

# ECE 471 – Embedded Systems

## Lecture 11

Vince Weaver

`http://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

22 September 2017

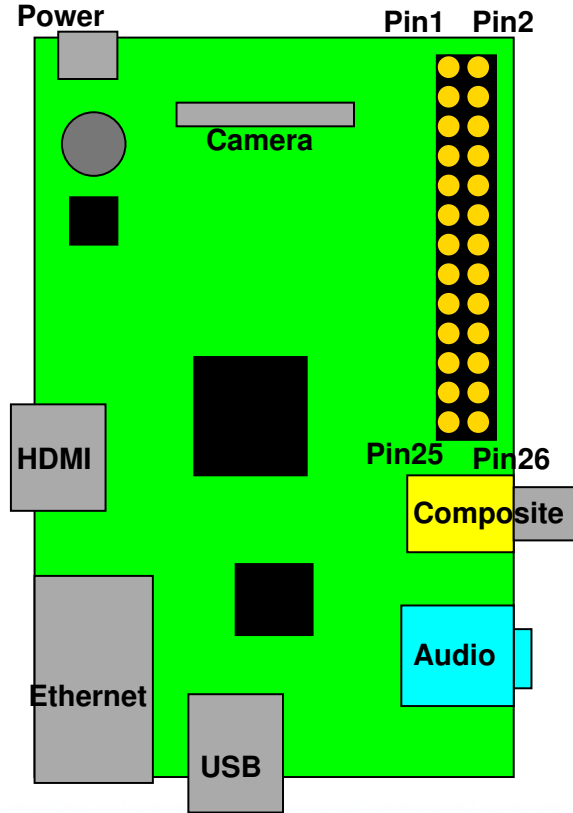
# Announcements

- HW#4 will be posted
- Will require an LED, a breadboard, some resistors and some jumper wires.  
I handed out some GPIO wires in class.

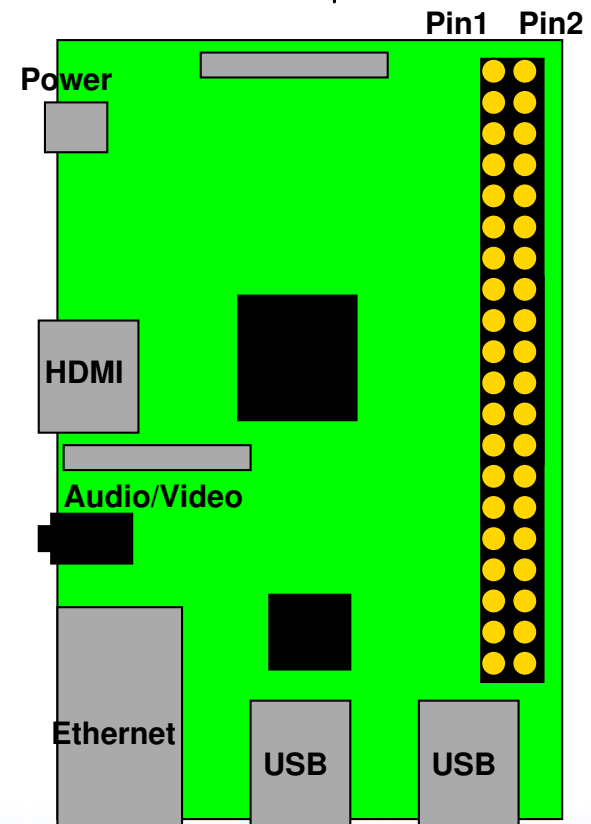


# Brief Overview of the Raspberry Pi Board

## Model B

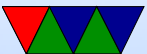


## Model B+



# Rasp-pi Header

- Model B has 17 GPIOs (out of 26 pins), B+ has 9 more (out of 40)
- 3.3V signaling logic. Need level shifter if want 5V or 1.8V
- Linux by default configures some for other purposes (serial, i2c, SPI)



# Rasp-pi Header

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4 (1-wire)	7	8	GPIO14 (UART_TXD)
GND	9	10	GPIO15 (UART_RXD)
GPIO17	11	12	GPIO18 (PCM_CLK)
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10 (MOSI)	19	20	GND
GPIO9 (MISO)	21	22	GPIO25
GPIO11 (SCLK)	23	24	GPIO8 (CE0)
GND	25	26	GPIO7 (CE1)
ID_SD (EEPROM)	27	28	ID_SC (EEPROM)
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21



# How you enable GPIO on STM32L

A lot of read/modify/write instructions to read current register values and then to shift/mask to write out updated bitfields.

- Enable GPIO Clock
- Set output mode for GPIO.
- Set GPIO type.
- Set pin clock speed.
- Set pin pull-up/pull-down
- Set or clear GPIO pin.

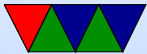


# “Bare Metal” on BCM2835 (Rasp-pi)

- Documented in BCM2835 ARM Peripherals Manual
- 53 GPIOs (not all available on board)
- Can use Wiring-Pi or libbcm2835 if you need speed
- Similar to how done on STM32L... but we have an operating system



# Letting the OS handle it for you





# Linux GPIO interface

- `Documentation/gpio/sysfs.txt`
- sysfs and string based



# A few low-level Linux Coding Instructions



# Enable a GPIO for use

To enable GPIO 17:

write "17" to /sys/class/gpio/export

To disable GPIO 17:

write "17" to /sys/class/gpio/unexport

```
char buffer[10];  
fd=open("/sys/class/gpio/export",O_WRONLY);  
if (fd<0) fprintf(stderr,"\tError enabling\n");  
strcpy(buffer,"17");  
write(fd,buffer,2);  
close(fd);
```



# Set GPIO Direction

To make GPIO 17 an input:

write "in" to `/sys/class/gpio/gpio17/direction`

To make GPIO 17 an output:

write "out" to `/sys/class/gpio/gpio17/direction`

```
fd=open("/sys/class/gpio/gpio17/direction",O_WRONLY);  
if (fd<0) fprintf(stderr,"Error!\n");  
write(fd,"in",2);  
close(fd);
```



# Write GPIO Value

To write value of GPIO 17:

```
write /sys/class/gpio/gpio17/value
```

```
fd=open("/sys/class/gpio/gpio17/value",O_WRONLY);  
if (fd<0) fprintf(stderr,"Error opening!\n");  
write(fd,"1",1);  
close(fd);
```



# Read GPIO Value

To read value of GPIO 17:

```
read /sys/class/gpio/gpio17/value
```

```
char buffer[16];
fd=open("/sys/class/gpio/gpio17/value",O_RDONLY);
if (fd<0) fprintf(stderr,"Error opening!\n");
read(fd,buffer,16);
printf("Read %c from GPIO17\n",buffer[0]);
close(fd);
```

Note: the value you read is ASCII, not an integer.

Also Note, if reading and you do not close after read you will have to rewind using `lseek(fd,0,SEEK_SET);` after your read.



# Delay

- Busy delay (like in 271).  
`for(i=0;i<1000000;i++);`  
Harder to do in C. Why?  
Compiler optimizes away.
- `usleep()` puts process to sleep for a number of microseconds. But can have issues if want exact delay.  
Why? OS potentially context switches every 100ms.
- Other ways to implement: Set up PWM? Timers?



# Using fopen instead?

- Need to `fflush()` after writes (linefeed not enough?)
- Need to `frewind()` after reads?





# C Pitfalls

- Be careful cut and pasting! Especially the size of strings you are sending with `write()`
- Know the difference between `'C'` and `"C"`
- Remember the strings we are reading/writing are ASCII  
`'0'` and `'1'` not integers



# Waiting for Input

- Busy loop. Bad, burns CPU / power
- `usleep()` in loop. Can delay response time.
- Interrupt when ready! `poll()`



# GPIO Interrupts on Linux

May need a recent version of Raspbian.

First write "rising", "falling", or "both" to  
`/sys/class/gpio/gpio17/edge`.

Then open and poll `/sys/class/gpio/gpio17/value`.

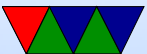
```
struct pollfd fds;
int result;

fd=open("/sys/class/gpio/gpio18/value",O_RDONLY);
fds.fd=fd;
fds.events=POLLPRI|POLLERR;
while(1) {
    result=poll(&fds,1, -1);
    if (result<0) printf("Error!\n");
    lseek(fd,0,SEEK_SET);
    read(fd,buffer,1); }
```



# Debouncing!

- Pull-up / Pull-down resistor. Why?
- Why the extra 1k resistor? (avoid short if set to output by accident)
- Noisy switches, have to debounce
- Manual, no built-in debounce like on STM32L



# Circuit Discussion

