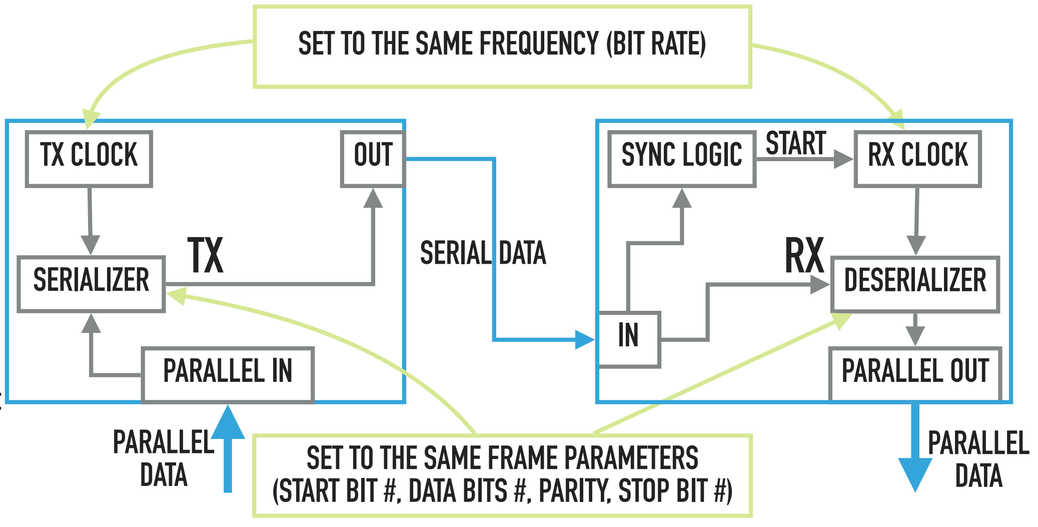
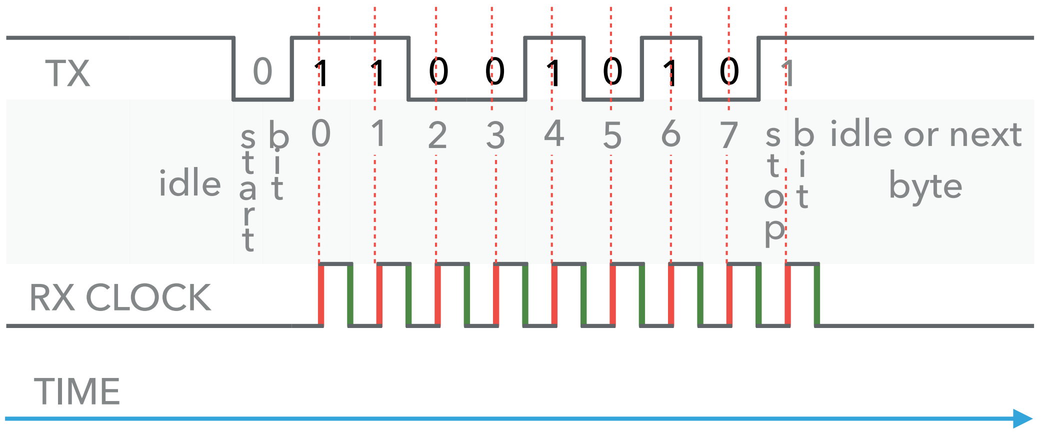
SERIAL INTERCONNECTIONS

- Example of transmission over an Asynchronous Serial link
  - The data line is held HIGH when the link is idle
  - When TX wants to transmit a frame, it sends the Start bit (LOW)
  - SYNC LOGIC in RX recognizes the HIGH-LOW transition as the start condition and starts its clock
  - On each pulse of TX CLOCK, the serializer in TX transmits one bit of the frame contained in the PARALLEL IN register through the SERIAL DATA line
  - On each rising edge (red) of RX CLOCK, the deserializer in RX samples one bit from SERIAL DATA and enqueues it in the PARALLEL OUT register
  - At the end of the frame, TX sends the Stop bit (HIGH) and RX recognizes the end of transmission
  - SERIAL DATA is kept HIGH until the next frame is sent

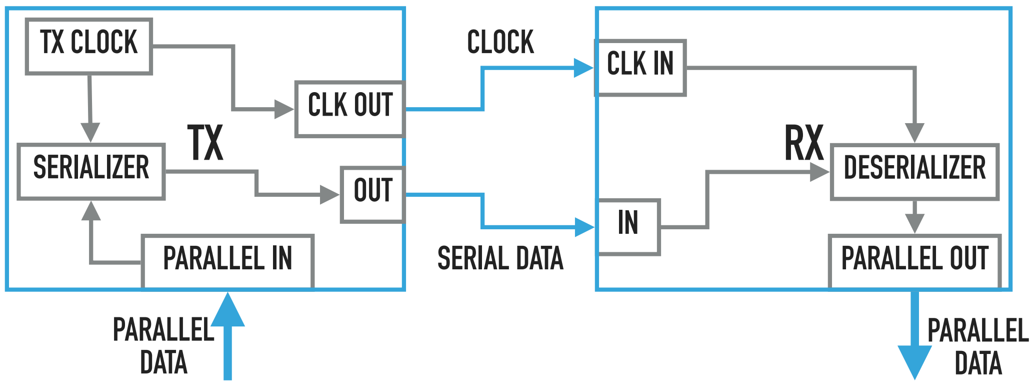


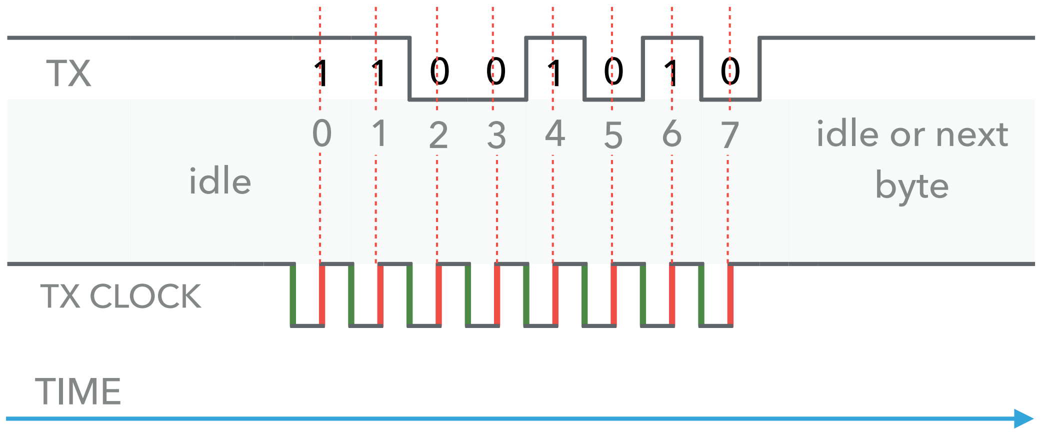


Transmission of value 0x53, ASCII character ‘S’, over an asynchronous serial link with parameters: 1 start bit, 8 data bits, no parity, 1 stop bit.
Least significant bit is sent first.

SERIAL INTERCONNECTIONS

- Example of transmission over a Synchronous Serial link with explicit clock signal
  - The SERIAL DATA and CLOCK are held HIGH when the link is idle
  - When TX wants to transmit a frame, it starts its clock
  - On each falling edge (green) of TX CLOCK, TX transmits one bit of the frame on the SERIAL DATA line
  - On each rising edge (red) of CLOCK, the deserializer in RX samples one bit from SERIAL DATA and enqueues it in the PARALLEL OUT register
  - SERIAL DATA and CLOCK are kept HIGH until the next frame is sent

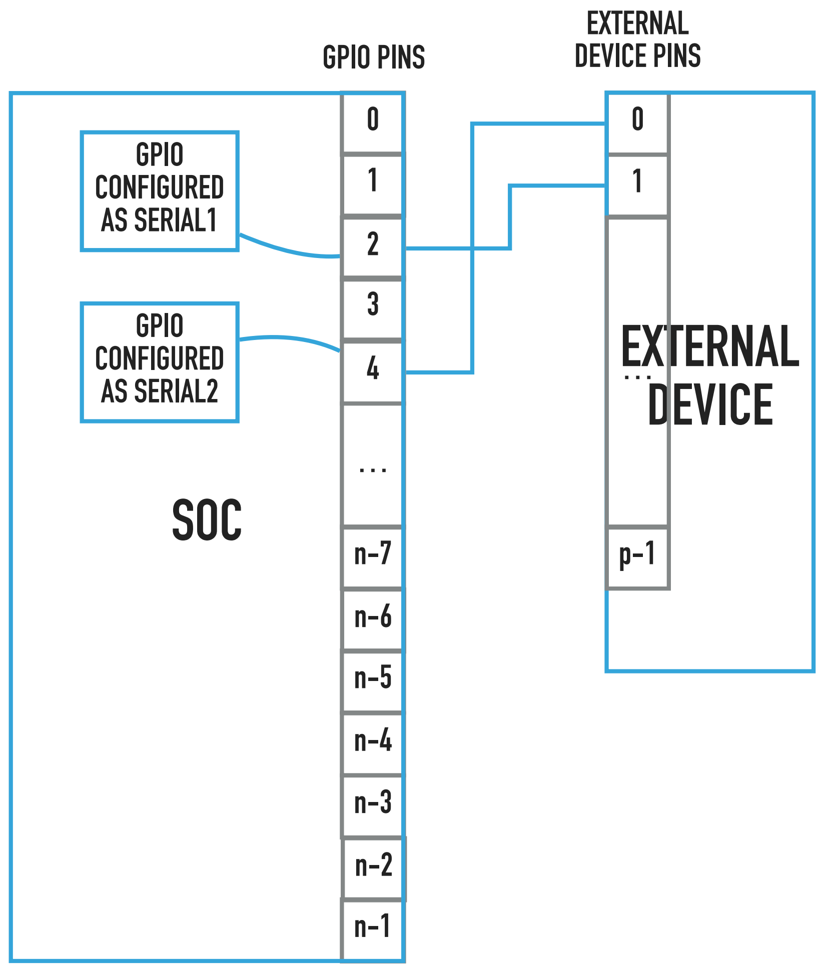




Transmission of value 0x53, ASCII character 'S', over a synchronous serial link.

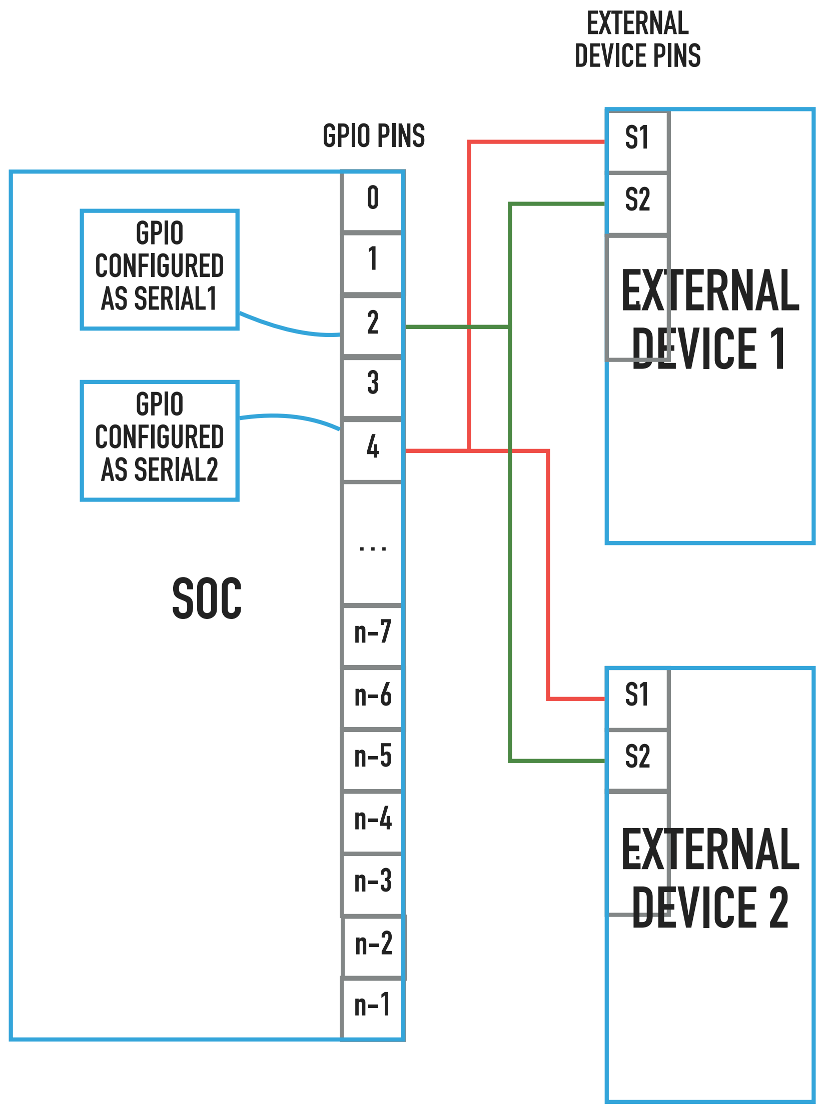
SERIAL INTERCONNECTIONS

- Several serial interconnection interfaces are available in SoCs and MCUs
  - Point-to-point
    - Universal Asynchronous Receiver/Transmitter (UART)
    - Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
      - May support several point-to-point and bus protocols (IrDA, Modbus)



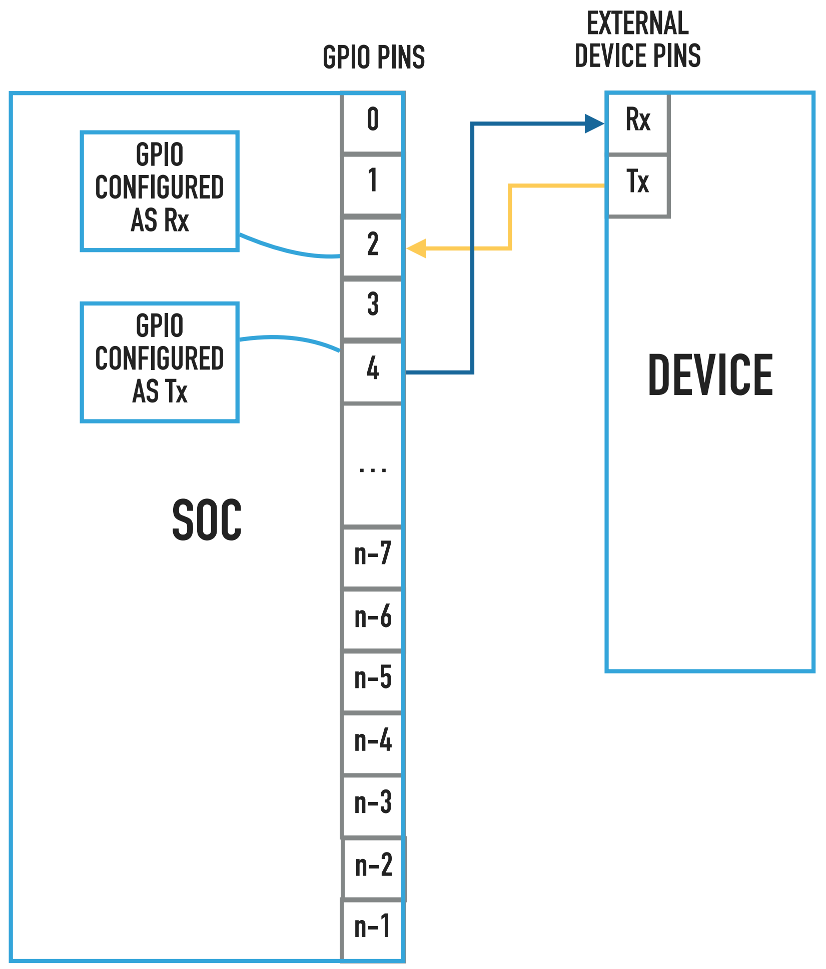
SERIAL INTERCONNECTIONS

- Buses
  - Modbus (RS-485)
  - Inter-integrated circuit - I^{2}C bus (Philips, NXP)
  - 1-Wire® (Dallas Semiconductor)
  - CAN (Controller Area Network) for automotive applications
- Sort of a bus
  - Serial peripheral Interconnect (SPI) (serial, point to point with optional selection signals)



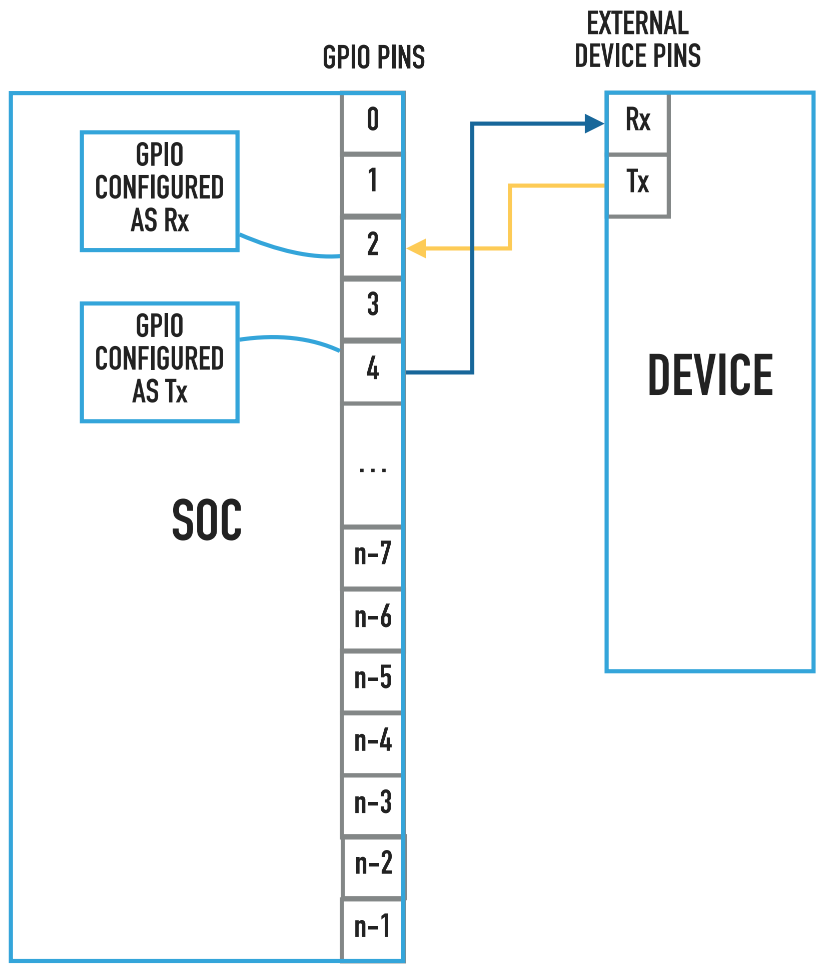
UART & USART

- UART and USART include optional control signals
  - At least two digital signals are required:
    - Transmit (Tx) - output signal
    - Receive (Rx) - input signal
    - The Tx pin in one device is connected to the Rx pin in the other
  - Bit rates range from few hundred to few million per second



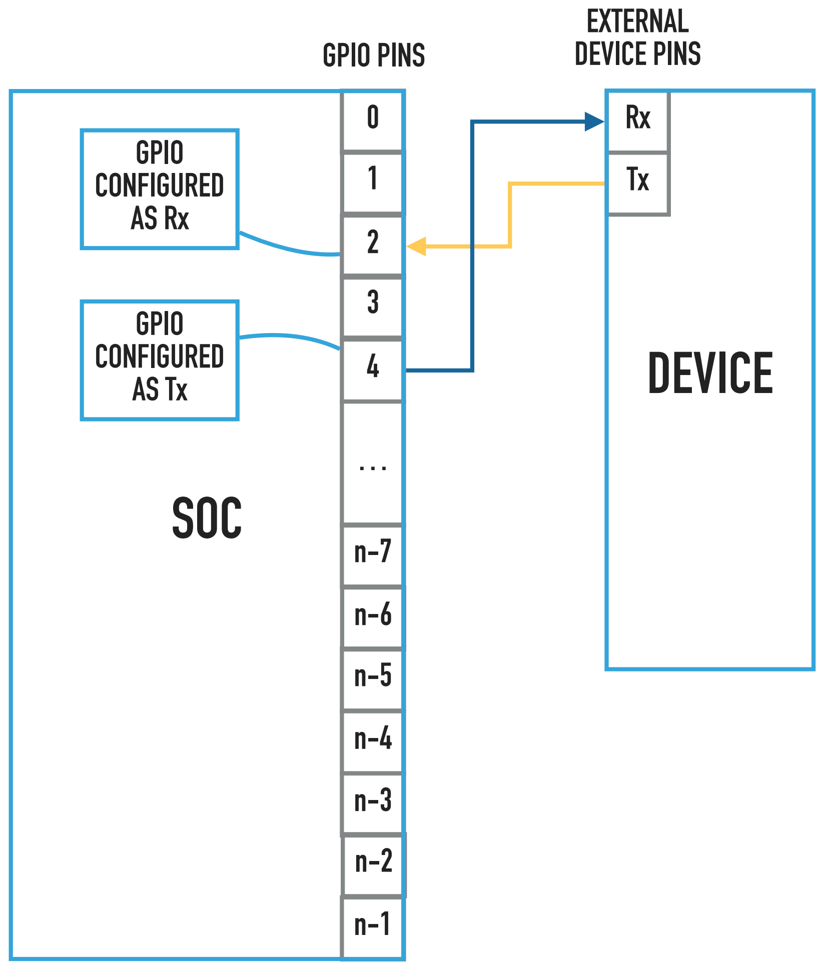
UART & USART

- Usually, UART and USART are managed by internal peripherals
  - In practice, every embedded system includes at least one of these peripherals, especially of the UART type
  - These peripherals are usually quite versatile and support several transmission protocols through a wide range of settings that include bit rates, frame length, error correction, flow control, electrical characteristics of signals, and optional signals management



UART & USART

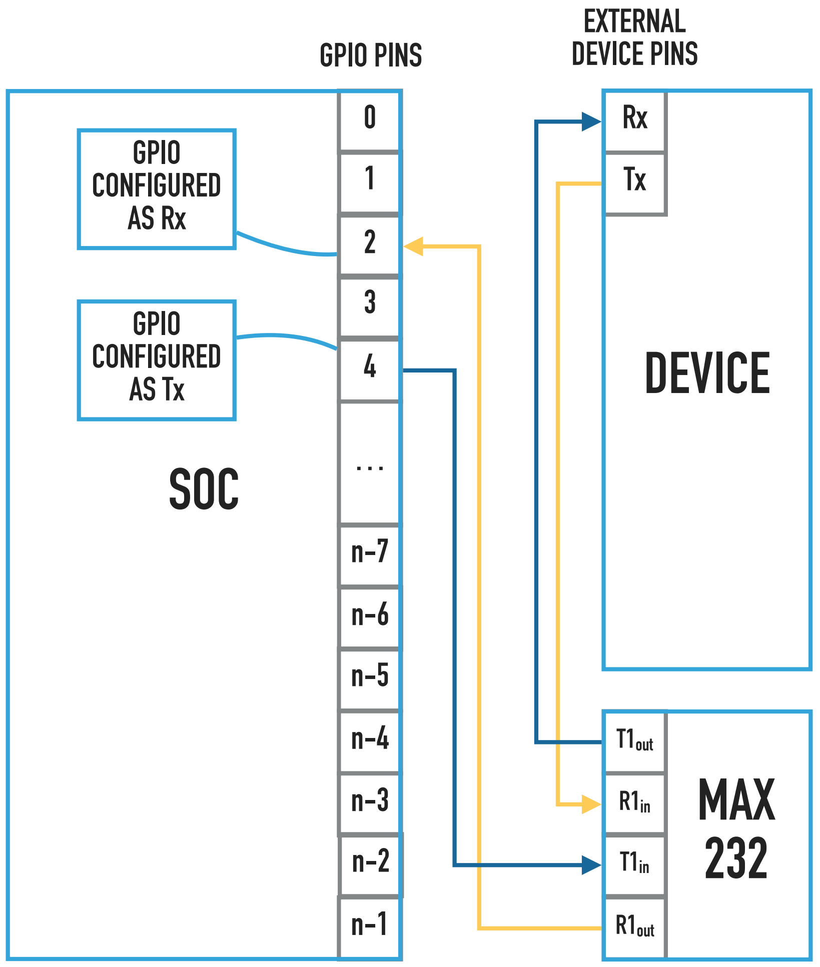
- Two devices can be connected only if both meet the same electrical specifications
- The electrical specifications of UART and USART may vary
  - There are standards such as RS-232 and RS-485
  - For short-range connections (short cables) RS-232 is a widely adopted interface
    - Valid voltage ranges are
      - 3V to 15V for logical zero
      - -15V to -3V for logical one
  - The serial interfaces found in PCs until the advent of USB mostly followed the RS-232 standard



Two directly connected devices sharing compatible electrical specifications

UART & USART

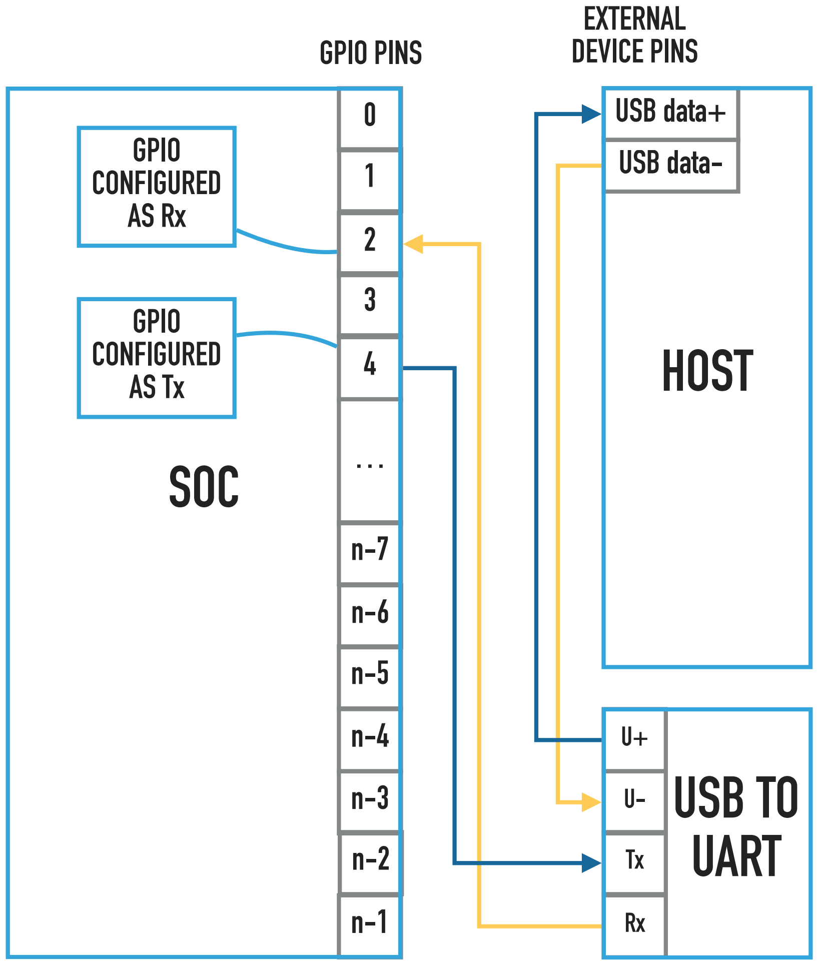
- On SoCs and MCUs, usually, the voltage levels of UART and USART signals are those used to supply chips with power:
  - Positive power-supply voltage **(Vcc)**, e.g. 5V or 3.3V, for logical one
  - **Ground** (**GND**) voltage for logical zero
  - These are called **Transistor-Transistor Logic (TTL)** levels
  - Voltage level translators are **required** to connect electrically different interfaces (e.g. a TTL UART to an RS-232) to avoid damages
  - Even connecting a device operating with Vcc=3.3V to another with Vcc=5V should be avoided



A bidirectional voltage level translator (MAX232)is used to connect the RS-232 port of the device to the TTL UART of the SoC. The same GND is used by the three devices.

UART & USART

- UART and USART are used for basic connectivity, logging and debugging
  - A terminal, nowadays a computer running a terminal emulator, can be connected to any of such interfaces for these purposes
  - UART and USART where included in PCs until the advent of USB
    - Now, adaptors (e.g. USB devices) are needed to connect a PC to a UART-provided systems



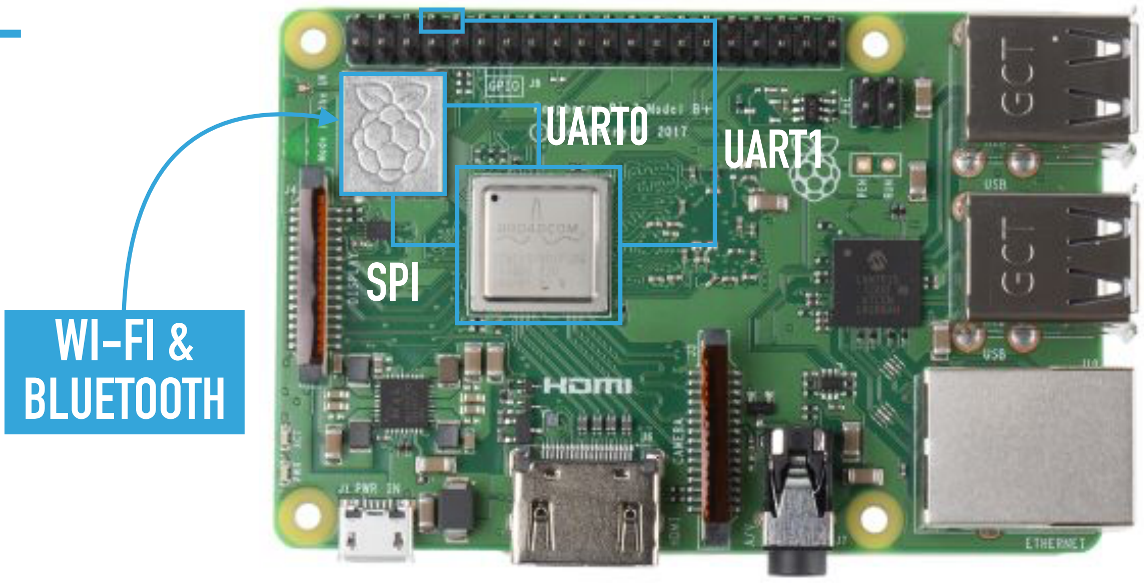
A USB adaptor is used to connect the host to the RS-232 port of the SoC. The devices must be properly grounded.

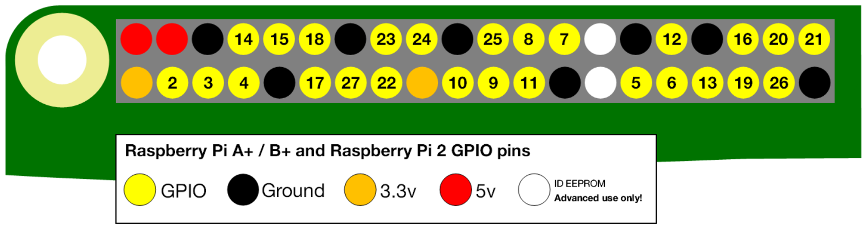
UARTS: RASPBERRY PI 3 B+

- The Raspberry Pi 3 B+ is provided with two UARTs:
    - A PL011 (UART0)
    - A mini UART (UART1)
  - Both are 3.3V devices so, in general, adaptors are needed to connect any of these ports to an RS-232 interface
  - UART0 is connected to the Bluetooth circuity
  - UART1 can be exposed to GPIO14 (TX) and GPIO15 (RX)
- The Wi-Fi circuitry is connected to the SoC through an SPI interface

PCB TRACES

Raspberry Pi 3 B+

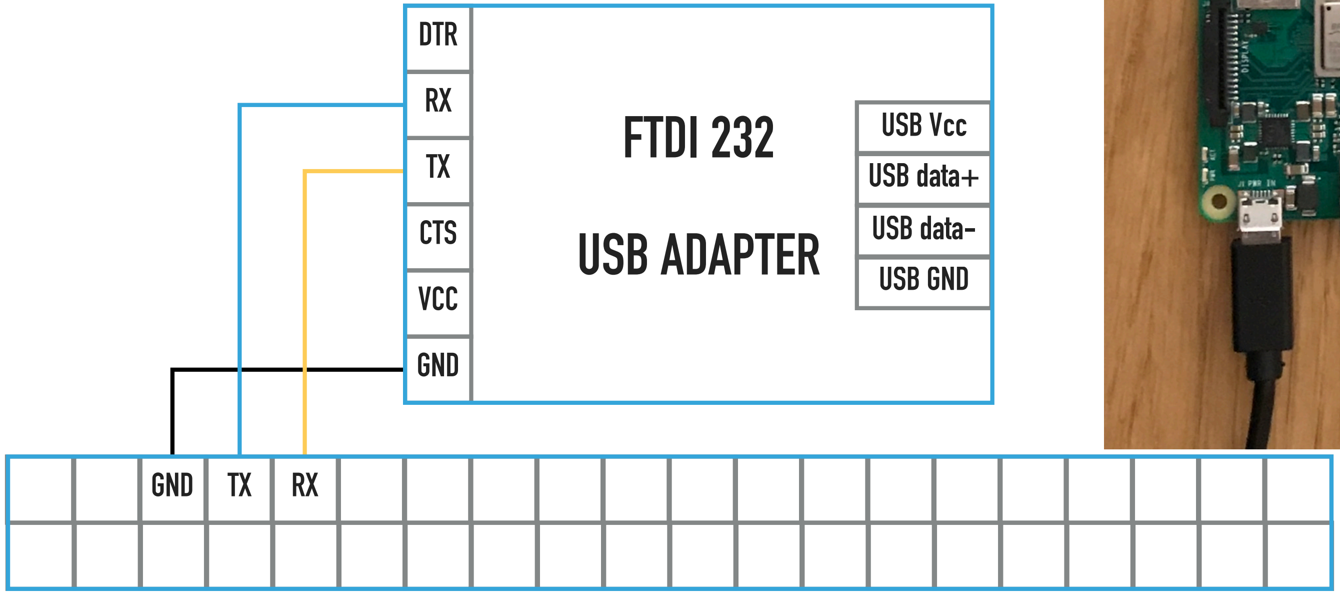


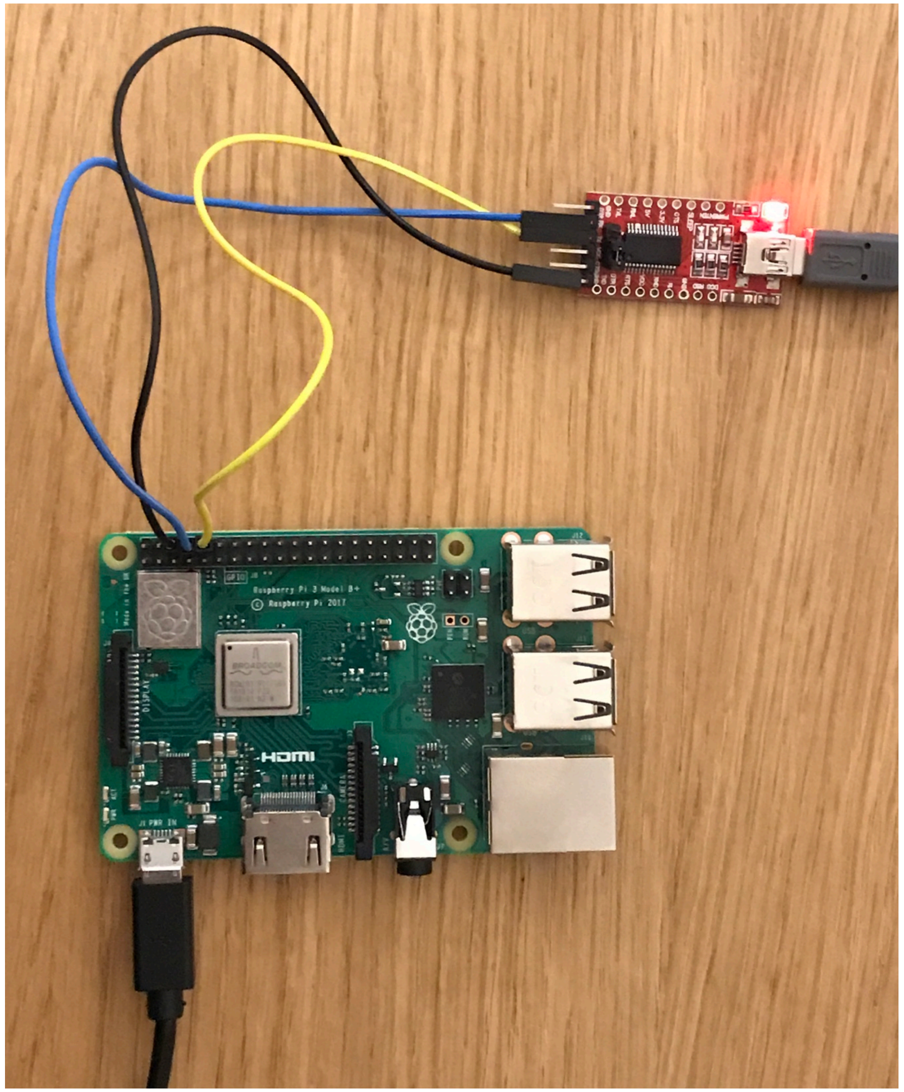


https://www.raspberrypi.org/documentation/configuration/uart.md

UARTS & RASPBERRY PI 3 B+

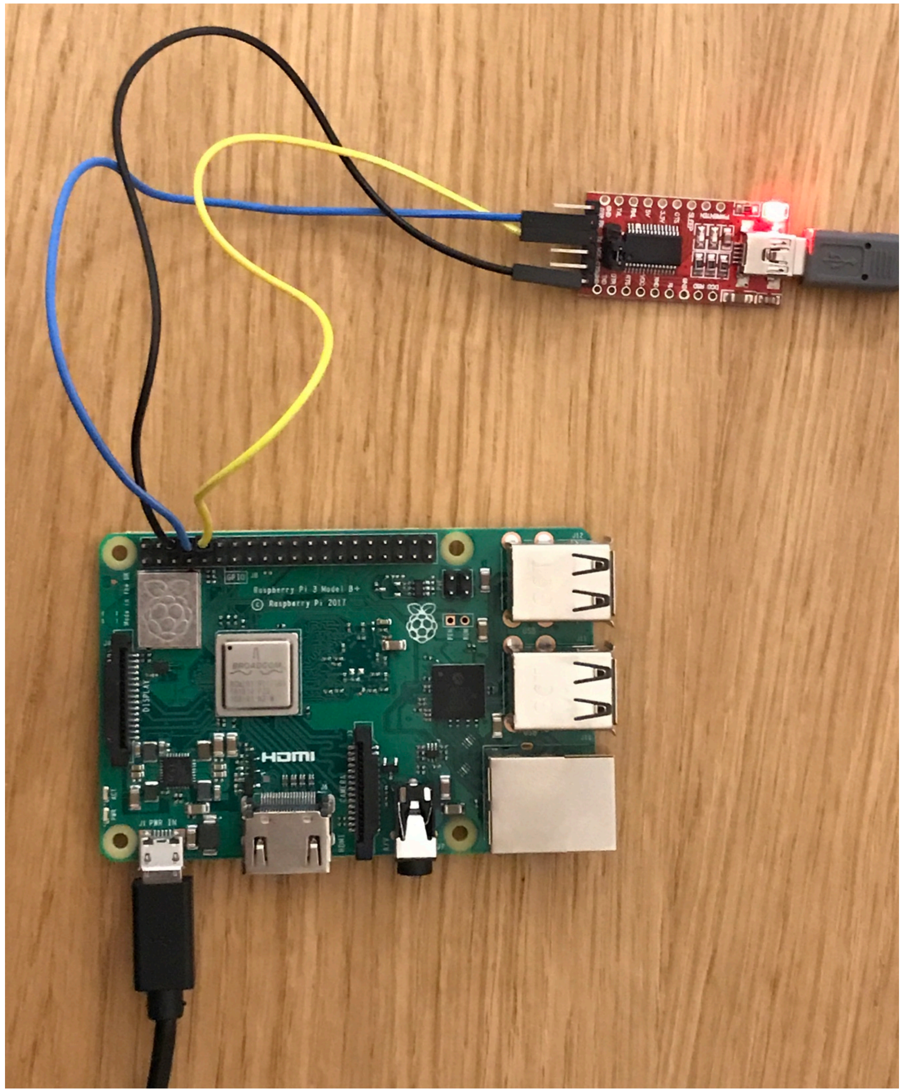
- An adaptor with voltage level translators permits to connect a USB-provided host computer to the Pi
- Proper grounding and powering is required
  - The adapter and the Pi in the picture share the same power supply (Vcc and GND pins of the USB connector)
- The host is augmented with a **Virtual Communication Port (VCP)** that is then used by the software interacting with the Pi





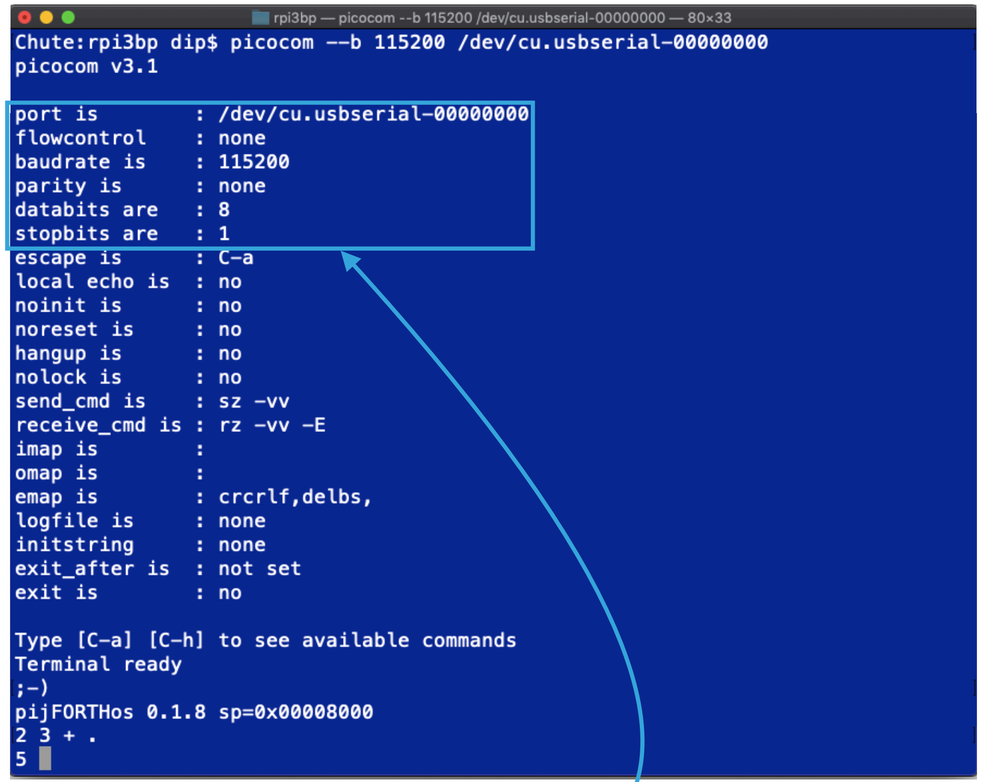
UARTS & RASPBERRY PI 3 B+

- With a VCP on the host and the proper software on the microSD card it is possible to bare-metal program the machine through:
  - Cross compilation
    - A specially crafted boot loader on the Pi and a companion program on the development machine are used to upload a program straight to the RAM of the Pi and execute it
  - Interactive on-target programming
    - A Forth environment on the Pi and a terminal emulator on the development machine are used to interact with the hardware through a command line interface



FORTH ON RASPBERRY PI 3 B+

- Interactive on-target programming
  - A Forth (pijFORTHos 0.1.8) is booted on the Pi
  - A terminal emulator (picocom v3.1) is launched on the development machine with the same VCP parameters as the Pi UART
    - 115200 bit/s, 1 stop bit, 8 data bits, no parity, no flow control
  - The Pi can be programmed through the serial interface


rpi3bp — picocom --b 115200 /dev/cu.usbserial-00000000 — 80×33

```
Chute:rpi3bp dip$ picocom --b 115200 /dev/cu.usbserial-00000000
picocom v3.1

port is         : /dev/cu.usbserial-00000000
flowcontrol     : none
baudrate is     : 115200
parity is       : none
databits are    : 8
stopbits are    : 1
escape is       : C-a
local echo is   : no
noinit is       : no
noreset is      : no
hangup is       : no
nolock is       : no
send_cmd is     : sz -vv
receive_cmd is  : rz -vv -E
imap is         :
omap is         :
emap is         : crcrlf,delbs,
logfile is      : none
initstring      : none
exit_after is   : not set
exit is         : no

Type [C-a] [C-h] to see available commands
Terminal ready
;-)
pijFORTHos 0.1.8 sp=0x00008000
2 3 + .
5 
```

FORTH ON RASPBERRY PI 3 B+

- The FORTH standard word @ (fetch) is used to read a word-sized value (32 bits) from memory. Fetch pops the memory address from the Top Of Stack (TOS), then pushes the read value on the stack
- The value of GPFSEL2 is thus read with the following code:

```
HEX
3F200008 @ U.
8000000
```

  - Register GPFSEL2 contains `0x08000000` so bits 29-27 (GPIO29) have value `0b001`. The pin is configured as an output so its level drives the green LED on the Pi

FORTH ON RASPBERRY PI 3 B+

- The FORTH standard word `!` (store) is used to write a word-sized value (32 bits) to memory. Store pops the memory address from the Top Of Stack (TOS) and the value to write from the element under the TOS
- The LED is switched on by writing a '1' to bit 29 of the `GPSET0` register, which controls GPIOs 0-29, with the following code:

```
20000000 3F20001C !
```

- The LED is switched off similarly by writing a '1' to bit 29 of the GPCLR0 register:

```
20000000 3F200028 !
```

FORTH ON RASPBERRY PI 3 B+

- A few constants are defined for the GPIO registers

```
3F200008 CONSTANT GPFSEL2
3F20001C CONSTANT GPSET0
3F200028 CONSTANT GPCLR0
```

- The phrase `CONSTANT` "`constantname`" defines a new word using for its name the next space-terminated string of characters in the input (e.g. `GPFSEL2`). The TOS (e.g. `3F200008`) is consumed and associated to `constantname`. When executed, `constantname` pushes its associated value onto the stack

- The LED is switched on using :

```
20000000 GPSET0 !
```

- The LED is switched off with:

```
20000000 GPCLR0 !
```

FORTH ON RASPBERRY PI 3 B+

- The new word LED is defined

```
: LED   20000000 GPSET0 GPCLR0 ;
```

  - When executed
    - the value `0x20000000` is pushed on the stack
    - `GPSET0` is executed
      - the execution of `GPSET0` pushes `0x3F00001C` on the stack
    - `GPCLR0` is executed
      - the execution of `GPSET0` pushes `0x3F000028` on the stack

FORTH ON RASPBERRY PI 3 B+

- After the execution of LED the stack contains the values:

```
LED
U.
3F200028 U.
3F20001C U.
20000000
```

- So to switch the LED on or off it is possible to use the values left on the stack by LED removing the unwanted register address for each case
- To turn the LED on the TOS value (address of `GPCLR0`) is DROPped

```
LED
DROP
!
```

- To turn the LED of the value under TOS is NIPped (`NIP` can be defined as `SWAP DROP`)

```
LED
NIP
!
```

FORTH ON RASPBERRY PI 3 B+

- Now, two more words may conveniently defined

```
: ON     DROP ! ;
: OFF   NIP ! ;
```

- After these words are defined, two FORTH phrases can be used to drive the LED instead of writing to registers

```
LED ON
```

- to turn the LED on and

```
LED OFF
```

- to turn it off
- This code defines a small programming language to drive the LED

FORTH ON RASPBERRY PI 3 B+

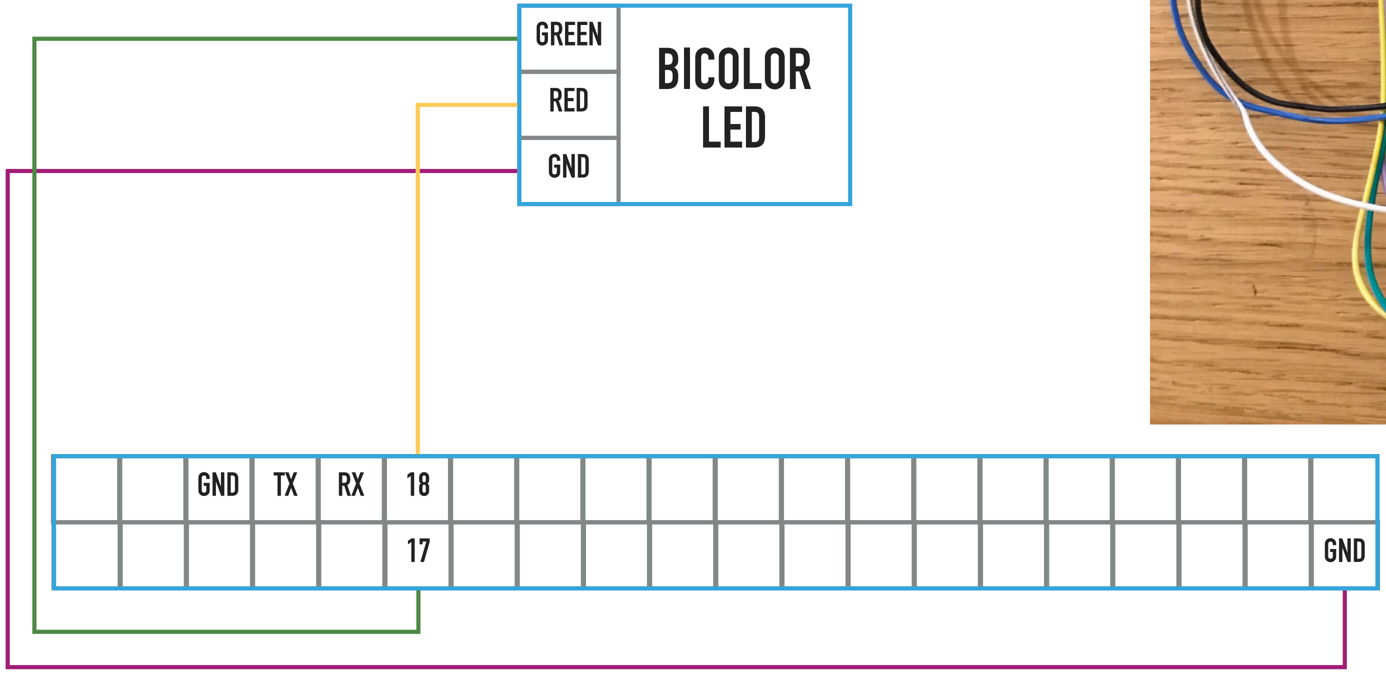
- Complete code of the new language to drive the green LED:

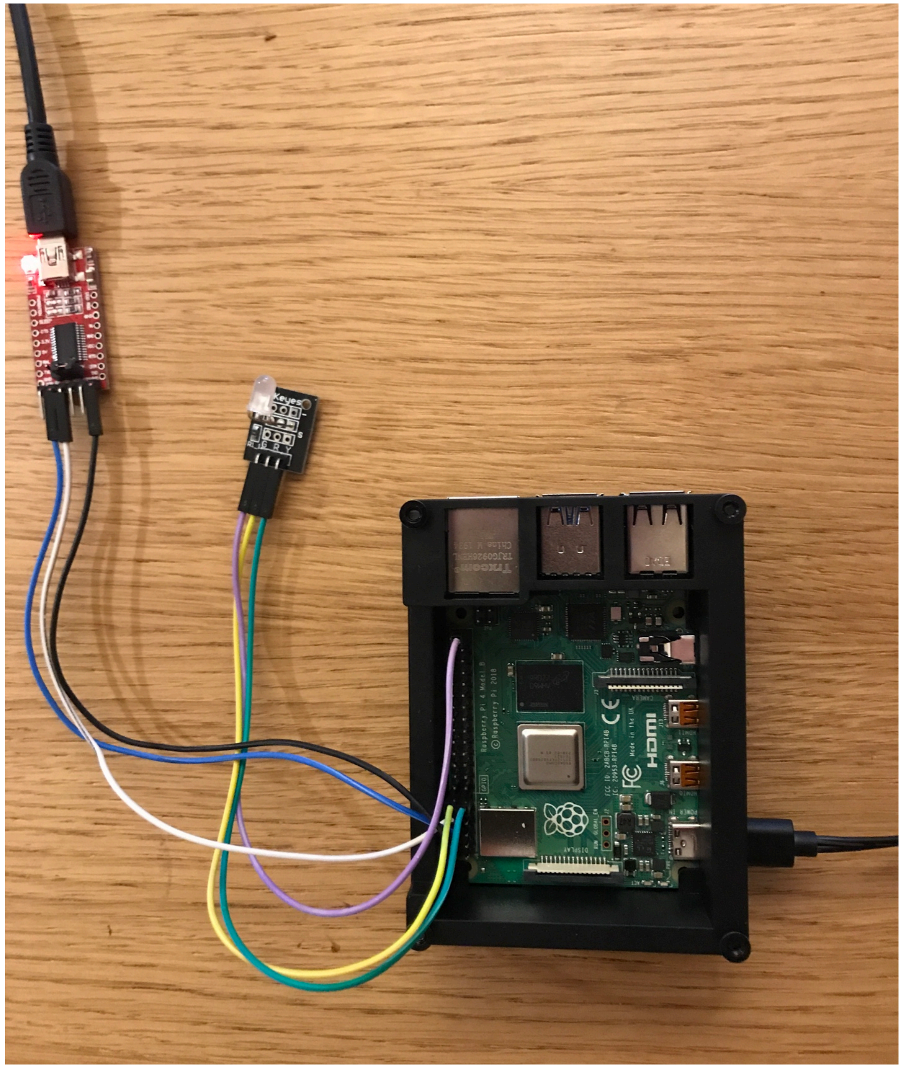
```
HEX
3F200008 CONSTANT GPFSEL2
3F20001C CONSTANT GPSET0
3F200028 CONSTANT GPCLR0

: LED   20000000 GPSET0 GPCLR0 ;
: ON    DROP ! ;
: OFF   NIP ! ;
```

- This code only works on the Raspberry Pi 3 B+ because the on-board LED is connected to GPIO 29
- In most applications GPIOs exposed in the GPIO connector are used
- Pi models may differ in the base Peripheral address (e.g. `0x3F000000` for the Pi 3, `0xFE000000` for the Pi 4) but the register offsets are the same

FORTH ON THE RASPBERRY PI 4 B+

- Another example: drive a bicolor LED connected to GPIO 17 (GREEN) and GPIO 18 (RED)





FORTH ON THE RASPBERRY PI 4 B+

- Another example: drive a bicolor LED connected to GPIO 17 (GREEN) and GPIO 18 (RED)
- First the GPFSEL1 value is read:

```
FE200004 @ U.
224000
```

- Register GPFSEL1 contains `0x00224000` so bits 21-23 (GPIO17) have value `0b001` and bits 24-26 have value `0b000`
  - GPIO17 is configured as an output
  - GPIO18 is configured as an input
- GPIO18 must be configured as an output by changing also bits 24-26 of GPFSEL1 to 0b001:

```
01224000 FE200004 !
```

FORTH ON THE RASPBERRY PI 4 B+

- Complete code of the language to drive the bicolor LED:

```
HEX
FE200004 CONSTANT GPFSEL1
FE20001C CONSTANT GPSET0
FE200028 CONSTANT GPCLR0

: GREEN   20000 ;
: RED     40000 ;
: LED     GPSET0 GPCLR0 ;
: ON      DROP ! ;
: OFF     NIP ! ;
```

- The code defines new commands to turn:
  - the green LED on:

```
GREEN LED ON
```

  - the green LED off:

```
GREEN LED OFF
```

  - the red LED on:

```
RED LED ON
```

  - the red LED off:

```
RED LED OFF
```