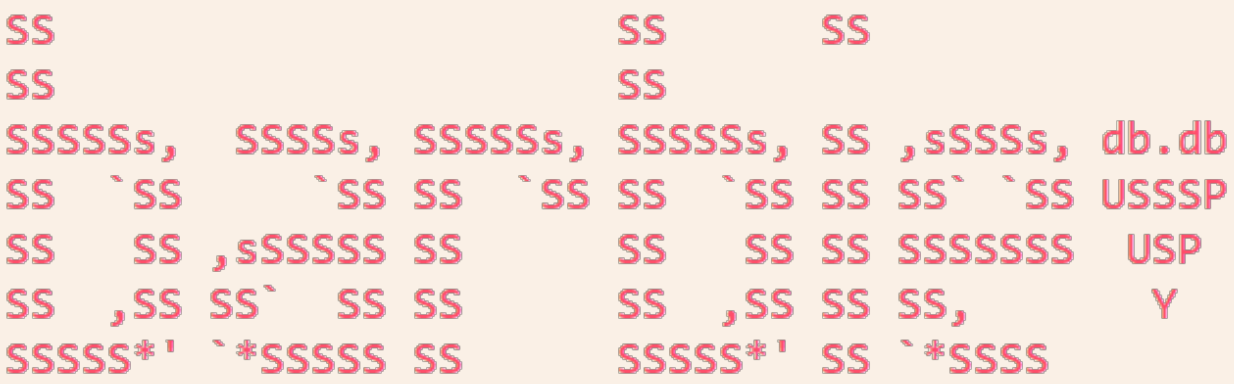

Hardware 101

2018-07-23 962 words 6 minute(s) hardware reversing

I have this $device – How to start?

Understanding your device

First of all: Look for Debug Ports

In fact, this should be the step zero step. I mean, you've got a wonderful piece of hardware but how do you communicate with it? To find all the available connections, I usually make a list of all physical ports I can access, I count all the pins I see and so on, always keeping in my mind that I want a *debug port*. The debug port is usually the one used to program the device at the factory and is sometimes left available for technical support and repair reasons.

This port is found on many embedded device: from printers to security cameras, from smart toasters to routers. Another important fact: everything gets easier, if we are handling a "complex" embedded system, as the probability of the system is running some Linux kernel increases – and Linux kernel is Open Source :D remember? But anyway, if you can't see any debug port right away, don't give up! Sometimes they are kind

of hidden and you may or may not need to break some seals and void some warranties! ;)

Debug ports are often serial ports, UART, and usually look like bundles of 4 or 6 pins. Sometimes, they are even labelled as debug ports, right on the PCB, just waiting for you to poke it. As they are mostly made for technical support, they are usually unprotected, sometimes you even have pin headers waiting for you to plug a cable. Otherwise, the pads (ie the small holes in the PCB where you can solder your components) will be unprotected and not connected to any device.

So, find the unused pads and solder some pin headers. If your PCB has more than one IC(chip?), check if you can find the main one and solder your pin headers on that debug port. The main chip is usually where you find the most important data. I always try to solder some pin headers to facilitate my access to the board.

Understand the Transmission Protocol

It is also important to understand how the communication protocol works. The *UART* communication protocol (Universal Asynchronous Receiver / Transmitter) is different than other common protocols like I2C basically for being a physical circuit in the microcontroller with just one function: transfer and receive data in a serial form. The good side of it is, since it is not a general purpose protocol like USB, it is possible to exchange data using only two wires. This kind of communication is called direct.

The data is transferred from the Tx (Transmitting) pin which is the source to the Rx pin (Receiving) to the destination, as illustrated in the pinout diagram below:

```
      +-------+        +-------+
      |       |        |       |
      |    Tx +--    --> Tx    |
      |       |  `..'  |       |
      |    Rx +--.'`.--> Rx    |
      |       |        |       |
      |   GND +--------+ GND   |
      |       |        |       |
      +-------+        +-------+
```

As the name suggests, the data transfer in UART is asynchronous, meaning that there is no clock tick available to synchronize the data sent and received. Instead, the source adds signal bits in the beginning and in the end of each data package sent. These bits are going to define the beginning and the end of a data package so that the destination understands where to start and where to end the reading process.

When the destination receives an initial bit, it starts to read the received data using a rate called *baud rate*. This baud rate is the data transfer rate in bps (bits per second) and both devices should be using the same rate (max 10% error) and data structure.

Finding the Pinout

The UART port can have two different pin layout. The 4-pin version have the following pins:

- Tx: this pin will be connected with the Rx of our receiver. Often have a value of 1 / true / on by default.
- Rx: this pin will be connected with the TX of our receiver. Often have a value of 1 / true / on by default.
- Vcc: POWER \o/ DO NOT CONNECT! Usually 3.3V or 5V.
- GND: aka Ground Pin, this pin will be connected with the GND of our receiver.

The 6-pin version includes two optional pins DTR and CTS. I will probably write about them later on this series. :)

Get your Multimeter!

Of course, you can just put wires between all free pads you can see – “trial & error” is also a method. ;) But depending on the device you want to understand, it can become pretty expensive. I usually prefer to analyze the board a bit further before sending random signals and put everything on fire – been there, done that! Not fun!

It is easier than you expect. A simple multimeter is enough to get most of the important information about our board. BUT you will NEVER hear me say “no” to an oscilloscope. :) The oscilloscope will show us exactly what’s happening on the electrical level so we can deduce which pin is what.

- The pin with a constant value of 0V is our GND.
- The pin with a constant value of 3.3V or 5V should be our Vcc.
- One of the pins with a floating value close to 0V should be our Rx.
    - (Makes sense? No? Think about Rx as a disconnected “listening” port)

Yay! Now that we know which pin is what, we just need to find out the *baud rate* to configure our receiver. For that, the easiest way is to use a logic analyzer. Connect your pins, start the option “Protocol analysis” and guess :) just keep increasing / decreasing the *baud rate* value until you start seeing plain text in the serial output.

With the pinout and the baud rate we can start communicating with our device! \o/

Post written originally in pt_br as a guest post to Mente Binaria and it wouldn’t be here without the precious help of Kylma.

Author: barbie aka Thaís