ECE531: Advanced Operating Systems – Homework 3
Device Driver for the Raspberry Pi Serial Port

Due: Friday, 26 September 2025, 5:00pm

This homework is about communicating to your bare-metal Pi system via a serial port.

1. **Download the homework code template**

   - Download the code from:
     `https://web.eece.maine.edu/~vweaver/classes/ece531/ece531_hw3_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece531_hw3_code.tar.gz`

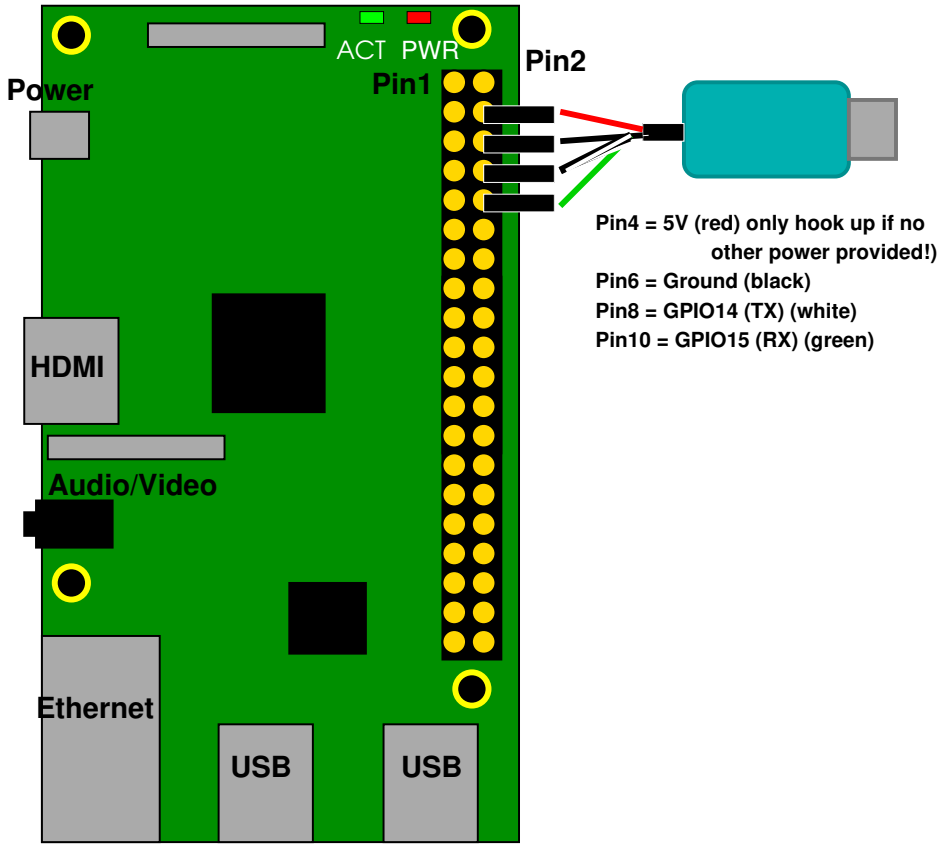

Figure 1: Raspberry Pi Serial Connection on Model-1B+

2. **Setup the USB-serial Cable on your Pi**

   - For the rest of the homeworks we will communicate with the Pi via the USB-serial connector
   - Connect the serial to USB converter to your Pi as shown in Figure 1.
   - On the Pi-1B+ models we are using it will probably be easiest to power your Pi via the serial cable as seen in Figure 1. If you do want to power it separately via the separate USB power connector be sure to not hook up the red wire from our USB-serial connector.

3. **Setup the USB-serial Cable on your Development Machine**

   - You will probably need to set some things up on your development machine before you can communicate with the Pi.
   - Once things are setup you will be able to communicate to the Pi via serial by just plugging the USB-end of the serial adapter into your development machine.

     (note that until you copy the completed HW#3 code to your device it won't send anything across the serial cable, so for the steps below don't be surprised if you just get a blank terminal at first).

     – On Linux the USB-serial device should be automatically detected (you can check the results of the `dmesg` command and it should report something similar to `cp210x converter now attached to ttyUSB0`
     – For Windows or MacOS you might have to install a custom driver. Details for doing that can be found on this page, there are links for MacOS and Windows instructions:
     `https://www.adafruit.com/products/954`
     The adapters I handed out in class are the CP2102 model.
     If you follow the link to the Silicon Labs webpage you will need to pick the "Downloads" tab on the page to find the driver downloads.
   - Setup a terminal program on your development machine.
     – We will program the Pi to use settings 115200 8N1 and software flow control.
     – For Windows I recommend "putty". See the adafruit link for download and usage info. There is also some info at the end of this document.
     – For MacOS you can use "screen". The adafruit link above has info on how to get this working and there's some additional information at the very end of this document.
     – For Linux there are two tools you can use. The easiest is also to use the "screen" tool. You might have to install it (`apt-get install screen`).
     Plug in the device then look at the end of the `dmesg` command to see what device it is. (Usually `/dev/ttyUSB0`).
     Then run `sudo screen /dev/ttyUSB0 115200`
     Control-A, followed by **K** will exit it.

     Alternately you can use the program "minicom". You might need to install it (`apt-get install minicom` on Debian or Ubuntu).
     Run minicom with something like:
     `minicom --color=on -D /dev/ttyUSB0`
     You may need to use "sudo" to get Administrator access. Menus are accessed by **control-A** then **Z** for help. You might have to disable hardware flow control (**control-A O**. It might also help to enable line wrapping (**control-A W**).

4. **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
- `led.c`, `led.h` – code to turn on/off ACT LED

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

   - Note: if you had to modify the `Makefile.inc` file to point to your cross compiler for HW#2 you will have to do it again for this assignment.
   - Edit `uart_init()` in the `serial.c` file. Much of this is already implemented for you.
   - You will need to add some MMIO calls. Unlike the examples shown in class, use routines called `bcm2835_write()` and `bcm2835_read()`. These routines take into account the `IO_BASE` value (to allow using the same routines on any Raspberry Pi).
   - 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.
     Use 48MHz as the UART clock.
   - At the end of the function be sure to set the `UART0_CR` register to enable the UART overall, transmit, and receive. (See class notes for more details)
   - 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 with the letters changed to uppercase. (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.
   - The code also lights up the ACT LED at startup. (the code that implements this is in `led.c` and called after the uart init in `kernel_main.c`).

     You can use this routine to help debug things as a last resort. (For example if the LED never turns on at boot it means somehow your code isn't running at all).
   - Getting the serial port working is critical, so if you get stuck here please ask for help right away.

6. **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. `printk()` is just the kernel version of `printf()`
   - Remember you might need to have a CR/LF line ending (`"\r\n"`) to get a newline.

7. **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 (it currently prints in decimal).
   - Fix the code in `printk.c` so the %x modifier works.
   - Once you have that working the code that prints the processor type should be properly in hex. Be sure to check various corner cases to make sure your code handles all possible hex results.

8. **Something Cool (1pt)** Be sure to describe what you did in the README file.

   Suggestions of things you can do:

   - Clear the screen (using escape characters) at boot.
   - Print a fancy boot message that is in color and has ASCII art.
   - Interpret the keys being pressed and do something interesting (turn on/off the LED, change the text color, etc.)
   - Add other formats to printk (%c, %s, %X, etc).

9. **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 in a serial connection?

   (c) What is inline assembly language?

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

   (e) 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?

10. **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.
      Be sure your name (and your partner's name if applicable) is in the file!

Some extra notes on getting serial ports working

- MacOS
  - Generally the linked tutorial to the Aafruit page should tell you what you need to know, be sure to keep clicking next to get past the Linux directions
  - To see if you have the proper driver installed, be sure the USB-serial device is connected, open a terminal and do a `ls /dev/cu*`. If things are working you'll see a file called something like `cu.usbserial-0001` or similar (it might have random hex digits after it).
  - Once you see that file, you can use the `screen` tool to connect to the serial port. Something like:
    `screen /dev/cu.usbserial-0001 115200`
    where cu.usbserial-0001 is the serial port you saw in the previous step.
  - If you get an error like "TERM too long – sorry" try running it as
    `TERM=vt102 screen /dev/cu.usbserial-0001 115200`
- Windows
  - For windows again the Adafruit directions should be more or less enough to get you going.
  - Ideally windows would auto-install the driver. If not, you should get an EXE file to click on when you download the drivers. If there is no EXE, you might have to right-click on an .inf file and pick "install"
  - Check the device manager and hopefully a USB serial connection has appeared. It should appear with a name something like COM7
  - Start putty and use the COM name from the previous step. Be sure you are connecting at 115200 bits per second