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

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

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 */
```

Layer model

An API

A device driver:

- opens and initialises a device for use
- reads and writes data as appropriate
- closes and shuts down the device

C stdlib

The C library has low level: `open()`, `read()`, and `write()` and higher level `putchar()`, `getchar()`, etc

An LED will have:

- as write
  - turn on
  - turn off
  - toggle (change state)
- as read (not really meaningful)

Major and minor device numbers

Historically Uinx used *major* and *minor* device numbers:

From Unix

Major number is the class of device, and looks up the functions (row in table)

Minor number is the identifier of that particular device

In practice the Major number is used as an index into a table of device drivers, and the minor number is passed as a parameter to the driver.

Example code

```
int read(unsigned int device)
{
    return devtable[major(device)].read(minor(device));
}
```

How to structure Device numbers

- Possible elements are:

  Device LED, Port, Bit, etc

  ID $0 \ldots 1$

  Connection Port and bit numbers

- using groups of bytes,

Major and minor numbers are unsigned 16bit numbers, packed into 32bits.

31 24 23 16	15 8 7 0
Major	Minor

API design and semantics

We have to design the API, it should have the operations

```
open(device, mode);
read(device);
write(device, data);
close(device);
```

We need to decide on data types and semantics

Semantics

Semantics describes the processes a computer follows when executing a program in that specific language.

In our case, how to interpret the values passed as parameters, and how to interpret the value returned by the function.

API design and semantics : BIT

```
bit = open(bitID, 'r')
```

Opens a bit for reading, the direction bit is set for input.

```
bit = open(bitID, 'w')
```

Opens a bit for writing, the direction bit is set for output.

- The returned value can be the index into the table of internal addresses, or a negative number to signify an error.
- The `bitID` signifies which port and bit number to open.
- The open function has to make sure the Port is also open.

API design and semantics : BIT

`r = write(bit, value)`

Writes to a bit, setting it to 1 or 0 as given by `value`.

0	clear the bit
1	set the bit
-1	toggle, change the state of the bit

The `bit` would be the ID returned by `open()`. The return value signifies success or failure.

`r = read(bit)`

Reads from a bit. The `bit` would be the ID returned by `open()`. The return value gives the value of the bit, 0 or 1.

API design and semantics : BIT

```
bit = open(ledID, 'w')
```

Opens an LED.

- The `ledID` signifies which LED to open.
- The driver opens the appropriate bit for writing
- It makes no sense to open an LED for reading!

```
r = write(led, value)
```

Writes to an LED, setting it as given by `value`.

0	turn off the LED
1	s turn on the LED
-1	toggle, change the state of the LED (flashing)

Design of device numbers

Ports and Bits

In the FRDM-K64F often we have to write to a particular bit on a particular port. The Bit-Alias region allows access to individual bits.

Device registers

The FRDM-K64F has registers to: write, set, clear, toggle, and read bits

Semantics of operations

Open	Port	assign clock signal to port, enabling port
	Other	opens the port device is attached to
Write	Bits	0 – clear the bit
		1 – sets the bit
		-1 – toggles the bit

Bit-Alias Address Calculations

Port base	$\texttt{000FF000}_{16}$
Port	$\texttt{000FF000}_{16} + P \times 40_{16} = \texttt{000FF000}_{16} + P \ll 6$
P reg	$\texttt{000FF000}_{16} + P \ll 6 + r \ll 2$
Bit offset	$P_r \times 32 + b \times 4$
	$P_r \ll 5 + b \ll 2$
	$(\texttt{FF000}_{16} + P \ll 6 + r \ll 2) \ll 5 + b \ll 2$
	$\texttt{FF000}_{16} \ll 5 + P \ll 11 + r \ll 7 + b \ll 2$

P 0...4 100

r 0...5 101

b 0...31 11111

31 16	15	13 11	10	9 7	6 2	
1FE	00	Port	0	r	bit	00

- Use Macros in code for readbility
- AWK script to calculate in parallel
- AWK results used to create Unit tests