Lecture 23

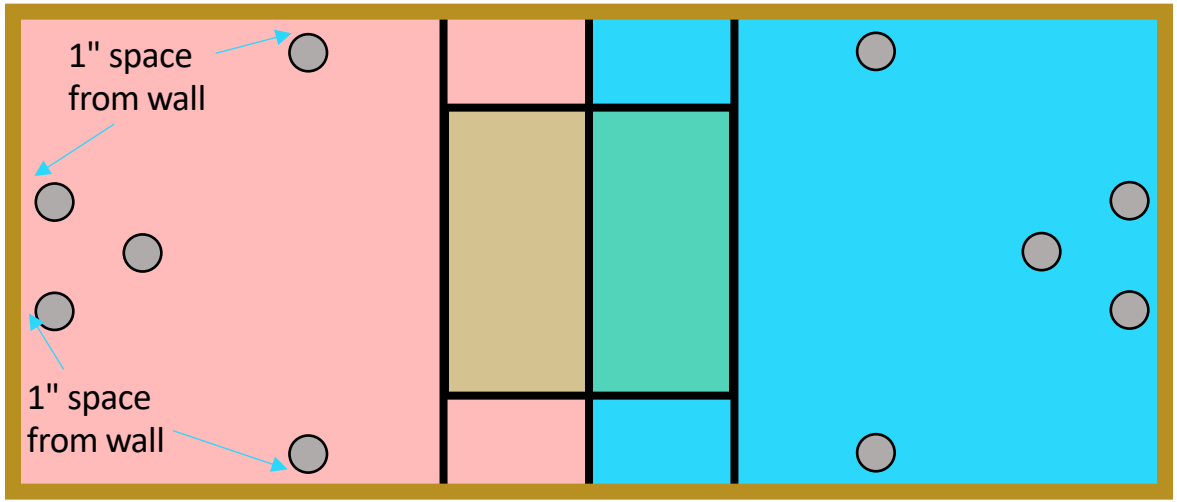
SPI and
Examples for UART and I2C

Agenda

- Sample Peripherals
- UART ESP32 to Teensy
- SPI ESP32 (master) to Teensy (slave) example.
- Memory on ESP32
- Extra bits

Rules Clarification

- One reset allowed per game (TA will move your robot back)
- Bonus circle is now **Bonus Square**
- Cans are set 1" off from wall (not touching wall)



Waldo Video

- Please watch the Waldo's.
- Pick your favorite (two or three) (not your own).
- Vote in chat for your favorites.
- Those will be incorporated into a final year video.

Waldo movie on canvas

01

Interfacing ESP32 and Teensy Logic Levels

29.2 DC Characteristics

Table 29-1. DC Characteristic, TA = -40°C to 85°C, VCC = 2.7V to 5.5V (unless otherwise noted)

Teensy

Symbol	Parameter	Condition	Min.[5]	Typ.	Max.[5]	Units
V_{IL}	Input Low Voltage, Except XTAL1 and Reset pin	V_{CC} = 2.7V - 5.5V	-0.5		$0.2V_{CC}$-0.1V[1] (LVTTL)	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 2.7V - 5.5V	-0.5		$0.1V_{CC}$[1]	
V_{IL2}	Input Low Voltage, RESET pin	V_{CC} = 2.7V - 5.5V	-0.5		$0.1V_{CC}$[1]	
V_{IH}	Input High Voltage, Except XTAL1 and RESET pins	V_{CC} = 2.7V - 5.5V	$0.2V_{CC}$+0.9V[2] (LVTTL)		V_{CC} + 0.5	
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 2.7V - 5.5V	$0.7V_{CC}$[2]		V_{CC} + 0.5	
V_{IH2}	Input High Voltage, RESET pin	V_{CC} = 2.7V - 5.5V	$0.9V_{CC}$[2]		V_{CC} + 0.5	
V_{OL}	Output Low Voltage[3],	I_{OL} = 10mA, V_{CC} = 5V I_{OL} = 5mA, V_{CC} = 3V			0.7 0.5	
V_{OH}	Output High Voltage[4],	I_{OH} = -10mA, V_{CC} = 5V I_{OH} = -5mA, V_{CC} = 3V	4.2 2.3			

Table 13: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter		Min	Typ	Max	Unit
C_{IN}	Pin capacitance		-	2	-	pF
V_{IH}	High-level input voltage		0.75×VDD[1]	-	VDD[1]+0.3	V
V_{IL}	Low-level input voltage		–0.3	-	0.25×VDD[1]	V
I_{IH}	High-level input current		-	-	50	nA
I_{IL}	Low-level input current		-	-	50	nA
V_{OH}	High-level output voltage		0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage		-	-	0.1×VDD[1]	V
I_{OH}	High-level source current (VDD[1] = 3.3 V, V_{OH} >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain [1, 2]	-	40	-	mA
		VDD3P3_RTC power domain [1, 2]	-	40	-	mA
		VDD_SDIO power domain [1, 3]	-	20	-	mA
I_{OL}	Low-level sink current (VDD[1] = 3.3 V, V_{OL} = 0.495 V, output drive strength set to the maximum)		-	28	-	mA
R_{PU}	Pull-up resistor		-	45	-	kΩ
R_{PD}	Pull-down resistor		-	45	-	kΩ
V_{IL_nRST}	Low-level input voltage of CHIP_PU to power off the chip		-	-	0.6	V

Logic levels are okay, but ESP32 is not 5V safe!

29.2 DC Characteristics

Table 29-1. DC Characteristic, TA = -40°C to 85°C, VCC = 2.7V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.(5)	Typ.	Max.(5)	Units
V_{IL}	Input Low Voltage, Except XTAL1 and Reset pin	V_{CC} = 2.7V - 5.5V	0.2*5 - 0.1= **0.9V**		$0.2V_{CC}$-0.1V(1) (LVTTL)	
V_{IH}	Input High Voltage, Except XTAL1 and RESET pins	V 0.2*5 + 0.9= **1.9V**	$0.2V_{CC}$+0.9 V(2) (LVTTL)		V_{CC} + 0.5	V

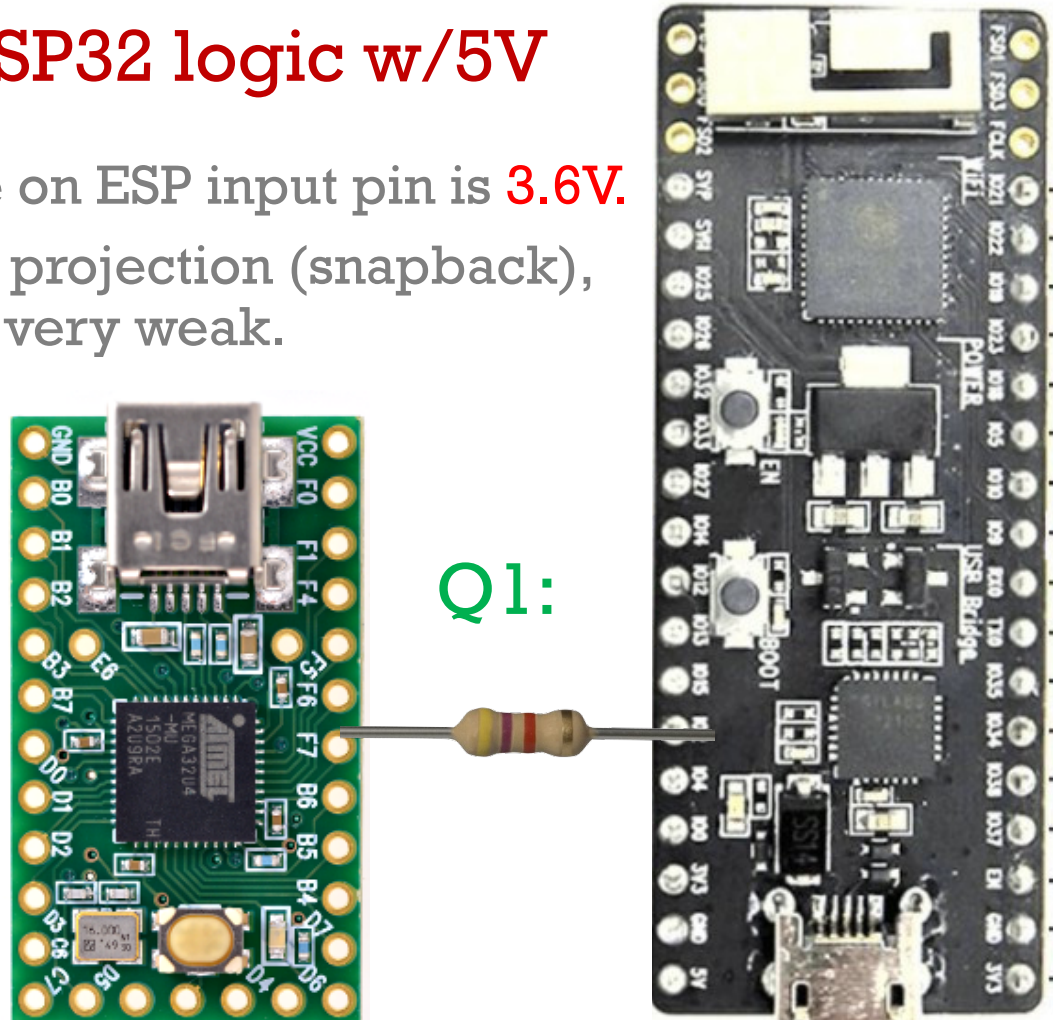
Table 13: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter	Min	Typ	Max	Unit
V_{OH}	High-level output voltage	0.8*3.3 = **2.64V** 0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage			0.1×VDD[1] 0.1*3.3 = **0.33V**	V
V_{IH}	High-level input voltage			VDD[1]+0.3 **3.3+.3V = 3.6V**	V

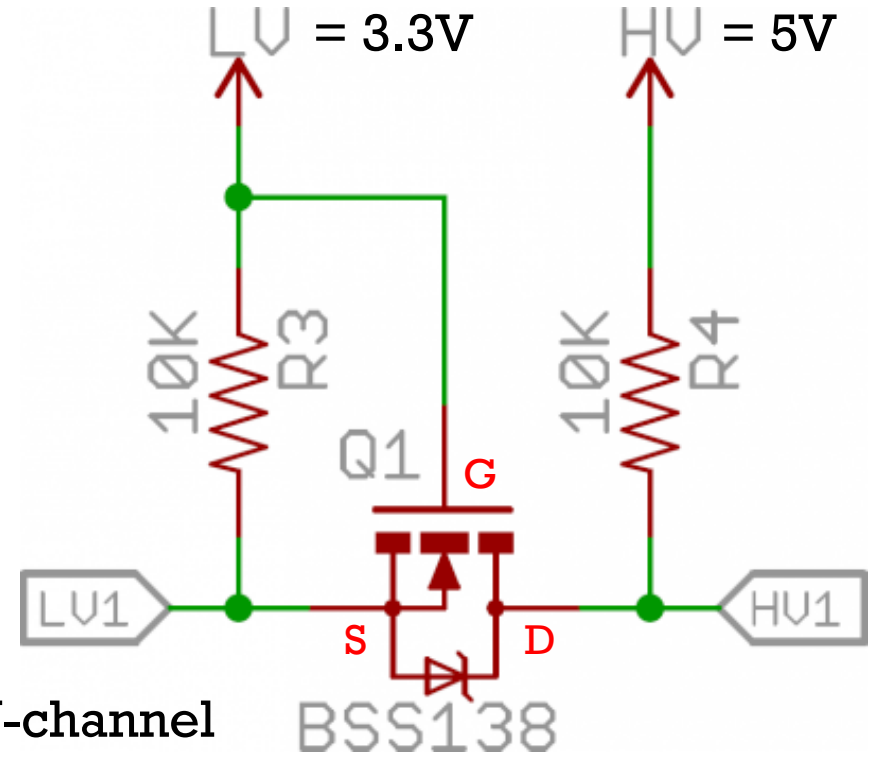
Dangers of driving ESP32 logic w/5V

- Maximum allowable voltage on ESP input pin is 3.6V.
- Pins have some overvoltage projection (snapback), but apparently they may be very weak.

- Online forum an Espressif employee: 0.3mA may be okay, but not guaranteed to be safe.
- e.g., 5K resistor between 5V and pin maybe okay, for a while.
- Better to use level shifter.



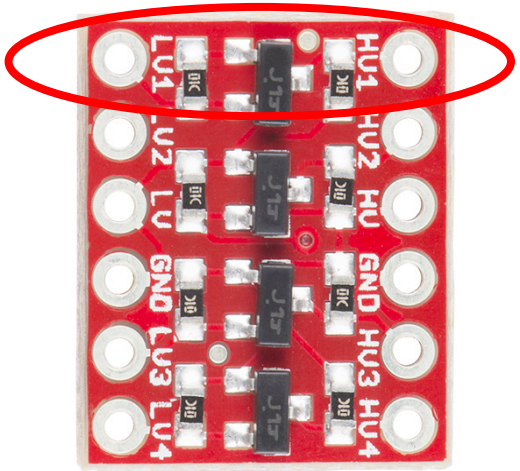
Level Shifter



N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- N-channel MOSFET
  - Becomes a low value resistor between S and D when $V_G > V_S + 2.5V$
  - Otherwise very high resistance between S and D
  - G-S and G-D are not connected (infinite resistance)



Level Shifter

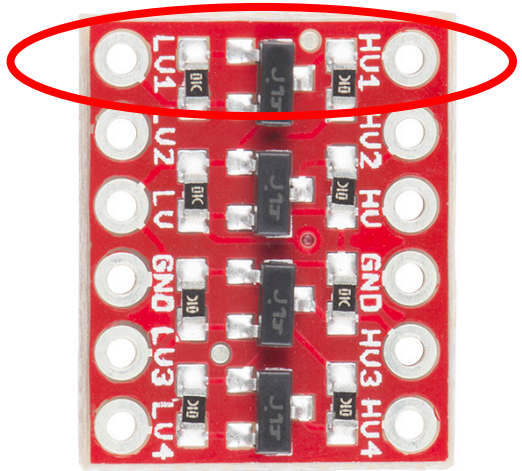
Four cases:

- LV1 is output HIGH driving HV1
- LV1 is output LOW driving HV1
- HV1 is output HIGH driving LV1
- HV1 is output LOW driving LV1

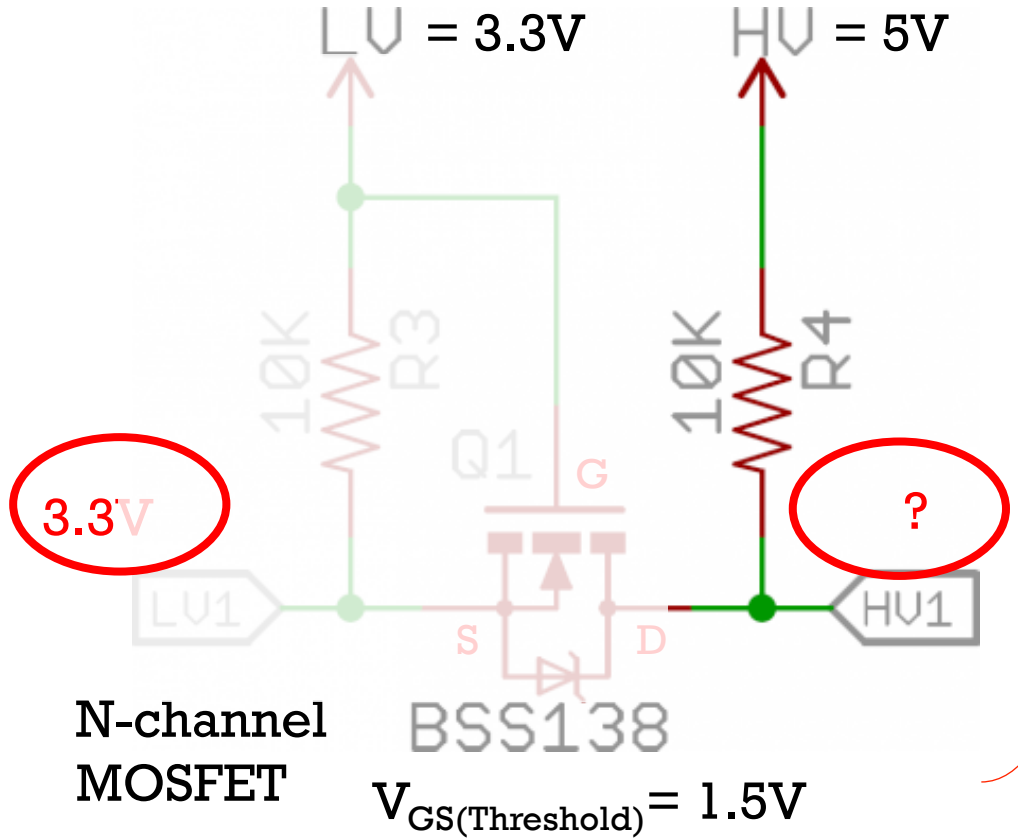
N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- N-channel MOSFET
  - Becomes a low value resistor between S and D when $V_G > V_S + 2.5V$
  - Otherwise very high resistance between S and D
  - G-S and G-D are not connected (infinite resistance)



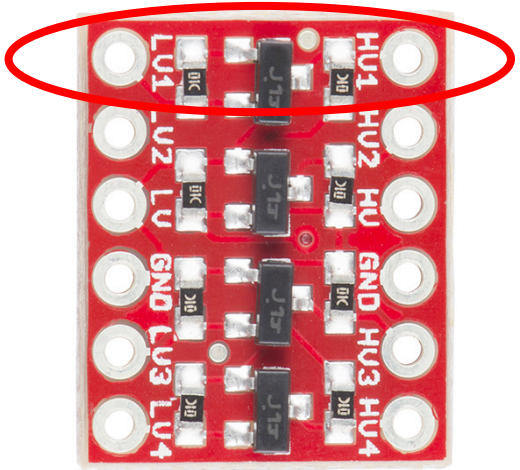
Q2: Level Shifter



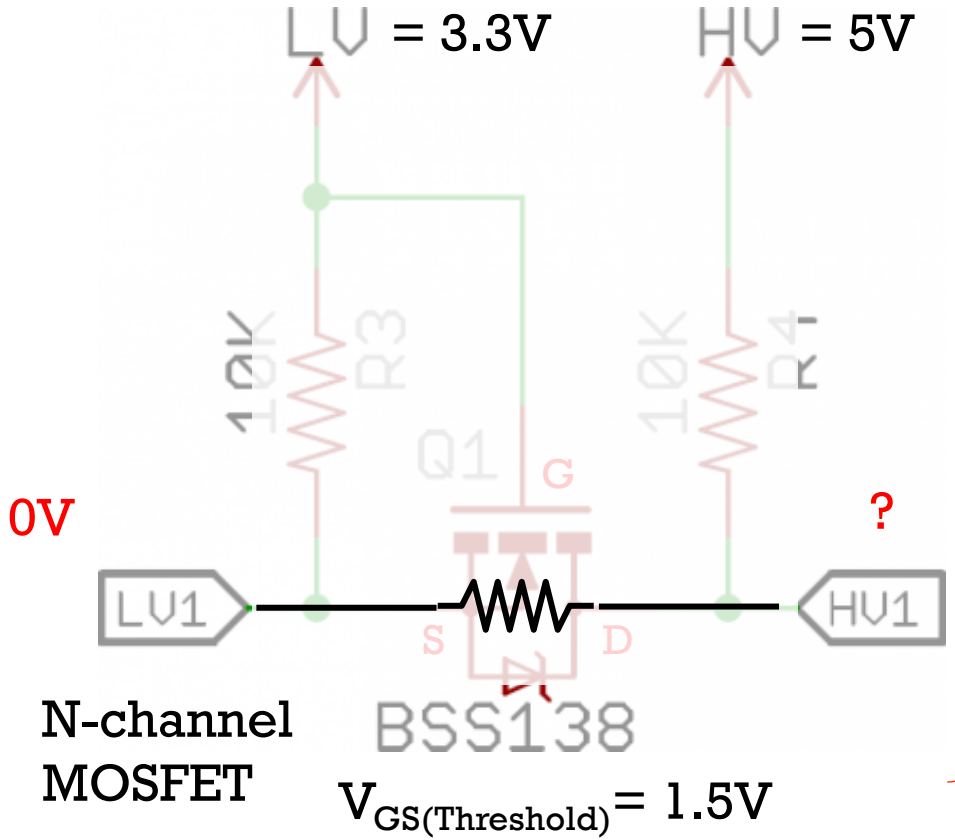
N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- With 3.3V at LV1, what voltage is at HV1 and what is the max current supplied to HV1 to maintain proper logic levels?



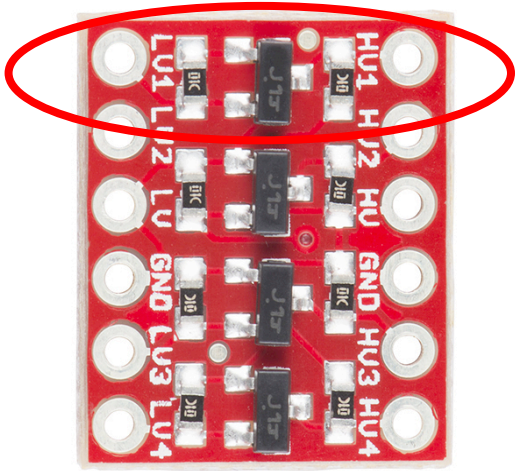
Q3 Level Shifter



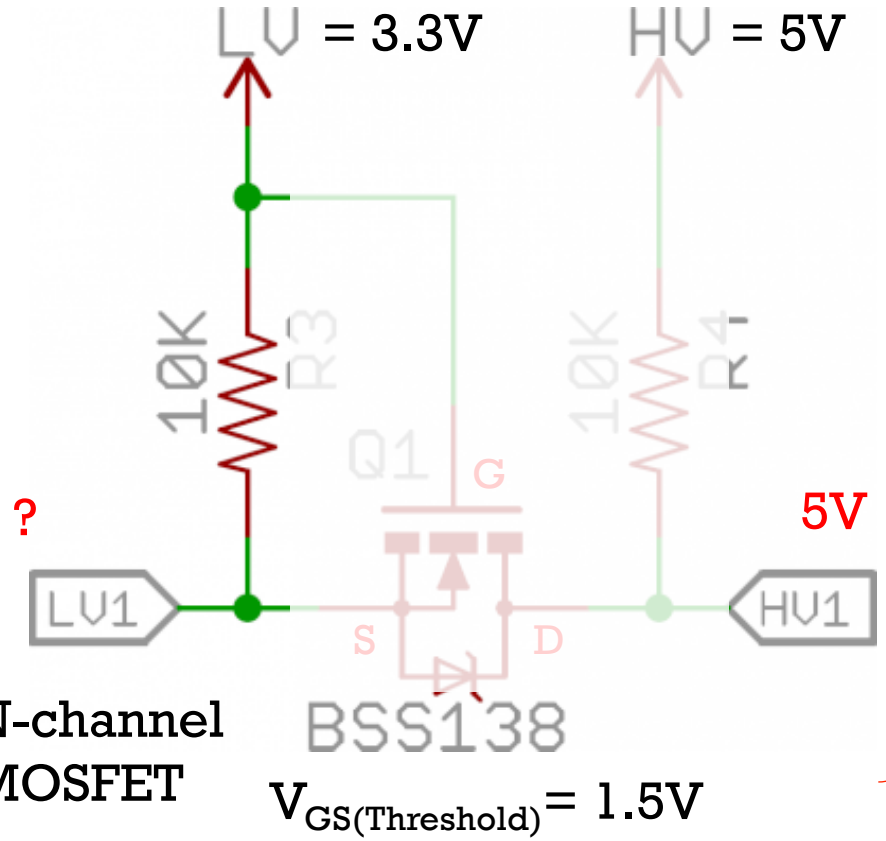
N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- With 0V at LV1, what voltage is at HV1 and what is the max current supplied to HV1 to maintain proper logic levels?



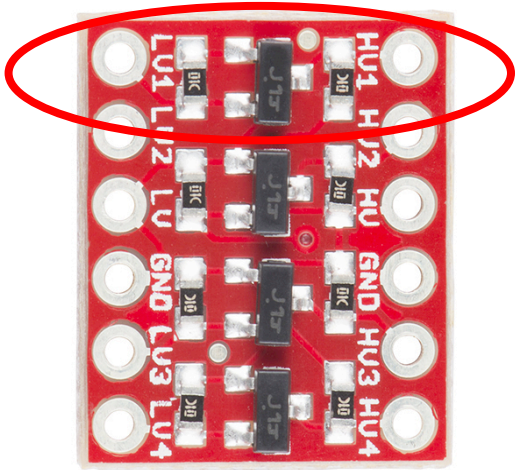
Q4: Level Shifter



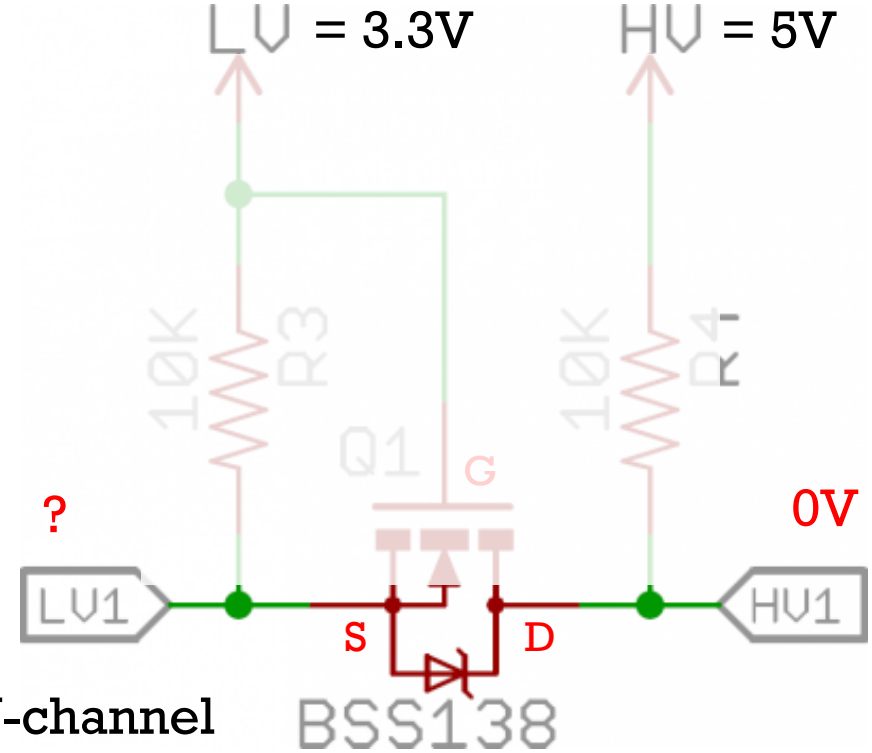
N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- With 5V at HV1, what voltage is at LV1 and what is the max current supplied to LV1 to maintain proper logic levels?



Q5: Level Shifter



N-channel MOSFET

$V_{GS(Threshold)} = 1.5V$

- With 0V at HV1, what voltage is at LV1 and what is the max current supplied to LV1 to maintain proper logic levels?

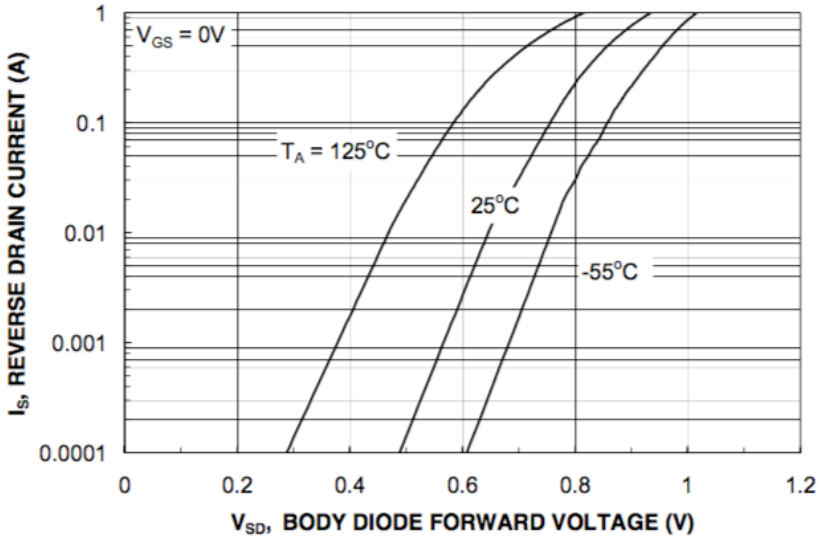


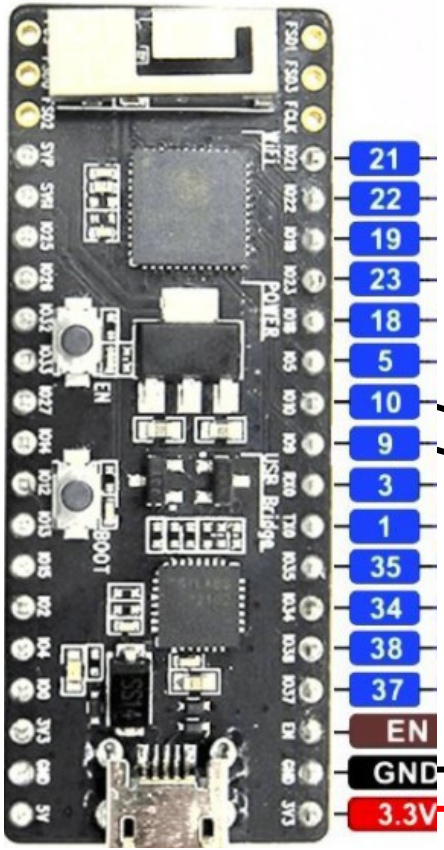
Figure 6. Body Diode Forward Voltage Variation with Source Current and Temperature.

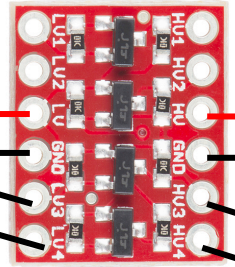
02

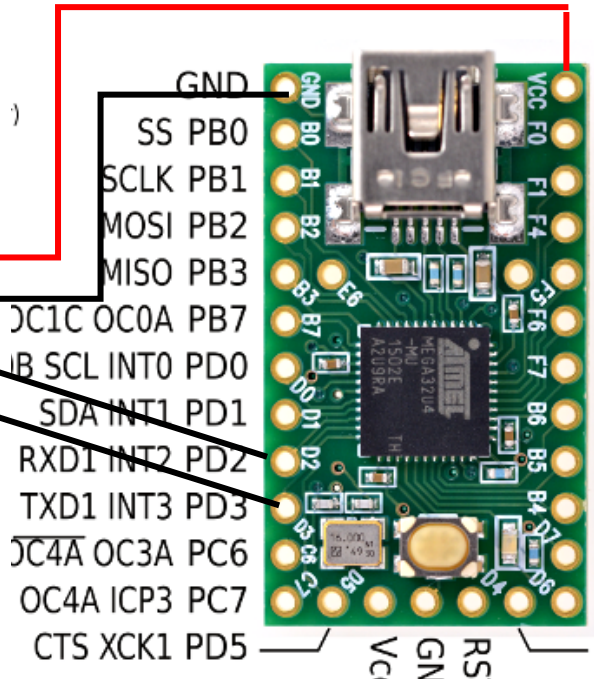
UART
ESP32 to Teensy

ESP32 to Teensy using UART

Connect:
GND to GND
RXD to TXD
TXD to RXD







Resources -> ESP32 Arduino Lecture Examples -> ESP32-UART.ino
Resources -> Teensy files -> Examples from Lecture -> Teensy_uart.c

UART ESP32 -> Teensy

- ESP32 Arduino code

```
#define RXD2 9
#define TXD2 10

void setup() {
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2);
}

int i;
void loop() {
  while (Serial2.available()) {
    Serial.print(char(Serial2.read()));
  }
  if (millis()%1000==1){
    Serial2.println(i++);
    Serial.printf("ESP32 write %d\n",i);
    delay(10);
  }
}
```

- Teensy - Needs uart.c uart.h
  https://www.pjrc.com/teensy/uart.html

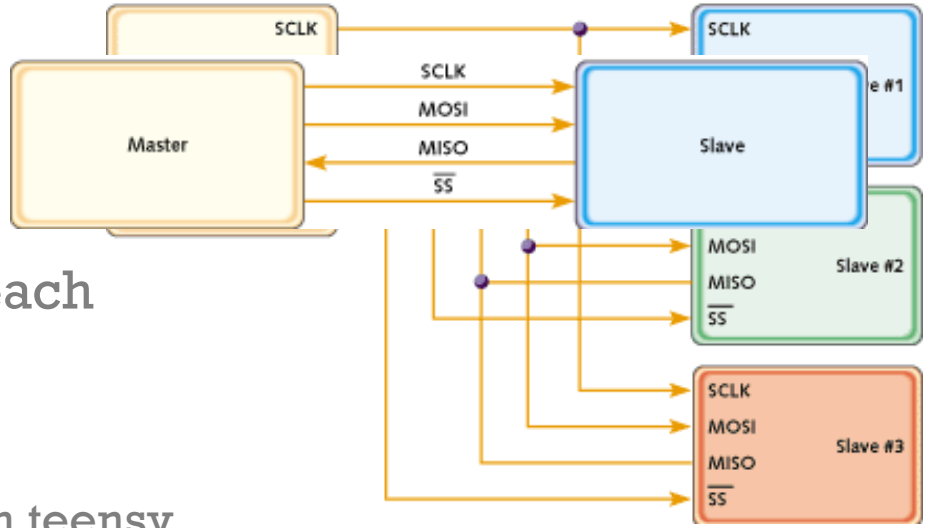
```
#include "teensy_general.h"
#include "uart.h"


int main(void) {
  uint8_t c;
  teensy_clockdivide(0);
  uart_init(115200);
  while (1) {
    if (uart_available()) {
      uart_putchar('.');    // put '.'

      c = uart_getchar(); // get a char
      uart_putchar(c);    // put a char
    }
  }
}
```

03

SPI
Serial Peripheral Interface

Serial Peripheral Interface (SPI)

- Synchronous comms started by Motorola.
- 4 wire:
  - clock: SCLK
  - master out slave in: MOSI
  - master in slave out: MISO
  - slave select: SS
- Multidrop: need one SS for each
- Generally faster than RS 232
- SPI is faster than I2C
  - theoretically 800K Byte/sec on teensy



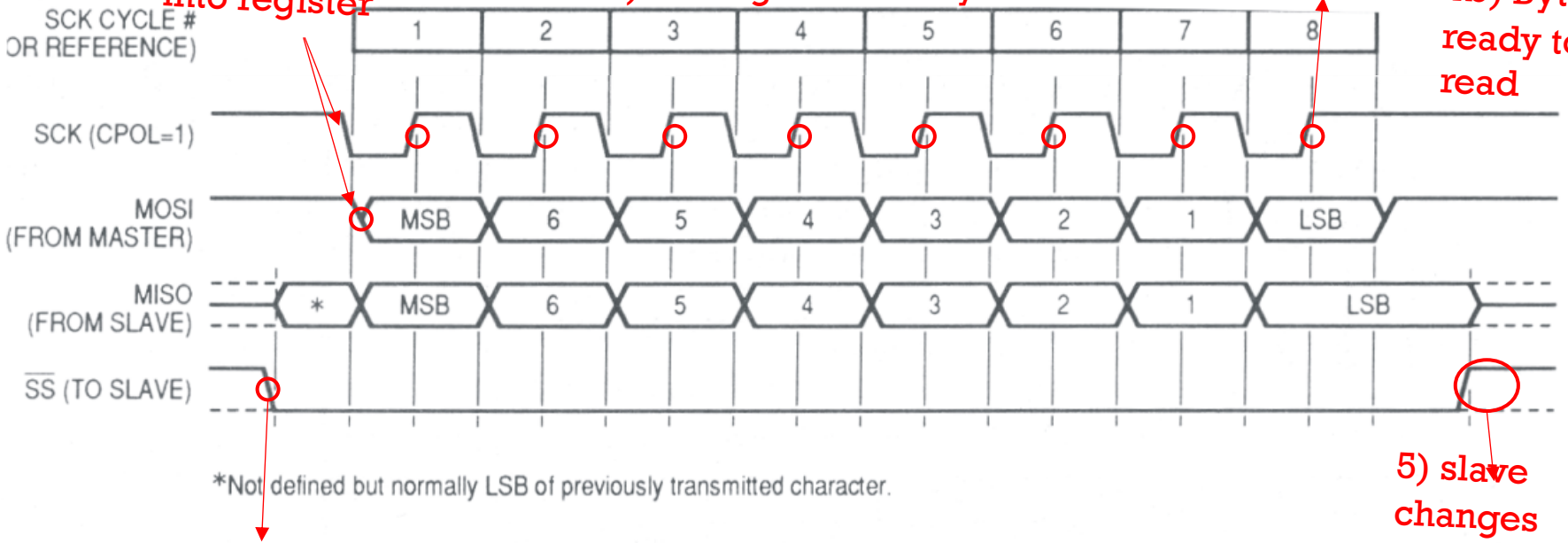
SPI Typical transfer sequence

2) Master writes one byte to send into register

3) storing bits as they come in

4a) interrupt is asserted on master and/or slave

4b) Byte is ready to be read



1) slave changes MISO to output

5) slave changes MISO to input

SS sometimes ignored (always asserted)
But transition insures proper byte alignment

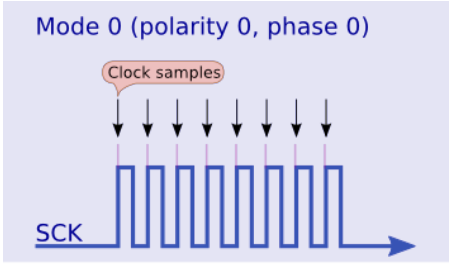
SPI options: clock polarity and bit order

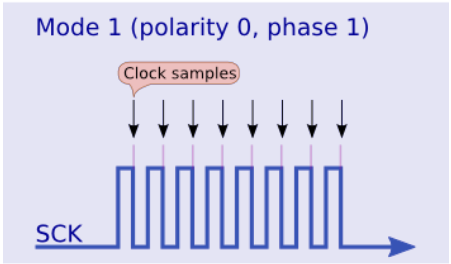
Important that master and slave have same options:

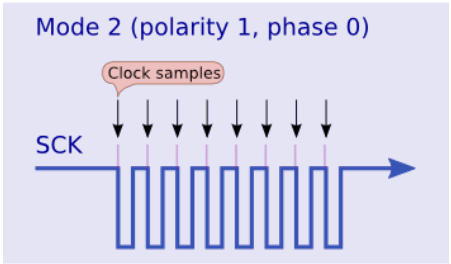
- CPOL: Clock Polarity
- CPHA: Clock Phase, rising or falling edge triggered

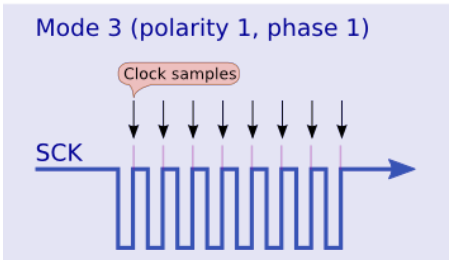
- DORD: Data Order, most significant bit first or least significant bit first.

- Commonly CPOL=1, CPHA=1 (AKA: mode 3)
- CPOL=0, CPHA=0 (Mode 0) is default on adruino.









SPI on ATmega32U4 - registers

- CPOL = CPHA = mode 0 (default) sets clock polarity and phase - rising/falling edge transfer
- Data Order DORD = 0 = MSB first (default)

Data Register

Bit	7	6	5				1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W				R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Master writes to initiate transfer

Master reads after transfer

interrupts enable bit order master polarity phase clock rate

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0		0	0	0	0	0	0	

Control Register

flag

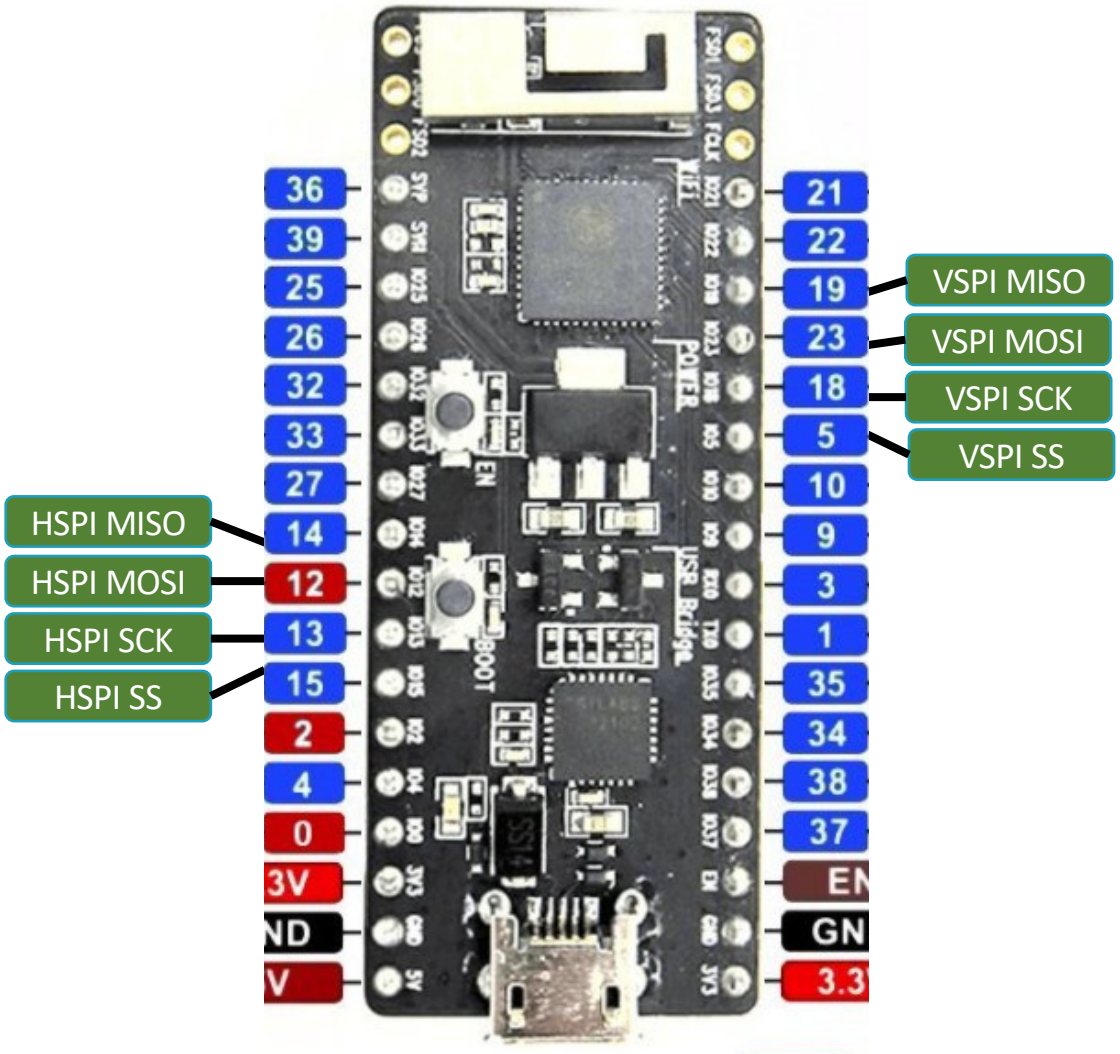
Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Status Register

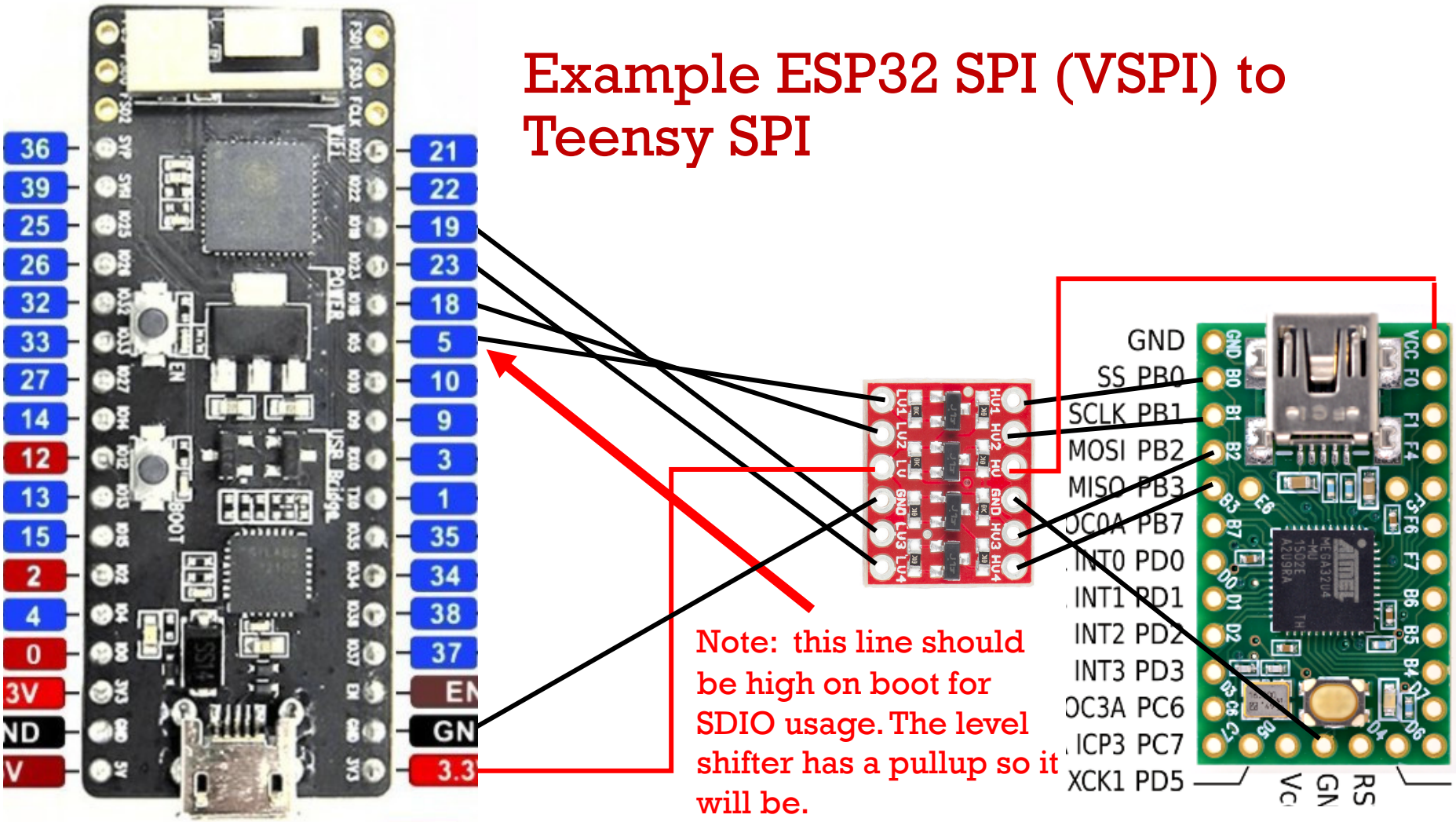
SPI on ESP32

There are four SPI. Two are used for on board memory. Two are available (VSPI, HSPI).

Most functions on ESP32 can be remapped to other pins.



Example ESP32 SPI (VSPI) to Teensy SPI



CANVAS files->Resources->Teensy Files->Teensy_SPI_slave.c

Sample C slave code (on Teensy)

```
#include "teensy_general.h"
#include "t_usb.h"

char buf[26];
// volatile since interrupt may change value
volatile unsigned char pos;
volatile char process_packet;

ISR (SPI_STC_vect)
{ // grab byte from SPI Data Register
  unsigned char c = SPDR;
  if (pos < sizeof buf) { // add to buffer if room
    buf [pos++] = c;
    if (c == '\n')   // set flag if newline
      process_packet = 1;
  }
  else pos = 0;
}  // end of interrupt routine SPI_STC_vect
```

```
int main() {
  int i;
  m_usb_init();
  set(DDRB,3); // set  MISO B3 as output,
  set(SPCR, SPE);  // enable SPI
  set(SPCR, SPIE); // enable SPI interrupt
  sei();  // enable all interrupts

  while(1) {
    if (process_packet) {
      for (i=0;i<pos;i++)
        m_usb_tx_char(buf[i]);
      pos = 0;
      process_packet = 0;
    }  // end of process_packet
  }  // end of loop
}
```

Q6 Why don't we use "clear(SPCR,MSTR)"

CANVAS files->Resources->ESP32 Arduino Lecture Examples->ESP32-SPI-master.ino

Sample Arduino Master Code (on ESP32)

```
#include <SPI.h>
static const int spiClk = 1000000; // 1 MHz
SPIClass * vspi = NULL;

void setup() {
  vspi = new SPIClass(VSPI);
  vspi->begin(); //default pins: SCLK = 18, MISO = 19, MOSI = 23, SS = 5
                 //alternatively set  pins e.g. vspi->begin(0, 2, 4, 33);
  pinMode(5, OUTPUT);  // use 5 for SS
}


void loop (void) {
  byte data = 0b01010101;
// junk data to illustrate usage

  vspi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
  digitalWrite(5, LOW); //pull SS slow to prep other end for transfer
  vspi->transfer(data);
  digitalWrite(5, HIGH); //pull ss high to signify end of data transfer
  vspi->endTransaction();delay(100);
}
```

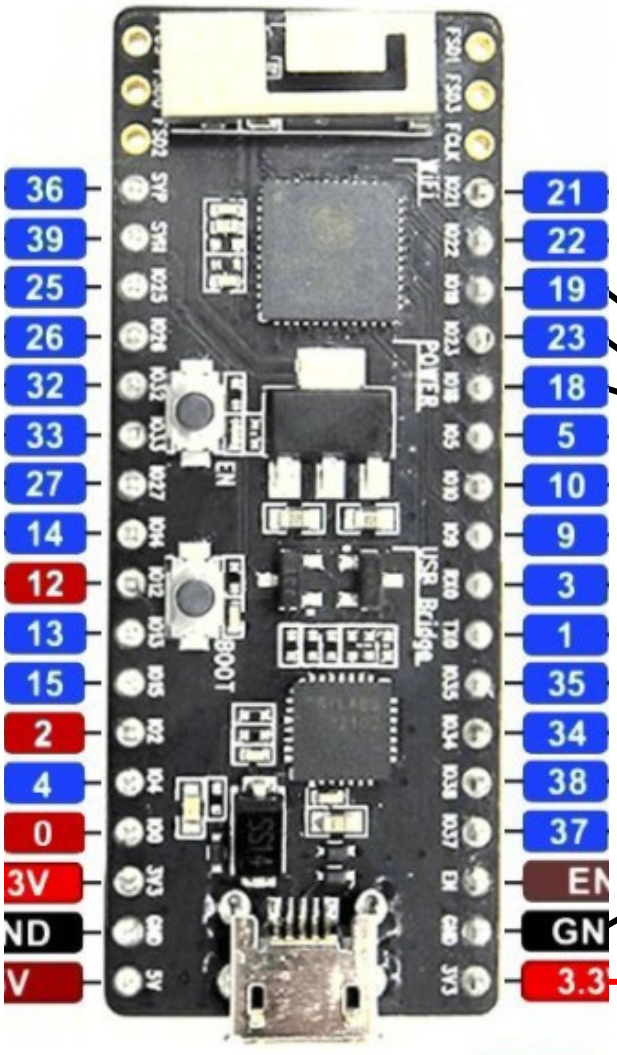
clock speed

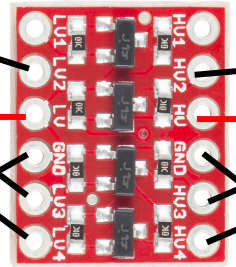
Bit order

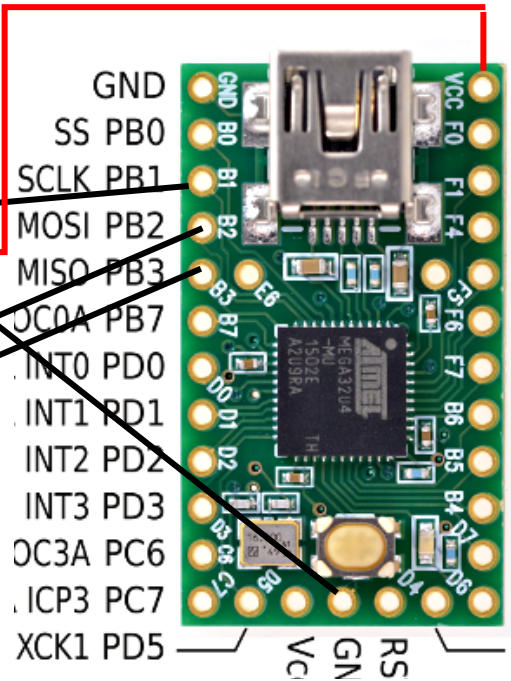
CPOL, CPHA mode

Example ESP32 SPI (VSPI) to Teensy SPI

Hack: If you have only one slave, and you are desperate, you could theoretically wire SS on slave to gnd. Though you risk desyncronizing transfers.

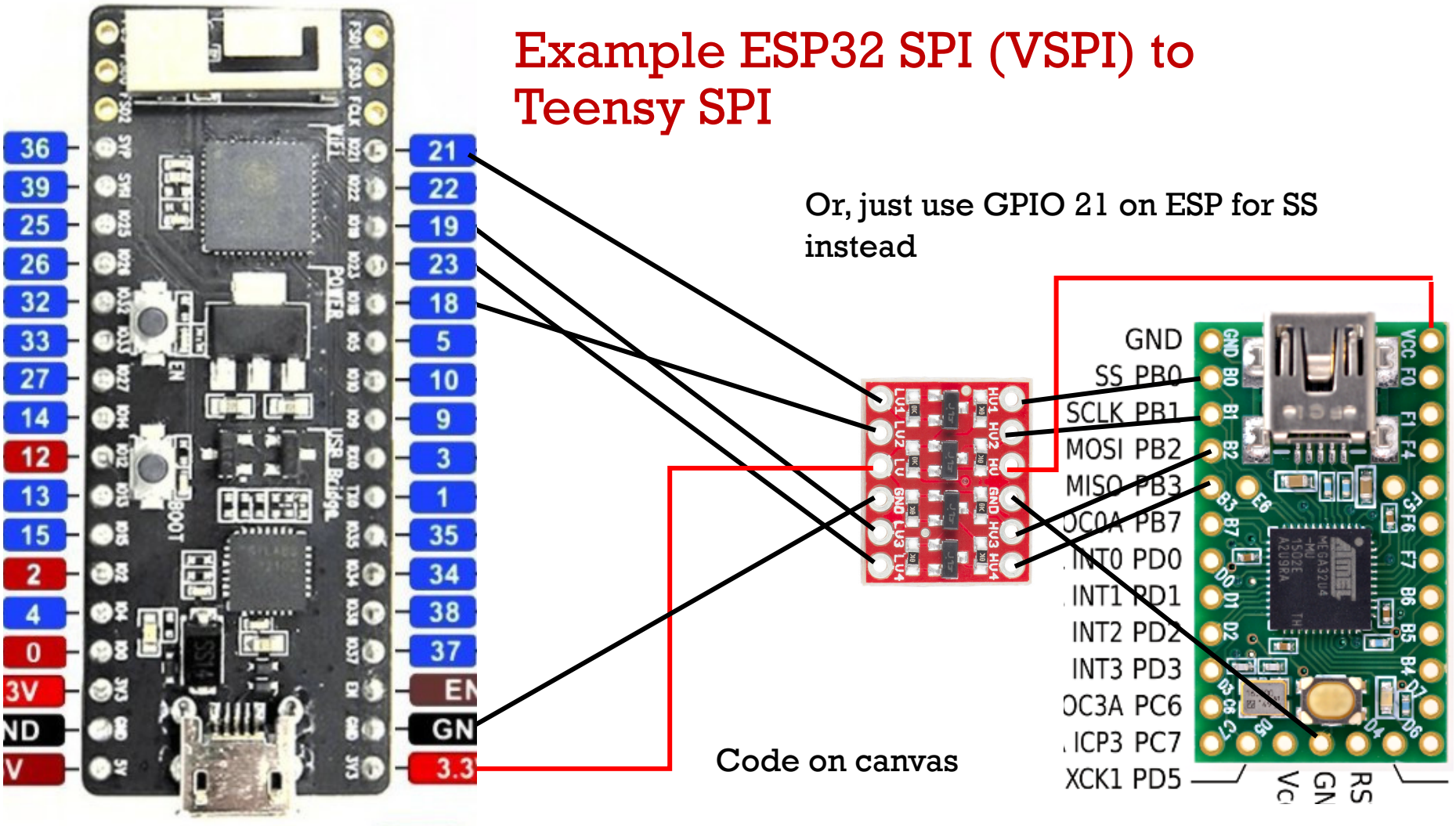






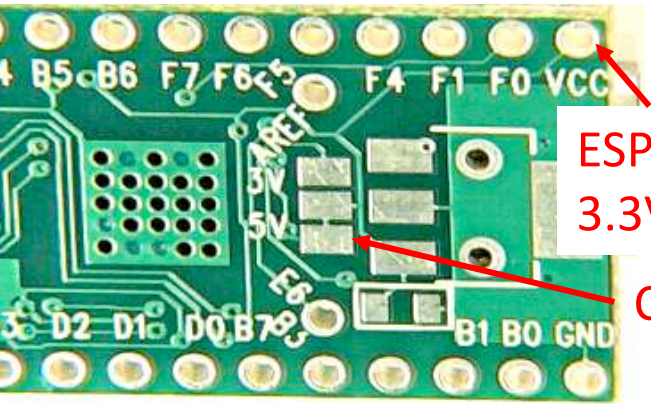
For ESP as master, it doesn't matter which GPIO we use for SS. So, just use a different one.

Example ESP32 SPI (VSPI) to Teensy SPI



3.3V conversion

- Can use ESP32 3.3V
- Modify teensy –
  - Cut trace to prevent damaging ESP32 when connecting USB
  - Add solder jumper

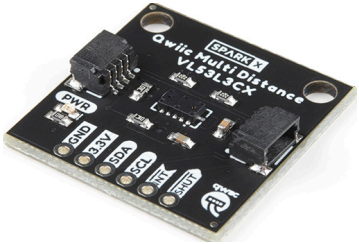


https://www.pjrc.com/teensy/3volt.html

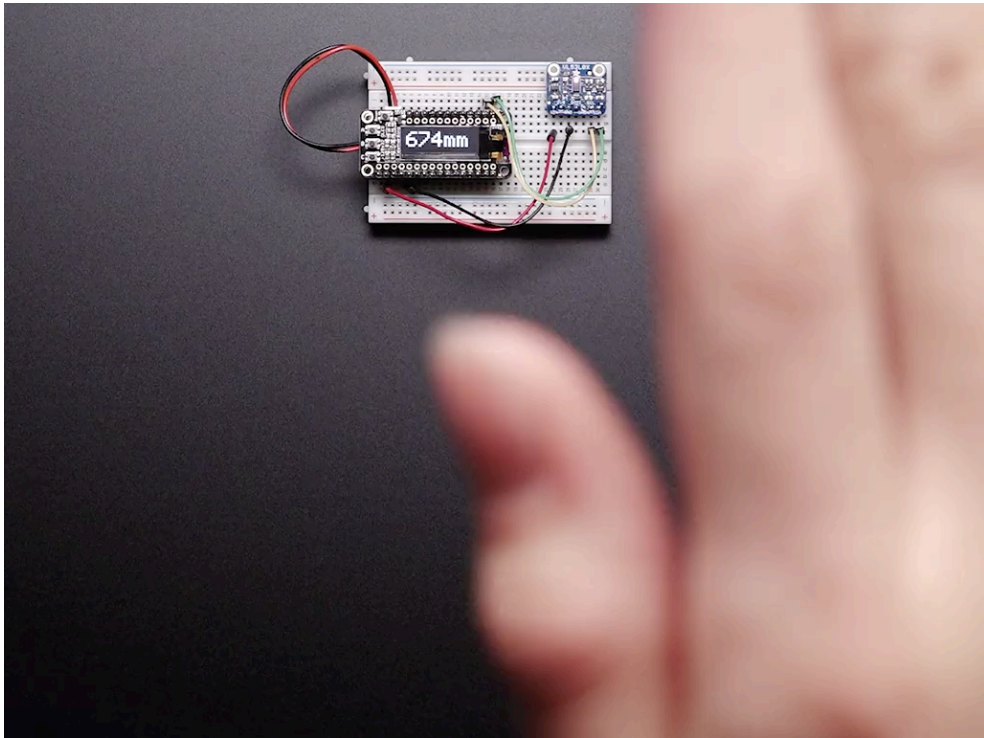
04

I2C Peripheral Example

TOF Range Finder (VL53L0X)



- Range: 5cm to 120cm
- Speed: ~30 samples/sec
- Beam width: ~25°
- $14.95
- Interface with **I2C**,
- Adafruit has Arduino library.
- Very accurate and linear

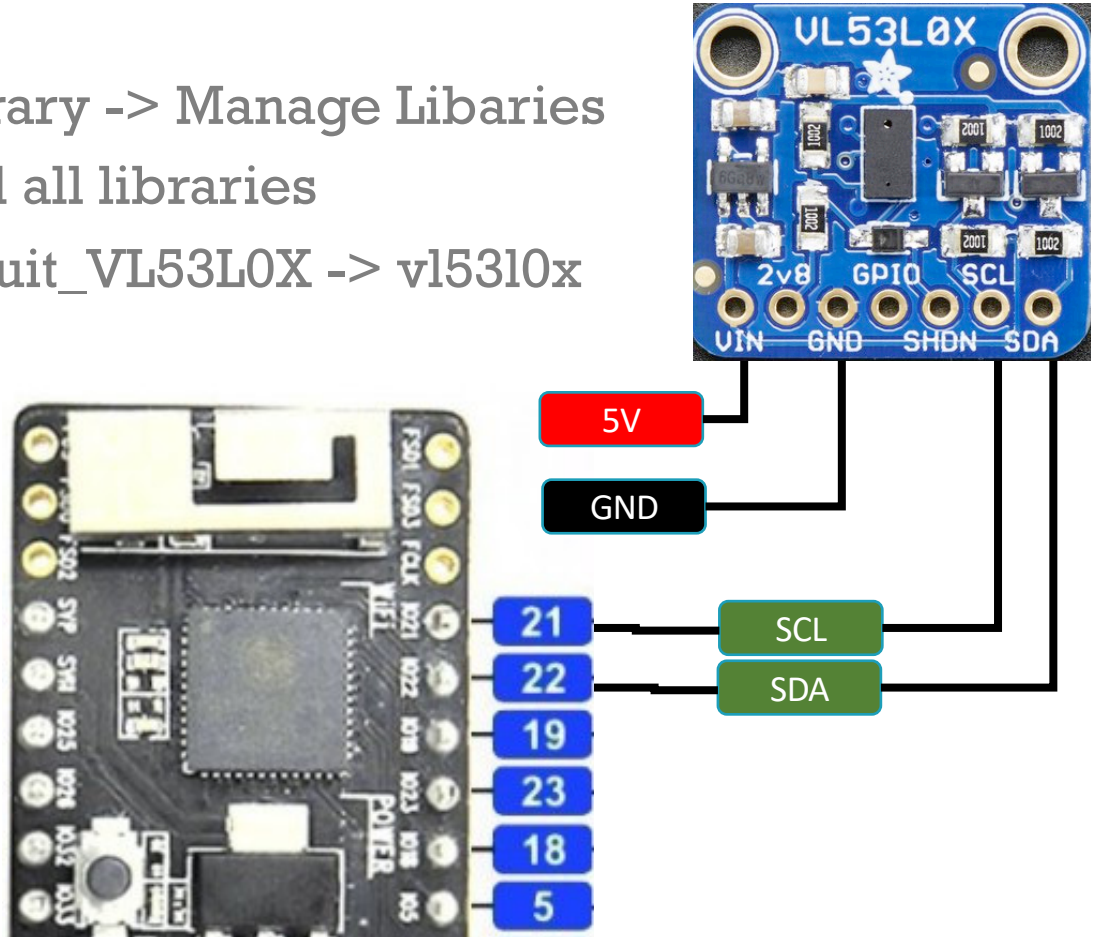


https://www.adafruit.com/product/3317

I2C example VL53L0X from Adafruit

- IDE Sketch -> Include Library -> Manage Libaries
- Search for VL53L0X install all libraries
- File -> Examples -> Adafruit_VL53L0X -> vl53l0x

- ESP32 default SCL is GIPO21
- ESP32 default SDA is GPIO22
- Attach SDA to SDA
- Attach SCL to SCL



Demo of ToF ranger with objects

PROS

- Easy to install library
- Range values independent of color
- Angle does not change the precision if value is received.
- You know when you have good values or no value.

CONS

- Color does affect ability to get values at all
- Angle of reflection affects ability to get a range value.

05

ESP32 Memory

Memory usage

```
#define RECORDSIZE 6
#define BUFFERSIZE RECORDSIZE*10000
uint8_t s[BUFFERSIZE];
```

- Sample program uses most of dynamic memory.
  - Uses 60000 bytes of data records:

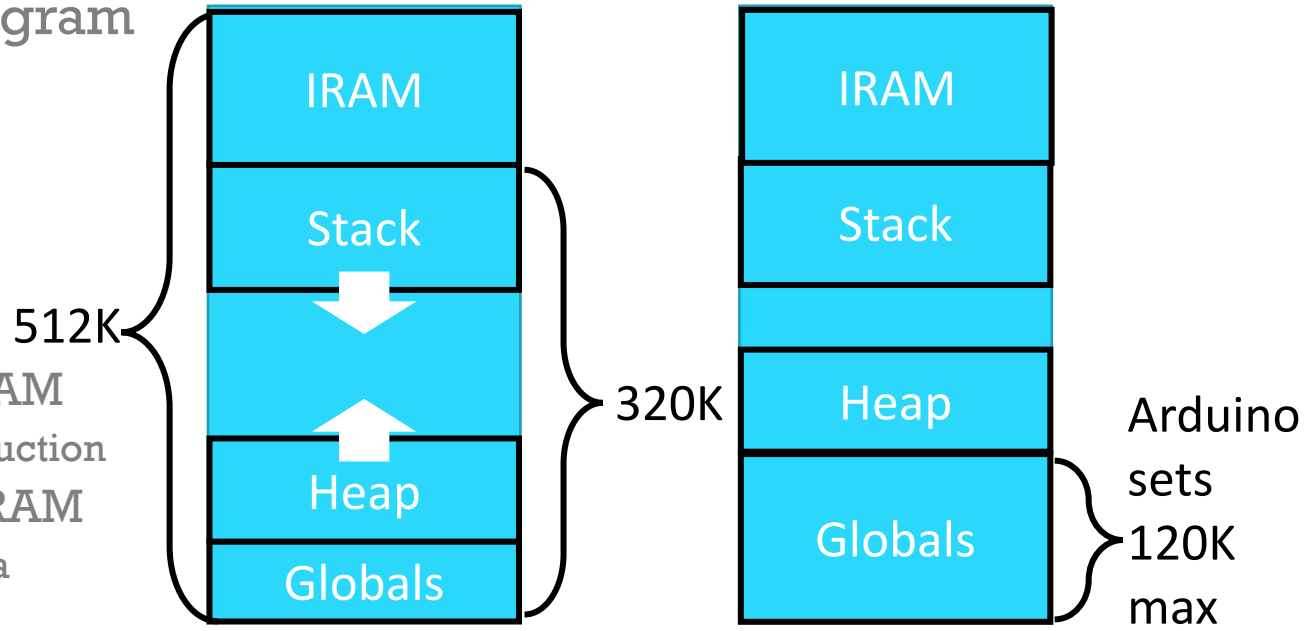
```
Global variables use 100508 bytes (30%) of dynamic memory, leaving
227172 bytes for local variables. Maximum is 327680 bytes.
```

  - 84068 bytes of data records is fine.
  - 84070 bytes of data records lead to error:

```
../xtensa-esp32-elf/bin/ld: acceldemo.ino.elf section `.dram0.bss' will
not fit in region `dram0_0_seg'
../xtensa-esp32-elf/bin/ld: region `dram0_0_seg' overflowed by 8 bytes
```

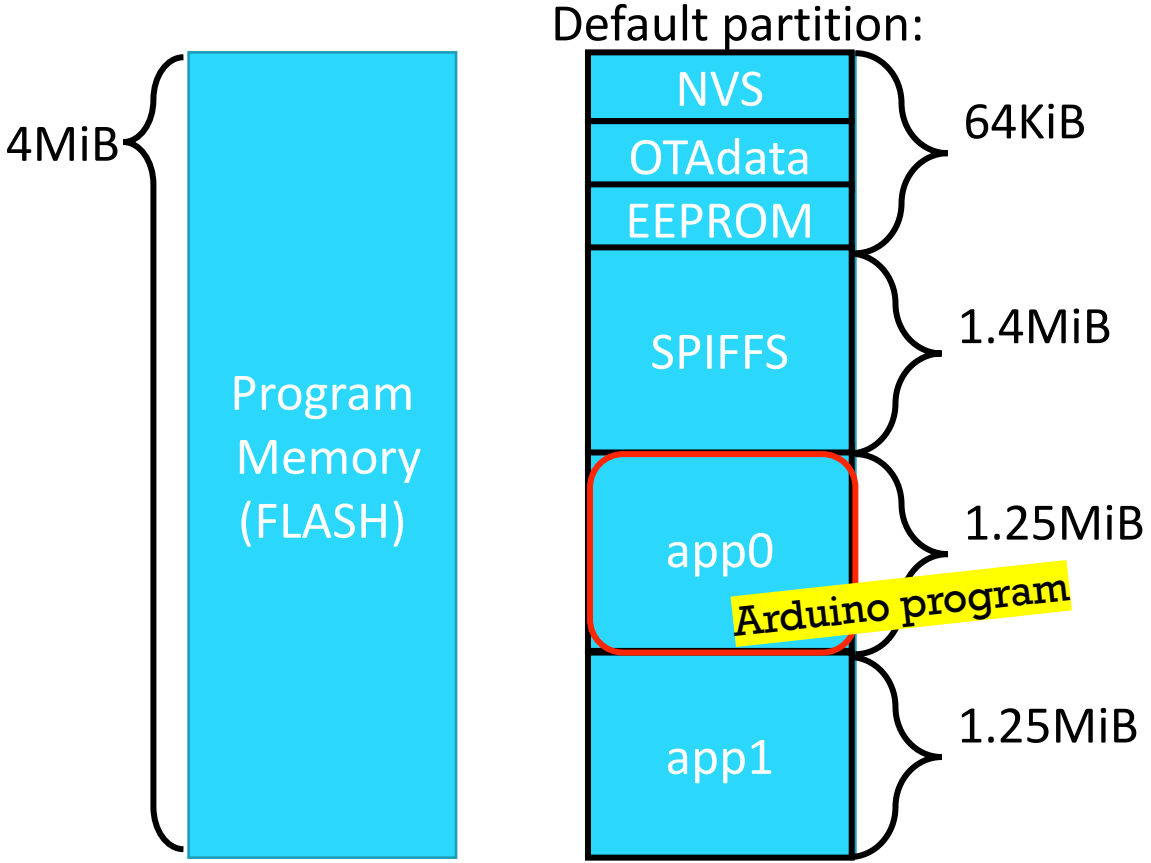
ESP32 Memory (Dynamic Memory allocation)

- 4MiB Flash Program
- 512KiB SRAM
  - 192 KiB for IRAM
    - Fast DP Instruction
  - 320 KiB for DRAM
    - RAM for Data



ESP32 Memory (Program Memory (FLASH))

- 4MiB Flash Program
  - 1.25 MiB allocated for Arduino App
  - All but 64KiB available
  - Partitioned
- 512KiB SRAM
  - 192 KiB for IRAM
    - Fast DP Instruction
  - 320 KiB for DRAM
    - RAM for Data



Typical Program Sizes

- Empty program – 192K bytes
- Uses Serial.print – add ~6K
- Uses ledc – add ~3K
- Uses interrupts – add ~2K
- Uses ADC – add ~3K
- Uses I2C – add ~25K
- Uses Wifi – add 450K

- Typical MEAM510 program will be about 700k

Partition table

- Uses "partitions" like file systems
- Can shift allocation around
- OTA = "Over the air" upgrades. (assumes double memory)

# Name	Type	SubType	Offset	Size	Size [decimal]
nvs	data	nvs	0x9000	0x5000	20,480
otadata	data	ota	0xe000	0x2000	8,192
app0	app	ota_0	0x10000	0x140000	1,310,720
app1	app	ota_1	0x150000	0x140000	1,310,720
eeprom	data	0x99	0x290000	0x1000	4,096
spiffs	data	spiffs	0x291000	0x16F000	1,503,232

Arduino program

Extra files

Total is 4M

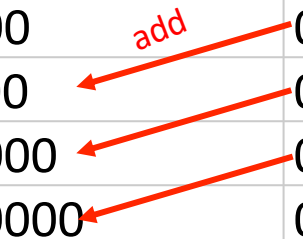
Changing FLASH memory allocation

- Copy "esp32/tools/partitions/default.csv" to a backup,
- Edit this default.csv file to reallocate memory

- Be sure sizes add to offset.

- Arduino IDE will recompile
  - Won't be reflected in IDE output
  - Need to manually check partition

# Name	Offset	Size
nvs	0x9000	0x5000
otadata	0xe000	0x2000
app0	0x10000	0x140000
app1	0x150000	0x140000



Routine to check size of partition

```
void printPartition() {
  esp_partition_iterator_t mp;
  const esp_partition_t *p;

  printf("Flash chip size: %d \n Partition Table:\n", spi_flash_get_chip_size());
  printf("type\t sub\t address\t size\t name \t encrypted\n");
  if (mp = esp_partition_find(ESP_PARTITION_TYPE_APP, ESP_PARTITION_SUBTYPE_ANY, NULL)) {
    do {
      p = esp_partition_get(mp);
      printf("%x \t %x \t %x \t %x \t %s - %i\r\n",
             p->type, p->subtype, p->address, p->size, p->label, p->encrypted);
    } while (mp = esp_partition_next(mp));
  }
  esp_partition_iterator_release(mp);
  if (mp = esp_partition_find(ESP_PARTITION_TYPE_DATA, ESP_PARTITION_SUBTYPE_ANY, NULL)) {
    do {
      p = esp_partition_get(mp);
      printf("%x \t %x \t %x    \t%x \t %s - %i\r\n",
             p->type, p->subtype, p->address, p->size, p->label, p->encrypted);
    } while (mp = esp_partition_next(mp));
  }
  esp_partition_iterator_release(mp);
}
```

06

Extra bits

Aside – on coding (all valid);

Which is better?

A:

```
int steps=1, bumpedflag=FALSE;
```

or

B:

```
int steps=1, bumpedflag;
```

or

C:

```
int steps=1;
bool bumpedflag;
```

Gobal variables are all initialized to 0 (not true for local variables). But explicitly assigning to 0 doesn't hurt and indicates intention

To use `bool` you must `#include <stdbool.h>`

Aside – on coding;

Which is better?

```
TCCR1B = (1<<CS10) | (1<<CS12);
```

Slightly faster and shorter
(smart compiler will join into one command)

vs

```
set(TCCRIB,CS10);
set(TCCRIB,CS12);
```

Is a little clearer,
(won't get confused with | or &)

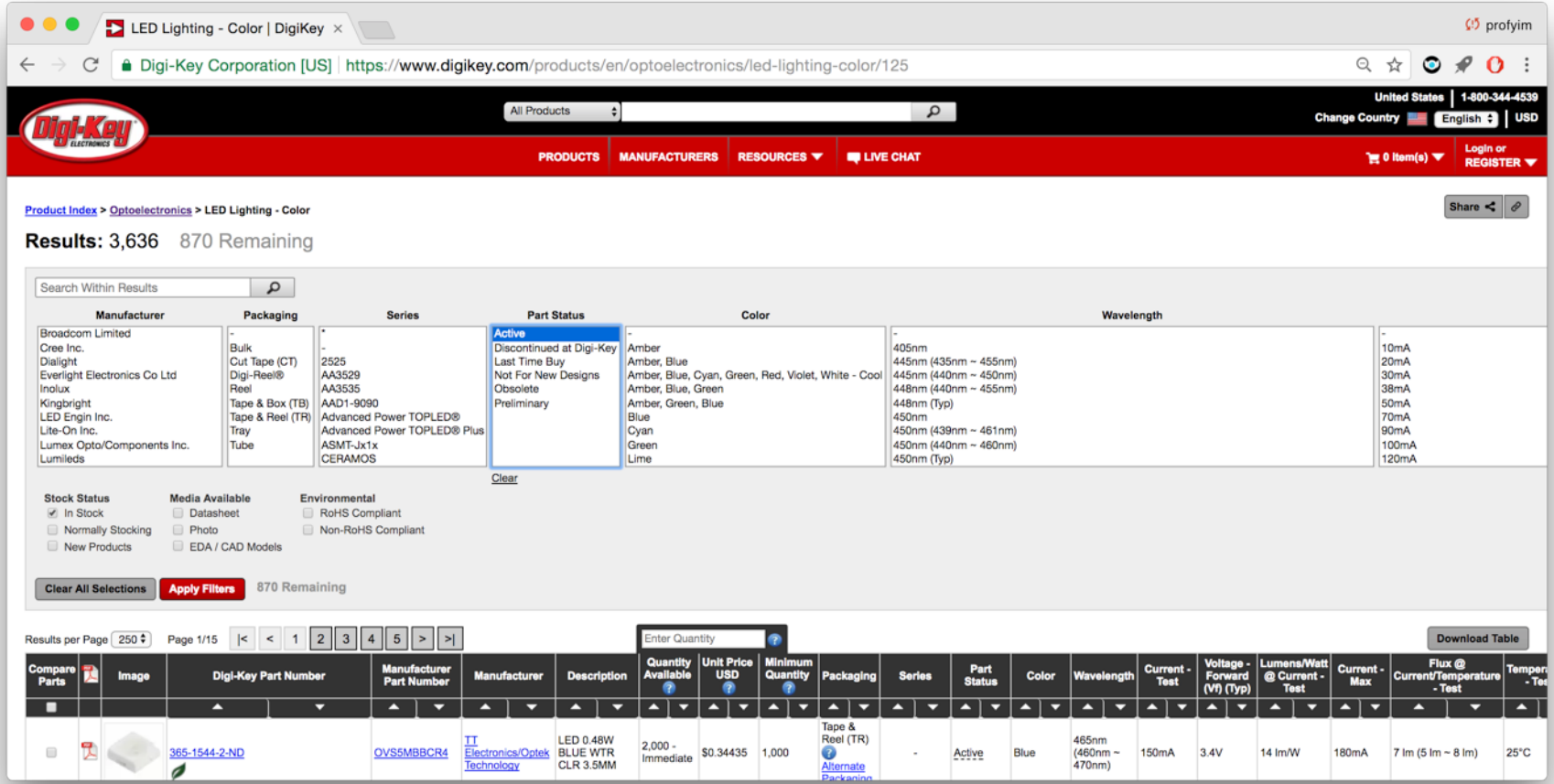
Aside – on coding 2;

From /usr/local/CrossPack-AVR/avr/include/**avr/sfr_defs.h** (on mac)

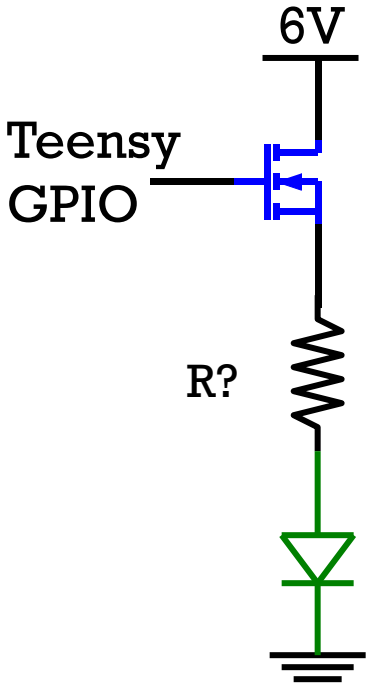
```
#define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
#define _BV(bit) (1 << (bit))
#define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))
#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
->
bit_is_set(sfr, bit)    ((*(volatile uint8_t *)(sfr)) & (1 << (bit)))
[mostly, skipping the part about casting the address pointer]

#define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
#define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
#define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))
#define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))
```

For C Casting tutorial see https://www.tutorialspoint.com/cprogramming/c_type_casting.htm

Digikey for LED



OSRAM GT PSLR31.13 (170 lumens)



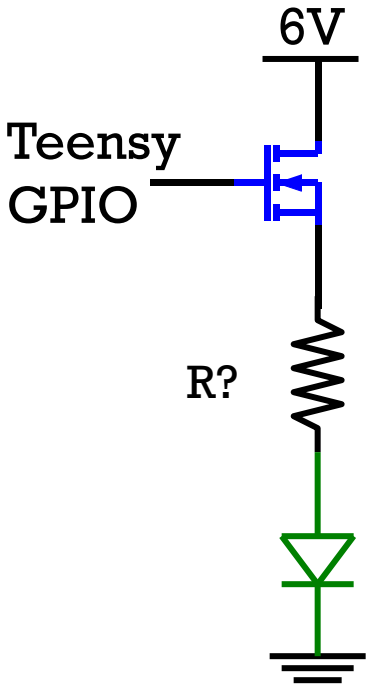
Q7: What value for R will maximize the brightness?

3mm x 3mm SMD

Parameter		Symbol	Values	Unit
Forward voltage 3) *page 21* Durchlassspannung 3) *Seite 21*	(min.) (typ.) (max.)	V_F V_F V_F	5.60 6.25 6.80	V V V

Parameter	Symbol	Values	Unit
Forward current Durchlassstrom (T_S = 25 °C)	I_F	10 ... 200	mA
Surge current Stoßstrom (t ≤ 10 µs; D = 0.005; T_S = 25 °C)	I_{FM}	300	mA

OSRAM GT PSLR31.13

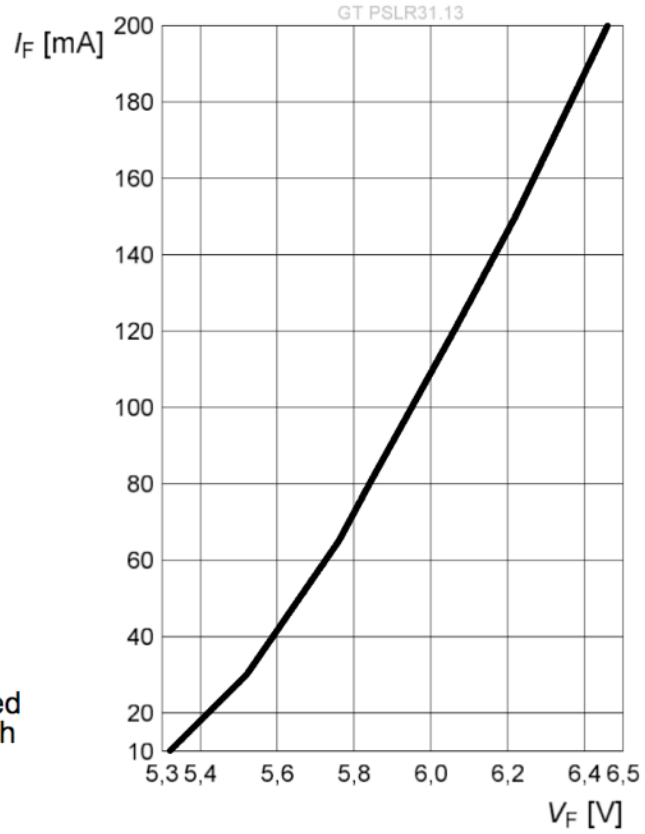


Forward Voltage Groups 3) *page 21*
Durchlassspannungsgruppen 3) *Seite 21*

Group Gruppe	(min.) V_F [V]	(max.) V_F [V]
B	5.60	5.80
C	5.80	6.00
D	6.00	6.20
E	6.20	6.40
F	6.40	6.60
G	6.60	6.80

3) **Forward Voltage:** The Forward voltage is measured during a current pulse duration of typically 1 ms with a tolerance of ± 0.05V .

Forward Current 5) *page 21* , 7) *page 21*
Durchlassstrom 5) *Seite 21* , 7) *Seite 21*
$I_F = f(V_F)$; $T_S = 25$ °C



Answer in CHAT

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I don't know now, but know what to do to get by
3. I understand some, but expect to get the rest later
4. I understand completely already

A. using UART between ESP32 and Teensy

B. using ToF sensor with I2C

C. SPI