Lecture 20

Failures and Debugging
Heat plus misc. coding stuff

Agenda

01. Heat and Heat sinks
02. Memory on ESP32
03. Using Internet DIY projects
04. Debugging Process

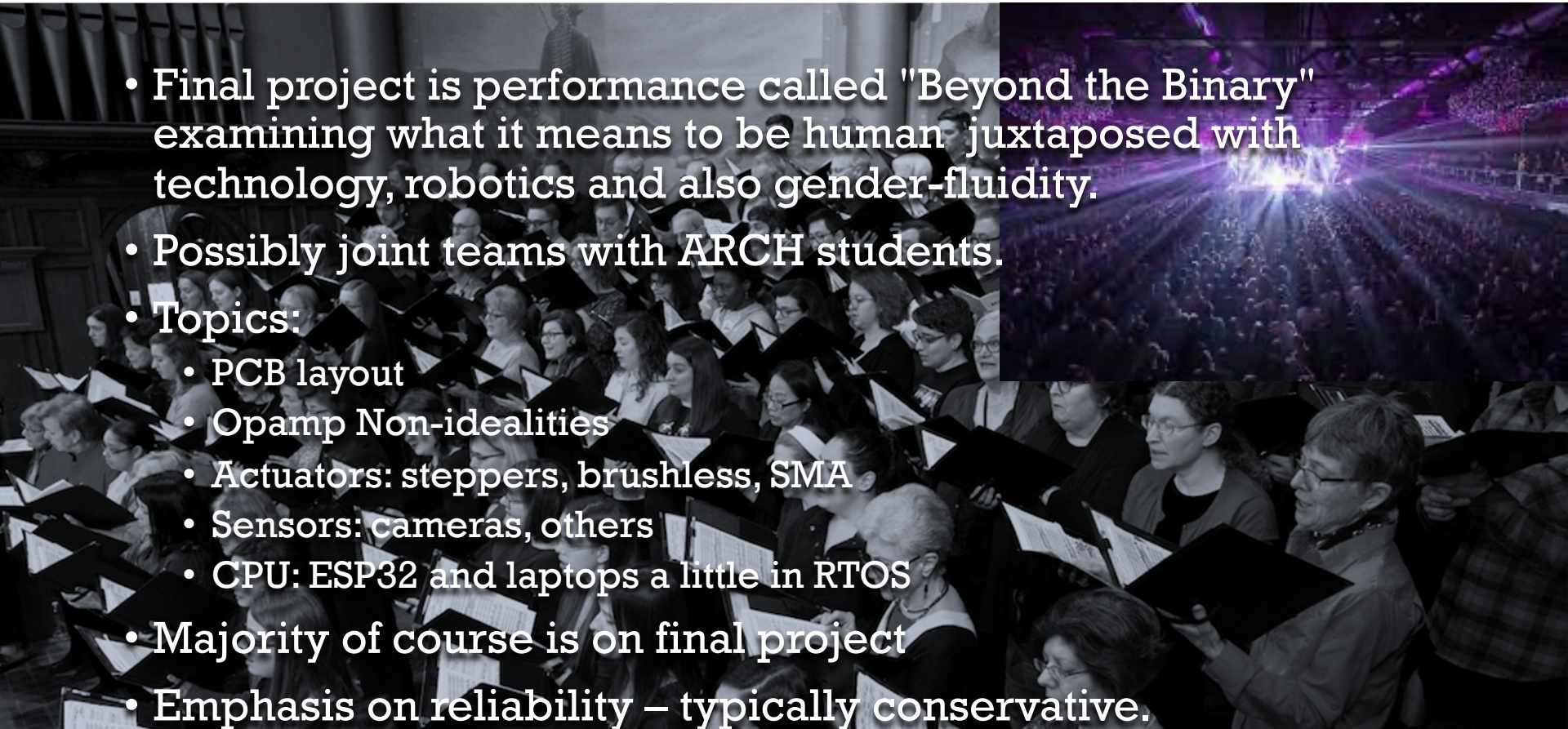
--- if we have time ---

05. Coding tips
06. Master-Slave / Client-Server Communications (AJAX)

Stuff

- Post regrade questions to Piazza (don't email TA's directly)
- Lab 4 races Sunday or Monday.
- Submit code with good comments (e.g. your name, reasonable file name etc.)
- Potential problem with GMlab router IP addresses. May need to change student IP from 198.162.1.XXX to 198.162.0.XXX

Adv. Registration:
MEAM/IPD516 Adv Mechatronics in Reactive Spaces



- Final project is performance called "Beyond the Binary" examining what it means to be human juxtaposed with technology, robotics and also gender-fluidity.
- Possibly joint teams with ARCH students.
- Topics:
  - PCB layout
  - Opamp Non-idealities
  - Actuators: steppers, brushless, SMA
  - Sensors: cameras, others
  - CPU: ESP32 and laptops a little in RTOS
- Majority of course is on final project
- Emphasis on reliability – typically conservative.

MEAM510 T-Shirts

- Designed by Current MEAM 510 IPD students
- Order forms will go out soon for your size
- You get one free for completing the class. You may purchase extras if desired.

01

Failure: Connectors, Heat and Heat Sinks

Connectors are the most common point of failure

- In Mechatronic systems.
- Assuming everything else is correct (software and hardware design etc.)

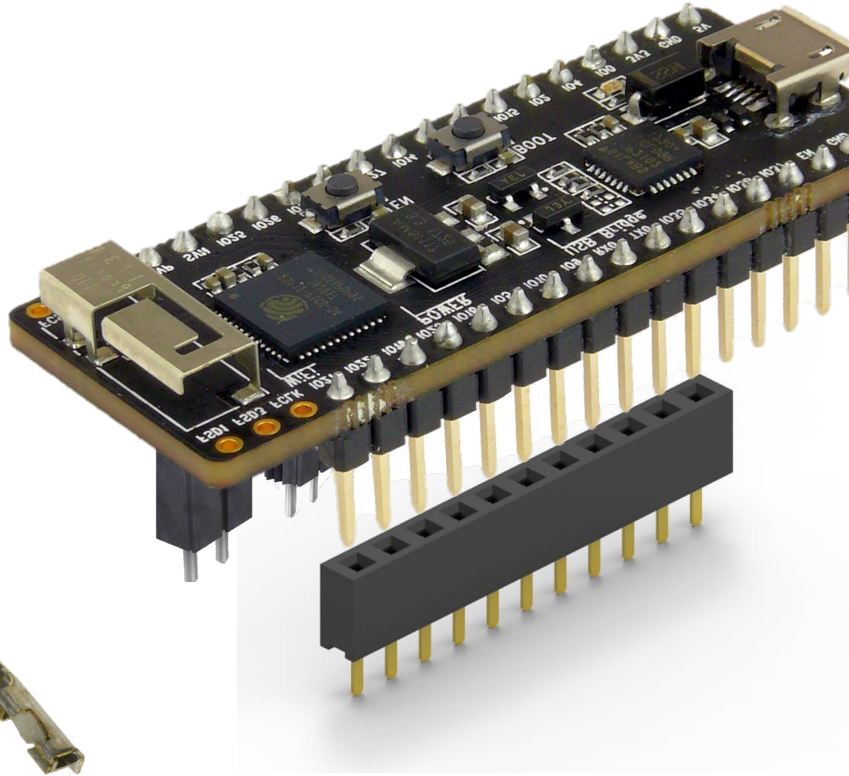
Connectors and sockets

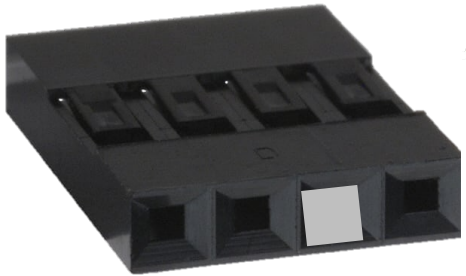
- Molex connectors are good for cables.
- They are "keyed" so you can't plug them in backwards
- The crimping tool is works well, though possible to crimp with pliers (as demonstrated in recitation).
- Crimper is wired to the table (it is expensive)
- What if the worst case power requirements is 5A and you want to use molex connectors to connect your battery?

Molex KK 254 series
22 to 30 AWG wire
Max 4.0 A per line.

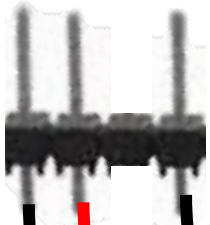
Connectors and sockets

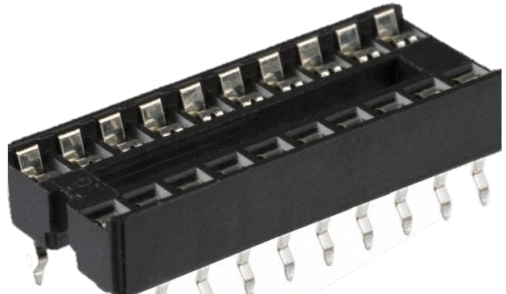
- Use sockets for IC's especially ESP32 and Teensy.
- Headers and connectors can make cables but good to either key them or make them symmetric



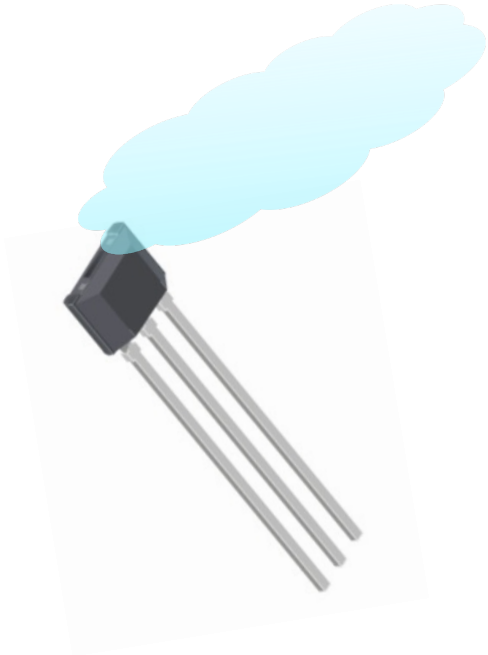


Amphenol MiniPV
Max 3A per line





When chips fail they usually get very hot



Which electronics get hot?

Power Transistors

- IRLB8721
- TIP32
- TIP102
- ULN2003*

Power circuits

- LM7805 – 5V Regulator
- SN754410* – H-Bridge

* DIP package in ministore

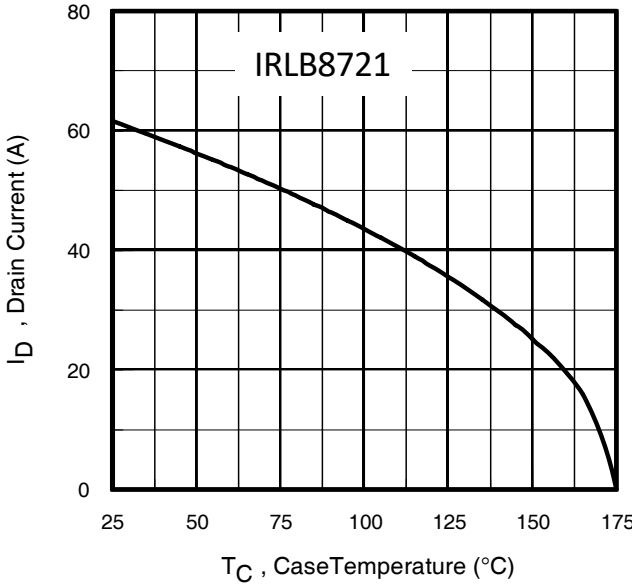


Fig 9. Maximum Drain Current vs. Case Temperature

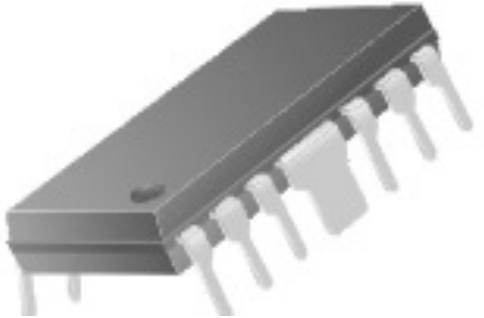
7.2 Recommended Operating Conditions SN754410

over operating free-air temperature range (unless otherwise noted)

		MIN	MAX	UNIT
V_{CC1}	Logic supply voltage	4.5	5.5	V
V_{CC2}	Output supply voltage	4.5	36	V
V_{IH}	High-level input voltage	2	5.5	V
V_{IL}	Low-level input voltage	–0.3 (1)	0.8	V
T_J	Operating virtual junction temperature	–40	125	°C
T_A	Operating free-air temperature	–40	85	°C

FAN8100

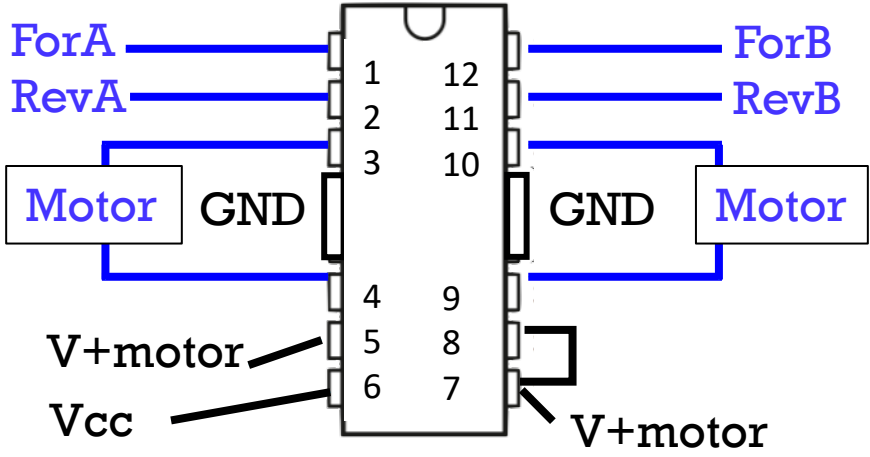
- Obsolete chip (some in ministore)
- Has better output Voltage than SN754410.
- Peak current 1.5A (same as yellow motors @ stall)
- Datasheet available on Canvas -> files-> Resources



gnd ---------- PWM ---- ForA

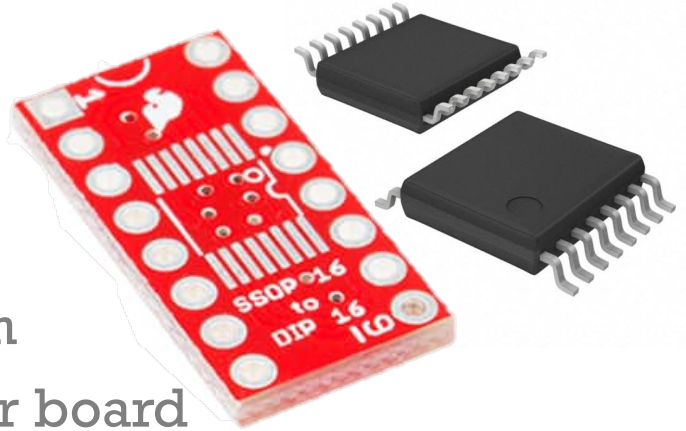
PWM ---------- gnd ---- RevA

Reverse Forward



FORWARDA	REVERSEA	OUT1A	OUT2A	Function
L	L	Z	Z	Stand-by (Stop)
H	L	H	L	Forward
L	H	L	H	Reverse
H	H	L	L	Brake

MOSFET H-bridge (LV8401V)

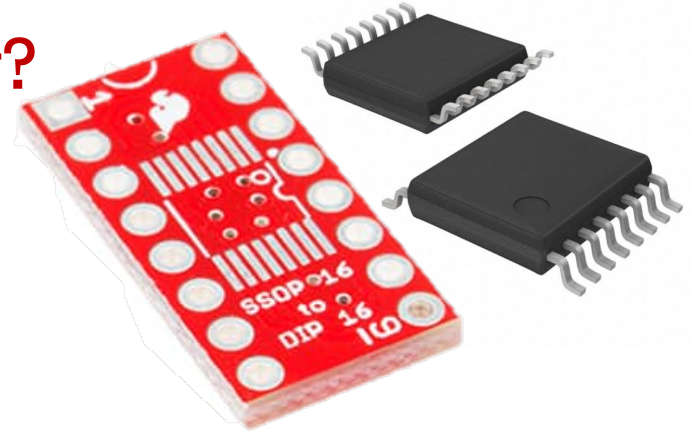


- I_{peak} = 3.8A, 1.2A continuous
- MOSFET so no V_{OH} or V_{OL}, $R_{DS(ON)}$ = 0.5ohm
- Problem SMD, need to solder onto adapter board

Q1) If we were driving a motor at 1A what is the loss in output voltage (effectively V_{CC} - V_{OH} - V_{OL}) for this MOSFET?

- Compare: SN754410 typ @ 1A, $V_{OH} = V_{CC}$-1.8 and V_{OL} = 1.2
  - (for 5V supply lose 3V with 2V on motor.)
- Compare: FAN8100 @1A not listed, $V_{OH} = V_{CC}$-0.7 and V_{OL} = 0.7
  - (for 5V supply lose 1.5V with ~3.5V on motor.)

What happens to the lost energy?



Generated as heat.

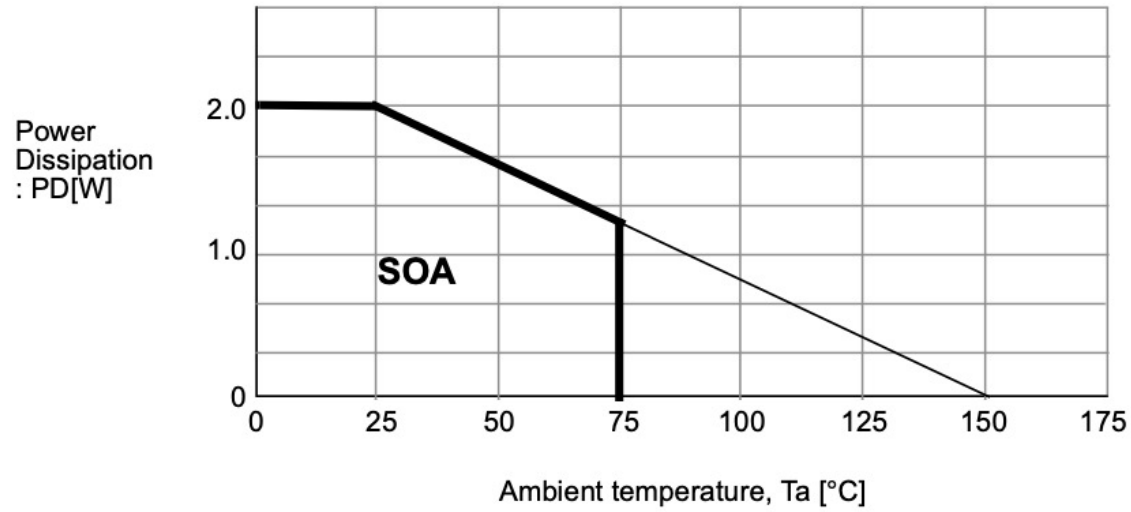
- FAN8100 3x worse than LV8401V
- SN754410 is 2x worse than FAN8100
- But LV8401V is MOSFET that will generate more heat from PWM

	Dissipated power
• @1A x 0.5ohm = 0.5V	~0.5W
• Compare: SN754410 typ @ 1A, $V_{OH} = V_{CC}$-1.8 and V_{OL} = 1.2	
• (for 5V supply lose 3V with 2V on motor.)	~3W
• Compare: FAN8100 @1A not listed, $V_{OH} = V_{CC}$-0.7 and V_{OL} = 0.7	
• (for 5V supply lose 1.5V with ~3.5V on motor.)	~1.5W

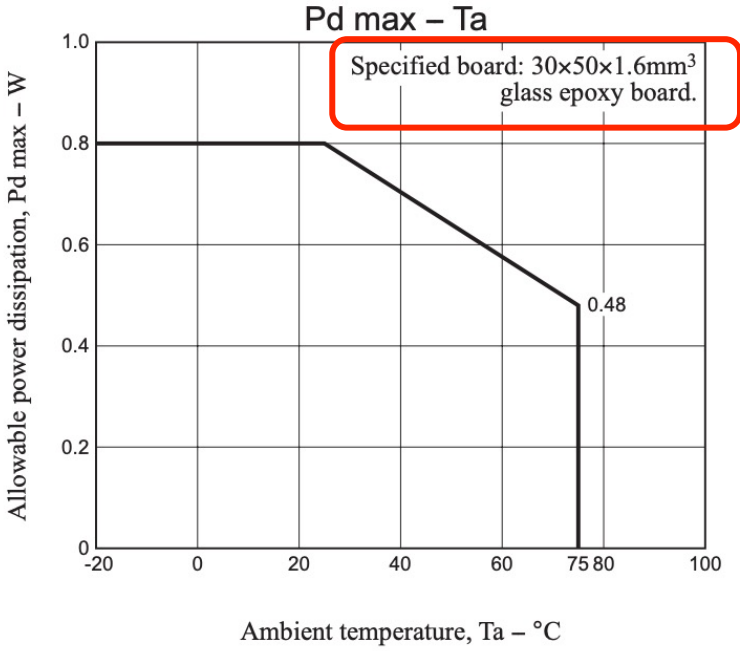
Thermal limits on motor drivers

FAN8100

- Thermal graphs wrt ambient depends on mounting
- Sometimes will include heat sink data

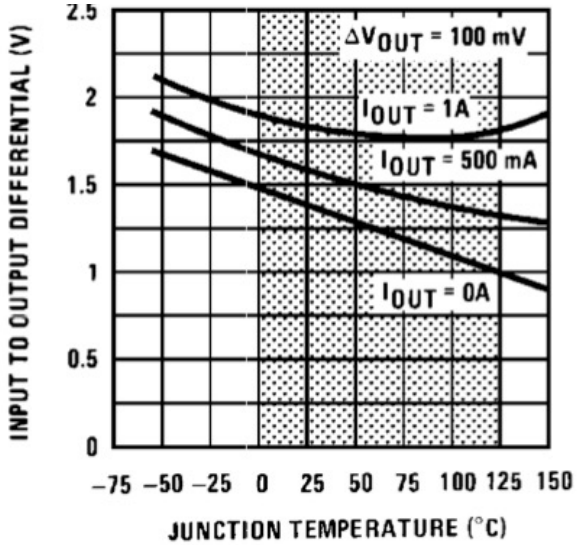


LV8401V



LM7805 Vreg Thermal issues

- Junction temperature depends heavily on package type and heat sink



Shaded area refers to LM340A/LM340, LM7805, LM7812, a LM7815.

Figure 11. Dropout Voltage

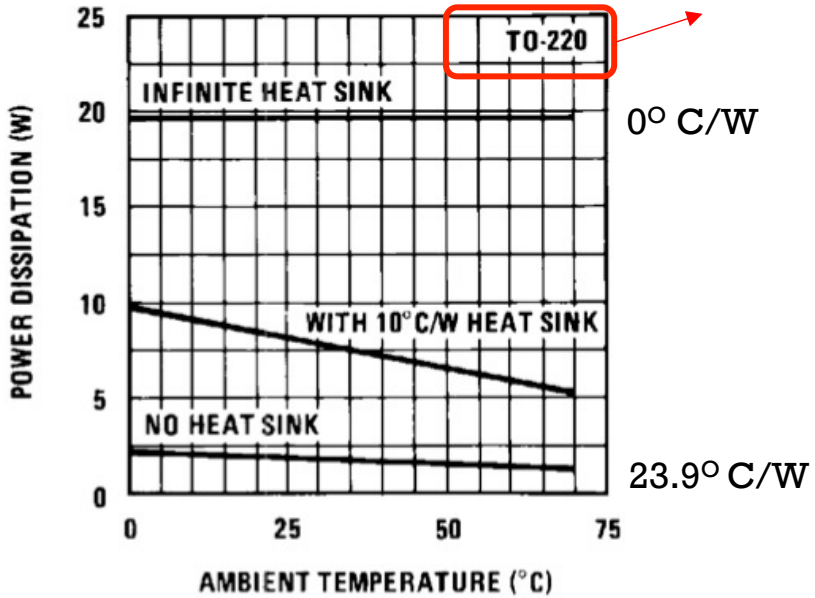


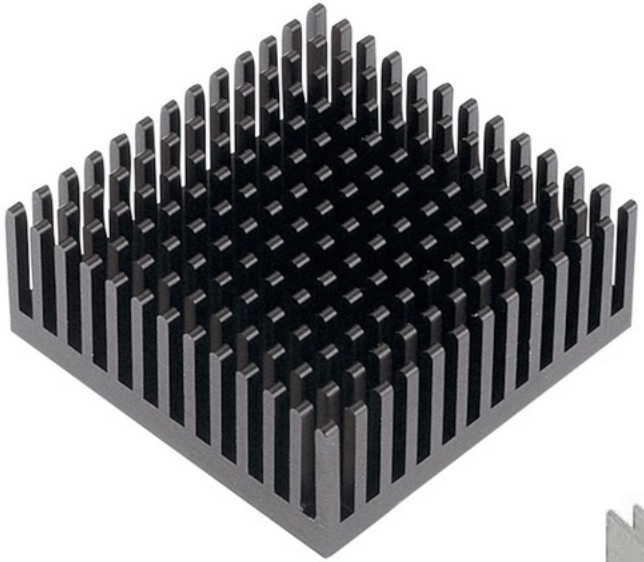
Figure 2. Maximum Average Power Dissipation

Q2: At room temp, what is the largest voltage we could use for 5V out @1A with a 5° C/W heat sink?

High Temperatures Kill Devices

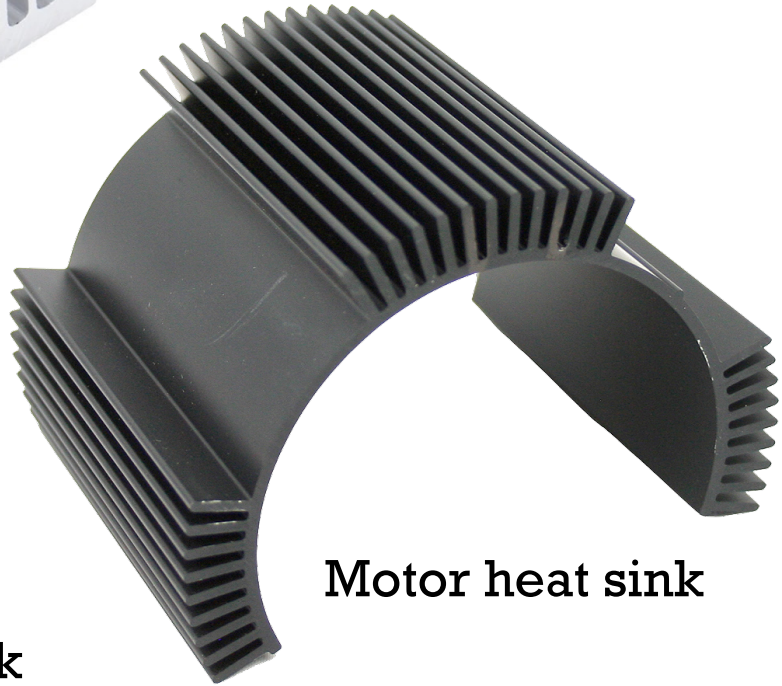
- Many devices have thermal shutdown (LM7805, SN754410, VL8401V, FAN8100). Others just burn up and stop working.
- Heat comes from dissipated power.
- For linear voltage regulators, higher voltage supply for fixed output voltage, linearly more power dissipated.
  - Ex: 13V in to 5V out (8 volts dissipated @ some current)
  - Ex: 9V in to 5V out (4 volts dissipated @ some current)
  - ½ as much power dissipated ½ as much heat generated. Less likely to reach thermal shutdown.
- How to reduce heat?

Heat Sinks



CPU heat sink

Relay heat sink

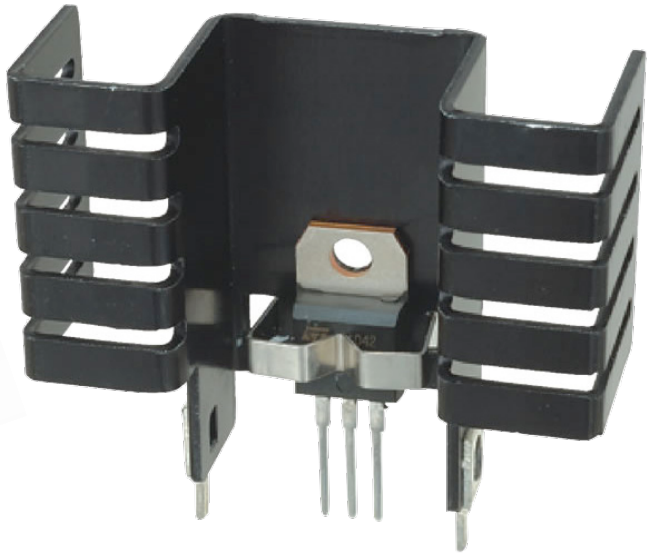


Motor heat sink

TO-220 Package

Thermal Resistance ~24°C/W

- LM7805
- MOSFETS in ministore
- TIP102
- TIP32



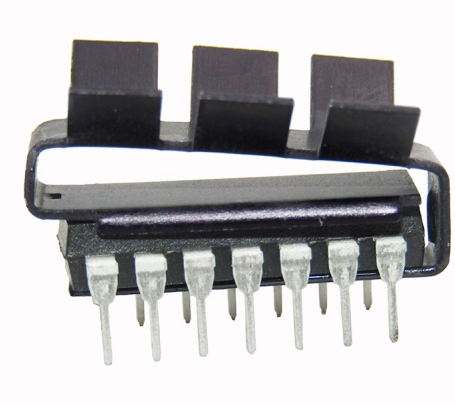
Thermal resistance:
32° C/W natural
10° C/W w/200 LFM

DIP16

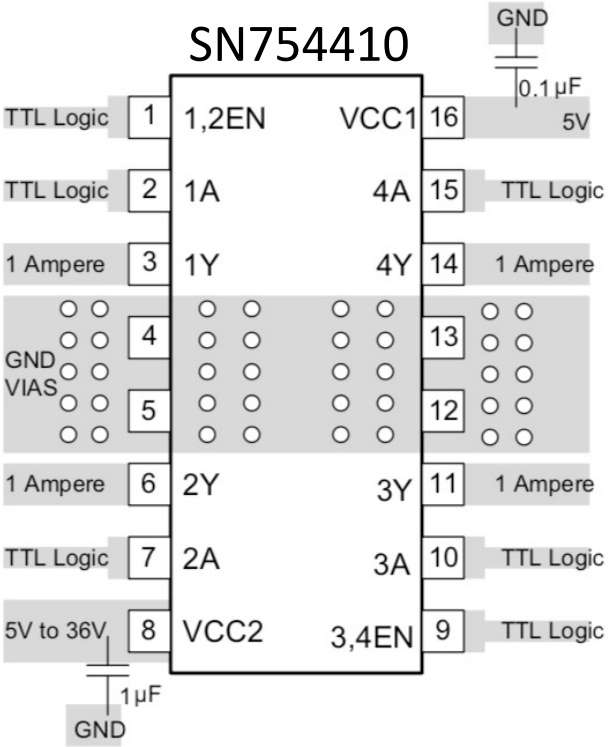
Thermal Resistance of ~60° C/W

- SN754410 H-Bridge
- FAN8100
- ULN2003 Darlington Pair Drivers

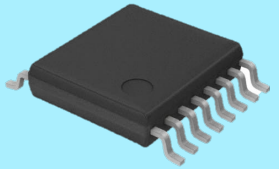
$0.59 on Digikey
48° C/W



$1.84 on Digikey
20° C/W



For reference
LV84014 uses SSOP16

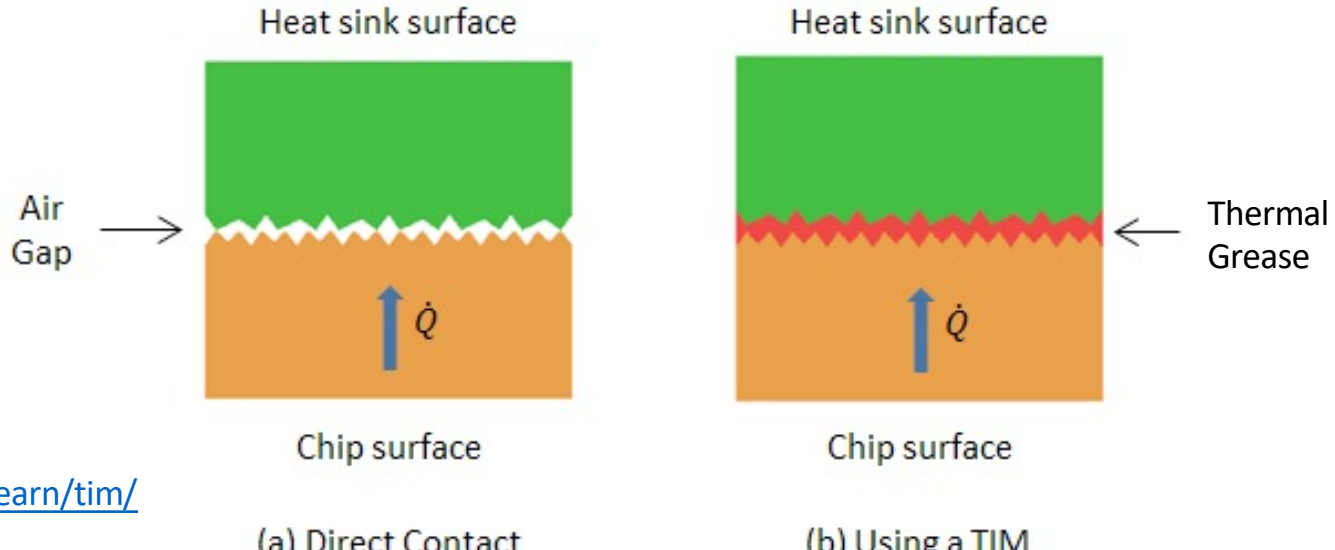


95° C/W

Thermal grease will make heat sinks work

- A dry interface will not conduct heat very well.
- Thermal grease greatly improves performance.
- Many types of grease work but, thermal grease will tend to last longer and not dry out.

https://myheatsinks.com/learn/tim/



Heat Summary

- When you are pushing limits on performance, heat can be a problem.
- Overheating most common cause of otherwise correctly designed circuits.
- Heat dissipation is function of package and environment (heat sinks and ambient temperature)
- Using heat sinks can have huge effect.

02

ESP32 Memory Size

(what happens when your program is too big?)

Memory usage

- What does this message mean?

```
  ESP.restart();
}
esp_now_register_send_cb(OnDataSent);

while (esp_now_add_peer(&peer1) != ESP_OK) {
  Serial.println("Pair failed");
```

Done uploading.

```
Sketch uses 669630 bytes (51%) of program storage space. Maximum is 1
Global variables use 31732 bytes (9%) of dynamic memory, leaving 2959
esptool.py v3.1
Serial port /dev/cu.SLAB_USBtoUART
```

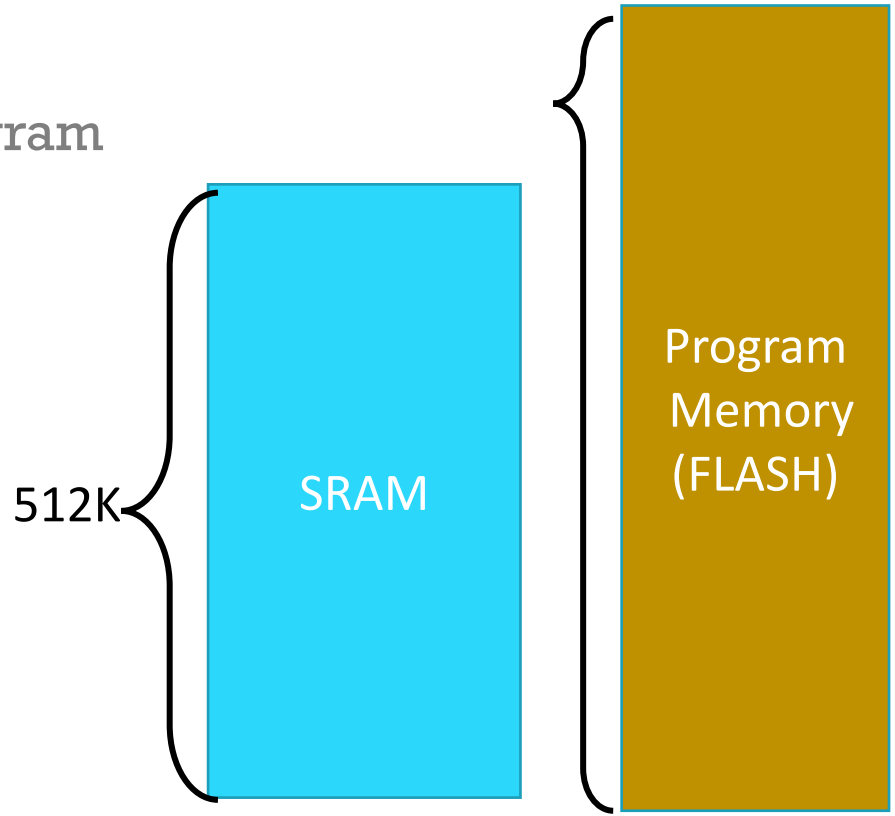
73 ESP32 PICO-D4 on /dev/cu.SLAB_USBtoUART

Sketch uses ## bytes (##%) of program storage space. Maximum is 1310720 bytes.

Global variables use ## bytes (##%) of dynamic memory, leaving #### bytes for local variables. Maximum is 327680 bytes.

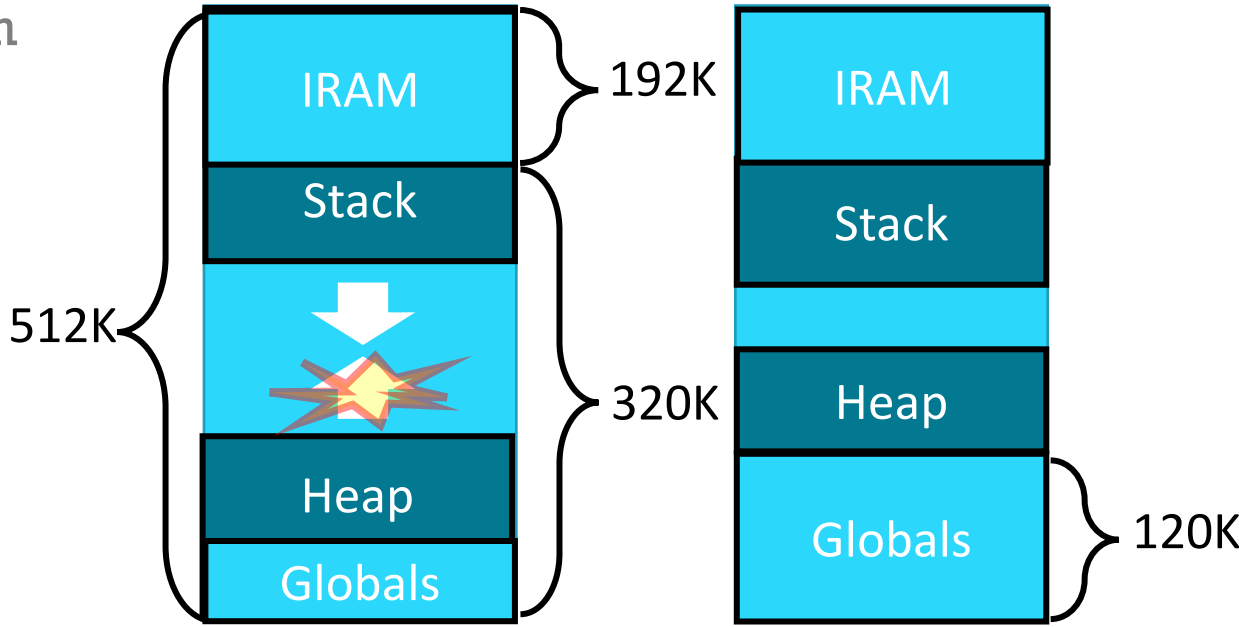
ESP32 Memory (Two Types)

- 4MiB Flash Program
- 512KiB SRAM



ESP32 Memory (Dynamic Memory allocation)

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



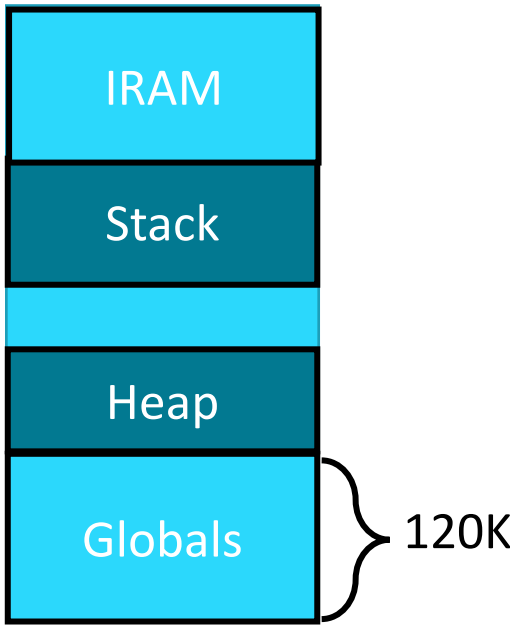
Globals allocated at compile time.
Heap & stack change at run time

Max globals set by memory segments

ESP32 Memory (Dynamic Memory allocation)

- When is SRAM size an issue in MEAM510?
- Large array variables
  - Used in subroutines grow stack, malloc() grows heap. Globals allocated in global section.

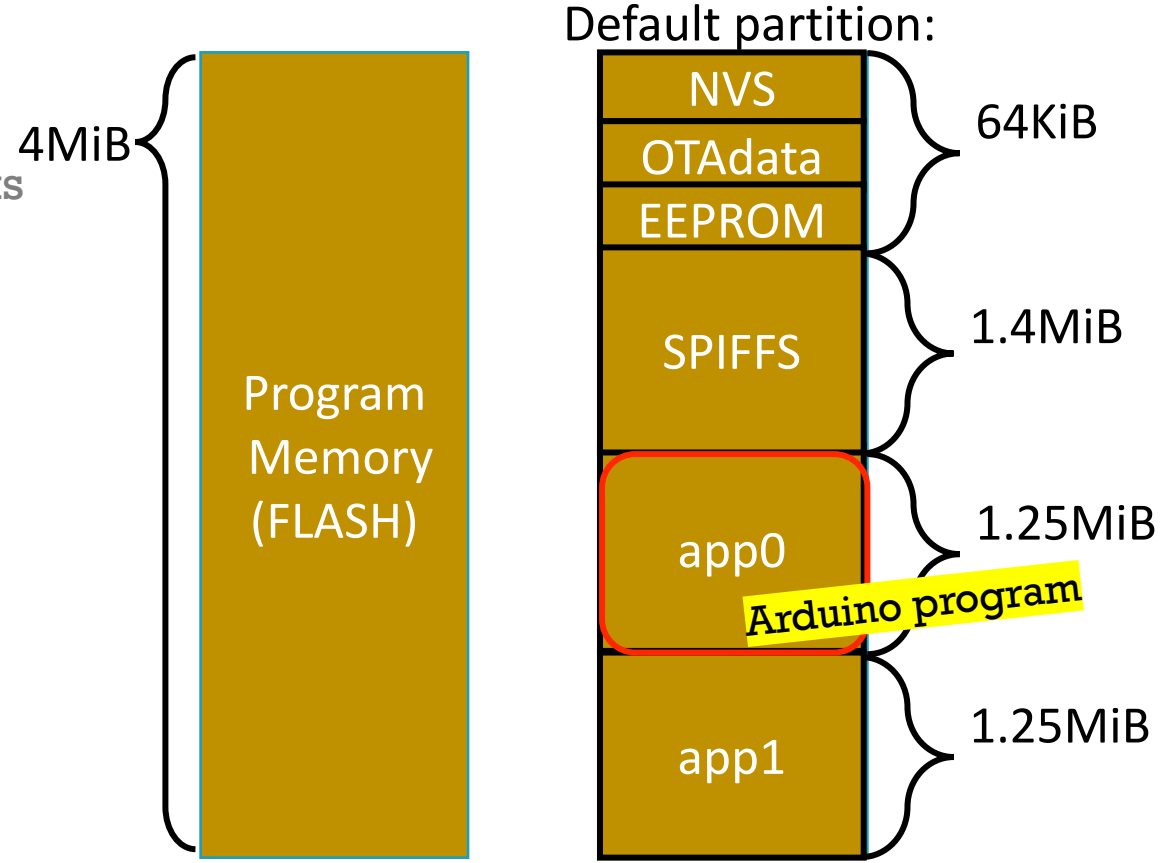
- 512KiB SRAM
  - 192 KiB for IRAM
    - Fast DP Instruction
  - 320 KiB for DRAM
    - RAM for Data



Max globals set by memory segments

ESP32 Memory (Program Memory (FLASH))

- 4MiB Flash Program
  - Partitioned into chunks
  - 1.25 MiB allocated for Arduino App
- 512KiB SRAM
  - 192 KiB for IRAM
    - Fast DP Instruction
  - 320 KiB for DRAM
    - RAM for Data



Dynamic Memory usage on ESP32

```
uint8_t s[112928];
void setup() { s[1]=1; }
void loop() { }
```

- Sample program uses most of dynamic memory.
  - Does nothing but allocate one big array.

```
Global variables use 124576 bytes (38%) of dynamic memory, leaving
203104 bytes for local variables. Maximum is 327680 bytes.
```

  - 112928 bytes of data records is fine. (**maximum total global memory**)
  - 112929 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
```

Q3) How can we use more of the 320K dynamic memory?

Typical Program Sizes on ESP32

- 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

For reference Teensy maximum file size 32K, typically 5-10K

- Typical MEAM510 program will be about 700k

Partition table can reallocate program memory

- 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

4,147,440

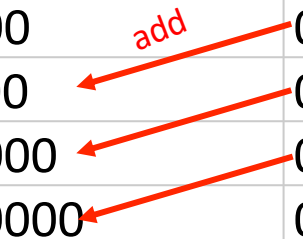
Changing FLASH memory allocation

- 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);
}
```

ESP32 Memory Summary

- Uses of memory:
  - PROGRAM memory
  - DYNAMIC memory
    - Allocated at compile time: IRAM (fixed), globals
    - Determined at run time: Stack and Heap
- Arduino says 320K dynamic memory, but globals actually limited to ~110k
  - There are ways to use more (e.g., malloc)
- ESP32 lists 4M program memory, but Arduino only makes available 1.25M
  - There are ways to use more (repartition)

03

Using internet DIY projects (Debugging Audio Example)

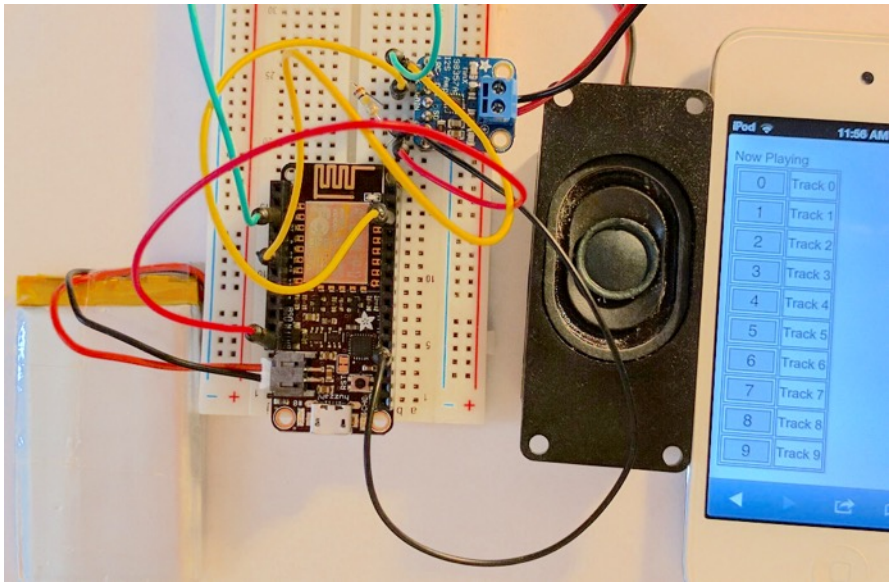
Navigating the Arduino Free Software World

1. You find some site that has software – download – test
2. Try to modify so it works for your case.
3. Find library that code relies on.

   http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use
4. Search for Library source code. (typically github)
5. Debug the mistakes from public domain software.
6. Learn and use new system.

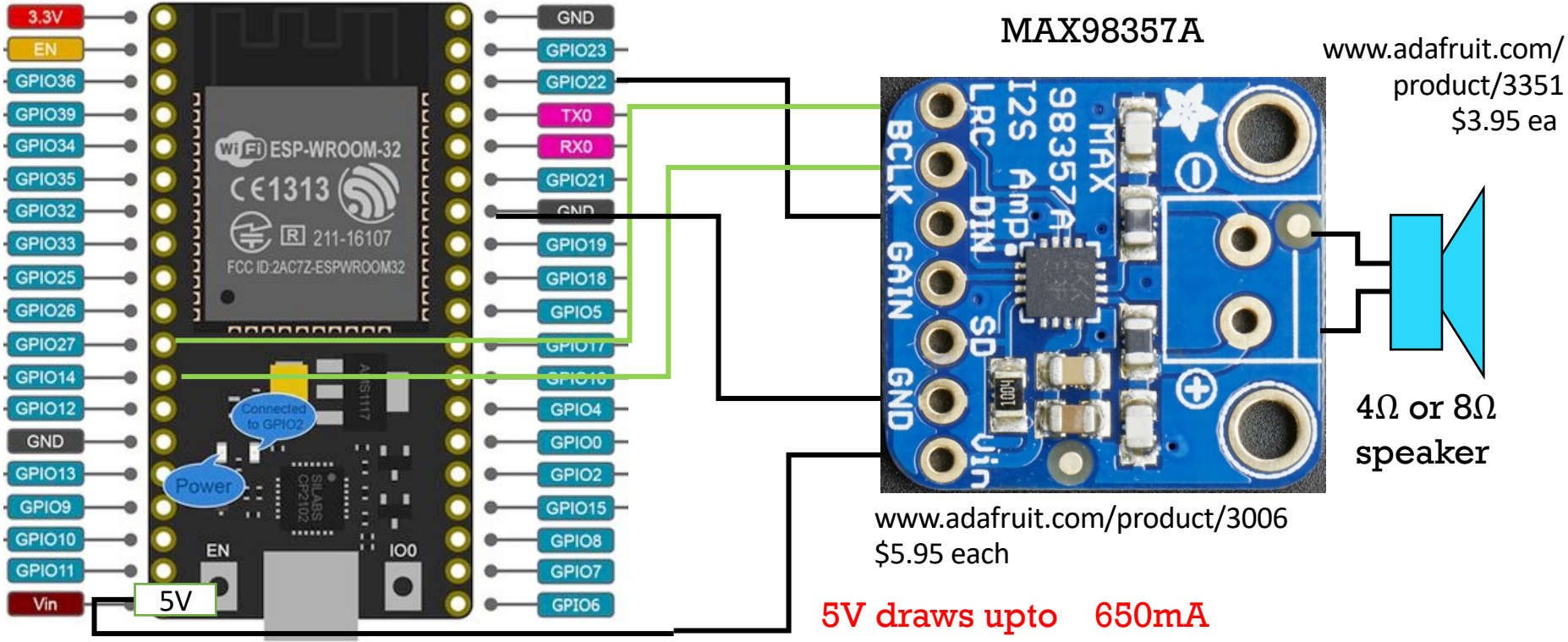
Arduino I2S Example Scenario (from 2018)

- Google I2S ESP8266
  - https://github.com/bbx10/SFX-I2S-web-trigger
  - I2S from SPIFFS file plus web interface
- Set up and run
  - Works!
- Change to fit
  - Remove web stuff
  - Upload different sound files
- Result
  - Crashes sometimes
  - Doesn't play some sound files



I2S on Arduino on ESP32

Arduino file on canvas -> files -> resources -> ESP32-i2sSounds



Arduino I2S Example Scenario: Bug #1

- Problem: crashes
  - Play same 3 files over and over (in loop())
  - After 10 plays it crashes. (every time).
- Google I2S library
  - github.com/esp8266/Arduino/blob/master/cores/esp8266/core_esp8266_i2s.c
- Examination of I2S C code shows `i2s.begin()` uses `malloc()` and `i2s.end()` uses `free()`.
- The Arduino code calls begin() and end() for every song file.
  - malloc() and free() tend to fragment memory and overtime things crash.
  - Solution: call begin() once in setup() instead of every song.

For embedded code – avoid repeated calls to malloc() and free(). Usually, you know how much space you need. Pre-allocate in a global if it fits, or call malloc() once.

Arduino I2S Example Scenario: Bug #2

- Problem: sometimes sound crackles or drops
  - Especially if moving wires.
- The I2S runs at high frequencies.
- The sound amp takes high currents (~700mA)
- Push on individual pins to see which is the problem.

Q4) What could be the issue?

- Solution: use better wire connections. Resolder pins.

Sometimes hardware is the problem – but usually suspect software first.

In general, software is consistent (except things like interrupts). If you have an intermittent problem, suspect a hardware issue.

Arduino I2S Example Scenario: Bug #3

- Problem: crashes
  - Simplifying cleaning up code after development…
  - Commenting out Serial.println() causes a crash.
- Must be a timing issue.
- Program writes bytes (e.g. a sound file) to a DMA (direct memory access) subsystem in big chunks. DMA writes those bytes one at a time to I2S port without processor babysitting.
- Problem DMA buffer was not clearing in time. Didn't think about wav_loop() calls happening quickly in succession.
- Solution: use `i2s_is_full()` routine instead of custom flags.

If removing println() makes something crash – suspect a timing issue.

04

Debugging Process
(via a failure example)

Standard Debugging process

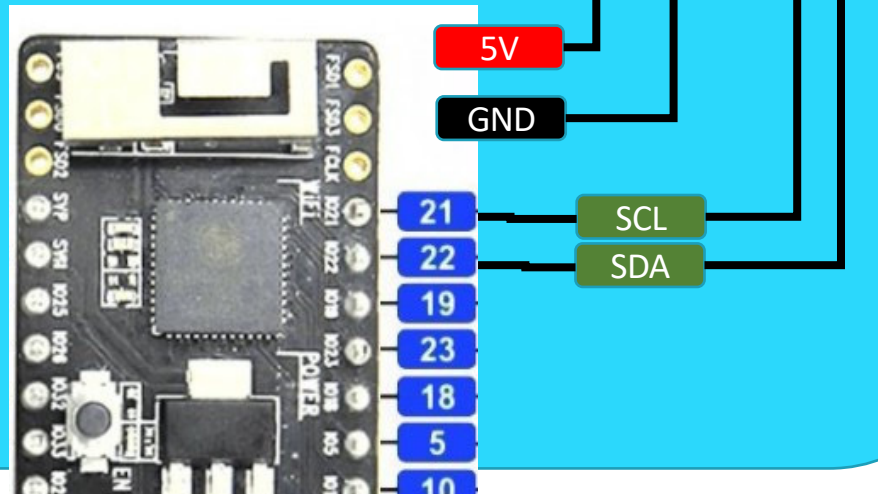
1. Start with most reliable:
    1. Check power and ground, trace power to and ground to your system.
2. Hardware components
    1. Check the voltage level of each output individually.
    2. Check inputs have proper effect (isolate and manually control input lines).
3. Software components
    1. Write short routines that you know work to check individual functions
4. Combine individual components one at a time until you isolate failure.

Last semester Lecture on Wiring

Time of Flight to I2S wiring

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

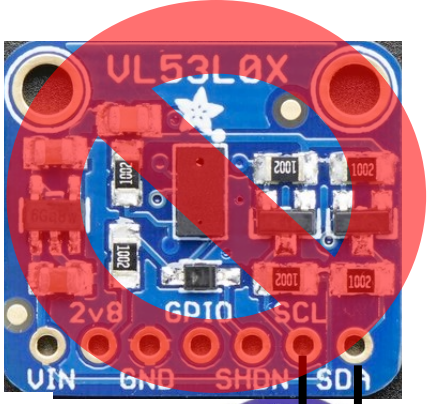
- ESP32 default **SCL** is **GIPO21**
- ESP32 default **SDA** is **GPIO22**
- Attach SDA to SDA
- Attach SCL to SCL



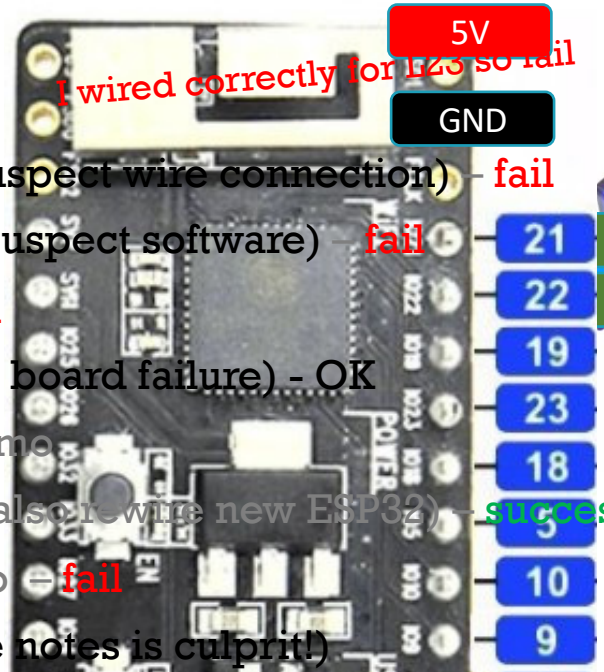
ToF Scanning Demo Development

I miss-wired here so it works

- Test VL53L0X using I2C and sample code on ESP32 - success
- Write slides for L23 L23 wiring is WRONG
- Add servo and scanning for this lecture
- Mount VL53L0X to servo and test - success
- Servo motion pulls wires out – rewire – fail
- Test wiring (recheck wiring with slides) (Suspect wire connection) – fail
- Test with previous VL53L0X sample code (Suspect software) – fail
- Test new ESP32 (Suspect ESP32 ports) – fail
- Check VL53L0X voltages (Suspect VL53L0X board failure) - OK
- Switch to ultrasonic sensor for scanning demo
- Try new I2C address (based on datasheet, also rewire new ESP32) – success!???
- Move back to old ESP32 connected to servo – fail
- Realize error in wiring TWICE! (Bad lecture notes is culprit!)



I wired correctly for L23 so fail

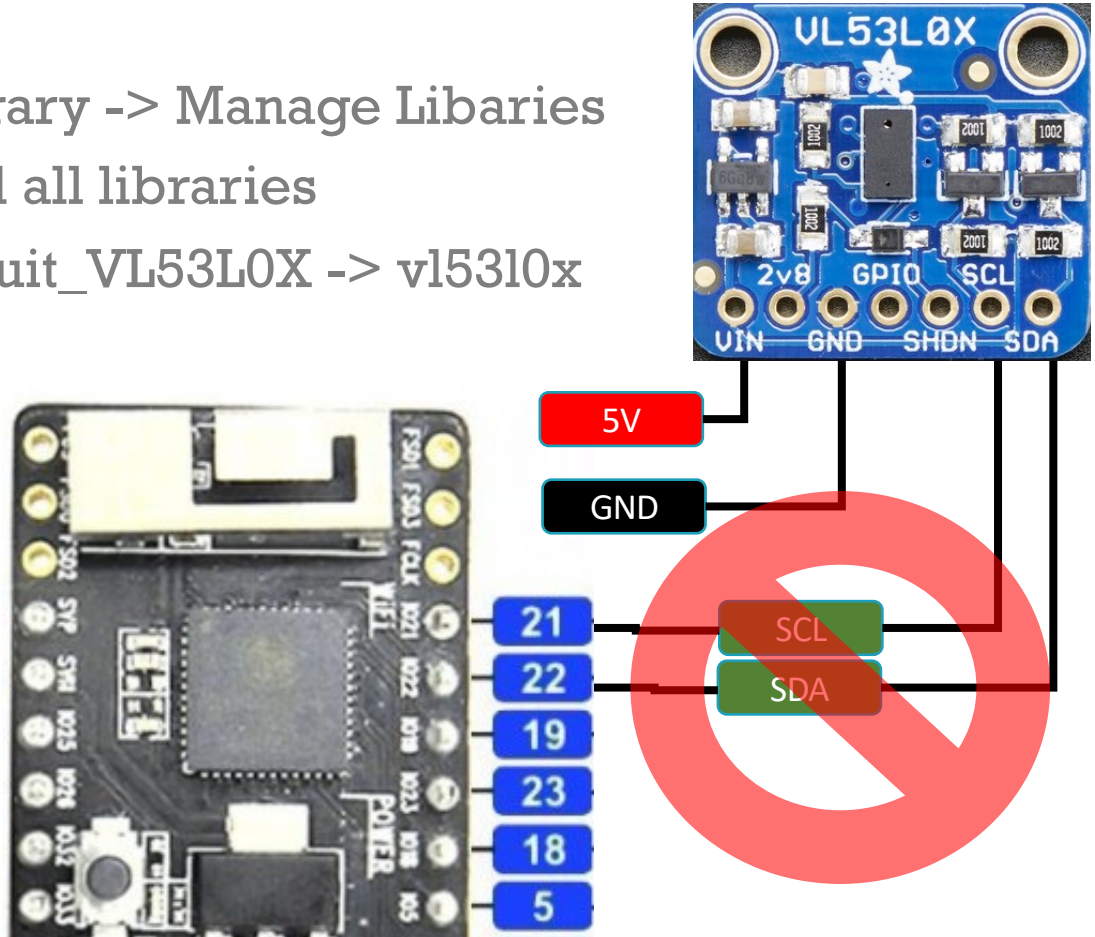


I miss- wire again!

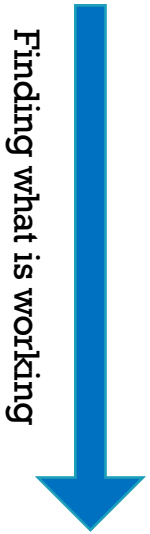
Slide on Wiring WRONG!

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

- ESP32 default SCL is ~~GIPO21~~ GPIO22
- ESP32 default SDA is ~~GPIO22~~ GPIO21
- Attach SDA to SDA
- Attach SCL to SCL



Order of trust (correct and working properly)



1. Corporate documentation
2. Test equipment:
   - E.g. DMM, Power Supply, Oscilloscope
3. Passive parts
   - Resistors, caps, wires
4. Simple active parts
   - LEDs, diodes, voltage regulators
5. Oscillosorta
6. Lecture documentation
7. Web documentation

90% of ERRORS DUE TO THESE:

12. Blown port on MCU
13. External hardware
    - H-bridge, logic, opamps, sensor, motor, driver
14. Wiring
    - breadboard interface, connectors,
15. Software

Debugging Summary

- Build up piece by piece starting with most reliable
- Suspect piece by piece starting with least reliable
- Sample Arduino code is everywhere. Quality is random.
- If you see random crashes for repeated code (without interrupts) suspect a memory leak or fragmentation. Look for overusing malloc() and free().
- If removing println() causes crashes, suspect a timing issue.
- If you have flakey hardware, push on wires to find suspect connections.

05

Extra Coding Tips

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

"teensy_general.h" Header File

```
[…]
#define teensy_led(val)     set(DDRD,6);\
        if(val==OFF){set(PORTD,6);}\
        else if(val==ON){clear(PORTD,6);}\
        else if(val==TOGGLE){toggle(PORTD,6);}
```

Allows macro to continue past the end

Subtle Bug in `teensy_LED`:

Normal use:

```
teensy_LED(ON); -> teensy_LED(1);
teensy_LED(OFF);-> teensy_LED(0);
```

Q5: What would cause this?

Breaks under this case:

```
int condition1 = 1, condition2 = 0;
teensy_LED(condition1 || condition2);
```

Note, this would not happen if `teensy_LED` was a subroutine.

Standard integer types

`int`	**on Teensy is 2 bytes (−32,768 to 32,767)**
`int`	**on ESP32 is 4 bytes (−2,147,483,648 to 2,147,483,647)**
`uint8_t`	1 byte unsigned, (i.e. unsigned char)
`uint16_t`	2 byte unsigned, (i.e. unsigned short int)
`uint32_t`	4 byte unsigned, (i.e. unsigned long int)
`uint64_t`	8 byte unsigned, (i.e. unsigned long long int)
`int8_t`	1 byte signed, (i.e. signed char)
`int16_t`	2 byte signed, (i.e. signed short int)
`int32_t`	4 byte signed, (i.e. signed long int)
`int64_t`	8 byte signed, (i.e. signed long long int)

Variable type conversion

- **Automatic promotion:** If two operands are different, the smaller is promoted to the larger. If operands are the same size but unsigned and signed, then signed is cast as unsigned. Float is bigger than int.

```
int si = -1;
unsigned int ui = 1;
if (    si <(int)ui) printf("true"); // Q6 Does it print true?
```

Mixing signed and unsigned may have unexpected results

```
float x;
short int y = 4;
x =   y/3.0;    // y is promoted to a float, x = 1.33333
x =         y/3;// Q7 What does x equal?
```

Promotion happens at evaluation of operation

https://wiki.sei.cmu.edu/confluence/display/c/INT02-C.+Understand+integer+conversion+rules

Variable type conversion -cont.

- Bitwise operations and promotions to int.

```
uint8_t port = 0x12;   // for reference 0x12 = b00010010
uint8_t result_8 = (uint8_t) (~port) >> 4; // Q8: What is result_8?
```

- Passed parameters are promoted according to prototype

```
hw_timer_t *timer;
float value = 12.0;
timerAlarmWrite(timer,  value, true);
```

Supposed to be an unsigned int 64 bit

- Use google to find code prototypes "esp32 timerAlarm"
- Prototype in esp32-hal-timer.h

```
timerAlarmWrite(hw_timer_t *timer, uint64_t value, bool reload);
```

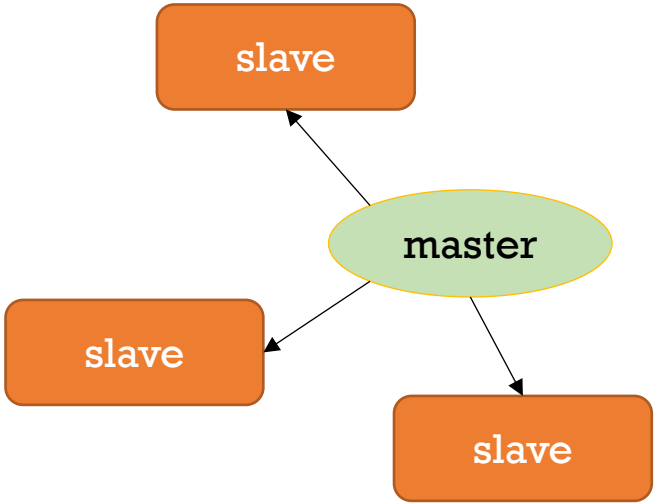
https://wiki.sei.cmu.edu/confluence/display/c/INT02-C.+Understand+integer+conversion+rules
https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-timer.h

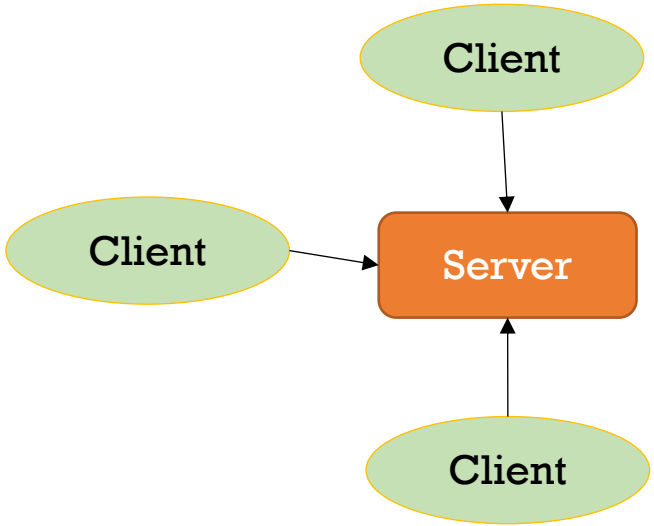
06

Master-slave Client-server Communication

Difference between Master-Slave and Client-Server

- Master and Client, initiate communication.
- Typically master-slave is many-slave to one-master. Client-server is many-client to one-server.



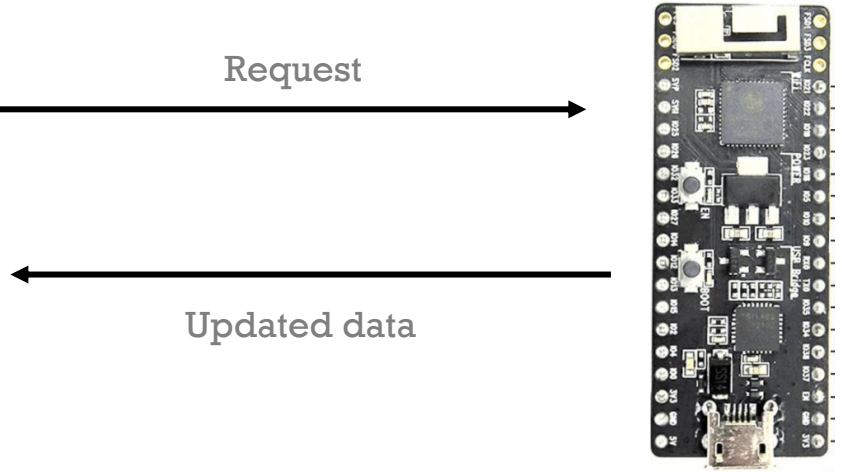


Web sites have a Client-Server Model

Client website
Initiates communication

Server code
Responds to client

Request



Updated data

Cannot send data without request from client

Updating info on website

- How does Oscillosorta update the display data being constantly updated by ESP32?
- AJAX – Asynchronous JavaScript And XML.
  - Read data from a web server - after the page has loaded
  - Update a web page without reloading the page
  - Send data to a web server - in the background
  - https://www.w3schools.com/js/js_ajax_intro.asp
- Two mechanisms
  - Timed calls setTimout(), setInterval()
  - XMLHttpRequest objects

AJAX

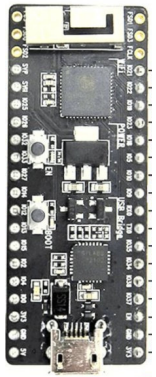
Browser

An event occurs...

- Create an XMLHttpRequest object
- Send HttpRequest

Server

Request for display info



Display info

https://www.w3schools.com/js/js_ajax_intro.asp

AJAX XMLHttp

REQUEST

```
function getData() {
  var xhttp = new XMLHttpRequest();

  xhttp.open("GET", "up", true);
  xhttp.send();
}
```

Handler URL text

RESPONSE

```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200 ) {
    // draw new data in dark green
    ch = this.responseText.split(","); // get ranging data
    ctx.strokeStyle = "#008800";
    drawDataCircles();
  }
};
```

AJAX XMLHttp template

```
function getData() {
  var xhttp = new XMLHttpRequest();

  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200 ) {
      callbackfunction ();
    }
  };

  xhttp.open("GET", URLtext, true);
  xhttp.send();
}
```

Event handler, like onchange or oninput on sliders

Check no errors

There are 4 readystates, you want the last one

AJAX Server Polling

- Use `setInterval()` to periodically request server data

```
setInterval( function, interval_in_ms);
setInterval(getData, 1000);
```

getData called once every 1000ms.

- Use `setTimeout()` inside a called function, timing based on function run time

```
function getData() {
  ...
  ...
  setTimeout(getData, 100);
}
```

getData called 100mS after the function runs

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. Heat effects on some circuits

B. Memory structures

C. Debugging sequence