# Control systems and Computer Networks

LEDs and Switches

Dr Alun Moon

Lecture 1

## Memory mapped IO

- Access to hardware is via read/writes to addresses
- Easier to build
- easier instruction set

## ARM

- ▶ IO is via read/write to 32bit registers
- ▶ alias region
  - read and write to each 32bit word
  - reads and writes to each bit in the IO registers

## Complex declarations

The complete declaration for a memory mapped I/O register is something like.

```
volatile uint32_t *const IOmap;
```

Which reads as... IOmap is a Constant Pointer to an unsigned 32 bit integer which is Volatile

**Constant pointer** the value of the pointer (address) is constant

   **Volatile** tells the compiler that the value at the address may change, so always fetch the value from memory.

## C arrays and pointers

Arrays and pointers in C have a close relationship;

### Arrays

```c
int modes[12]; /* array of 12 integers */
modes[5];      /* 5th element (count from 0) */
```

### Pointers

```c
int *data; /* pointer to an integer */
*data = 5; /* write to address */
data+1;    /* pointer to the next integer */
```

### Arrays and Pointers

```c
data = modes; /* array name is a pointer */
data[6] = modes[5]; /* pointers as arrays */
```

## Memory

We can model the memory as an array of bytes

```
uint8_t memory[SIZE];
```

The ARM is a 32bit architecture and so it may be more convenient to model the memory as an array of 32bit words

```
uint32_t wordmemory[SIZE/4];
```

## Pointer Arithmetic

Given

```
uint32_t *word;
```

Then

| | |
|---|---|
| word | is an address aligned to 4 bytes |
| *word | is an unsigned 32 bit integer at that address |
| word+1 | is the address of the *next* integer, 4 bytes on |
| *(word+1) | is the next 32 bit integer |
| word[0] | is the integer at the address in word |
| word[1] | is the next integer (at word+1) |

Pointer

arithmetic (and arrays) take into account the *size* of the thing pointed to.

### See also

the **sizeof** compile time operator

# Part I

# GPIO

## Port IO

Each port has

**Data out** sets the output

**Set** writing 1 sets the output (sets to 1)

**Clear** writing 1 clears the output (sets to 0)

**Toggle** writing 1 changes the output

**Input** reads the input

**Direction** set the pin as output or input

## Port Addresses

| Port | Base address | register | offset | action |
|------|--------------|----------|--------|--------|
| Port A | 0x400FF000 | Data out | 0x00 | sets bits to 0 or 1 |
| | | Set | 0x04 | 1 set bit, |
| | | | | 0 leaves bit unchanged |
| | | Clear | 0x08 | 1 clears bit |
| | | | | 0 leaves bit unchanged |
| | | Toggle | 0x0C | 1 toggles bit |
| | | | | 0 leaves bit unchanged |
| | | Input | 0x10 | reads bit state |
| | | Direction | 0x14 | 1 is output, 0 is input |
| Port B | 0x400FF040 | | | |
| Port C | 0x400FF080 | | | |
| Port D | 0x400FF0C0 | | | |
| Port E | 0x400FF100 | | | |

## Endianness

Arrangement of bytes in a multi-byte value (4 bytes in a 32 bit integer)

**Big Endian** Most significant bytes come first in memory

**Little Endian** Least significant bytes come first in memory

The ARM is Little Endian

**For register/32bit integer at address 400FF004**

| Address | byte | bits |
|---------|------|--------|
| 0×400FF004 | 0 | 0 – 7 |
| 0×400FF005 | 1 | 8 – 15 |
| 0×400FF006 | 2 | 16 – 23 |
| 0×400FF007 | 3 | 24 – 31 |

## An initial model

We can have an initial model, thinking of the I/O memory as an array of words.

```
enum registers {
    Output,Set,Clear,Toggle,Input,Direction
};
uint32_t *PortB = (uint32_t*)0x400FF00;
```

The ports registers are now simply array access

```
PortB[Direction] |= 1<<22;
```

Sets bit 22 in Port-B's direction register.