Lab 7 - UART

Objective

In this lab you will use the LaunchPad board, and the virtual serial port that runs over the debug USB cable.

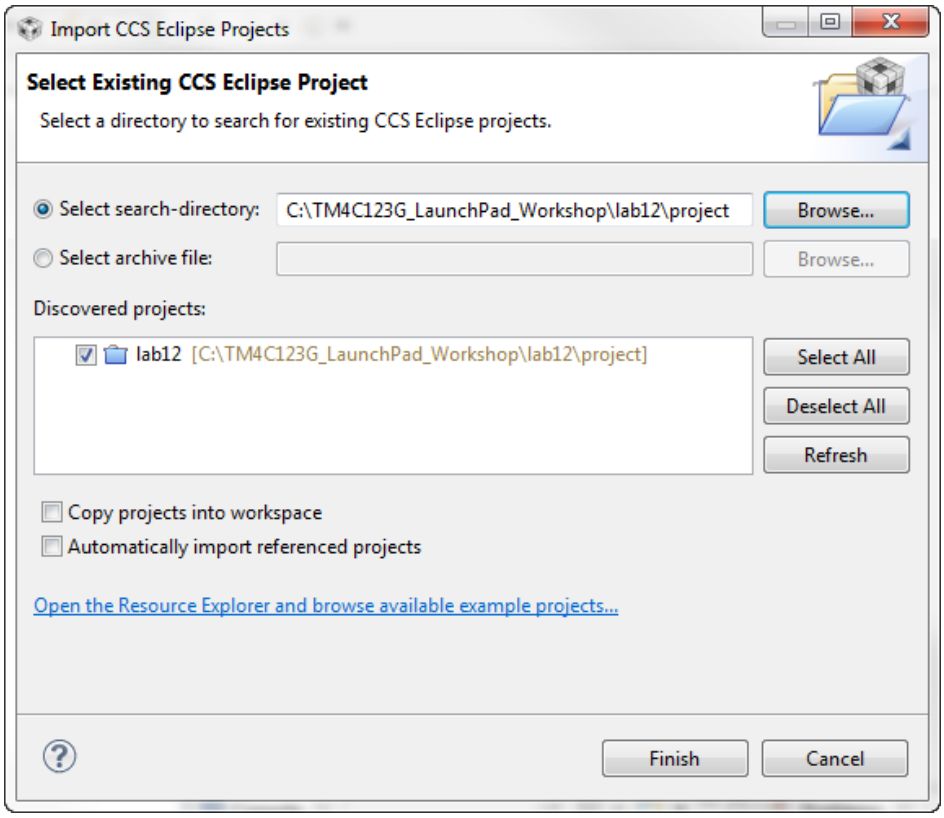
Procedure

Import Lab7

1. We have already created the Lab7 project for you with a **`main.c`** file, a startup file, and all the necessary project and build options set.

   ► Maximize Code Composer and click Project → Import Existing CCS Eclipse Project. Make the settings shown below and click Finish.

   Make sure that the “Copy projects into workspace” checkbox is **unchecked**.



2. ► Expand the project by clicking on the + or ▷ next to Lab7 in the Project Explorer pane. Double-click on **`main.c`** to open it for review. The code looks like the next page:

3. You need these two: **TARGET_IS_BLIZZARD_RA2** and **PART_**TM4C1233H6PM **included as** pre-defined variables.

4. The former one can help you correctly use functions which reside in ROM ; the later one is telling it the correct pin map. that's why you need these info. predefined.

```
#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));
    }

}
```

5. In **`main()`**, notice the initialization sequence for using the UART:
   - Set up the system clock
   - Enable the UART0 and GPIOA peripherals (the UART pins are on GPIO Port A)
   - Configure the pins for the receiver and transmitter using GPIOPinConfigure
   - Initialize the parameters for the UART: 115200, 8-1-N
   - Use simple "**`UARTCharPut()`**" calls to create a prompt.
   - An infinite loop. In this loop, if there is a character in the receiver, it is read, and then written to the transmitter. This echos what you type in the terminal window.

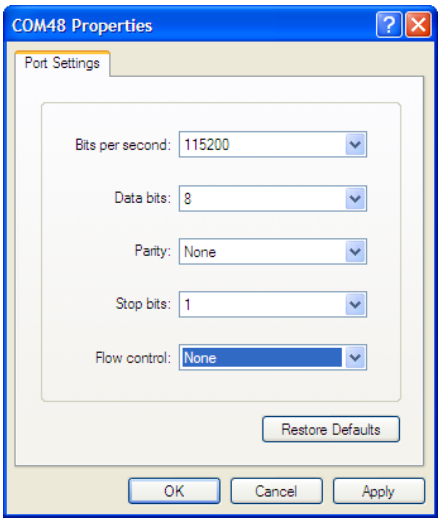
Build, Download, and Run the UART Example Code

You can also use the built-in Terminal in CCS for this section.

6. ► Click the Debug button to build and download your program to the TM4C123GH6PM flash memory.

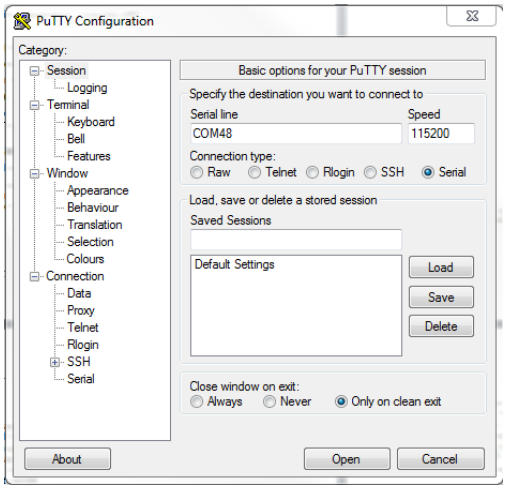
   We can communicate with the board through the UART, which is connected as a virtual serial port through the emulator USB connection. You can find the COM port number for this serial port back in chapter one of this workbook on page 18 or 19.

   In WinXP, ► open HyperTerminal by clicking Start → Run…, then type `hypertrm` in the Open: box and click OK. Pick any name you like for your connection and click OK. In the next dialog box, change the Connect using: selection to COM##, where ## is the COM port number you noted earlier from Device Manager. Click OK. Make the selections shown below and click OK.

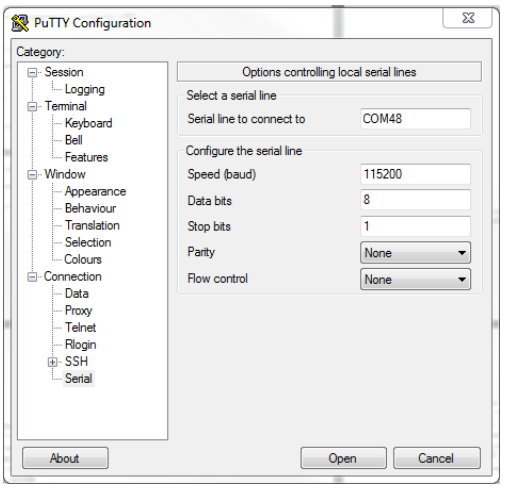


   When the terminal window opens click the Resume button in CCS, then type some characters and you should see the characters echoed into the terminal window. Skip to step 8.

7. In **Win7**, ► double-click on **`putty.exe`**. Make the settings shown below and then click Open. Your COM port number will be the one you noted earlier in chapter one.



   When the terminal window opens ► click the Resume button in CCS, then type some characters and you should see the characters echoed into the terminal window.



Using UART Interrupts

Instead of continually polling for characters, we'll make some modifications to our code to allow the use of interrupts to receive and transmit characters. In the first part of this lab, the only indication we had that our code was running was to open the terminal window to type characters and see them echoed back. In this part of the lab, we'll add a visual indicator to show that we received and transmitted a character. So we'll need to add code similar to previous labs to blink the LED inside the interrupt handler.

8. First, let's add the code in **`main()`** to enable the UART interrupts we want to handle. ► Click on the Terminate button to return to the CCS Edit perspective. We need to add two additional header files at the top of the file:

```
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
```

9. Now we need to add the code to enable processor interrupts, then enable the UART interrupt, and then select which individual UART interrupts to enable. We will select receiver interrupts (RX) and receiver timeout interrupts (RT). The receiver interrupt is generated when a single character has been received (when FIFO is disabled) or when the specified FIFO level has been reached (when FIFO is enabled). The receiver timeout interrupt is generated when a character has been received, and a second character has not been received within a 32-bit period. ► Add the following code just below the **`UARTConfigSetExpClk()`** function call:

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

10. We also need to initialize the GPIO peripheral and pin for the LED. ► Just before the function **`UARTConfigSetExpClk()`** is called, add these two lines:

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

11. ► Finally, we can create an empty **`while(1)`** loop at the end of main by commenting out the line of code that's already there:

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

12. ► Save the changes you made to `main.c` (but leave it open for making additional edits).

13. Now we need to write the UART interrupt handler. The interrupt handler needs to read the UART interrupt status register to know which specific interrupt event(s) just occurred. This value is then used to clear the interrupt status bits (we only enabled RX and RT interrupts, so those are the only possible sources for the interrupt). The next step is to receive and transmit all the characters that have been received. After each character is "echoed" to the terminal, the LED is blinked for about 1 millisecond. ► Insert this code below the include statements and above **`main()`**:

```
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
    }
}
```

14. We're almost done. We've added all the code we need. The final step is to insert the address of the UART interrupt handler into the interrupt vector table. ► Open the **`startup_ccs.c`** file. Just below the prototype for `_c_int00(void)`, add the UART interrupt handler prototype:

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

15. On about line 68, you'll find the interrupt vector table entry for "UART0 Rx and Tx". It's just below the entry for "GPIO Port E". The default interrupt handler is named `IntDefaultHandler`. ► Replace this name with `UARTIntHandler` so the line looks like:

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

16. Save your work. Your **main.c** code should look like this.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

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
    }
}

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);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //enable pin for LED PF2

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_UART0); //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX interrupts

    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) //let interrupt handler do the UART echo function
    {
//      if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
    }

}
```

17. ► Click the Debug button to build and download your program to the TM4C123GH6PM memory.
18. ► If you've closed it, open Hyperterminal or puTTY, and configure it as before. **You can also use the built-in Terminal in CCS for this section.**
19. ► Click the Resume button. Type some characters and you should see the characters echoed into the terminal window. Note the LED.
20. ► Close puTTY or HyperTerminal. Click the Terminate button to return to the CCS Edit perspective. ► Close the Lab7 project and minimize Code Composer Studio.

Follow the submission guideline to be awarded points for this Lab.

Task00: Execute the supplied code, display the temperatures in the built-in Graph Tool.

Task 01: Continuously display the temperature of the device (internal temperature sensor) on the a) hyperterminal, and b) GUI Composer (Temp Sensor) using a timer interrupt every 0.5 secs.

Task 02: Interaction/User Interface: Develop a user interface using UART to perform the following:

Enter the cmd: R: Red LED, G: Green LED, B: Blue LED, T: Temperature:

Based on the command (cmd) the program should turn ON Red LED when R is entered in the terminal, etc. Command of 'r' will turn off the Red LED.

Follow the submission guideline to be awarded points for this Lab.

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.
2. Create a Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named LABXX, with one document and one video link file for each lab, place modified c files named as LabXX-TYY.c.
3. If multiple c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).