

Admin

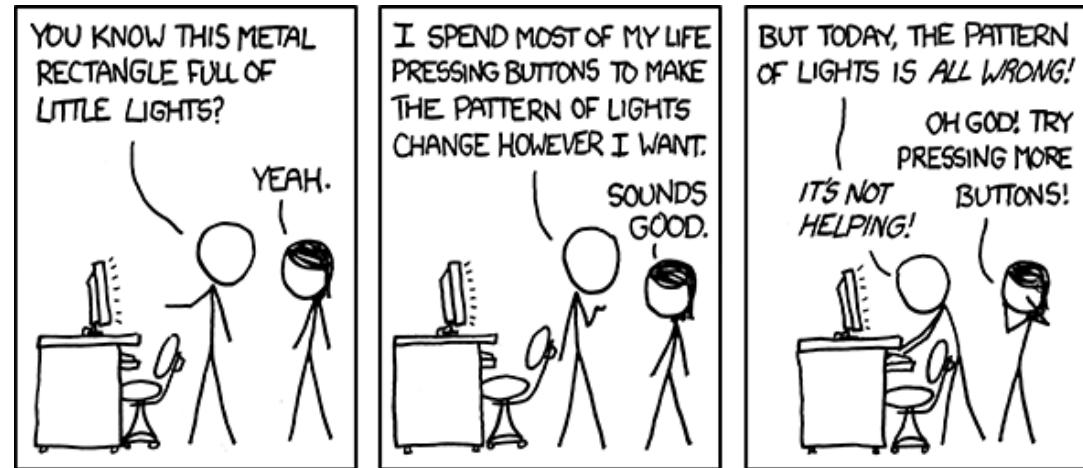
Assign 1 grades posted
style reflection due Thu

Assign 2 extension encouraged!

Lab 3 serial, gdb, strings

Assign 3 printf

Teach your Pi to talk, yeah!



Today: Serial communication

GDB debugger

Serial protocol

ASCII character set, strings, printing

Uart peripheral

C vs Assembly Smackdown

Why C?

- Variable names, type system
- Function decomposition, control flow
- Portable abstractions
- Consistent semantics
- Compiler back-end doing heavy lifting - yay!

Why assembly?

- Execution is always in asm, this is the real deal -- WYSIWYG
- Ability to drop down and review/debug asm is key
- Certain hardware features only accessible via asm
- Hand-code in asm for optimization or obtain precise timing

Abstraction FTW!

```
lui      a0,0x2000  
addi    a1,zero,1  
sw      a1,0x30(a0)  
  
loop:  
    xori    a1,a1,1  
    sw      a1,0x40(a0)  
  
    lui      a2,0x3f00  
delay:  
    addi    a2,a2,-1  
    bne    a2,zero,de  
    j       loop
```

```
void pause(int count) {  
    while (count-- != 0) {}  
  
void main(void) {  
    volatile unsigned int *PB_CFG0 = 0x2000030;  
    volatile unsigned int *PB_DATA = 0x2000040;  
  
    *PB_CFG0 = 1;    // config as output  
  
    while (1) {  
        *PB_DATA = 1;    // on  
        pause(0x3f00000);  
        *PB_DATA = 0;    // off  
        pause(0x3f00000);  
    }  
}
```

```
void main(void) {  
    timer_init();  
    gpio_init();  
    gpio_set_output(pin);  
  
    while (1) {  
        gpio_write(pin, 1);  
        timer_delay_ms(SECOND);  
        gpio_write(pin, 0);  
        timer_delay_ms(SECOND);  
    }  
}
```

good -> better -> best!

Gdb debugger

Debugger is incredibly useful

Allows you to run your program in a monitored context
Can set breakpoints, examine state, change values,
reroute control, and more

gdb has simulation mode where it pretends to be an RISC-V processor, running on your laptop 🙌

Pretty good approximation (not perfect, e.g. no peripherals)

Let's try it now!

○○ **Read our course guide** ○○

<http://cs107e.github.io/guides/gdb/>

Practice exercises in lab3

Invaluable tool for assign3

I just want to say thank you for teaching us how to use the debugger #611



Monica Diane Hicks

5 months ago in Assignments - A4



Because I NEVER would have found** the bug in my boggle function without it.

7



ENDORSED

255

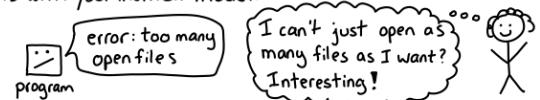
VIEWS

Sort by Relevant ▾

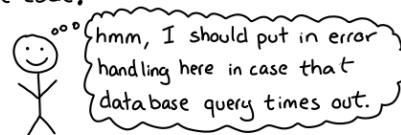
debugging: ♥ love your bugs ♥

(thanks to Allison Kaptur for teaching me this attitude!
she has a great talk called "Love Your Bugs".)

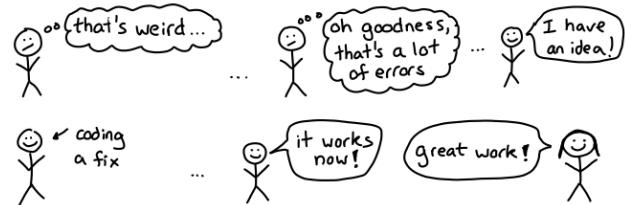
Debugging is a great way to learn. First, the harsh reality of bugs in your code is a good way to reveal problems with your mental model.



Fixing bugs is also a good way to learn to write more reliable code!



Also, you get to solve a mystery and get immediate feedback about whether you were right or not.

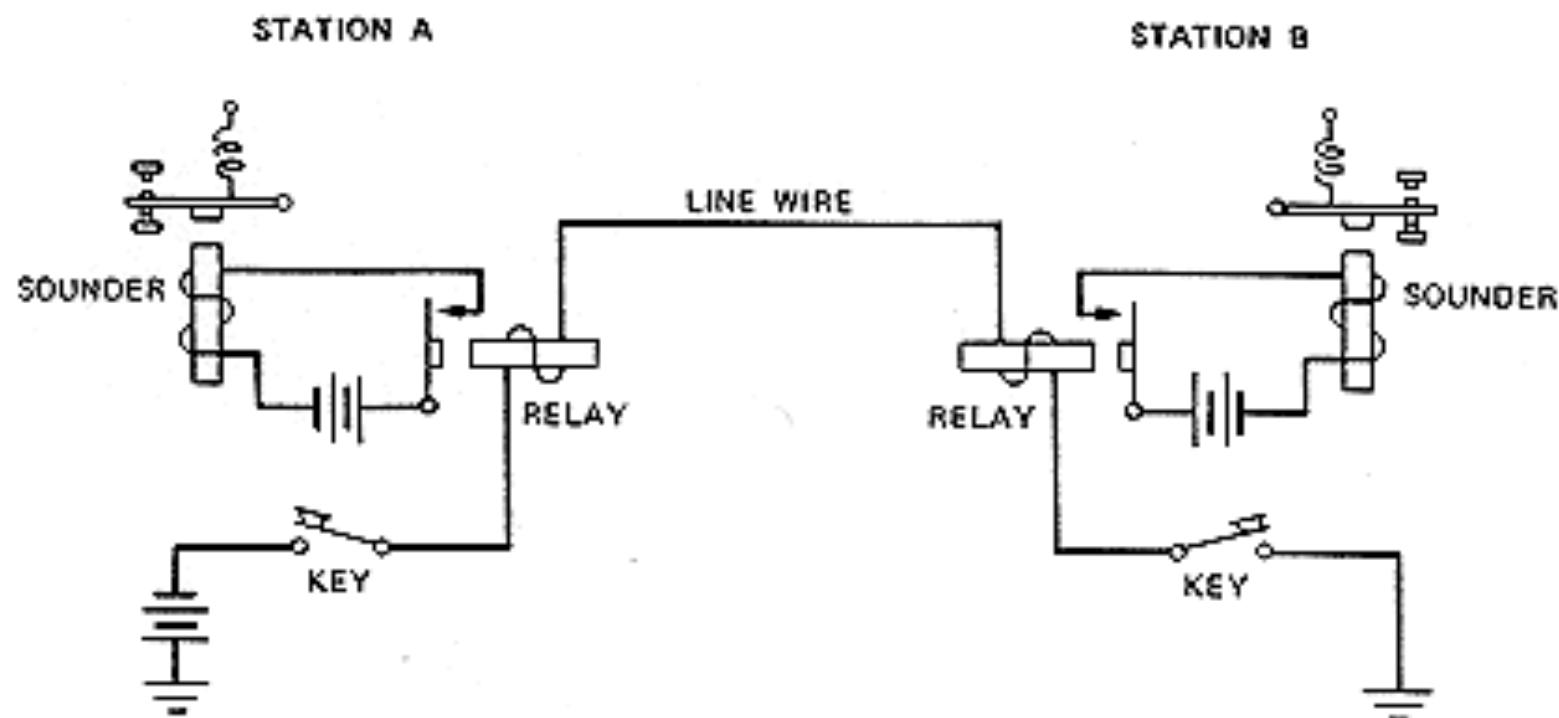


Nobody writes great code without writing + fixing lots of bugs. So let's talk about debugging skills a bit!

JULIA EVANS
@b0rk

Telegraph circuit

SIMPLEX TELEGRAPH



Elementary neutral telegraph circuit.

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	— — — — —

blink_gpio/blink.c
->
sos/sos.c

Code examples from lecture linked on website

<https://github.com/cs107e/cs107e.github.io/tree/master/lectures/Serial/code/>

Teletype



5-bit Baudot Code (1870)

LETTERS FIGURES		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	CARRIAGE RETURN	LINE FEED	LETTERS	FIGURES	SPACE	ALL-SPACE NOT IN USE
CODE ELEMENTS		-	?	:	WHO ARE YOU	3	%	@	£	8	BELL	()	.	,	9	0	1	4	,	5	7	=	2	/	6	+						
1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
2	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○				
3	○	○	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
4	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
5	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				

● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)

○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet

Baud = Number of symbols per second

https://en.wikipedia.org/wiki/Baudot_code

ASCII Code

```
$ man ascii
```

	2	3	4	5	6	7
<hr/>						
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

\0
65
37
30
31
73
63

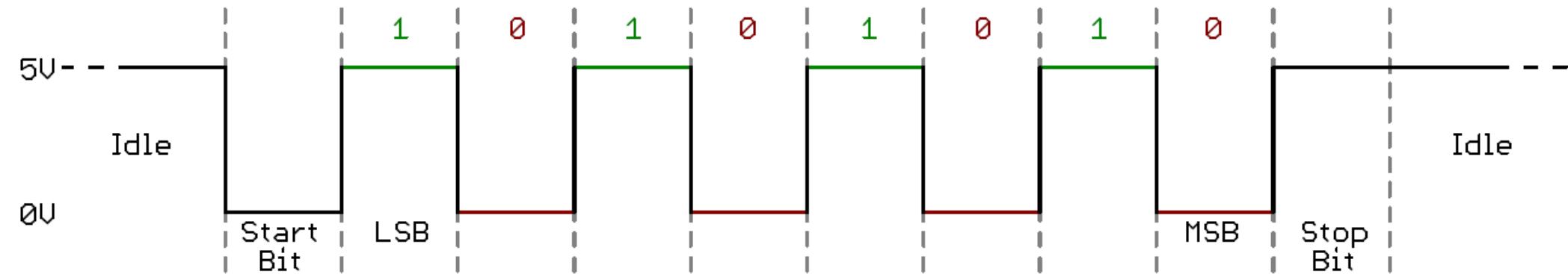
"cs107e" =

0x68 represents character 'h'

ASCII 7-bit

(8th bit used by extended char sets)

Asynchronous Serial Communication



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

9600 baud = 9600 bits/sec

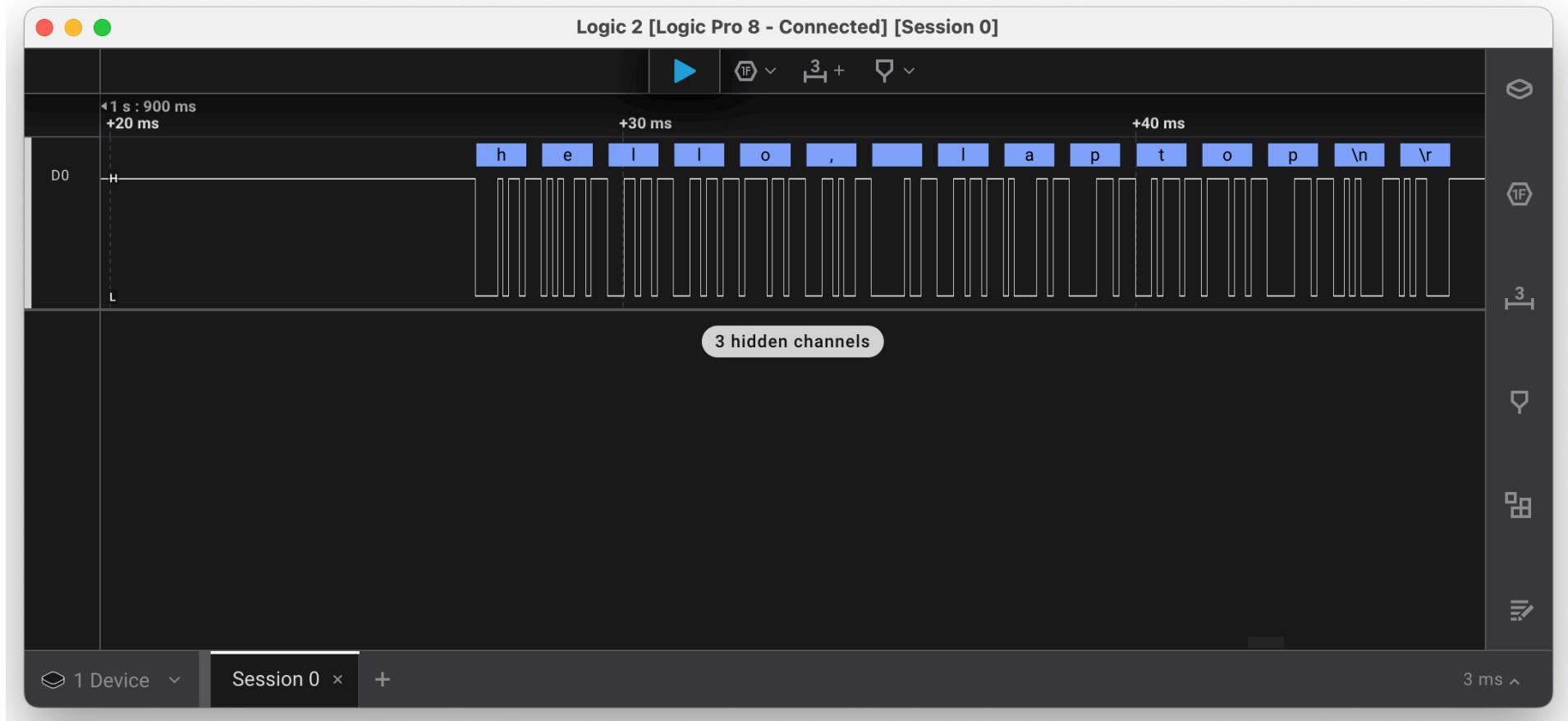
(1000000 usecs)/9600 ~ 104 usec/bit

sos.c -> serial.c

Code examples from lecture linked on website

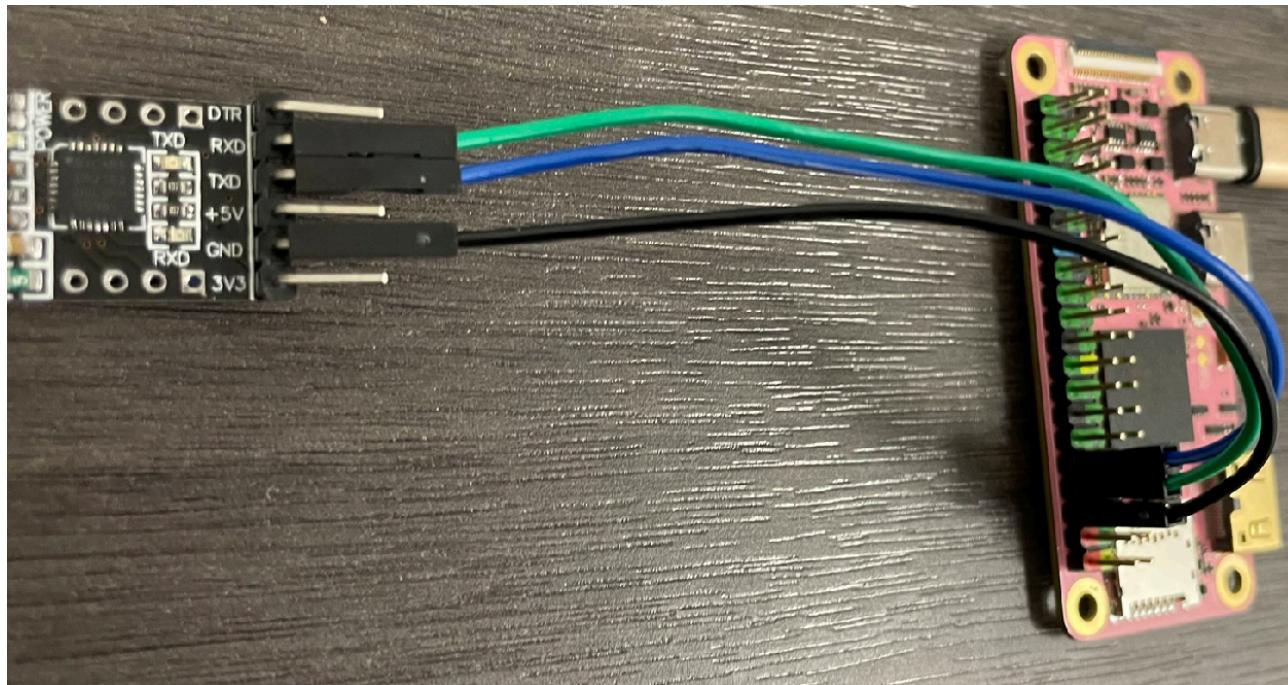
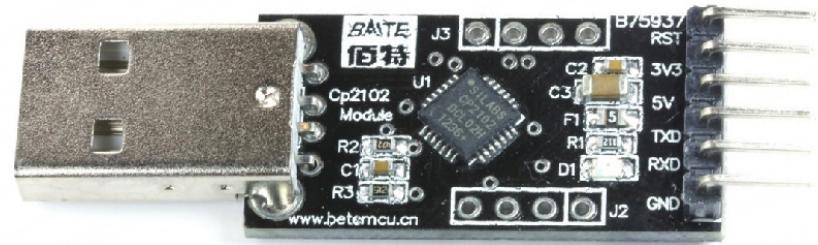
<https://github.com/cs107e/cs107e.github.io/tree/master/lectures/Serial/code/>

Logic Analyzer



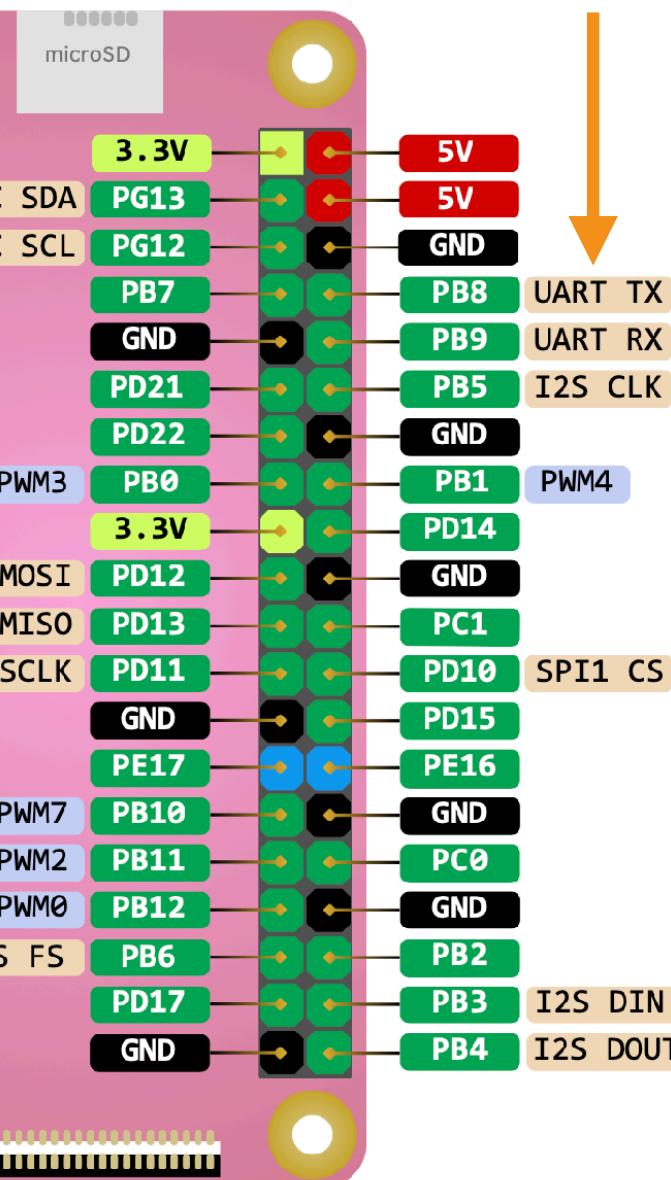
Making the invisible visible!

CP2102 USB-serial adapter



Ref: [CS107e uart guide](#)

GPIO Alternate functions

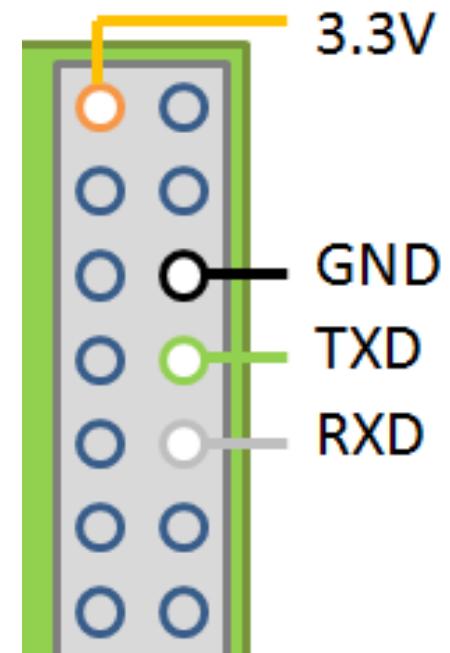
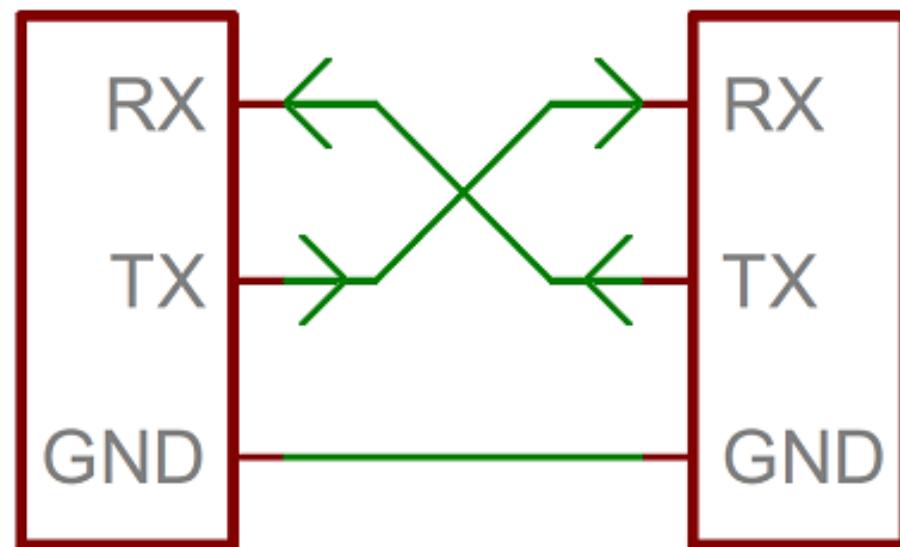
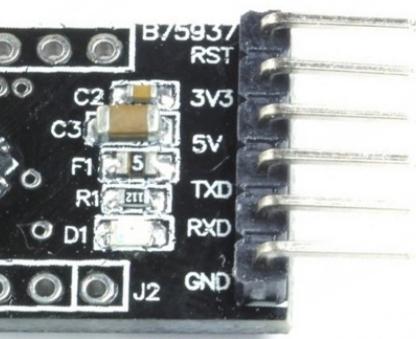


The tables show the configuration of GPIO pins PB9 and PB8 through two registers:

7:4	R/W	0xF	PB9_SELECT
			0000:Input
			0010:DMIC-DATA2
			0100:TWI2-SDA
			0110:UART0-RX
			1000:Reserved
			1110:PB-EINT9

3:0	R/W	0xF	PB8_SELECT
			0000:Input
			0010:DMIC-DATA3
			0100:TWI2-SCK
			0110:UART0-TX
			1000:Reserved
			1110:PB-EINT8

Ref: [DI-H User Manual p.1097](#)



```
$ tio -b 115200 /dev/cu.usbserial-0001
```

Universal Asynchronous Receiver-Transmitter

9.2 UART

9.2.1 Overview

The universal asynchronous receiver transmitter (UART) provides an asynchronous serial communication with external devices, modem (data carrier equipment, DCE). It performs serial-to-parallel conversion on the data received from peripherals and transmits the converted data to the internal bus. It also performs parallel-to-serial conversion on the data that is transmitted to peripherals.

The UART has the following features:

- Compatible with industry-standard 16450/16550 UARTs
- Supports IrDA-compatible slow infrared (SIR) format
- Two separate FIFOs: one is RX FIFO, and the other is TX FIFO
 - Each of them is 64 bytes (For UART0)
 - Each of them is 256 bytes (For UART1, UART2, UART3, UART4, and UART5)

```
struct {
    union {
        uint32_t rbr;      // receive buffer register
        uint32_t thr;      // transmit holding register
        uint32_t dll;      // divisor latch lo
    };
    union {
        uint32_t dlh;      // divisor latch hi
        uint32_t ier;      // interrupt enable register
    };
    union {
        uint32_t iir;      // interrupt identification register
        uint32_t fcr;      // FIFO control register
    };
    uint32_t lcr;          // line control register
    uint32_t mcr;          // modem control register
    uint32_t lsr;          // line status register
```

Ref: [DI-H User Manual p.899](#)

Register Name	Offset	Description
UART_RBR	0x0000	UART Receive Buffer Register
UART_THR	0x0000	UART Transmit Holding Register
UART_DLL	0x0000	UART Divisor Latch Low Register
UART_DLH	0x0004	UART Divisor Latch High Register
UART_IER	0x0004	UART Interrupt Enable Register
UART_IIR	0x0008	UART Interrupt Identity Register
UART_FCR	0x0008	UART FIFO Control Register
UART_LCR	0x000C	UART Line Control Register
UART_MCR	0x0010	UART Modem Control Register
UART_LSR	0x0014	UART Line Status Register
UART_MSR	0x0018	UART Modem Status Register
UART_SCH	0x001C	UART Scratch Register

Ref: [uart.c source](#)

C-strings

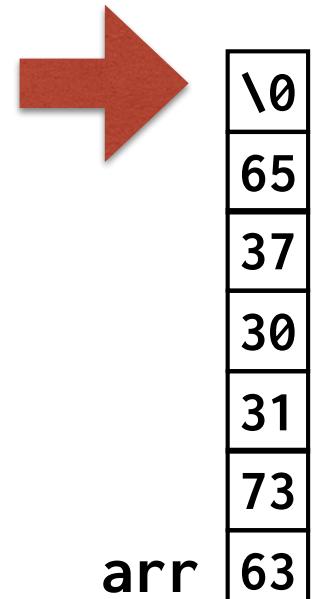
```
// Note '\0' at the end!
char arr[] = {'c', 's', '1', '0', '7', 'e', '\0'};
```

```
// short cut
char arr[] = "cs107e";
```

```
char ch = arr[1]; // ok!
ch = arr[25]; // not ok, what happens?
```

```
char *ptr = arr; // ok!
arr = ptr; // not ok, why?
```

```
const char *ptr = "cs107e";
ch = ptr[1]; // ok?
ptr[1] = 'd'; // ok?
```



Operations on C-strings

```
size_t strlen(const char *str) {
    int n = 0;
    while (str[n] != '\0') {
        n++;
    }
    return n;
}
```

```
// strlen("a")?
// strlen(NULL)?
// strlen(&ch)?
```

Strings module

```
#include "strings.h"
```

strcat(dst,src)	Concatenate src chars to end of dst
strlcat(dst,src,n)	Concatenate n chars of src to end of dst
strcpy(dst,src)	Copy dst chars to src
strncpy(dst,src,n)	Copy n chars of src to dst
strlen(s)	Return length of s, not counting '\0'
strcmp(s1,s2)	Lexicographic comparison of s1 with s2; return is < 0 if s1 < s2, 0 if s1 == s2, else > 0
strncmp(s1,s2,n)	Compare only the first n chars
strtonum(const char *str, const char **endptr);	

...

Printf module

```
#include "printf.h"

// print formatted output

printf("%d, %d\n", 1, 2);
printf("%x\n", 0x20000030);
printf("%c\n", 'a');
printf("%s\n", "hello");
```

Printf is actually all about strings...