- The hardware *and* software resources located in the MVB Linux lab(s). (e.g., MVB-1.15 or MVB-2.11) are managed by the Faculty IT Support Team, a subset of IT Services. 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: you can contact them, to report then resolve said problem, using details collated at

  https://www.bristol.ac.uk/it-services/contacts

- The worksheet is written *assuming* you work in the lab. using UoB-managed equipment. If, however, you prefer to use your own equipment, *unsupported*[a] alternatives *do* exist: you could a) manually install any software dependencies yourself, *or* b) use the unit-specific Vagrant[b] box by following instructions at

  https://www.github.com/danpage/COMS20001/blob/COMS20001_2019/vagrant/README.md

- We intend the worksheet to be attempted, at least partially, in the lab. slot. Your attendance is important, because the lab. slot represents a primary source of formative feedback and help. Note that, perhaps more so than in units from earlier years, *you* need to actively ask questions of and/or seek help from the lectures and/or lab. demonstrators present.
- The questions are roughly classified as either C (for core questions, that *should* be attempted within the lab. slot), A (for additional questions, that *could* be attempted within the lab. slot), or R (for revision questions, that support preparation for any viva and/or exam). Keep in mind that we only *expect* you to attempt 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).

[a] The implication here is that such alternatives are provided in a best-effort attempt to help you: they are experimental, and so *no* guarantees about nor support for their use will be offered.

[b] https://www.vagrantup.com

# COMS20001 lab. worksheet #1

**Q1[A].** 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 As a result, it is often presented as the only historical reference point.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 basically a research project, not a consumer "product" for example.

Interim technologies such as Bulletin Board Systems (BBSes)[1] 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 set of interviews: highlights include discussion of the FidoNet[2] email and message forum system (episode #4), and demise of the BBS 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[3] boards, we can construct a (fairly) accurate reproduction of the original example. Figure 1a illustrates the experiment. We assume it is possible to log-in to the primary (top) board somehow (e.g., via a wired or wireless network network connection, or an attached keyboard and monitor); the goal is then basically to log-in to the secondary board the primary board.

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)[4] to this device and hence the serial interface. The first step is to disable this behaviour on the primary board, so that we have exclusive access to the serial interface. Log-in to the primary board, and then execute the command

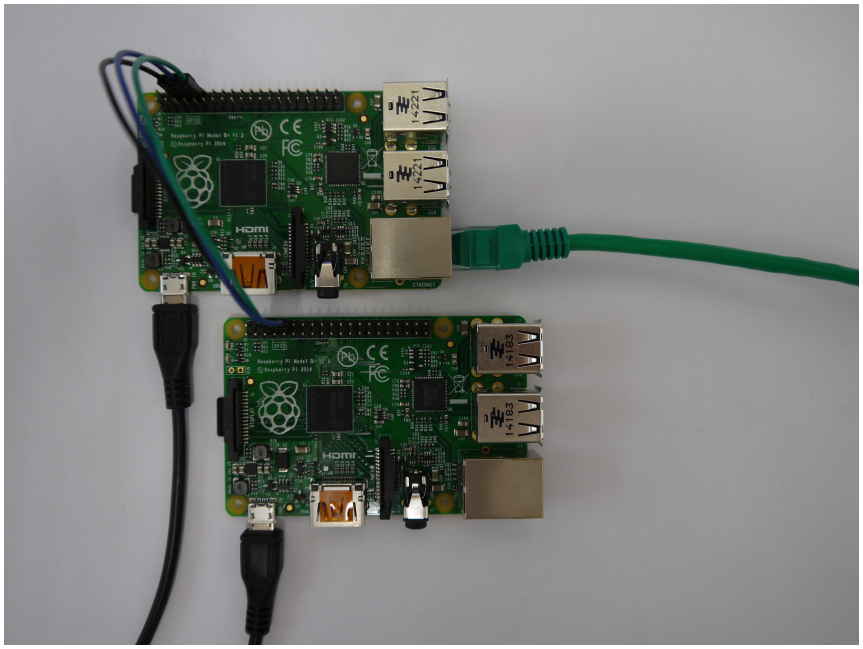
```
sudo nano /etc/inittab
```

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

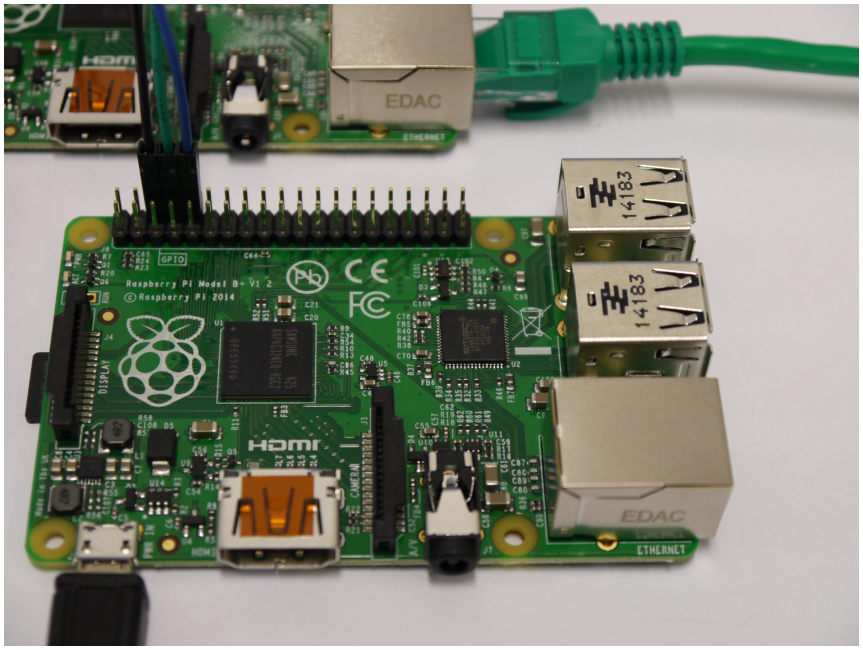
```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

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.

[1] `http://en.wikipedia.org/wiki/Bulletin_board_system`
[2] `http://en.wikipedia.org/wiki/FidoNet`
[3] The boards used here are model B+ but other models will work just as well.
[4] 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).



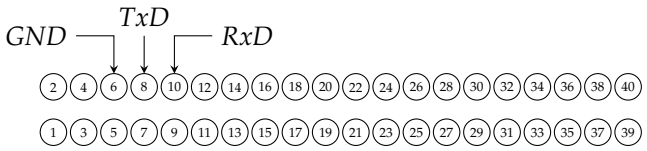
**(a)** *The primary (top) and secondary (bottom) boards; note the wired 802.3 connection allowing log-in to the primary board, but that the* only *connection to the secondary board is via the serial interface.*




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

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

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):

| Primary | | Secondary | 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 primary board to the receive (resp. transmit) pin of the secondary board; anything the primary board transmits on pin #8 is received by the secondary board 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 then to log-in to the secondary board from the primary board: we know the secondary board 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 primary board.

Log-in to the primary 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 secondary board, 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 secondary board: 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 primary and secondary boards (by sampling the *TxD* and *RxD* pins).