng cách, sau đó nhập vào hai dòng mã bên trong main () sau các dòng trong bước trước.

```
SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOF);  
GPIOPinTypeGPIOOutput (GPIO_PORTF_BASE,  
GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
```

APB đề cập đến các thiết bị ngoại vi Bus nâng cao, trong khi AHB đề cập đến sự trình diễn nâng cao. Trong phòng thí nghiệm của chúng ta, GPIO_PORTF_BASE là 0x40025000.

```
GPIO Port A (APB): 0x4000.4000  
GPIO Port A (AHB): 0x4005.8000  
GPIO Port B (APB): 0x4000.5000  
GPIO Port B (AHB): 0x4005.9000  
GPIO Cổng C (APB): 0x4000.6000  
GPIO Cổng C (AHB): 0x4005.A000  
GPIO Port D (APB): 0x4000.7000  
GPIO Port D (AHB): 0x4005.B000  
GPIO Cổng E (APB): 0x4002.4000  
GPIO Cổng E (AHB): 0x4005.C000  
GPIO Cổng F (APB): 0x4002.5000  
GPIO Cổng F (AHB): 0x4005.D000  
while () Loop
```

Cuối cùng, tạo một thời gian (1) vòng lặp chậm trễ.

SysCtlDelay () là một bộ đếm thời gian loop cung cấp trong TivaWare. Các tham số tính là vòng lặp đếm, không chậm trễ thực tế trong chu kỳ đồng hồ. Mỗi vòng lặp là 3 chu kỳ CPU. + Để viết với các chân GPIO, sử dụng các GPIO API chức năng gọi GPIOPinWrite. Bảo đảm để đọc và hiểu cách các chức năng GPIOPinWrite được sử dụng trong datasheet. Các tranh luận dữ liệu thứ ba không chỉ đơn giản là 1 hoặc 0, nhưng đại diện cho toàn bộ cổng dữ liệu 8-bit. Các số thứ hai là một mặt nạ bit-đóng gói của các dữ liệu được ghi.

Trong ví dụ của chúng tôi dưới đây, chúng tôi đang viết các giá trị trong biến ui8PinData cho cả ba chân GPIO được kết nối với các đèn LED. Được viết để dựa trên mặt nạ bit quy định. Các chu kỳ chỉ dẫn cuối cùng thông qua các đèn LED bằng cách làm ui8PinData bằng 2, 4, 8, 2, 4, 8 và như vậy. Lưu ý rằng các giá trị gửi đến các chân phù hợp vị trí của họ; "một" trong vị trí bit hai chỉ có thể đạt được pin chút hai trên cổng.

Bây giờ có thể là một thời điểm tốt để nhìn vào các thông số kỹ thuật cho thiết bị Tiva C Series của bạn. kiểm tra ra chojơng GPIO để hiểu được cách duy nhất các dữ liệu đăng ký GPIO được thiết kế và những ưu điểm của phương pháp này.

```
while (1)  
{  
GPIOPinWrite (GPIO_PORTF_BASE, GPIO_PIN_1 |
```

```

GPIO_PIN_2 | GPIO_PIN_3, ui8PinData);
SysCtlDelay (2000000);
GPIOPinWrite (GPIO_PORTF_BASE, GPIO_PIN_1 |
GPIO_PIN_2 | GPIO_PIN_3, 0x00);
SysCtlDelay (2000000);
if (ui8PinData == 8) {ui8PinData = 2;} else {ui8PinData =
ui8PinData * 2;}
}

```

Ta có chương trình hoàn chỉnh như sau:

```

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
uint8_t ui8PinData=2;
int main(void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYS
CTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
while(1)
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|
GPIO_PIN_2| GPIO_PIN_3, ui8PinData);
SysCtlDelay(2000000);
GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);
SysCtlDelay(2000000);
if(ui8PinData==8) {ui8PinData=2;} else
{ui8PinData=ui8PinData*2;}
} }

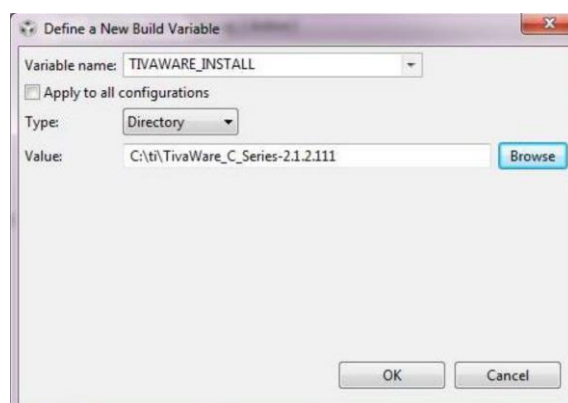
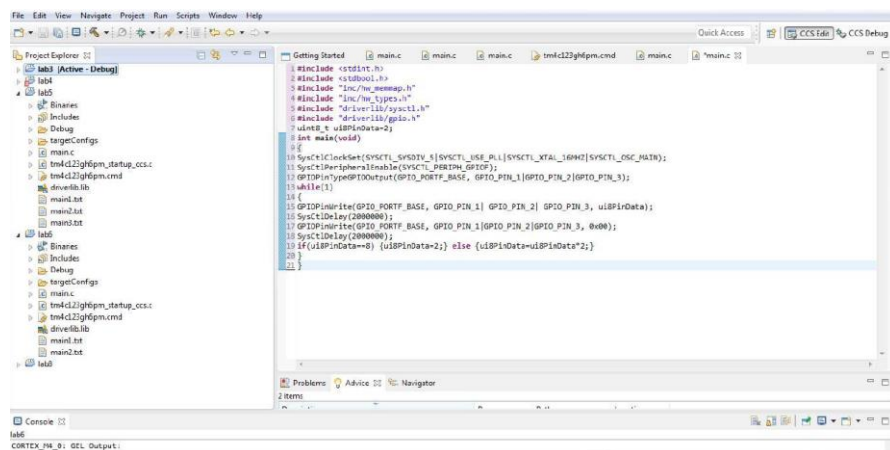
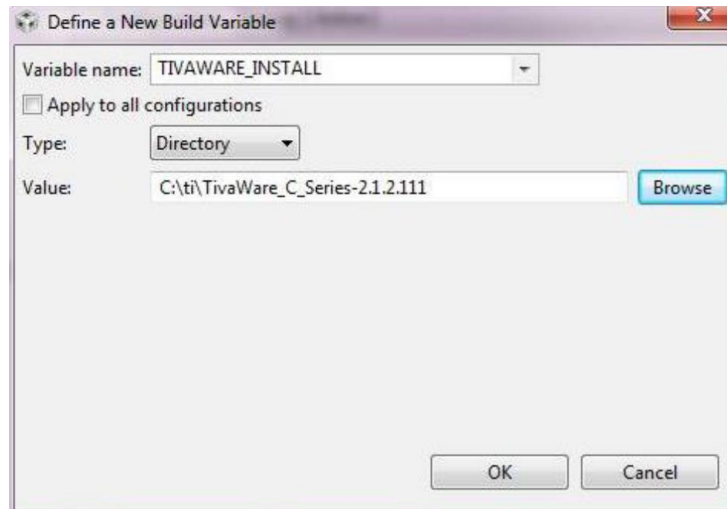
```

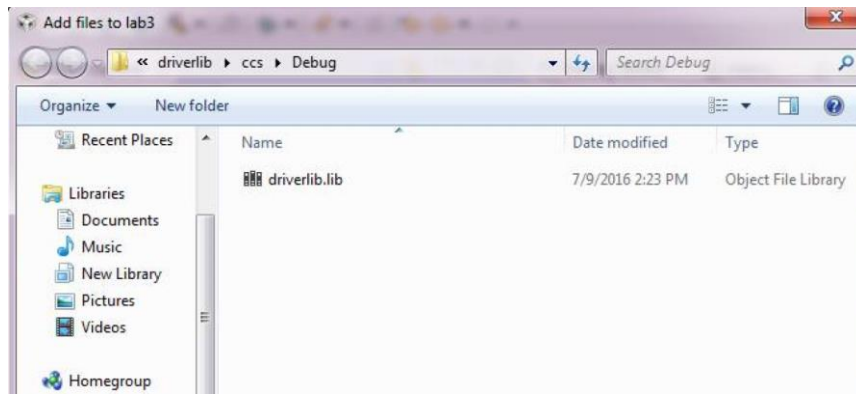
Ta có một sự chậm trễ trong 3-6 chu kỳ của thiết bị ngoại vi.
Kích hoạt tính năng GPIO

Config ADC
Config GPIO

Mã khởi động ngoài các tập tin chính bạn đã tạo ra, bạn cũng cần có một tập tin khởi động cụ thể tập tin này có chứa các bảng vector, ta cần khởi động để sao chép dữ liệu khởi tạo vào RAM và xóa phần bss, và ISRs lỗi mặc định. Các project mới Wizard sẽ tự động thêm một bản sao của tập tin Click đôi vào file tm4c123gh6pm_startup_ccs.c trong cửa sổ làm việc và xem xét. Không thực hiện bất kỳ thay đổi vào thời

điểm này. Đóng tập tin.
Kết quả tiến trình và thí nghiệm:





CHƯƠNG 4

Lab 4: Interrupts and the Timer

Trong bài thí nghiệm này, chúng ta sẽ thiết lập bộ đếm thời gian để tạo ra ngắt, và sau đó viết code mà đáp ứng ngắt nhấp nháy đèn LED. Chúng ta cũng sẽ thử nghiệm với việc tạo ra một hệ thống, bằng cách cố gắng để cấu hình một thiết bị ngoại vi trước khi kích hoạt nó.

Ta tiến hành mở project bằng cách chọn Project => Import Existing CCS Eclipse.

Project => chọn đường dẫn đến lab4 => chọn finish.

+ Click chuột vào tên lab4 để mở rộng dự án.

+ Chọn vào file main.c mở file main.c ta thêm các dòng khai báo file header dưới đây:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.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"
```

Ý nghĩa của một số file header:

tm4c123gh6pm.h: Các định nghĩa cho các gián đoạn và đăng ký thiết bị Tiva C Series trên bảng LaunchPad.

interrupt.h: Định nghĩa và mở rộng cho NVIC Controller (Interrupt) API

driverLib: Điều này bao gồm các chức năng API như IntEnable và IntPrioritySet.

timer.h: Định nghĩa và macro cho hẹn giờ API của driverLib. Điều này bao gồm API các chức năng như TimerConfigure và TimerLoadSet.

Hàm chính

```
int main(void)
{
    uint32_t ui32Period ;
}
```

Thiết lập xung clock

```
SysCtlClockSet(SYSCTL_SYSDIV_5/SYSCTL_USE_PLL/SYSCTL_
XTAL_16MHZ/SYSCTL_OSC_MAIN);
```

Cấu hình GPIO

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3);
```

Cấu hình timer

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
```

Tính toán delay

```
ui32Period = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
```

Cho phép ngắt

```
intEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
+ Bật timer
TimerEnable(TIMER0_BASE, TIMER_A);
While(1) loop
While
{ }
```

Giải quyết các vấn đề ngắt timer

```
void Timer0IntHandler(void)
{
// Clear the timer interrupt
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
// Read the current state of the GPIO pin and
// write back the opposite state
if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
{
GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}
else
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
} }
}
```

Ta được chương trình hoàn chỉnh như sau:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.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)
{
uint32_t ui32Period;
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_X
TAL_16MHZ|SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
```

```

TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
ui32Period = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);

```

```

96  IntDefaultHandler,          // PWM Generator 0
97  IntDefaultHandler,          // PWM Generator 1
98  IntDefaultHandler,          // PWM Generator 2
99  IntDefaultHandler,          // Quadrature Encoder 0
100 IntDefaultHandler,          // ADC Sequence 0
101 IntDefaultHandler,          // ADC Sequence 1
102 IntDefaultHandler,          // ADC Sequence 2
103 IntDefaultHandler,          // ADC Sequence 3
104 IntDefaultHandler,          // Watchdog timer
105 Timer0IntHandler,           // Timer 0 subtimer A
106 IntDefaultHandler,          // Timer 0 subtimer B
107 IntDefaultHandler,          // Timer 1 subtimer A
108 IntDefaultHandler,          // Timer 1 subtimer B
109 IntDefaultHandler,          // Timer 2 subtimer A
110 IntDefaultHandler,          // Timer 2 subtimer B
111 IntDefaultHandler,          // Analog Comparator 0
112 IntDefaultHandler,          // Analog Comparator 1
113 IntDefaultHandler,          // Analog Comparator 2
114 IntDefaultHandler,          // System Control (PLL, OSC, BO)
115 IntDefaultHandler,          // FLASH Control
116 IntDefaultHandler,          // GPIO Port F
117 IntDefaultHandler,          // GPIO Port G
118 IntDefaultHandler,          // GPIO Port H
119 IntDefaultHandler,          // UART2 Rx and Tx

```

```

while(1)
{
}
void Timer0IntHandler(void)

```

```

32 void ResetISR(void);
33 static void NmiISR(void);
34 static void FaultISR(void);
35 static void IntDefaultHandler(void);
36
37 //*****
38 //
39 // External declaration for the reset handler that is to be called when the
40 // processor is started
41 //
42 //*****
43 extern void _c_int00(void);
44 extern void Timer0IntHandler(void);
45 //*****
46 //
47 // Linker variable that marks the top of the stack.
48 //
49 //*****
50 extern uint32_t __STACK_TOP;
51
52 //*****

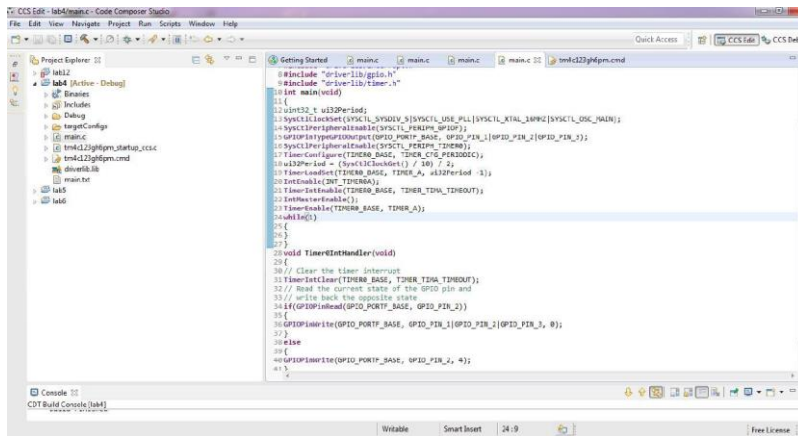
```

```

{
// Clear the timer interrupt
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
// Read the current state of the GPIO pin and
// write back the opposite state
if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
{
GPIOPinWrite(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}
else
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}
}

```

}}



```
8 #include "driverlib/gpio.h"
9 #include "driverlib/timer.h"
10 int main(void)
11 {
12     uint32_t ui32PerIOD;
13     SysCtlLockset(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHz|SYSCTL_OSC_MAIN);
14     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
15     GPIOWriteGPIOOutput(GPIO_PORT_A_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
16     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
17     TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
18     ui32PerIOD = (SysCtlClock() / 10) / 2;
19     TimerLoadSet(TIMER0_BASE, TIMER_A, ui32PerIOD - 1);
20     IntEnable(INT_TIMER0A);
21     TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
22     TimerIntEnable();
23     TimerEnable(TIMER0_BASE, TIMER_A);
24     while(1)
25     {
26     }
27 }
28 void Timer0IntHandler(void)
29 {
30     // Clear the timer interrupt
31     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
32     // Read the current state of the GPIO pin and
33     // write back the opposite state
34     GPIOWrite(GPIO_PORT_A_BASE, GPIO_PIN_2);
35 }
36 GPIOWrite(GPIO_PORT_A_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
37 }
38 else
39 {
40     GPIOWrite(GPIO_PORT_A_BASE, GPIO_PIN_2, 1);
41 }
```

Kết quả thí nghiệm:



CHƯƠNG 5

Lab 5: ADC12

Trong bài thí nghiệm này chúng ta sẽ sử dụng ADC12 và mẫu trình tự để đo lường các dữ liệu từ trên chip như cảm biến nhiệt độ. Chúng ta sẽ sử dụng Code Composer để hiển thị các giá trị thay đổi.

Ta tiến hành mở project bằng cách chọn Project => Import Existing CCS Eclipse Project => chọn đường dẫn đến lab4 => chọn finish.

Click chuột vào tên lab4 để mở rộng dự án.

Chọn vào file main.c mở file main.c ta thêm các dòng khai báo file header dưới đây:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
```

Thiết lập hàm chính main bằng cách thêm ba dòng dưới đây:

```
int main (void)
{ }
```

Thêm dòng mã sau đây vào bên trong hàm main ():

```
uint32_t ui32ADC0Value [4];
```

Chúng ta sẽ cần một số biến để tính nhiệt độ từ các dữ liệu cảm biến. Đầu tiên biến là để lưu trữ trung bình của nhiệt độ. Các biến còn lại được sử dụng để lưu trữ các giá trị nhiệt độ Celsius và Fahrenheit.

Thêm các dòng sau vào cuối cùng:

```
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
```

Thiết lập hệ thống clock một lần nữa để chạy ở 40MHz. Thêm các dòng sau:

```
SysCtlClockSet (SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
```

Và thêm dòng này sau:

```
SysCtlPeripheralEnable (SYSCTL_PERIPH_ADC0);
```

Đối với các phòng thí nghiệm này, chúng ta sẽ chỉ cho phép các ADC12 để chạy ở tốc độ mặc định của 1Msps.

Thêm một dòng cho khoảng cách và thêm dòng mã này:

```
ADCSequenceConfigure (ADC0_BASE, 1,
ADC_TRIGGER_PROCESSOR, 0);
```

Thêm ba dòng sau đây sau các dòng trên:

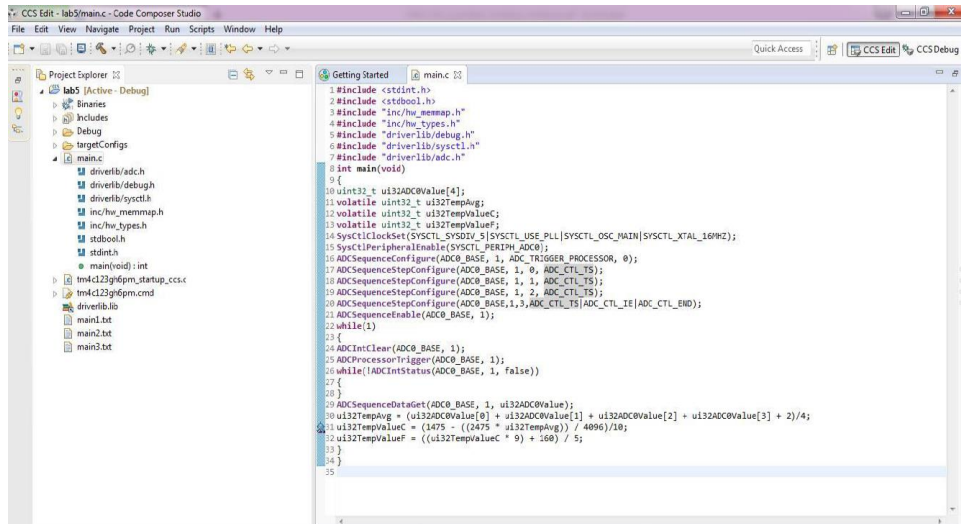
```
ADCSequenceStepConfigure (ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure (ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure (ADC0_BASE, 1, 2, ADC_CTL_TS);
```

Thêm một dòng cho khoảng cách và nhập ba dòng mã:

```
while (1)
{ }
```

Ta có một chương trình hoàn chỉnh như sau :

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
int main(void)
{
    uint32_t ui32ADC0Value[4];
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCSequenceConfigure(ADC0_BASE, 1,
        ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0,
        ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1,
        ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2,
        ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS|
        ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);
    while(1)
    {
        ADCIntClear(ADC0_BASE, 1);
        ADCProcessorTrigger(ADC0_BASE, 1);
        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {}
        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        ui32TempAvg = (ui32ADC0Value[0] +
            ui32ADC0Value[1] + ui32ADC0Value[2] +
            ui32ADC0Value[3] + 2)/4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg))
            / 4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) /
            5;
    }
}
```

Kết quả thí nghiệm:

CCS Debug - lab5/main.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

(x)= Variables Expressions Registers

Expression	Type	Value	Address
(x)= ui32TempValueC	unsigned int	23	0x20000054
(x)= ui32TempValueF	unsigned int	73	0x20000058
Add new expression			

Quick Access CCS Edit CCS Debug

(x)= Variables Expressions Registers

Expression	Type	Value	Address
(x)= ui32TempValueC	unsigned int	23	0x20000054
(x)= ui32TempValueF	unsigned int	73	0x20000058
ui32ADCOValue	unsigned int[4]	0x20000040	0x20000040
(x)= ui32TempAvg	unsigned int	2053	0x20000050
Add new expression			

CHƯƠNG 6

Lab 6: Low Power Modes

Trong bài thí nghiệm này, chúng ta sẽ sử dụng các module ngủ đông để đặt các thiết bị trong một trạng thái năng lượng thấp. Sau đó chúng ta sẽ thiết lập chúng hoạt động lại từ cả pin đánh thức và Real-Time Clock (RTC). Chúng ta cũng sẽ đo và vẽ để xem những ảnh hưởng của chế độ năng lượng khác nhau.

tm4c123gh6pm.h: Các định nghĩa cho các gián đoạn và đăng ký thiết bị Tiva C Series trên bảng LaunchPad.

interrupt.h: Định nghĩa và mở rộng cho NVIC Controller (Interrupt) API

driverLib: Điều này bao gồm các chức năng API như IntEnable và IntPrioritySet.

timer.h: Định nghĩa và macro cho hẹn giờ API của driverLib. Điều này bao gồm API các chức năng như TimerConfigure và TimerLoadSet.

Thiết lập xung clock

```
SysCtlClockSet(SYSCTL_SYSDIV_5/SYSCTL_USE_PLL/SYSCTL_XTAL_16MHZ/SYSCTL_OSC_MAIN);
```

Cấu hình GPIO

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3, 0x08);
```

Chúng ta cần phải đi vào chế độ ngủ đông. Các HibernateRequest () chức năng yêu cầu các mô-đun Hibernation để vô hiệu hóa các điều bên ngoài, loại bỏ điện từ xử lý và tắt cả các thiết bị ngoại vi. Nếu điện áp pin là thấp (hoặc tắt) hoặc nếu ngắt đang được phục vụ, chuyển sang chế độ ngủ đông có thể bị trì hoãn. Nếu điện áp pin không có mặt, việc chuyển đổi sẽ không bao giờ xảy ra.

Ta có chương trình hoàn chỉnh:

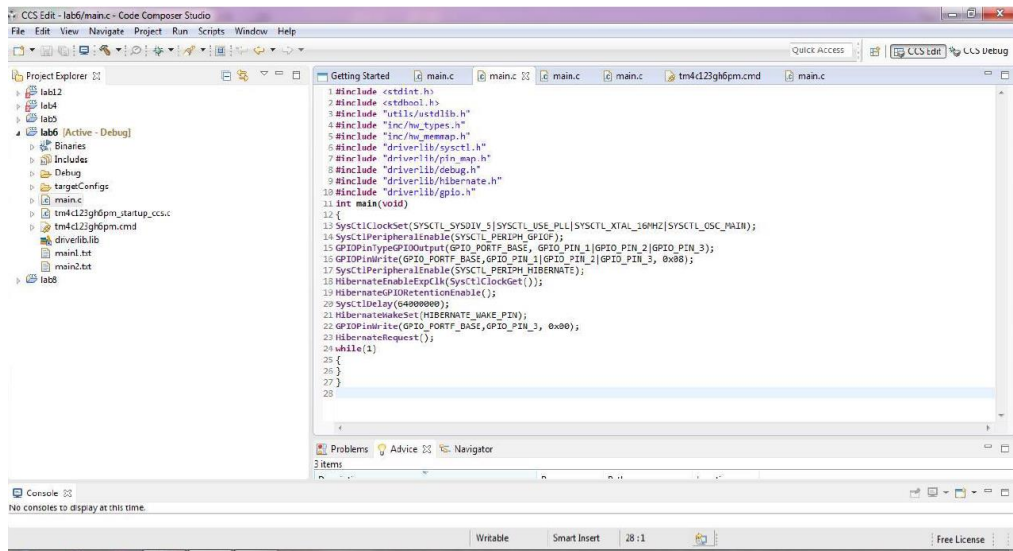
```
#include <stdint.h>
#include <stdbool.h>
#include "utils/ustdlib.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/debug.h"
#include "driverlib/hibernate.h"
#include "driverlib/gpio.h"

int main(void) { SysCtlClockSet(SYSCTL_SYSDIV_5/SYSCTL_USE_PLL/SYSCTL_XTAL_16MHZ/SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
```

```

GPIO_PIN_1/GPIO_PIN_2/GPIO_PIN_3);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1/GPIO_PIN_
2/GPIO_PIN_3, 0x08);
SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);
HibernateEnableExpClk(SysCtlClockGet()); HibernateGPIORetentionEnable();
SysCtlDelay(64000000); HibernateWakeSet(HIBERNATE_WAKE_PIN);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, 0x00);
HibernateRequest(); while(1)
{}
}

```



Kết quả thí nghiệm:





CHƯƠNG 7

Lab 7: USB

Trong bài thí nghiệm này, Chúng ta sẽ thử nghiệm với việc gửi dữ liệu qua lại trên một kết nối với cổng USB.

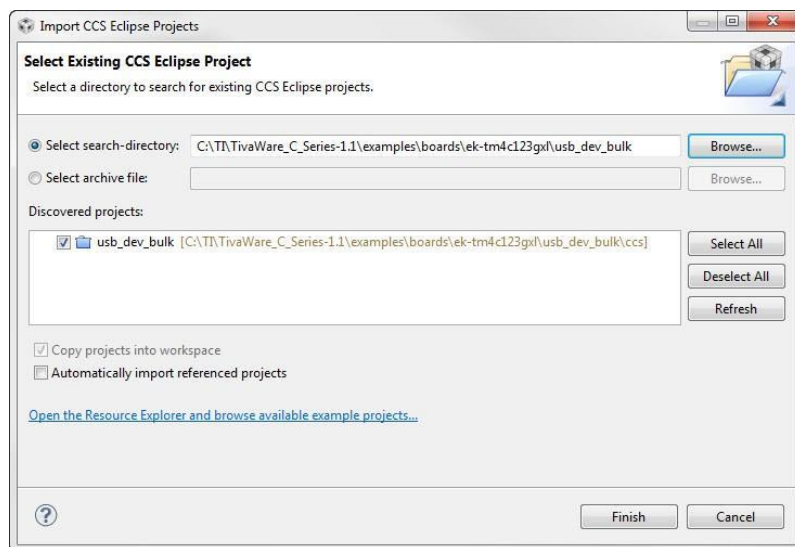
Có bốn loại chuyển dữ liệu / thiết bị đầu cuối trong kết nối USB: chuyển điều khiển (với lệnh và trạng thái), ngắt chuyển (để nhanh chóng nhận được sự chú ý của host), truyền đẳng thời (chuyển liên tục và định kỳ các dữ liệu) và chuyển số lượng lớn (chuyển lớn, dữ liệu bùng phát).

Trước khi chúng ta bắt đầu với đoạn code này, chúng ta hãy cài usb_bulk_example cho quá trình thí nghiệm. Chúng ta sẽ sử dụng một ứng dụng dòng lệnh máy chủ Windows để chuyển dây qua kết nối USB cho Ban LaunchPad. Chương trình này sẽ thay đổi từ chữ thường thành hoa và ngược lại, sau đó chuyển dữ liệu lại cho host.

Lấy Project mẫu

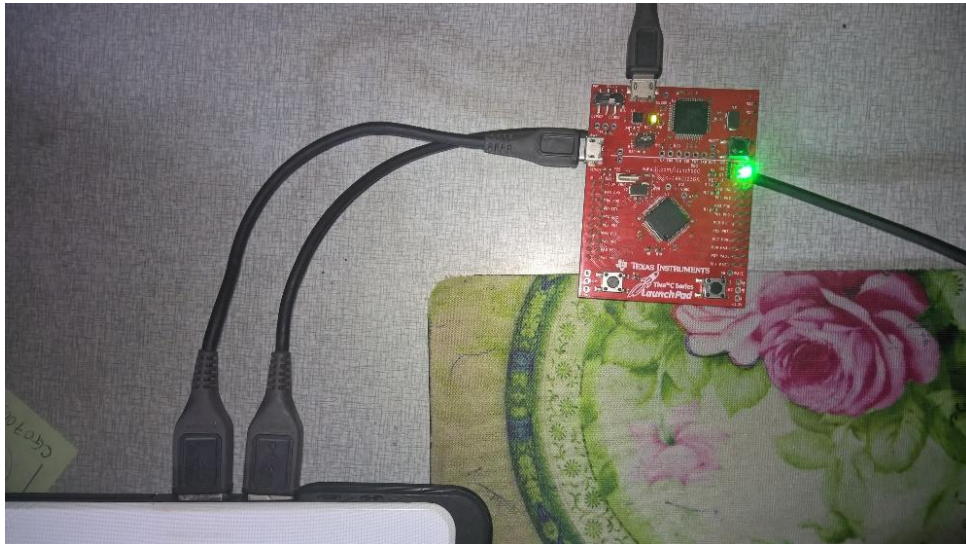
usb_bulk_example project là một ví dụ mẫu của tivaware. Khi bạn nhập project, lưu ý rằng nó sẽ được tự động sao chép vào không gian làm việc của bạn, bảo quản các tập tin ban đầu. Nếu bạn muốn truy cập các tập tin của project thông qua Windows Explorer, file bạn đang làm việc trên trong thư mục không gian làm việc của bạn, không phải là thư mục TivaWare. Nếu bạn xóa các project trong CCS, các project nhập khẩu vẫn sẽ ở trong không gian làm việc của bạn trừ khi bạn nói với các hộp thoại để xóa các tập tin từ ổ đĩa.

Tiến hành cài đặt như hình dưới, sau đó click ► *Finish*



trong Code Composer Studio, nếu usb_dev_bulk.c không có sẵn, expand usb_dev_bulk project trong Project Explorer pane và double-click vào usb_dev_bulk.c để mở.

Chương trình có 5 lựa chọn:



SysTickIntHandler – an ISR that handles interrupts from the SysTick timer to keep track of “time”.

EchoNewDataToHost – a routine that takes the received data from a buffer, flips the case and sends it to the USB port for transmission.

TxHandler – an ISR that will report when the USB transmit process is complete.

RxHandler – an ISR that handles the interaction with the incoming data, then calls the EchoNewDataToHost routine.

main() – chủ yếu là khởi tạo, nhong một vòng lặp trong khi giữ một mắt trên các số byte chuyên

chú ý rằng UARTprintf() APIs sprinkled throughout the code. This technique “instruments” the code, allowing us to monitor its status via a serial port.

chạy chương trình để kết nối với Stellaris Virtual Serial Port. Sắp xếp các cửa sổ thiết bị đầu cuối để nó chiếm không quá một phần tư của màn hình của bạn và vị trí của nó ở phía trên bên trái của màn hình của bạn.

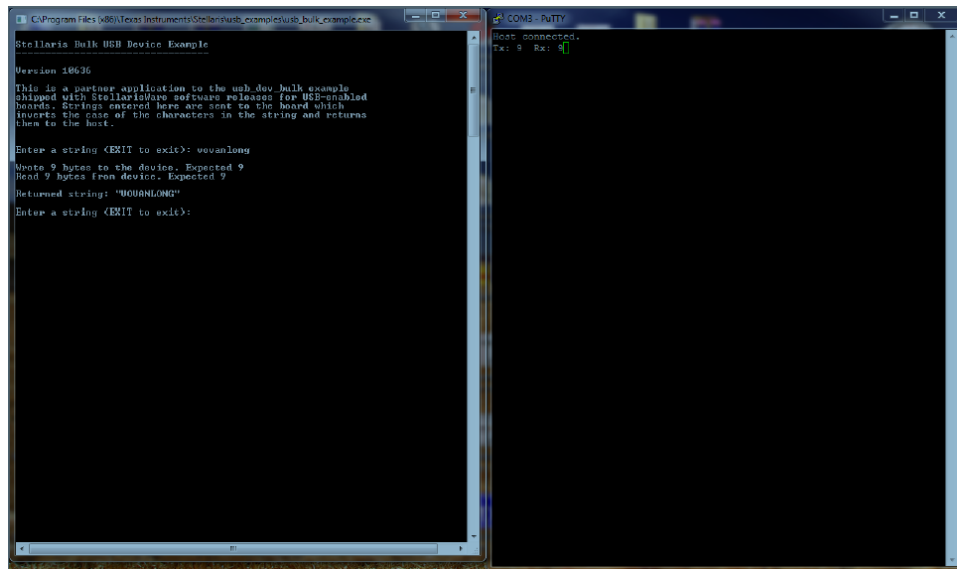
Thay đổi kích thước CCS để nó chiếm nửa dưới của màn hình của bạn. Nhấp vào nút Debug để xây dựng và tải về mã và kết nối lại với LaunchPad của bạn. Khởi mã bằng cách nhấn vào nút Resume.

Hãy bắt đầu USB Bulk Example Windows như bước 5. Đặt cửa sổ ở góc trên bên phải màn hình của bạn.

Lưu ý các trạng thái trên màn hình thiết bị đầu cuối của bạn và gõ một cái gì đó, giống như trong màn hình vào USB Bulk và nhấn enter. Lưu ý rằng chương trình thiết bị đầu cuối sẽ hiển thị

```
USB Bulk Example
Stellaris Bulk USB Device Example
Version 9107
This is a partner application to the usb_dev_bulk example
shipped with StellarisWare software releases for USB-enabled
boards. Strings entered here are sent to the board which
inverts the case of the characters in the string and returns
then to the host.
Enter a string <EXIT to exit>:
```

Kết quả thí nghiệm:



The image shows two side-by-side windows. The left window, titled 'C:\Program Files (x86)\Texas Instruments\Stellaris\usb_comp\usb_bulk_example.exe', contains the following text:

```
Stellaris Bulk USB Device Example  
  
Version 18626  
  
This is a partner application to the usb_dev_bulk example  
shipped with StellarisWare software releases for USB-enabled  
boards. Strings entered here are sent to the board which  
inverts the case of the characters in the string and returns  
them to the host.  
  
Enter a string (EXIT to exit): weuanlong  
Wrote 9 bytes to the device. Expected 9  
Read 9 bytes from device. Expected 9  
Returned string: "MOU8ANL0NG"  
Enter a string (EXIT to exit):
```

The right window, titled 'COM3 - PuTTY', shows the serial communication results:

```
Host connected.  
Tx: 9 Rx: 9
```

CHƯƠNG 8

LAB 8: Bộ nhớ và MPU

Ghi dữ liệu vào bộ nhớ FLASH in-system.

Đọc/ ghi EEPROM

Sử dụng MPU

Bit-banding

Bên trong hàm main(), xung nhịp hệ thống được cấu hình ở 40MHz, các pin kết nối với 3 LED ở chế độ xuất, và các LED này đều ở trạng thái tắt, chờ khoảng 2s. Sau đó bước vào vòng lặp vô tận.

Build project: chỉ build project mà không download chương trình vào bộ nhớ của Tiva TM4C.

Mở file lab8.map

Section MEMORY CONFIGURATION và SEGMENT ALLOCATION MAP:

MEMORY CONFIGURATION

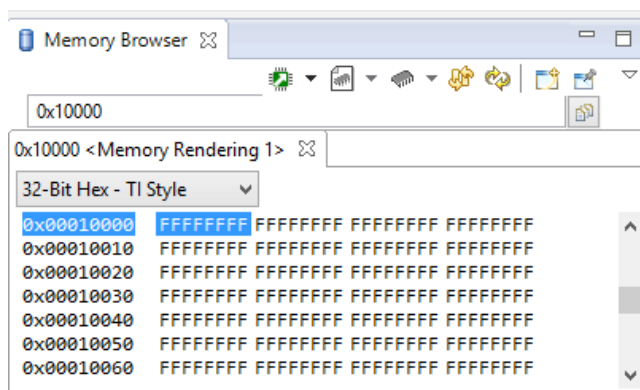
name	origin	length	used	unused	attr	fill
FLASH	00000000	00040000	000007a6	0003f85a	R X	
SRAM	20000000	00008000	00000214	00007dec	RW X	

SEGMENT ALLOCATION MAP

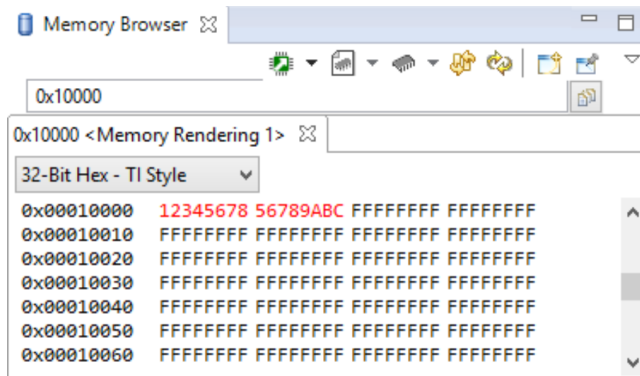
run origin	load origin	length	init length	attrs	members
00000000	00000000	000007a8	000007a8	r-x	
00000000	00000000	0000026c	0000026c	r--	.intvecs
0000026c	0000026c	0000051a	0000051a	r-x	.text
00000788	00000788	00000020	00000020	r--	.cinit
20000000	20000000	00000200	00000000	rw-	
20000000	20000000	00000200	00000000	rw-	.stack
20000200	20000200	00000014	00000014	rw-	
20000200	20000200	00000014	00000014	rw-	.data

Dung lượng bộ nhớ flash sử dụng là 0x07a8 bắt đầu ở 0x0.

Resume và kết quả trong Memory Browser tại địa chỉ 0x10000:



Resume để tiếp tục chạy:



Dữ liệu trong các biến được ghi vào ở địa chỉ bắt đầu 0x10000, LED đỏ bật sáng. Sau khi hàm EEPROMMassErase() được gọi, giá trị trong bộ nhớ sẽ chứa F-s:

Name	Type	Value	Location
pui32Data	unsigned int[2]	0x200001E8	0x200001E8
[0]	unsigned int	0x12345678 (Hex)	0x200001E8
[1]	unsigned int	0x56789ABC (Hex)	0x200001EC
pui32Read	unsigned int[2]	0x200001F0	0x200001F0
[0]	unsigned int	0xFFFFFFFF (Hex)	0x200001F0
[1]	unsigned int	0xFFFFFFFF (Hex)	0x200001F4

Sau khi Resume, LED xanh dương bật sáng, và dữ liệu được cập nhật:

Name	Type	Value	Location
pui32Data	unsigned int[2]	0x200001E8	0x200001E8
[0]	unsigned int	0x12345678 (Hex)	0x200001E8
[1]	unsigned int	0x56789ABC (Hex)	0x200001EC
pui32Read	unsigned int[2]	0x200001F0	0x200001F0
[0]	unsigned int	0x12345678 (Hex)	0x200001F0
[1]	unsigned int	0x56789ABC (Hex)	0x200001F4

BIT-BANDING

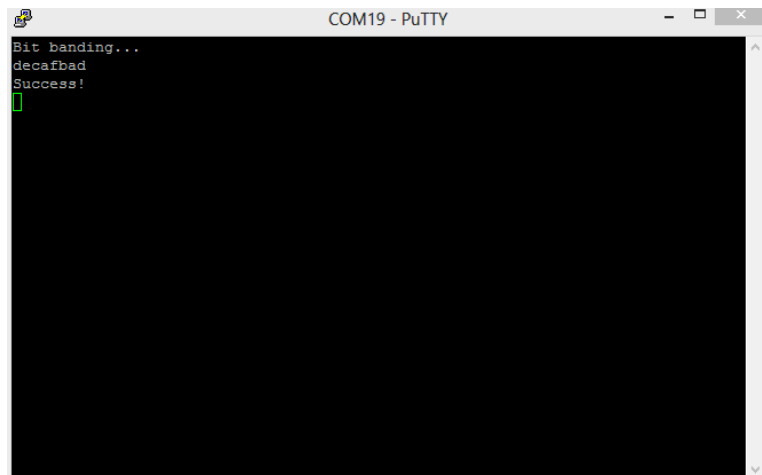
Import project bit-banding.

Mở file bitband.c : UART đọc sử dụng để xuất kết quả.

Build và nạp chương trình.

Click Resume.

Kết quả thí nghiệm:



```
COM19 - PuTTY
Bit banding...
decafbad
Success!
█
```

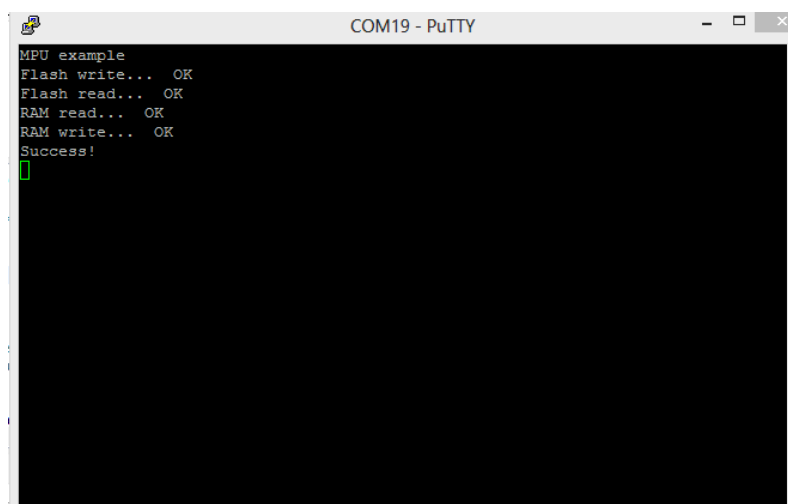
MEMORY PROTECTION UNIT (MPU)

Import project mpu_fault

Mở file mpu_fault.c

Build và nạp chương trình.

Kết quả thí nghiệm:



```
COM19 - PuTTY
MPU example
Flash write... OK
Flash read... OK
RAM read... OK
RAM write... OK
Success!
█
```

CHƯƠNG 9

LAB 9: FPU

Tìm hiểu về FPU trên TM4C123G.

Import project lab9

Mở file main.c, thêm vào đoạn code sau:

```
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/fpu.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
#define SERIES_LENGTH 100
float gSeriesData[SERIES_LENGTH];
int32_t i32DataCount = 0;
int main(void)
{
    float fRadians;
    ROM_FPULazyStackingEnable();
    ROM_FPUEnable();
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
        SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);
    fRadians = ((2 * M_PI) / SERIES_LENGTH);
    while(i32DataCount < SERIES_LENGTH)
    {
        gSeriesData[i32DataCount] = sinf(fRadians * i32DataCount);
        i32DataCount++;
    }
    while(1)
    {
    }
}
```

Bên trong hàm main ():

Khai báo một biến có kiểu *float* để tính giá trị *sin*.

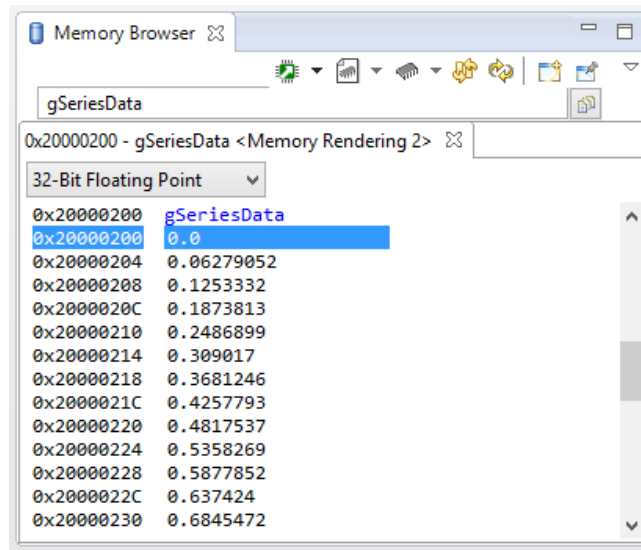
Enable Lazy Stacking.

Cấu hình xung nhịp hệ thống 50MHz.

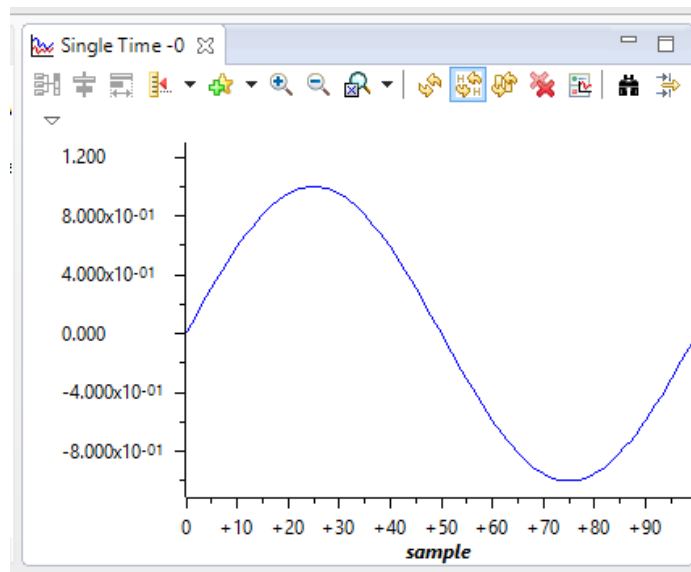
Vòng lặp while tính giá trị của sin và lưu vào mảng.

Build và nạp chương trình.

Dữ liệu của biến gSeriesData trong cửa sổ Memory Browser:



Dùng công cụ Graph:



Thời gian tính toán floating-point:

[illegible]

CHƯƠNG 10

LAB 10: Graphics library

Trong bài lab này chúng ta sẽ liên kết board và màn hình kenTec. Viết một chương trình sử dụng thư viện graphic.

Màn hình KenTec:



Copy đoạn code sau vào đầu của file pic.c trong library:

```
#include "glib/glib.h"

const unsigned char g_pucImage[] = { IMAGE_FMT_4BPP_COMP, 96, 0,
64, 0,
15, 0x00, 0x02, 0x00, 0x18, 0x1a, 0x19, 0x28, 0x2a, 0x28, 0x38,
0x3a, 0x38, 0x44, 0x46, 0x44, 0x54, 0x57, 0x55, 0x62, 0x65, 0x63,
0x72, 0x75, 0x73, 0x81, 0x84, 0x82, 0x93, 0x96, 0x94, 0xa2, 0xa5, 0xa3,
0xb3, 0xb6, 0xb4, 0xc4, 0xc7, 0xc5, 0xd7, 0xda, 0xd8, 0xe8, 0xeb, 0xe9,
0xf4, 0xf8, 0xf5,
0xff, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07,
0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
0x07, 0x07, 0xfc, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x03, 0x77, 0x23, 0x77,
0x77, 0xe9, 0x77, 0x78, 0x70, 0x07, 0x07, 0xc1, 0x77, 0x2c, 0x04, 0xde, 0xee,
0xee, 0xee, 0xe9, 0x3c, 0xee, 0xa1, 0x07, 0x07, 0x77, 0x2c, 0x03, 0xcf, 0x00,
0xee, 0xee, 0xee, 0xef, 0xee, 0xef, 0xfe, 0xa0, 0xf0, 0x07, 0x07, 0x77, 0x2c,
0x03, 0xcf, 0xee, 0xee, 0x4f, 0xee, 0xe9, 0xee, 0xa0, 0x07, 0x07, 0x77, 0x2c,
0x04, 0x03, 0xcf, 0xee, 0xee, 0xee, 0xe9, 0xee, 0x90, 0xf0, 0x07, 0x07, 0x77,
0x2c, 0x03, 0xcf, 0xee, 0xee, 0x4f, 0xee, 0xe9, 0xee, 0x90, 0x07, 0x07, 0x77,
0x2c, 0x04, 0x03, 0xcf, many, many more lines of this data ...
0x77, 0x2c, 0x19, 0xfe, 0xee, 0xef, 0x03, 0xee, 0xee, 0xee, 0xee, 0xfb, 0x20,
0x07, 0x07, 0xc1, 0x77, 0x2c, 0x05, 0xdf, 0xee, 0xee, 0xee, 0xe9, 0x78, 0xf9,
0x07, 0x07, 0x77, 0x2d, 0x01, 0x8d, 0xee, 0x2f, 0xee, 0xee, 0xe9, 0xf7, 0x07,
0x07, 0x77, 0x2e, 0x00, 0x39, 0xef, 0xee, 0xee, 0xee, 0xee, 0xf7, 0xf0,
0x07, 0x07, 0x77, 0x2e, 0x06, 0xdf, 0xee, 0xee, 0x0f, 0xee, 0xee, 0xee, 0xf6,
0x07, 0x07, 0x77, 0x2f, 0x01, 0x7d, 0xfe, 0xee, 0xee, 0xee, 0xee, 0xf7, 0x07,
0xe0, 0x07, 0x77, 0x2f, 0x17, 0xdf, 0xee, 0xee, 0xee, 0x3c, 0xee, 0xf7, 0x07,
0x07, 0x77, 0x2f, 0x01, 0x7d, 0x03, 0xee, 0xee, 0xee, 0xee, 0xf9, 0x10, 0x07,
0x07, 0xc0, 0x77, 0x2f, 0x05, 0xad, 0xee, 0xfe, 0xee, 0xfc, 0x78, 0x20, 0x07,
0x07, 0x77, 0x2f, 0x00, 0x27, 0x9d, 0x0f, 0xed, 0xee, 0xec, 0x40, 0x07, 0x07,
0x77, 0x2f, 0x01, 0x00, 0x00, 0x28, 0x9a, 0xcc, 0xa9, 0x30, 0x07, 0xff, 0x07,
0x77, 0x2f, 0x07, 0x07, 0x07, 0x07, 0x07, 0xc0, 0x07, 0x07, };
```

Mở file main.c ta soạn đoạn code sau:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "glib/glib.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"
extern const unsigned char g_pucImage[]; tContext sContext; tRectangle sRect;
void ClrScreen(void);
#ifdef DEBUG void __error__(char *pcFilename, unsigned long ulLine) { }
#endif
int main(void) {
SysCtlClockSet(SYSCTL_SYSDIV_4/SYSCTL_USE_PLL/SYSCTL_OSC_MAIN/SYSCTL_XTAL_16MHZ);
Kentec320x240x16_SSD2119Init();
GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119); ClrScreen();
GrImageDraw(&sContext, g_pucImage, 0, 0);
GrFlush(&sContext);
SysCtlDelay(SysCtlClockGet()); // Later lab steps go between here
// and here ClrScreen();
while(1) { }
void ClrScreen() { sRect.sXMin = 0; sRect.sYMin = 0; sRect.sXMax = 319;
sRect.sYMax = 239;
GrContextForegroundSet(&sContext, ClrBlack); GrRectFill(&sContext, &sRect);
GrFlush(&sContext); }
```

Kết quả thí nghiệm:



CHƯƠNG 11

LAB 11 : SPI Bus and the Olimex LED BoosterPack

Led ma trận:



Bảng thông số:

Olimex Header Pin	Olimex Function		LaunchPad Header Pin	LM4F120H5QR Pin Name	Pin Function
J1-7	SR_SCK	→	J2-10	PA2	SSI0CLK
J1-6	SR_LATCH	→	J2-9	PA3	SSI0Fss
J2-7	SR_DATA_IN	→	J1-8	PA5	SSI0Tx
J1-2	A_IN	→	J2-3	PE0	AIN3
J1-3	BUZ_PIN1	→	J1-9	PA6	GPIO
J1-4	BUZ_PIN2	→	J1-10	PA7	GPIO
J2-1	Ground	→	J2-1	Ground	-
J1-1	Vcc	→	J1-1	Vcc	-

Dưới đây là đoạn code trong file main.c:

```
#include "inc/hw_memmap.h"
#include "inc/hw_ssi.h"
#include "inc/hw_types.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#define NUM_SSI_DATA 8 const unsigned char ulDataTx[NUM_SSI_DATA] =
{0x88, 0xF8, 0xF8, 0x88, 0x01, 0x1F, 0x1F, 0x01};
unsigned short g_pusTxBuffer[16];
// Bit-wise reverses a number. unsigned char Reverse(unsigned char ucNumber)
{ unsigned short ucIndex; unsigned short ucReversedNumber = 0;
for(ucIndex=0; ucIndex<8; ucIndex++) { ucReversedNumber =
```

```

ucReversedNumber << 1; ucReversedNumber /= ((1 << ucIndex) &
ucNumber) >> ucIndex; } return ucReversedNumber; }
int main(void) { unsigned long ulindex; unsigned long ulData;
SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
GPIOPinConfigure(GPIO_PA2_SSI0CLK);
GPIOPinConfigure(GPIO_PA3_SSI0FSS);
GPIOPinConfigure(GPIO_PA5_SSI0TX);
GPIOPinTypeSSI(GPIO_PORTA_BASE,GPIO_PIN_5/GPIO_PIN_3/GPIO_PIN_2);

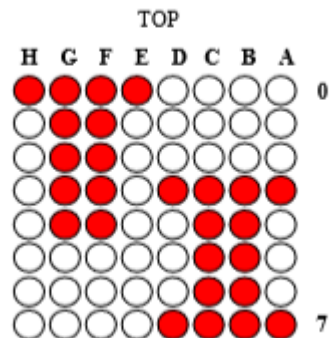
```

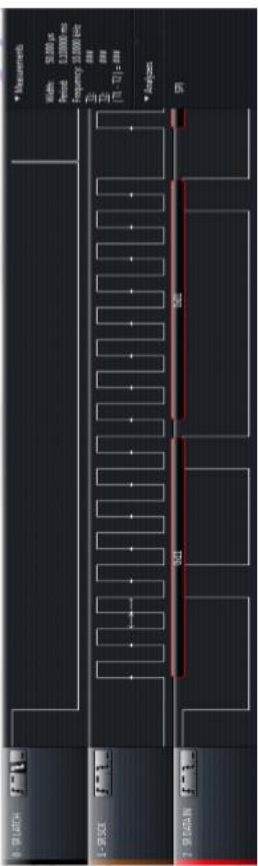
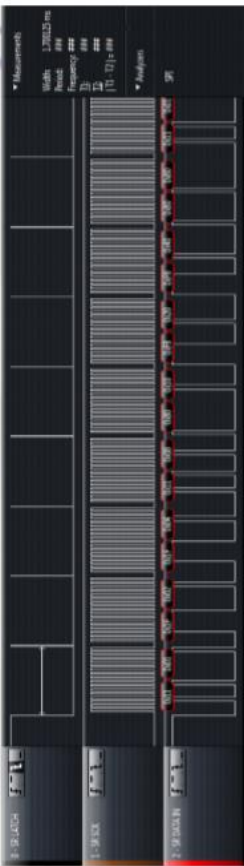
```

SSISetExpClk(SSIO_BASE,SysCtlClockGet(),SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER,10000,16); SSISetEnable(SSIO_BASE);
while(1) { for(ulindex = 0; ulindex < NUM_SSI_DATA; ulindex++) {
ulData = (Reverse(ulDataTx[ulindex]) << 8) + (1 << ulindex);
SSIPut(SSIO_BASE, ulData); while(SSIBusy(SSIO_BASE)) { }
}
}
}

```

{A7-0, B7-0, C7-0, D7-0, E7-0, F7-0, G7-0, H7-0}





CHƯƠNG 12

LAB 12: UART

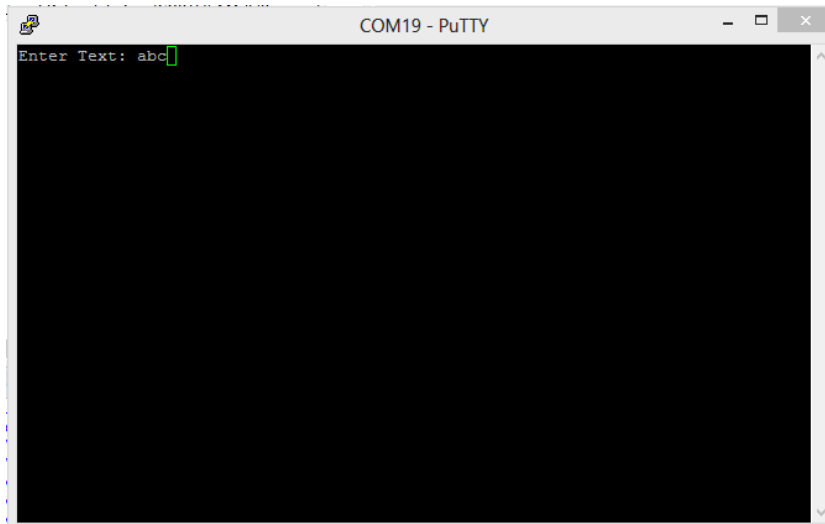
Thiết lập và gửi nhận dữ liệu UART dùng polling và interrupt.

Import project lab12

Mở file main.c

```
#include <stdint.h>
#include <stdbool.h>
#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) {
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL /
        SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    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));
    }
}
```

Trong hàm main():
Cấu hình xung nhịp hệ thống.
Cấu hình các chân truyền và nhận UART.
Cấu hình tham số UART: 115200, 8-1-N.
Build và nạp chương trình.
Kết quả:



Interrupt

Thêm đoạn code sau vào phần *include header*: `#include "inc/hw_ints.h"`
`#include "driverlib/interrupt.h"`

Thêm vào đoạn code sau ngay sau hàm `UARTConfigSetExpClk()`:

```
IntMasterEnable();  
IntEnable(INT_UART0);  
UARTIntEnable(UART0_BASE, UART_INT_RX /  
UART_INT_RT);
```

Cấu hình GPIO pin cho LED:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);  
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,  
GPIO_PIN_2);
```

Bên trong vòng lặp `while`, xóa bỏ dòng:

```
if  
(UARTCharsAvail(UART0_BASE))UARTCharPut(UART0_BASE, UARTCharGet(UAR  
T0_BASE));
```

Lưu lại file `main.c`.

Thêm vào đoạn code sau để xử lý ngắt UART:

```
void UARTIntHandler(void)  
{
```

```

uint32_t ui32Status;
ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts
while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
{
    UARTCharPutNonBlocking(UART0_BASE,
    UARTCharGetNonBlocking(UART0_BASE));
    //echo character
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink
    LED
    SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
}
}

```

Mở file startup_ccs.c để thêm vào khai báo trình xử lý UART:

```
extern void UARTIntHandler(void);
```

Thay đổi trình xử lý ngắt mặc định bằng trình xử lý ngắt UART ở UART0 interrupt vector:

```
UARTIntHandler, // UART0 Rx and Tx
```

Build và nạp chương trình.

Kết quả: ký tự nhập vào hiển thị trên terminal và LED nhấp khi nhận được dữ liệu.

