# Talking to a Serial Port

**ECE598: Advanced Operating Systems – Homework 3**

Spring 2016

**Due: Thursday, 11 February 2016, 9:30am**

This homework is about communicating to your standalone program via a serial port.

1. **Download the homework code template**

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece598_2016s/ece598_hw3_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece598_hw3_code.tar.gz`

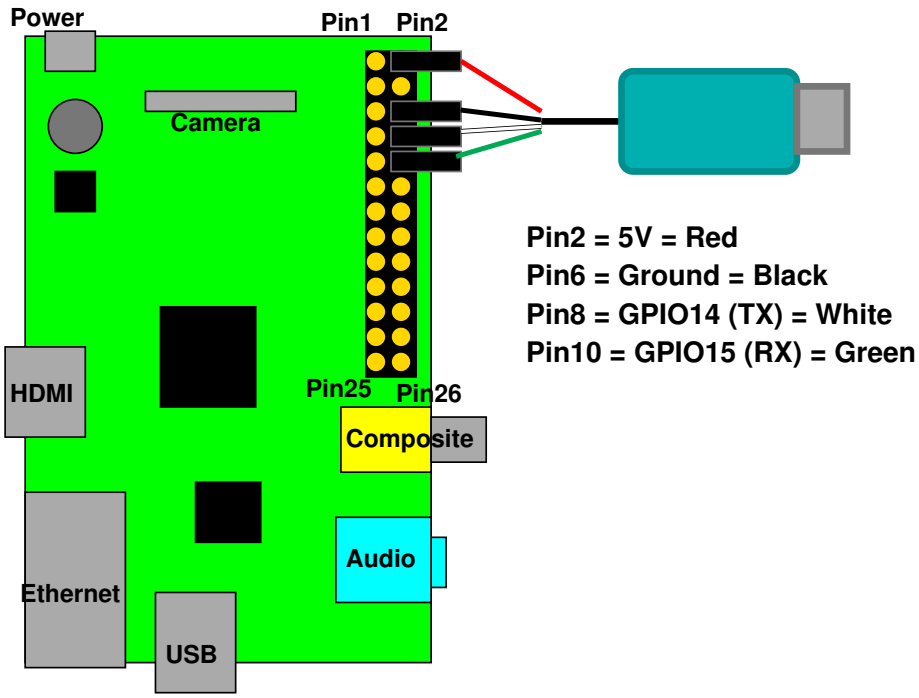


Figure 1: Raspberry Pi Serial Connection

2. **Get serial input/output working**

   - Connect the serial to USB converter to your Pi as shown in Figure 1. For the model B+ the connectors are similar (the GPIO header is longer but the top half of the pins are the same).
   - If you hook up the red wire (5V) you do *not* need to separately power via the USB micro port. However, powering your board this way bypasses some of the power filtering circuitry. A safer way to do things is to leave the red wire disconnected and still power your device via the separate USB micro connector.

- Additional details for using the adapter can be found here:
  `http://www.adafruit.com/products/954`
  specifically:
  `https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable`
  If you are connecting to a Windows machine you may need to download the PL2303 driver as described in that link. The same goes for OSX. Linux should come with the PL2303 driver by default.
- Setup a terminal program on your development machine.
  - For Windows I recommend "putty". See the adafruit link for download and usage info.
  - For Linux, minicom works. You might need to install it (`apt-get install minicom` on Debian or Ubuntu).
    Plug in the device then look at the end of the `dmesg` command to see what device it is. (Usually `/dev/ttyUSB0`).
    Run minicom with something like:
    `minicom --color=on -D /dev/ttyUSB0`
    You may need to be root. Menus are accessed by **control-A** then **Z** for help. You might have to enable sw flow control. It might also help to enable line wrapping (**control-A W**).
  - OSX. There are terminal programs out there, I haven't had a chance to find a good one. The adafruit link recommends screen, which should work.

3. **Familiarize yourself with the provided code**

   I provide the following files:

   - `bcm2835_periph.h` – useful #defines based on the Broadcom 2835 Peripherals manual
   - `boot.s` – assembly boot code that sets up the stack and clears the BSS
   - `console_io.c`, `console_io.h` – common entry point for console printing code
   - `delay.h` – inline assembly for creating delay loops
   - `kernel.ld` – linker script
   - `kernel_main.c` – the main entry point to the code
   - `mmio.h` – inline assembly for doing I/O accesses
   - `printk.c`, `printk.h` – code for the printk routine
   - `serial.c`, `serial.h` – the UART code

4. **Get your serial port working (2pt)**

   - Edit `uart_init()` in the `serial.c` file. Much of this is implemented for you.
   - Modify the code so it properly sets the `UART0_IBRD` and `UART_FBRD` values for 115200 operation. We went over the calculations on how to do this in class.
   - Modify the code so it sets the `UART0_CR` register at the end to enable the UART overall, transmit, and receive.
   - Compile the code, using `make`
   - Copy the generated `kernel.img` to your SD card and boot it as per HW#2.

- If all goes well you should be able to type things in your terminal and have them echoed back to you. (Note, pressing Enter might not work as expected because of the linefeed/carriage return issue discussed in class).
- Note at boot that it will prompt you to press a key before continuing. This is a bit of a hack; without it if you are using screen or putty and also powering your board from the serial port then sometimes the boot messages will scroll by before the terminal program is ready to display them.
- Getting the serial port working is critical, so if you get stuck here please ask for help right away.

5. **Print a message at bootup (2pt)**
   - Modify the `kernel_main.c` file to print a boot message of your choice.
   - You can do this simply by calling the provided `printk()` function with your string to print.
   - Remember you might need to have a CR/LF line ending (`"\r\n"`) to get a newline.
6. **Get printk printing hex values (2pt)**
   - Right after your boot message we print the hardware type, which can be found in the `r1` variable.
   - We want to use `printk()` to do this in hexadecimal, but `printk()` as provided does not properly support the %x modifier.
   - Fix the code in `printk.c` to support the %x modifier.
   - Add a printk statement in `kernel_main.c` to print the value in hex.
7. **Something Cool (1pt)** Suggestions of things you can do:
   - Clear the screen (using escape characters) at boot.
   - Print a fancy boot message that is in color.
   - Interpret the keys being pressed and do something interesting (turn on/off the ACT LED, change the text color, etc.)
8. **Answer the following questions (3pt)**

   Put your answers in the README file.

   (a) Why do we use the serial port for communication with the Pi in this homework rather than USB, HDMI video, or ethernet?

   (b) What are parity bits good for?

   (c) By default most systems these days use settings of 115200 bits/second, 8-bits, no-parity, 1 stop bit. Why is parity disabled?

   (d) What is inline assembly language?

   (e) A common way to write a simple command interpreter is to use the standard C `strtok()` function. Why can't we use that in this homework?

   (f) Which terminal program and operating system did you use for this homework? Is it working well or are there annoying issues with getting it working?
9. **Submit your work**
   - Run `make submit` in your code directory and it should make a file called `hw3_submit.tar.gz`. E-mail that file to me by the deadline.