# TI ARM Lab 6
# UART (without Interrupt)

National Science Foundation
Funded in part, by a grant from the National Science Foundation
DUE 1068182

## Acknowledgements
Developed by Craig Kief, Brian Zufelt, and Jacy Bitsoie at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. *Funded by the National Science Foundation (NSF).*

## Lab Summary
This lab introduces the concepts of the UART and how it is implemented on the ARM processor.

## Lab Goal
The goal of this lab is to continue to build upon the skills learned from previous labs. This lab helps the student to continue to gain new skills and insight on the C code syntax and how it is used in the TI supplementation of the ARM processor. Each of these labs will add upon the previous labs and it is the intention of the authors that students will build with each lab a better understanding of the ARM processor and basic C code and syntax. Even though these tutorials assume the student has not entered with a knowledge of C code, it is the desire that by the time the student completes the entire series of tutorials that they will have a sufficient knowledge of C code so as to be able to accomplish useful projects on the ARM processor.

## Learning Objectives
The student should begin to become familiar with the concept of the UART and ways to communicate through as serial interface such as Putty.

A Universal Asynchronous Receiver/Transmitter, abbreviated UART, is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as RS-232. In its simplest form, a UART can be thought of as a two wire system where one line is transmit and the other is receive. The parallel data is reformatted into a serial stream. The *universal* designation indicates that the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling etc.) are handled by a driver circuit external to the UART.

A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers such as the TI Stellaris ARM we are working with. Everyone that works commonly with hardware debugging is well aware of the power of something like a serial connection between a computer and a piece of hardware. It allows a designer to be able to see into the system that is being created and tested. In this case, the objective will not be to fully understand a UART (that can be done by using Google), it is to understand how to do a project with a UART on the Stellaris board and then how to communicate through that connection.

## Grading Criteria
N/A

### Time Required
Approximately one hour

### Lab Preparation
It is highly recommended that the student read through this entire procedure once before actually using it was a tutorial. By understanding the final goal, it will be easier to use this as a tutorial and as a learning guide. The other tricky part is related to how your software was loaded. While this series of tutorials was being developed, a new way to load the software was developed. This new method simplifies the linker and compiler options so that you no longer have to do the process as shown in tutorial 4.

### Equipment and Materials
It is assumed that the student has already completed prior labs and the software is installed properly.

| Software needed | Quantity |
|---|---|
| Install the required tutorial software from the autoinstaller located at http://www.cosmiac.org/Microcontrollers.html For this lab, the Putty software is also required. This is a free executable package that helps with a variety of serial capabilities. | 1 |
| Hardware needed | Quantity |
| The hardware required is the TI Stellaris LaunchPad Kit and the Digilent Orbit board | 1 |

### Additional References
Current TI ARM manuals found on TI web site: www.ti.com/lit/ug/spmu289a/spmu289a.pdf .

### Lab Procedure : Install/Connect board to computer

Plug in the supplied USB cable to the top of the Evaluation Kit and Digilent Orbit board as shown in Figure 1. Ensure the switch on the board is set to "DEBUG" and not "DEVICE". It is assumed that the evaluation board is properly installed and operating. Launch the Code Composer software. The easiest way to find it, is to go to Start, All programs and then look in the area identified in Figure 2.

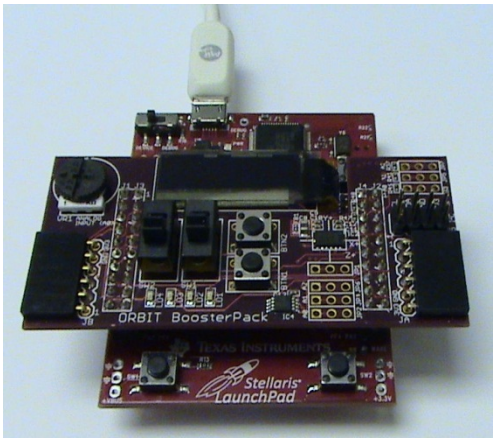


Figure 1. Stellaris and Orbit Combination

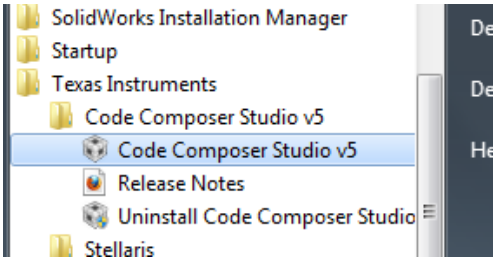


Figure 2. Code Composer

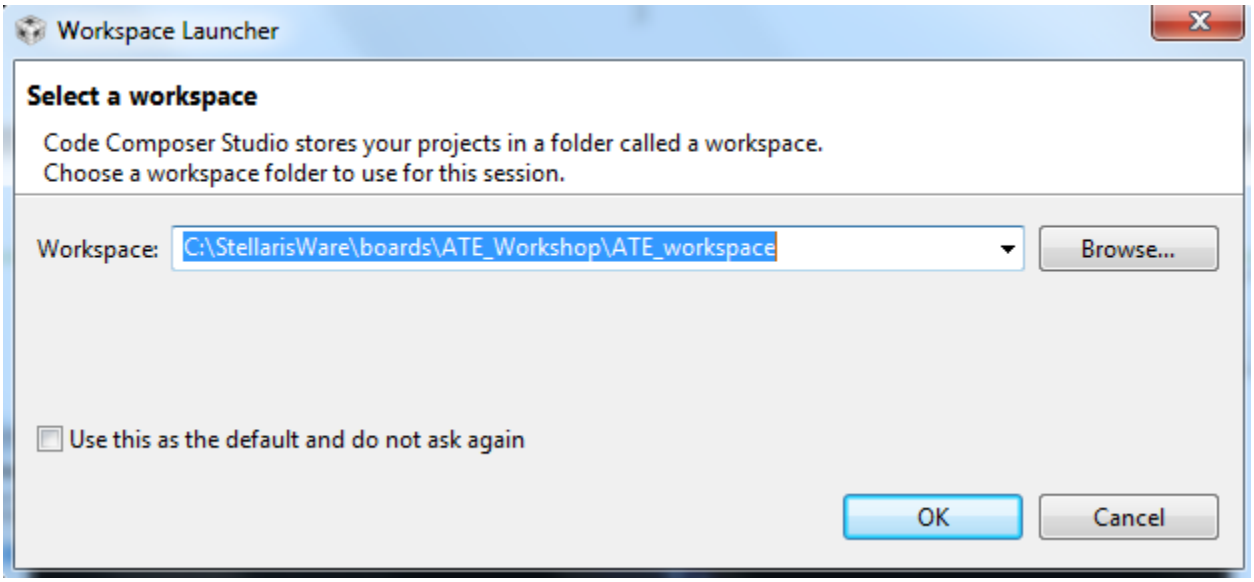


Figure 3. Workspace Launcher

The designer will be presented with the choices shown in Figure 3. The Code Composer wants to know where the workspace is located. If the auto installer was used, it will automatically load up shells that were used for the workshop. In that case, use the path as shown in the picture in Figure 3. The student will know if they have used the auto installer program as they will be presented with a picture as shown in Figure 4 where the shells of all lab projects are installed. What is desired is to have the student step through the Lab_6 exercise.

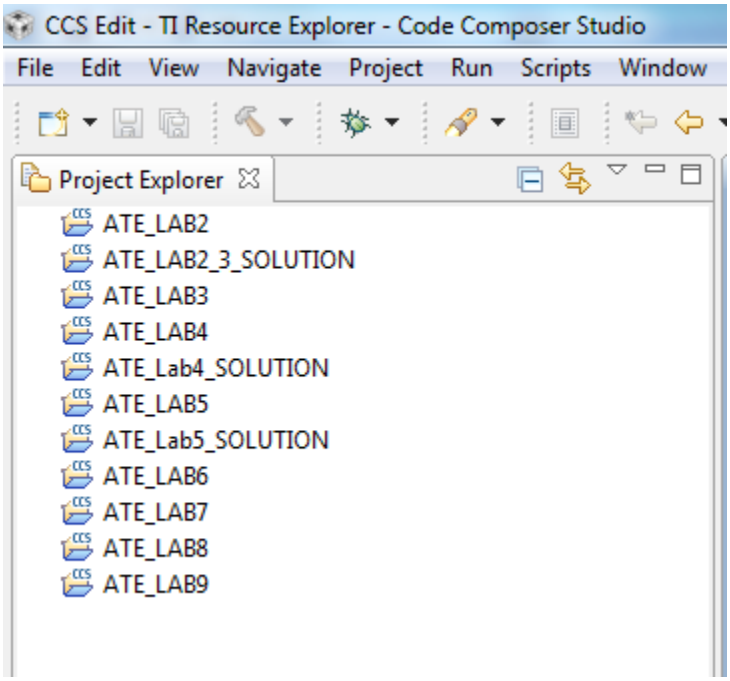

Figure 4. Shell of Projects

This has created the shell for all the tutorials. Expand the ATE_LAB6 project.

Open the main.c file. If you used the installer, it will have the main.c code in there. If not, the code is contained in attachment 1 at the end of the tutorial.

**#define** PART_LM4F120H5QR

Since this define is included above, the specific part is specified to the compiler. Stellaris has an entire line of microcontrollers. Port A may have a different address and function for each different family of parts. For our part, port A is a UART0 (the chip has two UARTs on it), other things, and GPIO. UART RX is on Port A, pin 0 (PA0).

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);              // Enable UART hardware
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);              // Enable Pin hardware

GPIOPinConfigure(GPIO_PA0_U0RX);            // Configure GPIO pin for UART RX line
GPIOPinConfigure(GPIO_PA1_U0TX);            // Configure GPIO Pin for UART TX line
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);     // Set Pins for UART
```

Compile and run the program.

Click on the debug icon and then run the program. Next it is necessary to run the UART software. One of the most common and free programs is called Putty. Using Google it is very easy to find the downloader and installer. It is a very small program. This test will use the same cable and connection as is used to program the board. First, hit the red square to halt the debugger. Open Putty.

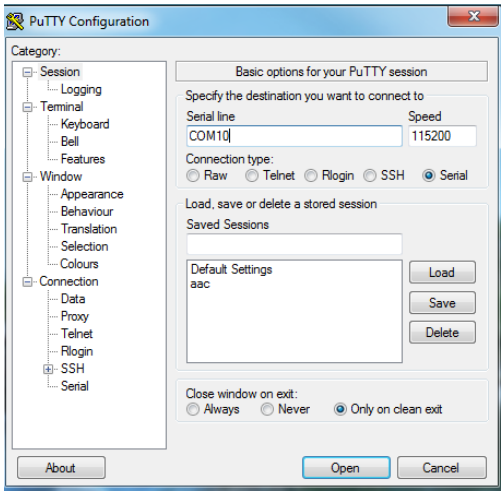


Figure 5. Putty

The window shown above will be presented. Choose the correct comm port for your machine. You may have to go to the device manager to confirm this. Make sure the serial button is chosen and that the speed is set for 115200 (as was set in the main.c file). Click Open. Hit reset on the stellaris board. It is located in the upper right corner of the board. Go back to the Putty.

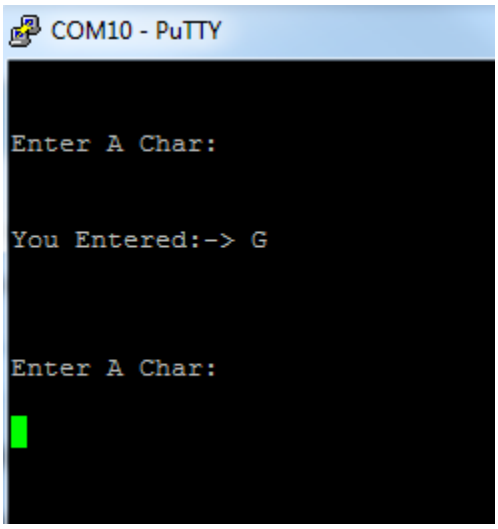


Figure 6. UART Output

As shown in Putty. The text that you had in main.c is now displayed. Type in a character and it will be displayed back.

Challenge – Change the words or baud rate

Attachment 1: main.c file

```
/****************************************************

Project : Orbit Lab 6 ATE (UART)
Version : 1.0
Date    : 2/20/2013
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This source provides an introduction to communication concepts.
The student will enable one of the GPIO ports to function
as a UART. This will allow the board to have a basic serial
port to transmit/ Receive data/commands.


****************************************************
Chip type               : ARM LM44F120H5QR
Program type            : Firmware
Core Clock frequency  : 80.000000 MHz
****************************************************/

// Define needed for pin_map.h
#define PART_LM4F120H5QR

#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) {

        long C;               // Temp Character holder

        SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);  // Set up
Clock

        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);              // Enable UART hardware
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);              // Enable Pin hardware

        GPIOPinConfigure(GPIO_PA0_U0RX);            // Configure GPIO pin for UART RX line
        GPIOPinConfigure(GPIO_PA1_U0TX);            // Configure GPIO Pin for UART TX line
        GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);     // Set Pins for UART

        UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,        // Configure UART to 8N1 at
115200bps
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));


        while (1)
        {

                UARTCharPut(UART0_BASE, '\n');              //
                UARTCharPut(UART0_BASE, '\n');              //
                UARTCharPut(UART0_BASE, '\r');              //
                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, 'A');        //      -----> Print Start Text
            UARTCharPut(UART0_BASE, ' ');        //
            UARTCharPut(UART0_BASE, 'C');        //
            UARTCharPut(UART0_BASE, 'h');        //
            UARTCharPut(UART0_BASE, 'a');        //
            UARTCharPut(UART0_BASE, 'r');        //
            UARTCharPut(UART0_BASE, ':');        //
            UARTCharPut(UART0_BASE, '\n');               //
            UARTCharPut(UART0_BASE, '\n');               //
            UARTCharPut(UART0_BASE, '\r');               //

            C = UARTCharGet(UART0_BASE);

            UARTCharPut(UART0_BASE, '\n');               //
            UARTCharPut(UART0_BASE, '\r');               //
            UARTCharPut(UART0_BASE, 'Y');        //
            UARTCharPut(UART0_BASE, 'o');        //
            UARTCharPut(UART0_BASE, 'u');        //
            UARTCharPut(UART0_BASE, ' ');        //
            UARTCharPut(UART0_BASE, 'E');        //
            UARTCharPut(UART0_BASE, 'n');        //
            UARTCharPut(UART0_BASE, 't');        //      -----> Print Start Text
            UARTCharPut(UART0_BASE, 'e');        //
            UARTCharPut(UART0_BASE, 'r');        //
            UARTCharPut(UART0_BASE, 'e');        //
            UARTCharPut(UART0_BASE, 'd');        //
            UARTCharPut(UART0_BASE, ':');        //
            UARTCharPut(UART0_BASE, '-');        //
            UARTCharPut(UART0_BASE, '>');        //
            UARTCharPut(UART0_BASE, ' ');        //
            UARTCharPut(UART0_BASE, C);                  // <------- Notice the use of a variable
            UARTCharPut(UART0_BASE, '\n');               //
            UARTCharPut(UART0_BASE, '\n');               //
            UARTCharPut(UART0_BASE, '\r');               //

    }
}
```

Attachment 1: Block Diagram of the Pins Used in Projects