

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 and Brian Zufelt from the COSMIAC Research Center at the University of New Mexico. Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. Originally Funded by the National Science Foundation.

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. The objective of this tutorial is to have the student finish with an understanding of how to develop a UART in the Tiva platform as well as how to exercise some of the ASCII code.

Each of these labs add upon the previous labs and it is the intention of the authors that the students will build (with each lab) a better understanding of the ARM processor and basic C code. 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.

It is assumed that the “student” will be installing the packages in accordance with the instructions in Lab1. That process will provide the framework and load all the support files/projects for the associated workshop. Each Lab will have two files. For example, there is a generally a Lab x project and a Lab x Solution project. The only difference between these two projects is the code in the main.c file. The exception is in the interrupt project (Lab 5) where the ...startup_ccs.c file is also edited in the solution. The reason this was done this way was to provide the student with two choices. They can follow the tutorial in the Lab and type in the associated code identified in the tutorial. This will allow for errors to be made and learning to be achieved through the debugging process. The alternative is that the student has a fall back option if everything goes bad and they can just look at/copy the source file contents from the “Solution” projects.

Learning Objectives

The student should begin to become familiar with the concept of the UART and ways to communicate through a 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 transmitting and the other is receiving. 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 Tiva 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 Tiva board and then how to communicate through that connection.

Time Required

Approximately 1.5 hours

Lab Preparation

It is highly recommended that the student read through this procedure once before actually using it was a tutorial. By understanding the final goal it will be easier to use this as a tutorial. This Lab will introduce the Tiva C Series TM4C123G LaunchPad Evaluation Kit which is a low-cost evaluation platform for ARM® Cortex™-M4F-based microcontrollers from Texas Instruments. The design of the TM4C123G LaunchPad highlights the TM4C123GH6PM microcontroller with a USB 2.0 device interface and hibernation module.

The EK-TM4C123GXL evaluation kit also features programmable user buttons and a Red, Green, RGB LED for custom applications. The stackable headers of the Tiva C Series TM4C123G LaunchPad BoosterPack XL Interface make it easy and simple to expand the functionality of the TM4C123G LaunchPad when interfacing to other peripherals with Texas Instruments' MCU BoosterPacks.

Equipment and Materials

Access to the four software packages: Code Composer, Tivaware, Driver file and ATE Workshop installer are required and should have been installed in accordance with Lab 1. In addition, the user should have access to the Tiva evaluation kit (EK-TM4C123GXL) and (for Labs 3-9) the Digilent Orbit board. It is assumed that the student has already completed Lab 1 and the software is installed.

| Software needed | Quantity |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| The installation files are covered in Lab 1. The software package can be downloaded as one large zip file from this URL: http://cosmiac.org/thrust-areas/education-and-workforce-development/community-portal/ | 1 |
| Hardware needed | Quantity |
| The hardware required is the Tiva LaunchPad Kit. The kit and the ORBIT daughter card can be purchased from the Digilent Corporation www.digilentinc.com | 1 |



Additional References

This page is for general information: <http://www.ti.com/tool/ek-tm4c123gxl> on the Tiva LaunchPad Kit. If the link above is no longer valid, just google the Tiva LaunchPad and it will pop up. The full set of tutorials is available on this website: <http://cosmiac.org/thrust-areas/education-and-workforce-development/microcontrollers/ate-developed-material/>



Lab Procedure : Install/Connect board to computer



Figure 1. ARM and ORBIT Combination

The image shown in Figure 1 is the correct mating of the LaunchPad and Orbit BoosterPack boards. There is no change from Lab 5.



Figure 2. Code Composer Icon

Launch Code Composer from the icon as shown in Figure 2 and where prompted, choose the workspaces location to store your project (as shown in Figure 3).

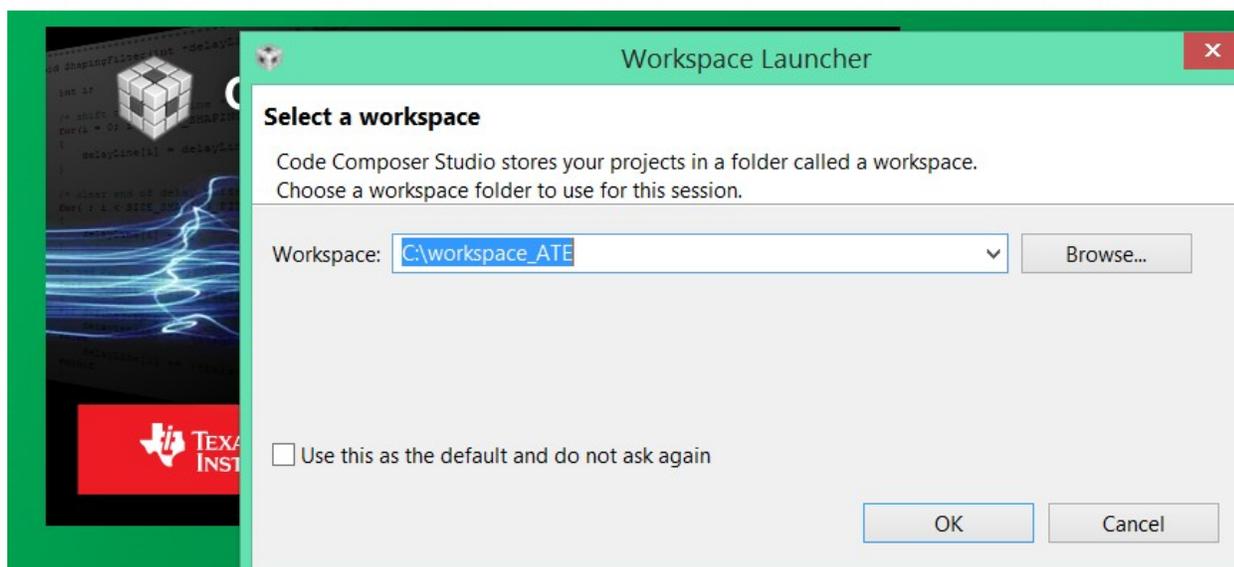


Figure 3. Workspace Selection

Since the installer for the workshop has been run prior to this, the user will be presented with the following view (Figure 4) where all lab projects exist.

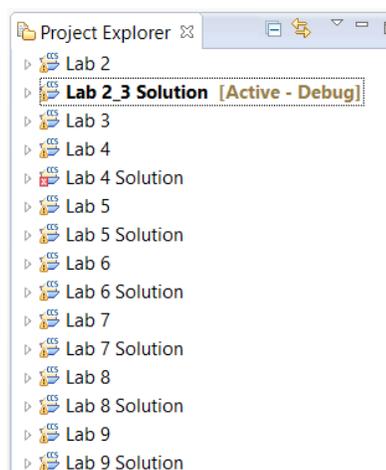


Figure 4. CCS Starting Point

The laboratory material is created to have the students type in a lot of the code. Only by typing the code and then debugging the errors will a user ever really understand how to do projects. For the sake of this activity, the source code is provided at the end of the tutorial. In Lab 6, open main.c.

```
#define PART_TM4C123GH6PM
```

What is presented to the student is a shell of lab 6. Since the define shown above is included as part of our project, the specific part is known to the compiler. Tiva has an entire line of microcontrollers. As can be seen in Figure 5, our specific part actually has eight different UARTs on it.

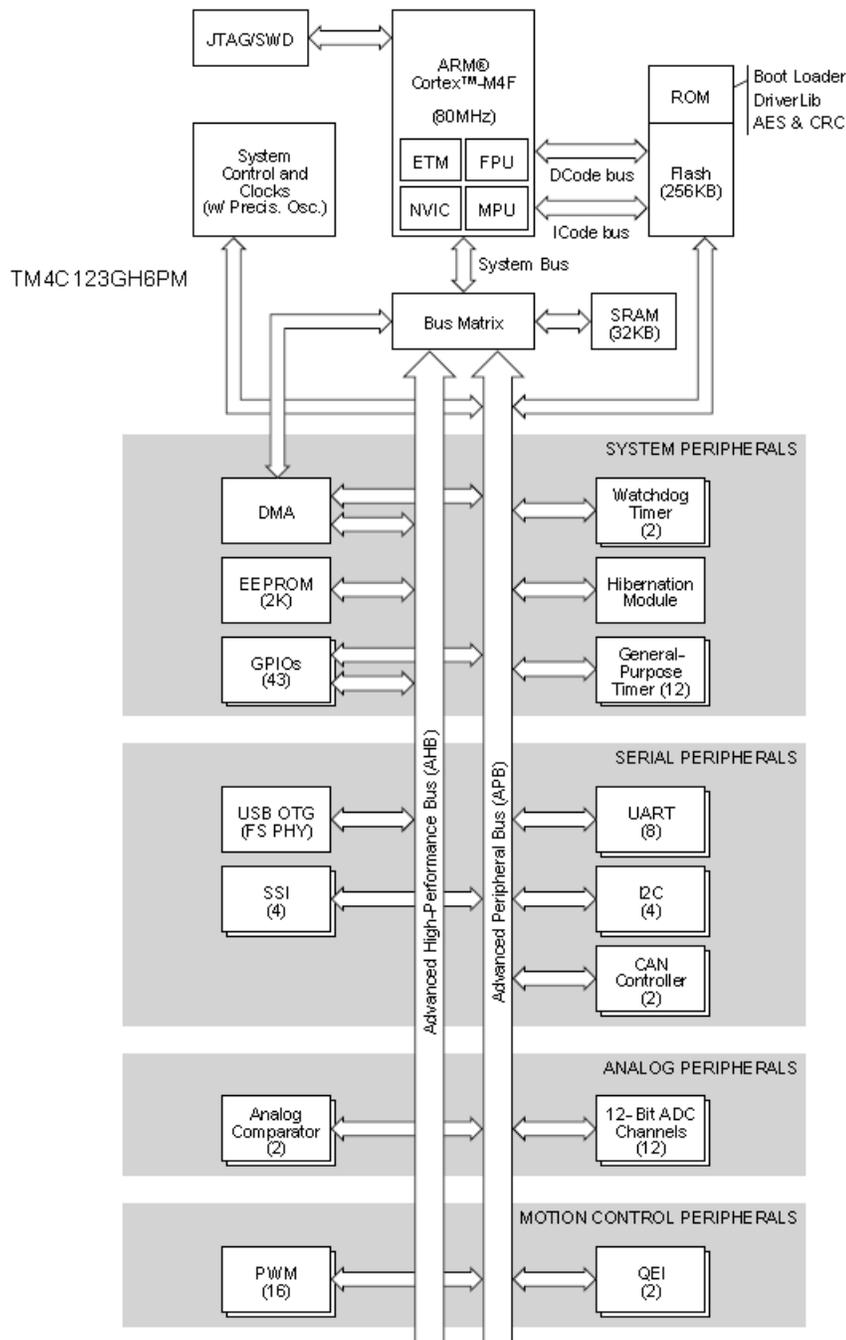


Figure 5. Chip High Level Diagram

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 eight UARTs on it), other things, and GPIO (general purpose inputs and outputs). UART RX is on Port A, pin 0 (PA0).

```
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));

```

There is nothing shocking in the code above (that is given to the student in the shell of Lab 6). What is new is the creation of a temporary value holder called C. After that, what is accomplished is the standard part of enabling ports and in this case, UARTs. After that, the regular role of configuring pins. For those familiar with UARTs, the final line of 8None1 is very common configuration for a UART as is the bit rate of 115,200.

The next portion of the Lab is the replacement of this information shown below with actual code.

```

/*****
 * Using the UARTCharPut(UART0_BASE, "CHAR") and UARTCharGet(UART0_BASE) functions to output data
 * and read data from the UART. Each function only transfers one 8-bit character.
 *****/

```

By going below, above or replacing this comment code block, add the following code. The user should be able to read from the top to the bottom and see: new line, return, Enter A Char:, a couple of new lines and then a return.

```

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, ' '); // -----> 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'); //

}
```

Once the program has printed to the screen “Enter A Char:” then it goes into a wait state.

```
C = UARTCharGet(UART0_BASE);
```

When someone enters a character into the keyboard, it is stored into the temporary value called “C” for later recall. After it is stored, the next set of code places some text onto the screen and then finally recalls the contents of “C” for the user to see. After you have typed/pasted in the code, compile and run the program.

Click on the debug icon and then run the program. Next, it is necessary to run the UART interface software for visualization of the inputs and outputs. One of the most common and free programs is called Putty. Using Google it is very easy to find the downloadable execution file (www.putty.com). 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.

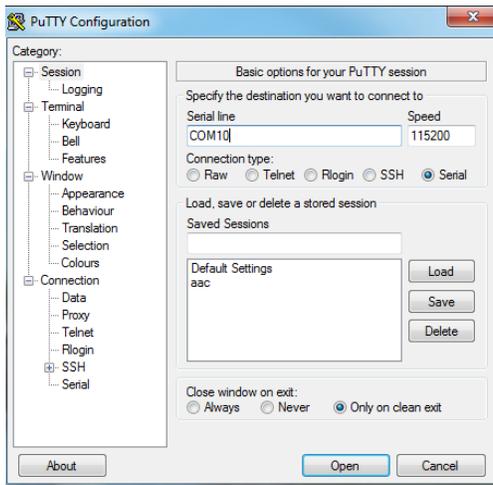


Figure 6. Putty

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 tiva board or the return key on the laptop keyboard a few times. The Tiva reset is located in the upper right corner of the board. Go back to the Putty.

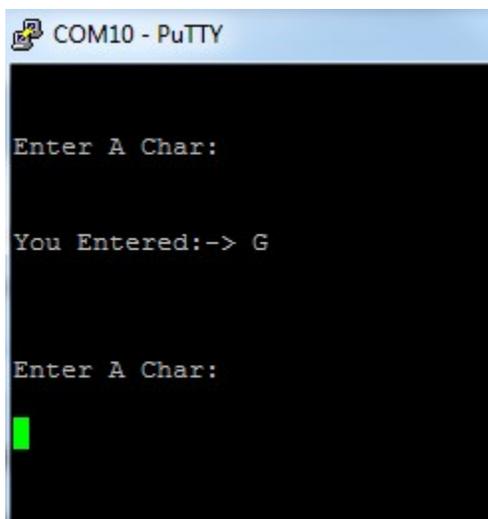


Figure 7. UART Output

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 as shown in Figure 7. You may have to hit return a couple of times to see the display initially. Challenge – Change the words or baud rate.



Attachment 1: main.c file

```

/*****
Project : Orbit Lab 6 ATE (UART)
Version : 2.0
Date   : 2/20/2015
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 TM4C123GH6PM
Program type   : Firmware
Core Clock frequency : 80.000000 MHz
*****/

// Define needed for pin_map.h
#define PART_TM4C123GH6PM

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.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"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.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)
    {

        /*****
        * Using the UARTCharPut(UART0_BASE, "CHAR") and UARTCharGet(UART0_BASE) functions to output data
        * and read data from the UART. Each function only transfers one 8-bit character.
        *****/

    }

}

```



Attachment 2: Lab 6 Solution main.c

```

/*****

Project : Orbit Lab 6 ATE (UART)
Version : 2.0
Date   : 2/20/2015
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 TM4C123GH6PM
Program type    : Firmware
Core Clock frequency : 80.000000 MHz
*****/

// Define needed for pin_map.h
#define PART_TM4C123GH6PM

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.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"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.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, 'r'); //
        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 3: Block Diagram of the Pins Used in Projects

