ECE 271 – Microcomputer Architecture and Applications Lecture 2

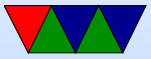
Vince Weaver

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

vincent.weaver@maine.edu

24 January 2019

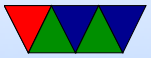
Preparing for the Lab



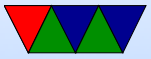
Bare-Metal vs Operating System

- Usually an operating system abstracts away direct device access. Why?
- For security, for safety (programs accidentally touch memory they shouldn't)

```
printf("Hello world\n");
c-library
write(1,"Hello world\n",12);
syscall()
operating system
looks up stdout
    network->network stack
    graphics->graphics stack
    serial port -> serial
        loop over buffer, output byte
        write byte to a mmap register FIFO in I/O space
```

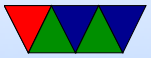


was earlier set up much as we are doing in class

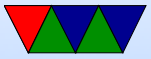


General Purpose Input/Output (GPIO)

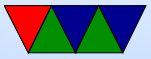
- Read textbook Chapter 14
- GPIO pins can be configured 4 ways
  - Digital input (0 or 1 based on some threshold)
    What threshold?
  - Digital output (0 or Vdd)
    How much current can you drive? What happens if you short it?
  - Analog functions
    DAC (digital-analog-conversion), ADC



- Alternate Functions:
  PWM, LCD driver, timer, USART, SPI, I2C, USB
- Why alternate functions? "only" 100 pins on chip, can have more functions than that so you can map them to the pins (but it does have limitations, might not be possible to have all combinations of supported I/O at once)



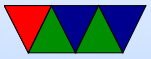
GPIOs on STM32L4



GPIO Inputs

- When configured input, appears as high-impedance (floating) to outside
- When left floating, what value is there if you read it with CPU?

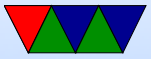
  Mention Apple II 6502 "floating bus" hack
- How can you force it to have a value when floating?
- Pull-up/Pull-down resistors
  - Will pull to high or low value.
  - When external voltage applied, it can overpower these.



- With a pull-up/down resistor, will have current flow. This wastes power, so often they have a relatively high value.
- These are "weak" pull-ups that can be overpowered Don't want to use them in high-speed cases. Why? RC time constant?

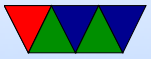
- Can use "strong" external resistors if really need pullups.

- Schmitt-trigger – to give cleaner signal on input
  Include plot like Textbook Figure 14-4?



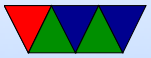
GPIO Outputs

- Push-pull vs open-drain.
  Include diagram?
- Open drain. Wired-AND / Wired-or
- Why low-slew bad? Fourier series, lots of harmonics?



Memory-mapped I/O

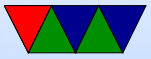
- Some CPUs have specific I/O instructions that have an I/O port range and a special instruction to output to it
- Most CPUs these days do I/O via memory-mapped I/O
- A range of address space in the memory range is interpreted sort of as registers, and these are used to program the I/O
  Use standard load/store instructions to access it



Memory-mapped I/O on Cortex-M

- The ARM Cortex-M4 board has mmio registers
- Peripheral MMIO Starts at `0x40000000`
- Mention physiscal vs virtual memory

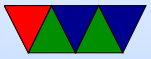
0xe000.0000	(3.75GB)	ARM internal peripherals
0x4000.0000	(1GB)	peripherals
0x2000.0000	(512MB) 96k	primary SRAM
0x1000.0000	(256MB) 32k	part of SRAM
0x0800.0000	(128MB) 1MB	Flash (stack ptr and vectors at bottom)



Memory-mapped I/O in C

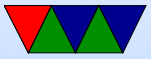
- If you want to read or write to address `0x4800.0414` can you do this?
  Could you do this in Java or Python? why not
  Could you do this in assembly language?
  Great power comes great responsibility. This is fun part of computing.
- C has pointers. Powerful. Mysterious. Easy to get wrong.
-

```
uint32_t value;
```



```
uint32_t *pointer;
pointer= &value;     // pointer now has address of value
*pointer=42;         // dereferece this address and store to it
```

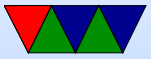
- Can we just set `pointer=0x48000414; *pointer=whatev`
- Yes... but modern C compilers make this difficult. If C compiler sees you write to memory but not use the result it might optimize it away. *p=1; *p=2; *p=3; Might see this and say that's pointless, let's just set p to 3, gives right final result. But if points to I/O space it might be important that all 3 get set.
- Big hack is the `volatile` keyword, tells C compiler that what is being pointed to can chance so you should



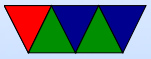
always read or write from it even if it looks like it doesn't matter.

```
#define GPIO_BASE   0x48000000
volatile uint32_t *gpio;
gpio = (uint32_t *)GPIO_BASE;

gpio[2]=0xdeadbeef;
```

- Textbook recommends making a volatile struct and using that. Not sure I like that. Forcing "packed structs" depends on C version and can not work in some case due to padding. Also read-modify-write issues with 16-bit subfields?
- I like inline assembly

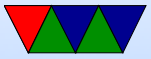


```
static inline void mmio_write(uint32_t address, uint32_t data) {
        uint32_t *ptr = (uint32_t *)address;
        asm volatile("str␣%[data],␣[%[address]]" :
                : [address]"r"(ptr), [data]"r"(data));
}

static inline uint32_t mmio_read(uint32_t address) {
        uint32_t *ptr = (uint32_t *)address;
        uint32_t data;
        asm volatile("ldr␣%[data],␣[%[address]]" :
                [data]"=r"(data) : [address]"r"(ptr));
        return data;
}
```



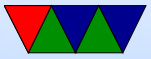
Setting bits in STM32L4 Registers

- To enable GPIO we will have to write some registers
- We will need to set or clear some bits in these registers
- What registers do we set?
  - See documentation (1800 pages!)
  - Can you always trust documentation?



Bit-manipulation in C

- Setting a bit: OR
- Clearing a bit: AND with MASK
- Toggling a bit: XOR
- Read-modify-write and issues therein



Setting register values

- read-modify write
- In C:

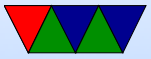
```
a=memory[x];
a=a|0x1;
memory[x]=a;
```

or

```
memory[x]|=0x1;
```

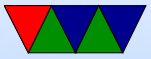
- Use of magic constants!
- Getting bit in right place.

```
0x10 or 1<<4
1UL<<4
```



Provided Code

- Setting up clock (clocks in general)
- Setting up stack/vectors
- Copying memory to RAM (if needed)



Debouncing

- What is it
- Why needed
- Hardware
- Software