This week

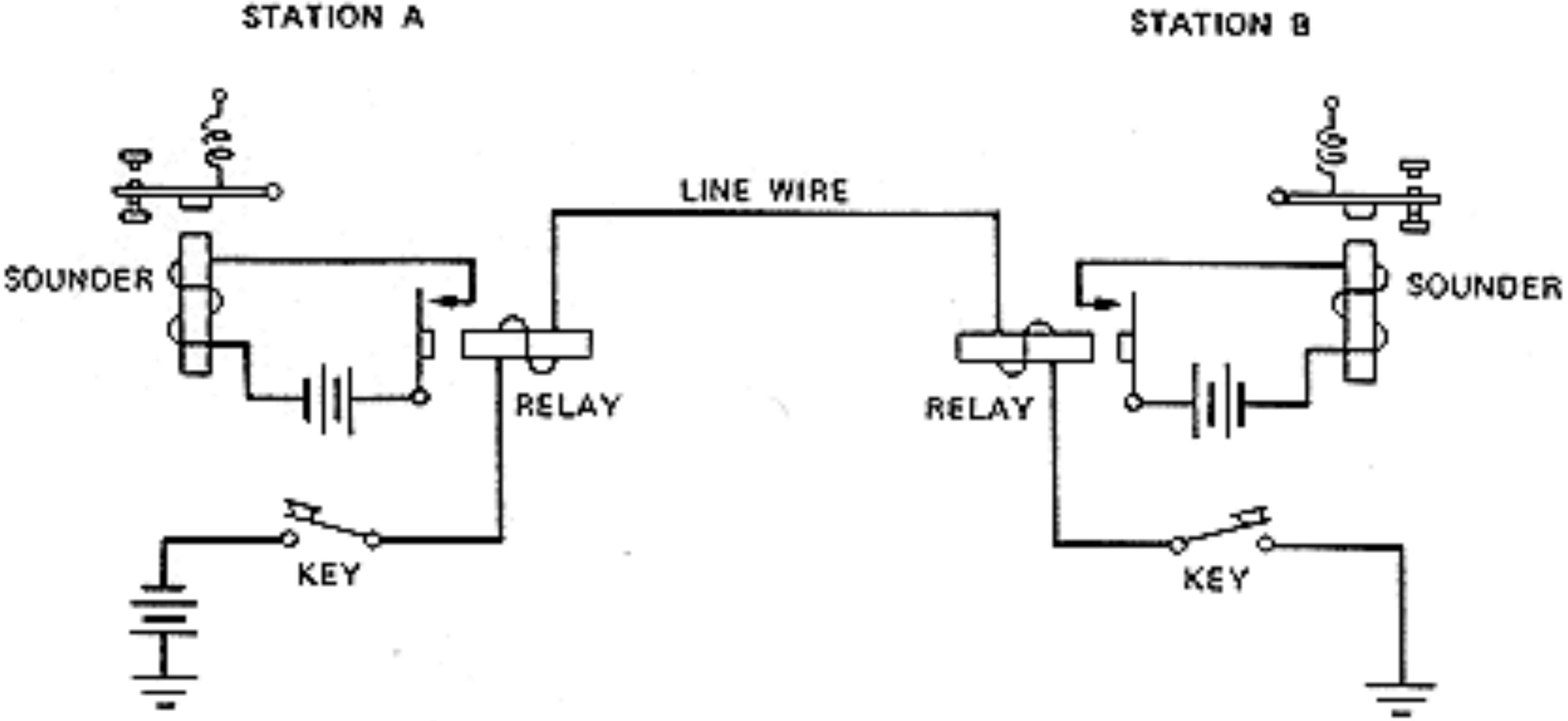
- assign2 due Tues

  (does TSA approve of breadboard clocks?)
- lab3, assign3 coming up

Goals for today

- Leftovers: full stack frame
- Next up: **Communication**
  - Serial protocol, uart
  - Ascii character codes
  - C-strings
  - GDB in simulation mode

Simplex Telegraph



Elementary neutral telegraph circuit.

http://people.seas.harvard.edu/~jones/cscie129/nu_lectures/lecture5/elecmag_tel/morse_tel.html

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	● ▬
B	▬ ● ● ●
C	▬ ● ▬ ●
D	▬ ● ●
E	●
F	● ● ▬ ●
G	▬ ▬ ●
H	● ● ● ●
I	● ●
J	● ▬ ▬ ▬
K	▬ ● ▬
L	● ▬ ● ●
M	▬ ▬
N	▬ ●
O	▬ ▬ ▬
P	● ▬ ▬ ●
Q	▬ ▬ ● ▬
R	● ▬ ●
S	● ● ●
T	▬
U	● ● ▬
V	● ● ● ▬
W	● ▬ ▬
X	▬ ● ● ▬
Y	▬ ● ▬ ▬
Z	▬ ▬ ● ●
1	● ▬ ▬ ▬ ▬
2	● ● ▬ ▬ ▬
3	● ● ● ▬ ▬
4	● ● ● ● ▬
5	● ● ● ● ●
6	▬ ● ● ● ●
7	▬ ▬ ● ● ●
8	▬ ▬ ▬ ● ●
9	▬ ▬ ▬ ▬ ●
0	▬ ▬ ▬ ▬ ▬

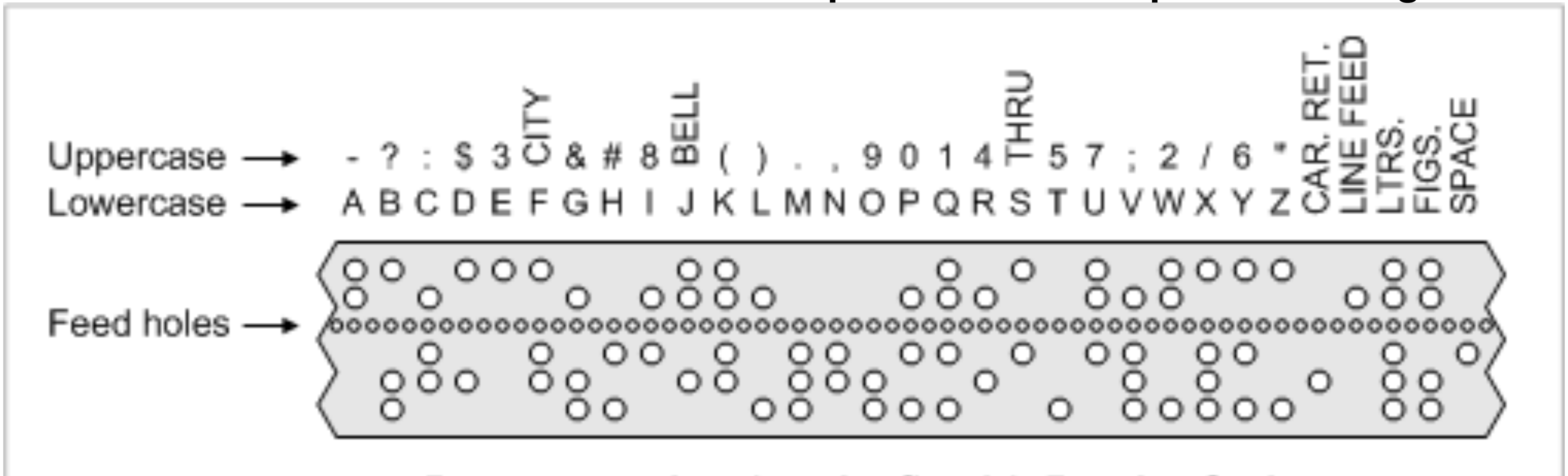
sos.c

Teletype

http://www.smecc.org/police_-__fire_-_civil_defense_communications.htm

Baudot Code

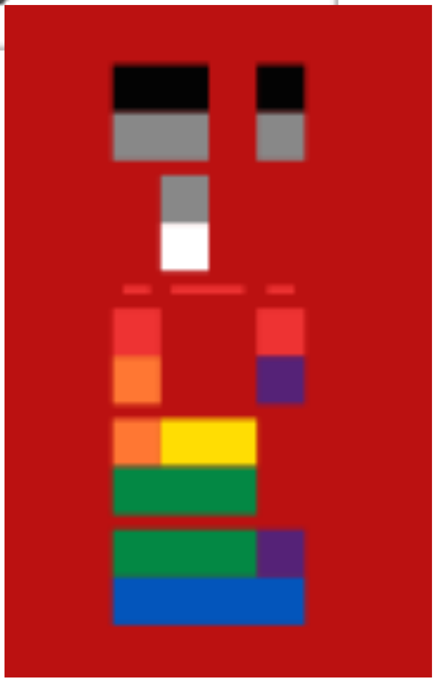
https://savzen.wordpress.com/tag/baudot/


Uppercase →
- ? : $ 3 CITY & # 8 BELL ( ) . , 9 0 1 4 THRU 5 7 ; 2 / 6 " CAR. RET. LINE FEED LTRS. FIGS. SPACE
Lowercase →
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Feed holes →

Paper tape showing the five-bit Baudot Code

Baud: Number of symbols per second

Who knew Coldplay were nerds?

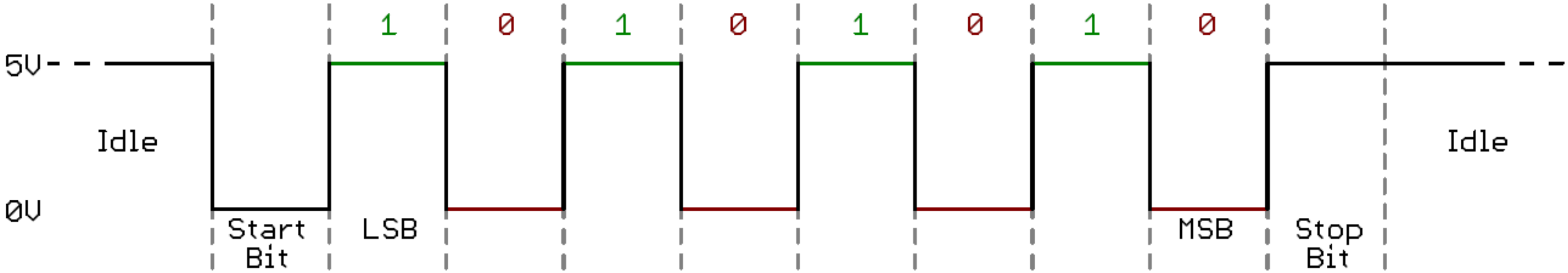


```
% ascii
   2 3 4 5 6 7
 -------------
0:   0 @ P ' p
1: ! 1 A Q a q
2: " 2 B R b r
3: # 3 C S c s
4: $ 4 D T d t
5: % 5 E U e u
6: & 6 F V f v
7: ' 7 G W g w
8: ( 8 H X h x
9: ) 9 I Y i y
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o DEL
```

```
0x30 = ‘0’
0x31 = ‘1’
0x40 = ‘@‘
0x41 = ‘A’
```

Asynchronous Serial Communication

(implicit clock)



1 start bit (0), 8-bits (data), 1 stop bit (1)

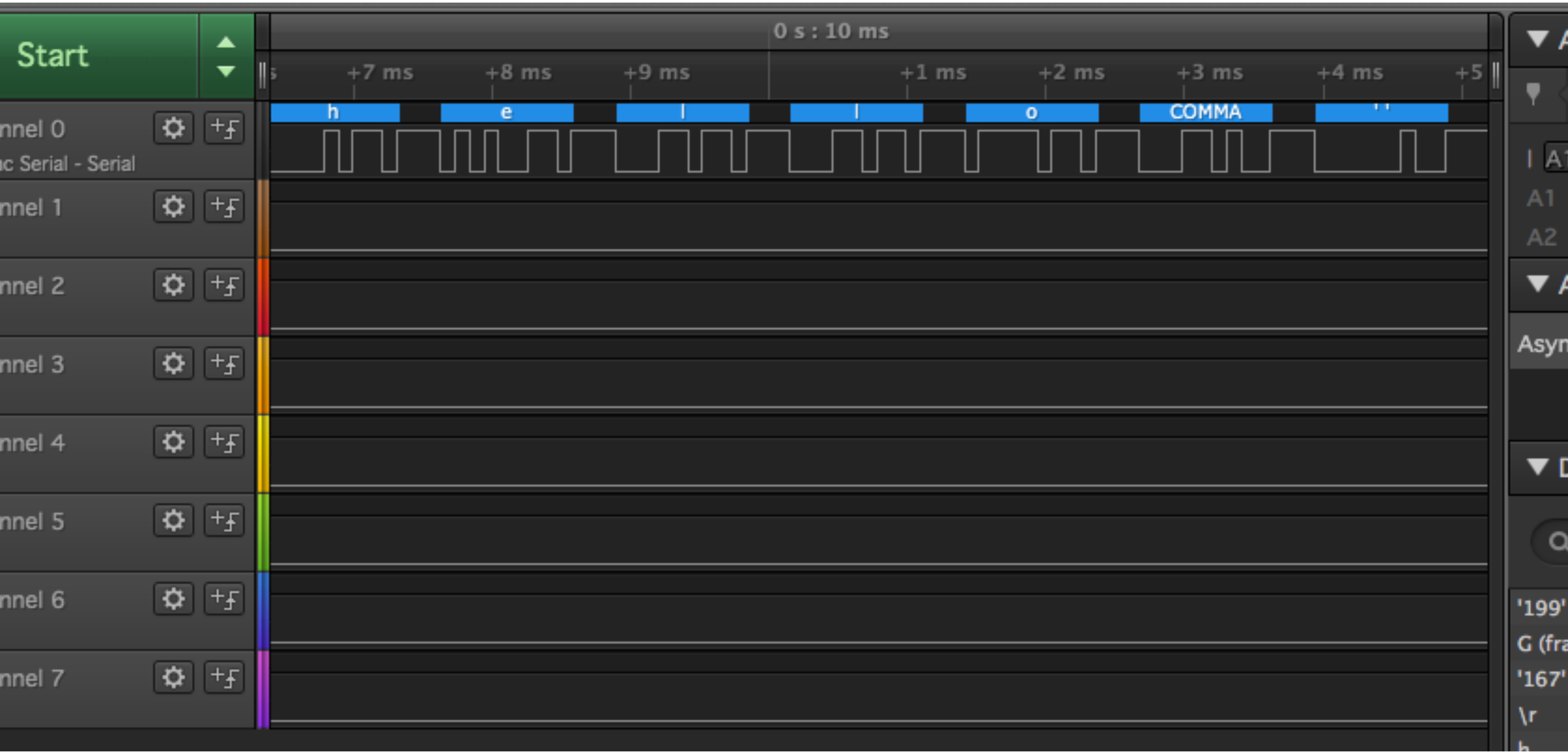
9600 baud = 9600 bits/sec

(1000000 usecs)/9600 ~ 104 usec/bit

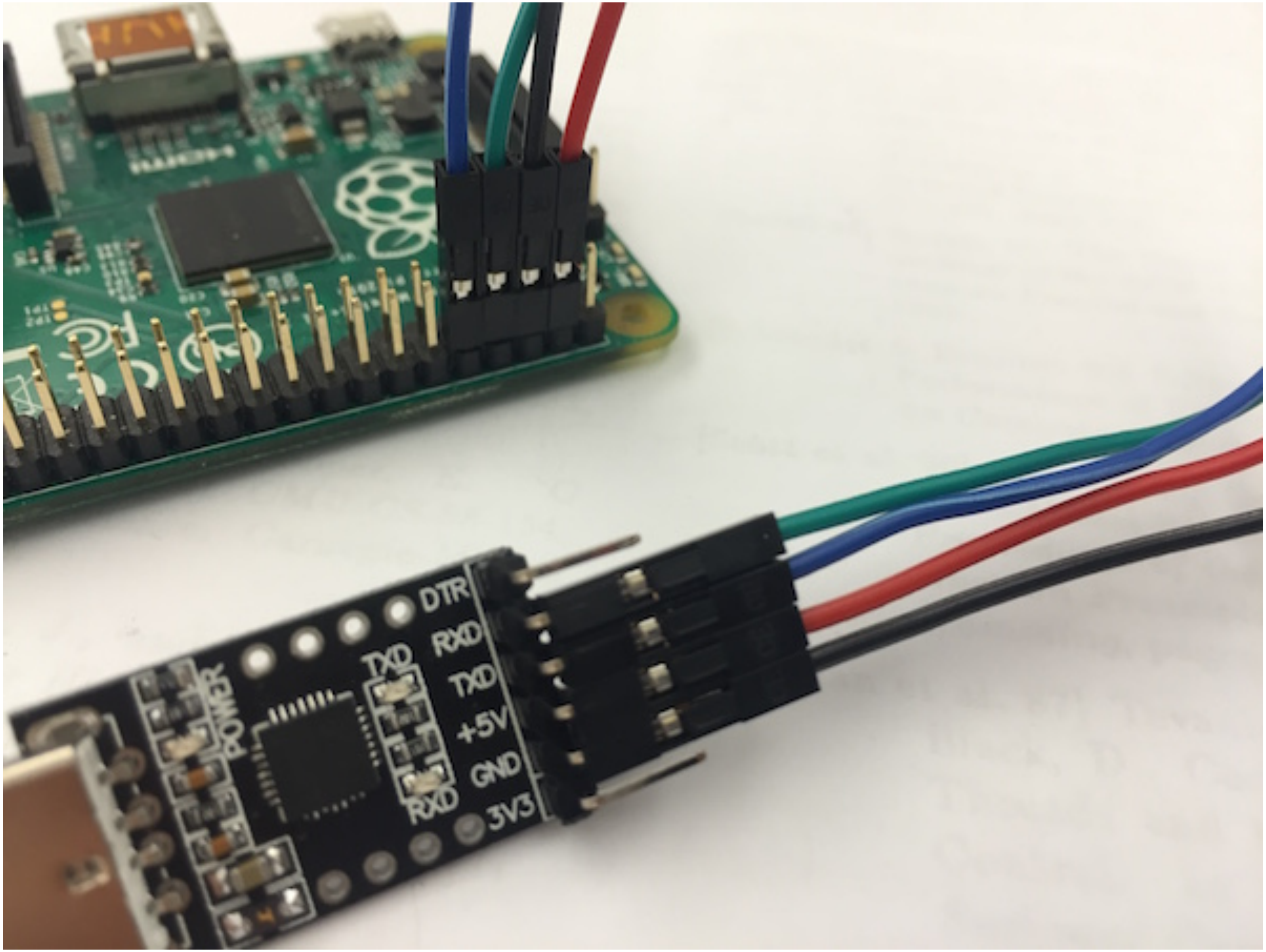
https://learn.sparkfun.com/tutorials/serial-communication

serial.c

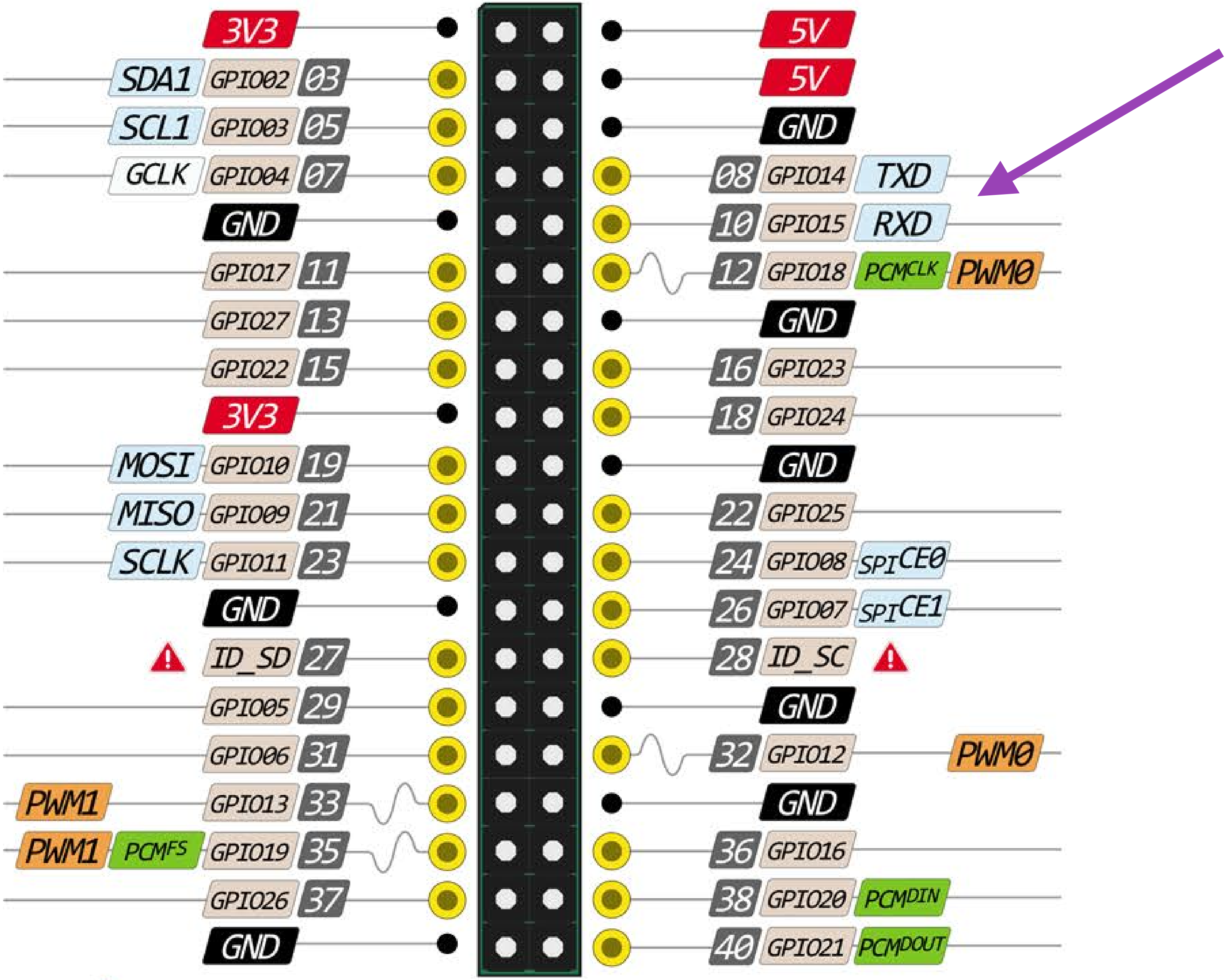
Logic Analyzer!

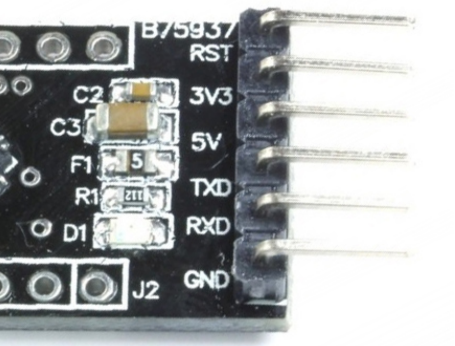


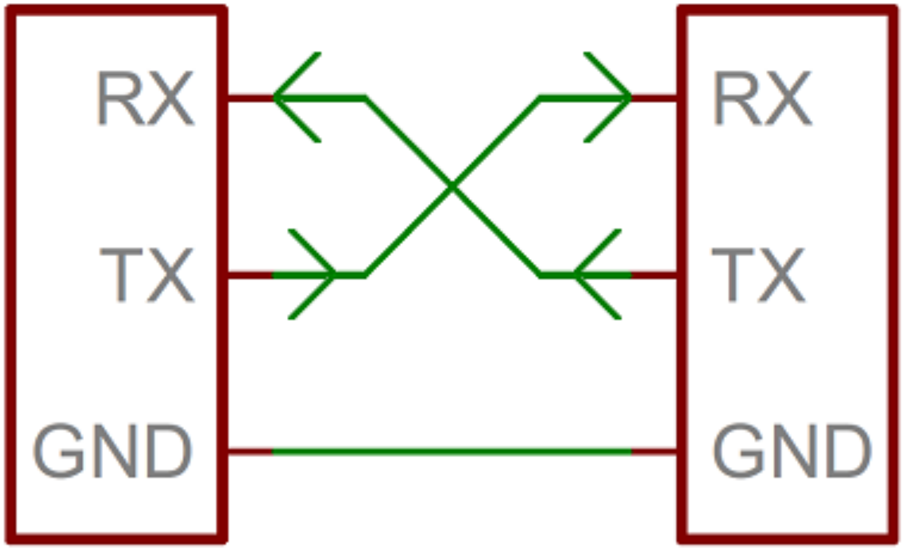
```
// hot wire TX

// device = tty (teletype)
// baud rate = 9600

% screen /dev/tty.SLAB_USBtoUART 9600

CTRL-A K - to exit
```

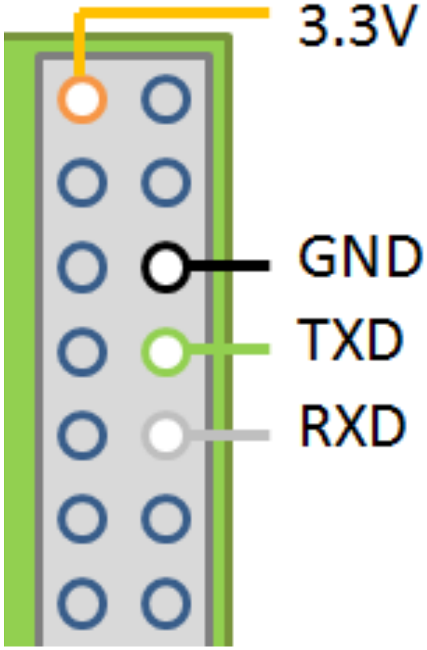

DTR
RXD
TXD
+5V
GND
3V3
TXD
RXD
POWER

GPIO Alternate Functions









```
% screen /dev/tty.SLAB_USBtoUART 115200
```

`uart.c`

Universal Asynchronous Receiver-Transmitter

GPIO ALT Function

BCM2835 has 54 general-purpose I/O pins. Every pin can be input, output, or one of 6 special functions (ALT0-ALT5), specific to each pin.

PIN	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO14	TXD0	SD6				TXD1
GPIO15	RXD0	SD7				RXD1

```
// BCM2835-ARM-Peripherals.pdf
// Sec 2: Mini-UART, SPI0, SPI1, pp 8-19
struct UART {
    unsigned data; // I/O Data
    unsigned ier;  // Interrupt enable
    unsigned iir;  // Interrupt identify/fifo
    unsigned lcr;  // line control register
    unsigned mcr;  // modem control register
    unsigned lsr;  // line status
    unsigned msr;  // modem status
    unsigned scratch;
    unsigned cntl; // control register
    unsigned stat; // status register
    unsigned baud; // baud rate register
} ;
```

echo.c

loop back test

String Functions in standard library

`strlen(s)`	Return number of chars in s, not counting ‘\0'
`strcmp(s1,s2)`	Compare s1 with s2; Return negative, zero, or positive
`strncmp(s1,s2,n)`	Compare only the first n characters of s1 and s2
`strcpy(dst,src)`	Copy **src** to **dst**; Note the direction of the copy!
`strncpy(dst,src,n)`	Copy first **n** characters of **src** to **dat**
`strcat(s1,s2)`	Concatenate **src** to **dst**
`strncat(s1,s2,n)`	Concatenate at most **n** characters of **src** to **dat**
`strchr(s,ch)`	Return pointer to first occurrence of ch in s; NULL if none
`strrchr(s,ch)`	Return pointer to last occurrence of ch in s; NULL if none
`strstr(s1,s2)`	Return pointer to first occurrence of s1 in s2; NULL if none

Debuggers

A debugger is invaluable tool:

monitor program
examine (and change!) runtime state
re-direct control
view disassembly
and more!

But … running bare metal, we’re mostly tool-free (unless we write it ourselves)

gdb can run in *simulation mode*

“pretends” to be an ARM processor
provides a model of the behavior, but not exact
e.g. no GPIO or peripherals

How to: GDB in simulation mode

```
% arm-none-eabi-gdb program.elf
```

Note: use of .elf version, not raw .bin

```
GNU gdb (GDB) 7.8.1
Copyright (C) 2014 Free Software Foundation
...
(gdb) target sim
(gdb) load
(gdb) run
```

Can now step through program, examine memory, …
Helps to understand what code is doing
Not everything debuggable (GPIO, peripherals)

strings.c

(gdb in simulation mode)

C-strings

Read:

The most expensive 1-byte mistake,

Did Ken, Dennis, and Brian choose wrong with 0-terminated text strings?

Poul-Henning Kamp

http://queue.acm.org/detail.cfm?id=2010365