Point-to-point serial communication

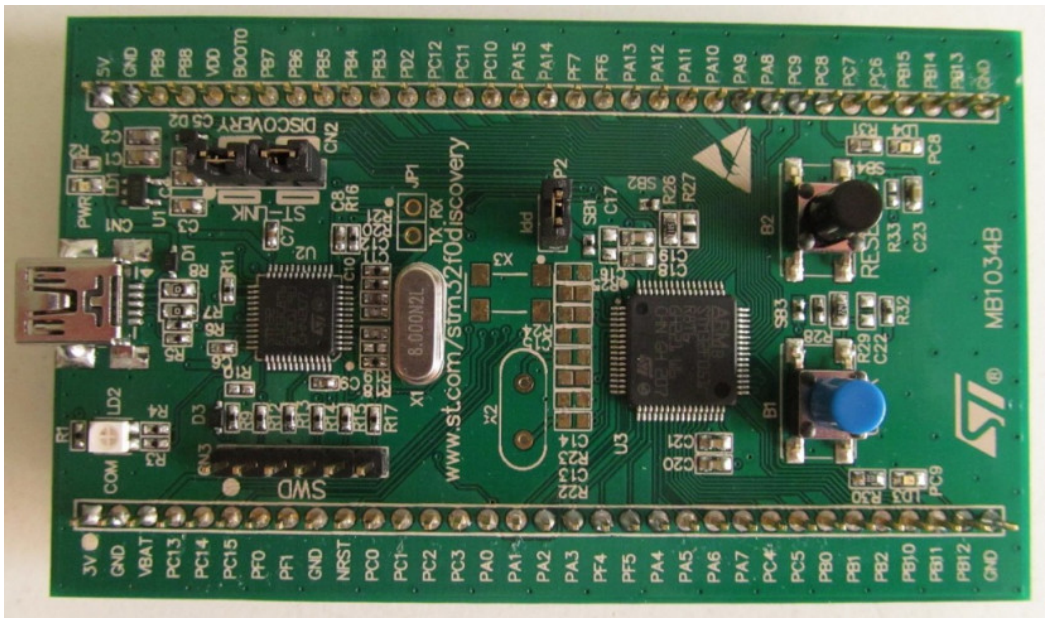
This challenge is about creating a simple serial terminal interface to access diagnostic information from an embedded micro-controller. Such simple serial interfaces help service engineers in the field with their trouble-shooting efforts. The introductory text below may introduce new terms and concepts, and it is up to you to find out what they mean and how to use them.


```
By Gorthmog - Own work: This file has been extracted from another
file: DEC VT100 terminal.jpg, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=78497051
```

Debugging embedded devices can be difficult. Not the least because they usually don't have a full-sized keyboard and a display directly attached. While debugging may be a challenge during product development, it becomes a

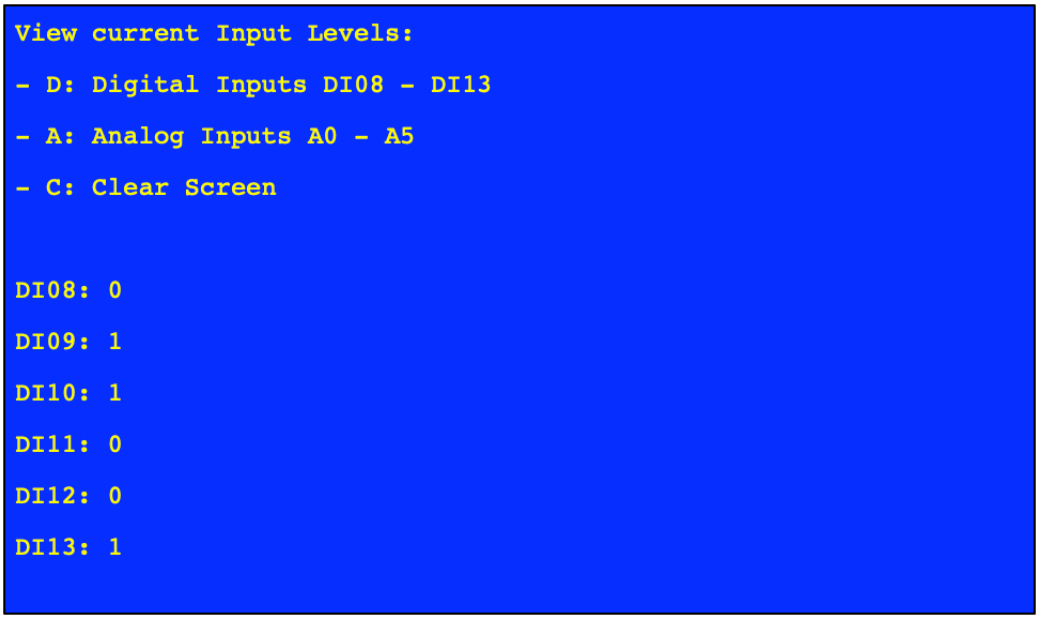
nightmare when you're in the field as a service engineer. And if this isn't enough, such embedded devices often cannot be simply taken to some service department, as they are an integral part of a washing machine, or a car.

This is the reason that almost all embedded devices can provide some diagnostic information via a simple serial interface, usually a UART (or USART, for Universal Serial Asynchronous Receiver Transmitter). This UART can be reached directly with logic levels, or sometimes they are attached via a 9-pin RS-232 port. Nowadays they may also be reached via a USB port. The chip behind the socket then translates from USB to UART serial and back.




```
Did you see the UART interface on this board? It is located directly
next to crystal X1.
```

The UART usually runs at a standard bitrate, and through this an ASCII terminal service program is used to control some diagnostic tests, often showing simple menus, like shown below.

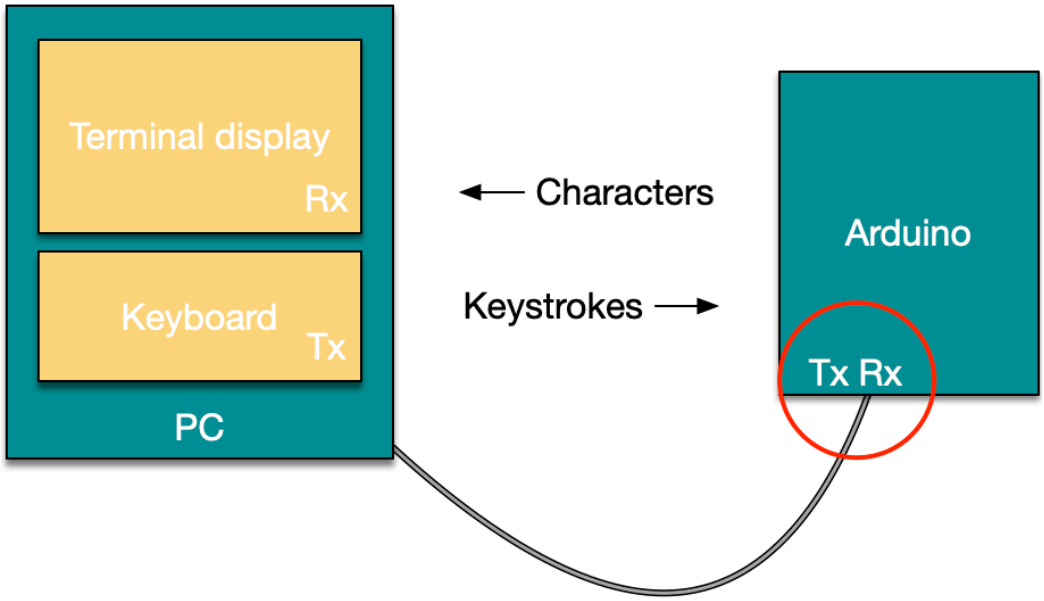



```
A menu created by a simple terminal server
```

The challenge for you is to design and build a setup with a *simple ASCII terminal server*, and a *software UART module* via which it can be reached from a standard laptop running a serial terminal program. The UART should be running at 9600 bps and 8n1 character format. You may **not** use Arduino pins 0 and 1, and you may also **not** use libraries like SoftwareSerial.

1. Making a simple ASCII terminal server

What really is a terminal server? In an embedded device it is just a small program that listens to a specific serial interface (our UART). It reacts upon characters (commands) that are received via that UART and it sends back (transmits) responses. The terminal server runs in an Arduino (for example), and on your laptop you will have to run a terminal client that connects via a serial interface (COM-port) to the Arduino.



So, the first step is to make a program on your Arduino that creates a menu similar to the one shown above at the *simple terminal server*. The terminal server listens to the Arduino hardware UART, and it only starts when the character 'S' is received. Then it shows a menu, providing the user with a number of options:

- Character 'D' received: show the levels of the digital input pins 8 through 13.
- Character 'A' received: show the levels of the analog input pins 0 through 5.
- Character 'C' received: clear the screen and stop the terminal server.

This shouldn't be too difficult, and the program can be used later to test your implementation of the software UART. Please note that your program immediately responds to received characters, therefore the Arduino serial monitor will not work correctly. You will need to use a serial terminal program on your laptop.

Estimated effort: 1 working day.

2. Making a design for the software UART

Now it's time to start thinking about a design for a UART. So, find out what a UART is, how it operates and, especially, what the requirements will be for communicating with your device from a standard laptop. This should result in knowing how many IO pins you need for full duplex communication and how to connect them to a laptop.

Because you will have to create a design, and later an implementation of a software UART. A number of resources are already brought together in the 'presentations' folder, and in the 'toolbox' section you will find background information from which the presentations take their input. And then there is of course the internet, other students, and teachers. Use it all when researching this.

Now, with respect to the design, we expect that you at least create a *state machine* for *transmitting* and one for *receiving*. And we also expect you to annotate these state machines with some explanatory text.

You probably also want to think now about an application programming interface (API): Which functions would a user (programmer) need to use your UART?

Finally, when researching UARTs, you will find information on hardware UART modules. If you're building a software UART, it is important that you realise that you don't need to build everything exactly the same way as is done for hardware UART modules. You only need your implementation to show the same external behaviour, the internal workings may be different.

NOTE: AD students can limit themselves to supporting transmitting and receiving characters in 8n1 format. BA students create a design that supports 5 to 8 data bits, 1 or 2 stop bits, and none, even or odd parity.

Estimated effort: 2 working days.

3. Implementing the UART module

After you made a design, you start implementing the UART module. You will experience a lot of issues. Often these will have to do with time and how you interpret time within your microcontroller and in your software. Our advice is that you make small steps and that you also try to *use your logic analyser* to find out what is actually happening.

Making small steps, verifying that each step that is taken is also correctly working, will help you to find out easier where you may have made an error, or where you made a mistake in reasoning. So don't rush it, because that will lead to more delays than taking it in small steps.

Estimated effort: 4 working days.

4. Knowledge transfer

After creating your own software UART module, you will have a communication solution for the rest of your working life. You will be surprised to see how often such a simple serial communication solution is used in devices of all sizes (from deep embedded into a simple portable FM radio, to wafer-steppers from ASML).

It is therefore VERY important that you document each and every step of this challenge, starting at the beginning at finding out what a UART actually is, all through the design phases, and finally building your actual source code. This also implies that you *include references* to information sources of any kind.

We expect you to hand in all documentation and source code to Canvas.

5. Making it fancy

So, suppose you're ready and there's still a lot of time left. Then there are some nice additional challenges related to this subject:

- You may want to create a VT100 terminal emulation. This means that you will have to check out what is meant with ANSI escape sequences, and how they can be used in terminal programs.
- You may want to check out what is actually meant with UTF-8 encoding, and what you can do with UTF-8 encoded ASCII characters.
- In principle, it is okay to have an implementation without the use of interrupt service routines (ISRs). But you may also create an implementation that includes the use of ISRs. Please note that your terminal server must still be working correctly.
- You may want to check what the fastest bitrate is that you can reach while still being robust.
- And more... (up to you)