# COMS20001 lab. worksheet: week #19

- Both the hardware and software in MVB-2.11 is managed by the IT Services Zone E team. If you encounter a problem (e.g., a workstation that fails to boot, an error when you try to use some software, or you just cannot log into your account), they can help: either talk to them in room MVB-3.41, submit a service request online via

  http://servicedesk.bristol.ac.uk

  or talk to the dedicated CS Teaching Technologist, Richard Grafton, in room MVB-2.07.
- We intend this worksheet to be attempted, at least partially, in the associated lab. session. Your attendance is important, since this session represents a central form of feedback and help for COMS20001. Perhaps more so than in units from earlier years, *you* need to actively ask questions of and seek help from either the lectures and/or lab. demonstrators present: passively expecting them to provide solutions is less ideal.
- The questions are roughly classified as either L (for coursework related questions that should be completed in the lab. session), or A (for additional questions that are entirely optional). Keep in mind that we only *expect* you to complete the first class of questions: the additional content has been provided *purely* for your benefit and/or interest, so there is no problem with nor penalty for totally ignoring it (since it is not directly assessed).

**Q1[A].** Justified by the volume of associated material, one *non*-goal cited by the lecture(s) was coverage of history as a specific topic. Despite this, you may still benefit from even limited study of the topic: a non-exhaustive list of suggestions appears below, each allowing some insight into the unit as a whole.

a There is a *huge* range of books and online resources dedicated to presenting a history of the Internet, but good examples include:

- Hafner and Lyon [1] offer an accessible, easy to read narrative that describes transition of the ARPANET project to *an* internet then *the* Internet; unlike some books, it strikes a fairly good balance between technical detail and the personalities involved.
- Russell [2] gives a good overview of the bureaucracy surrounding standardisation efforts, including contrast between development of the OSI and Internet models.
- The UCLA Kleinrock[1] Center for Internet Studies

  http://internethistory.ucla.edu/

  houses an extensive digital archive, including copies of many original technical reports that detail the design of ARPANET.
- The Computer History Museum has a curated exhibition at

  http://www.computerhistory.org/nethistory/

  with some interesting artefacts (even if the discussion is at a fairly basic level).

b In a technical sense, ARPANET is the obvious precursor to the modern Internet. As a result, it is often presented as the only historical reference point. At times however, the history of ARPANET can lack a personal dimension: ARPANET was essentially a research project, not a consumer "product" for example. Interim technologies such as Bulletin Board Systems (BBSes)[2] filled this gap. An 8-part documentary

http://www.bbsdocumentary.com/

which you can now find online at

https://archive.org/details/BBS.The.Documentary

captures both the technical and social history of BBSes through a range of interviews: various highlights include discussion of the FidoNet[3] email and message forum system (episode #4), and demise of BBSes and/or rise of consumer Internet use (episode #7) prompting many to transition into early ISPs.

**Q2[A].** Note that you *cannot* use the lab. workstations to complete the tasks in this question, so you will need to make use of your own equipment. In particular, you will need

a two RaspberryPi[a] boards, and
b three jumper wires (ideally with female-to-female connectors).

Assuming use of Raspbian, executing `sudo apt-get install minicom` should install all software necessary on the RaspberryPi boards. **Take care** when making connections between the GPIO pins: since they normally have *no* electrical protection, it is possible to irreparably damage the board if said pins are connected incorrectly (e.g., if an output pin is connected to a driving voltage, or any pin to too high a voltage).

[a] The example relates to use of a Model B+; others will work with minor alterations at most.

Recall from the lecture(s) that TIA-232-F is (or perhaps *was*) a common way to realise serial communication between devices; the concrete example we used was communication between a PDP-11 and Teletype 33 terminal. While there is some value in experimentally reproducing the example, doing so is made harder by lack of an associated interface in modern workstations or laptops (due to better alternatives such as USB). However, many embedded devices *do* often use a similar approach. Their implementation differs a little from TIA-232-F, st. almost no additional hardware is required to support similar functionality: they just use

- a minimal 3-pin (rather than 9- or 25-pin) interface including *GND*, *TxD* and *RxD*, and
- TTL-friendly (e.g., 0V and 3.3V) voltage levels to represent 0 and 1.

So, the question is *which* embedded device we could use to try it out? Fortunately, RaspberryPi offers a convenient solution: not only does it include GPIO (general-purpose I/O) pins which can be used for the interface, it also includes support in most OS distributions (e.g., Raspbian) to log-in to the board over said interface. Put another way, given *two* RaspberryPi[4] boards, we can construct a (fairly) accurate reproduction of the original example. Figure 1a illustrates the experiment: the top board is the master, which we assume can be logged-in to somehow (e.g., using a wired or wireless network network connection, or simply an attached keyboard and monitor), and the bottom is the slave. The goal is basically to log-in to the slave from the master.

a By default, Raspbian has a device entry `/dev/ttyAMA0` which is attached to the GPIO-based serial interface; again by default, it attaches a terminal (or TTY)[5] to this device and hence the serial interface. The first step is to disable this behaviour on the master board, so that we have exclusive access to the serial interface. Log-in to the master board, and then execute the command

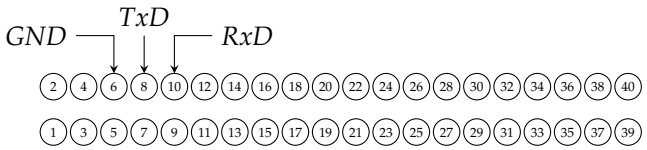
```
sudo nano /etc/inittab
```

Toward the bottom of the file, you should find a line similar to

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

which is responsible for spawning (and indeed *re*spawning, if it is ever terminated) the TTY: add a '#' character to the start of this line to comment it, then save the file.

b These boards have a 40-pin GPIO header, which Figure 1b shows a close up of. The pin assignment (limited to the the pins we are interested in) is as follows:




Note that if you have a different model of RaspberryPi then this *may* differ, but probably not: the three *GND*, *TxD* and *RxD* pins seem to have been kept in the same place even if the header size changes.

As a precaution, power-off both boards at this point. The next step is to connect the serial interfaces together, using the jumper wires, as follows (noting the right-hand column refers to the wire colours used in Figure 1a and Figure 1b):

[1] http://en.wikipedia.org/Leonard_Kleinrock
[2] http://en.wikipedia.org/wiki/Bulletin_board_system
[3] http://en.wikipedia.org/wiki/FidoNet
[4] The boards used here are model B+ but other models will work just as well.
[5] There is an excellent overview of the UNIX-based TTY mechanism here http://www.linusakesson.net/programming/tty/. The short version, however, is that the device entry is how user processes utilise the underlying serial interface (via the kernel, which manages the hardware using an appropriate device driver).

| Master | | Slave | Colour |
|---|---|---|---|
| pin #6 (*GND*) | ⟷ | pin #6 (*GND*) | black |
| pin #8 (*TxD*) | ⟷ | pin #10 (*RxD*) | blue |
| pin #10 (*RxD*) | ⟷ | pin #8 (*TxD*) | green |

So, put simply, we make the boards have a shared ground (which acts as a common reference point for voltage levels) and connect the transmit (resp. receive) pin of the master to the receive (resp. transmit) pin of the slave; anything the master transmits on pin #8 is received by the slave on pin #10 (and vice versa). Once the connections are in place, power-on both the boards (waiting a while until their boot sequences complete).

c The final step is to log-in to the slave board from the master board: we know the slave has a TTY attached to the serial interface at that end of the connection, so we just need to communicate with this from the serial interface on the master board.

Log-in to the master board, then executing the command

```
minicom -o -b 115200 -D /dev/ttyAMA0
```

st. `minicom` communicates via `/dev/ttyAMA0` (the device entry associated with the serial interface), without any form of initialisation, at 115200baud (which matches the TTY on the slave device, per the entry in `/etc/inittab`).

`minicom` is not the most friendly software in the world. When I tried, I had to experiment a little with the terminal options; when I explicitly selected 115200/8*N*1 and an ANSI terminal type, and pressed return a few times, the familiar log-in prompt appeared (but before this, I got garbage characters). So if you persevere a little, this now allows you to log-in to the slave device: and all you needed was three jumper wires!

d If you are *really* ambitious, it might be interesting/useful to inspect the raw, on-the-wire signal being transmitted. Normally the best way to do this would be via an oscilloscope or logical analyser, but of course you may not have access to such equipment. Fortunately, if you have a *third* RaspberryPi you can use *it* as a crude logic analyser. Various implementations of this concept exist; I found

http://github.com/richardghirst/Panalyzer

the easiest to get working, doing so by first installing a matched version of Raspbian, namely

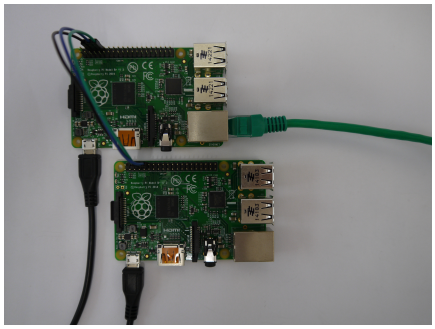
http://downloads.raspberrypi.org/raspbian/images/raspbian-2014-06-22/

to avoid having to compile anything myself. If you can do the same, and although it only allows a short sample period, you can use it to capture data being transmitted between the master and slave boards (by sampling the *TxD* and *RxD* pins).

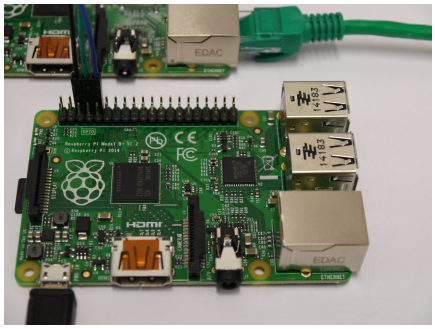
## References

[1] K. Hafner and M. Lyon. *Where wizards stay up late: the origins of the Internet*. Simon & Schuster, 1998.

[2] A.L. Russell. 'Rough consensus and running code' and the Internet-OSI standards war. *IEEE Annals of the History of Computing*, 28:48–61, 2006.



**(a)** *The master (top) and slave (bottom) boards; note the wired 802.3 connection allowing log-in to the master board, but that the* only *connection to the slave is via the serial interface.*



**(b)** *A close-up of the slave board showing the* 40*-pin GPIO header.*

**Figure 1:** *A RaspberryPi-based version of the serial communication example.*