Microprocessor Block Diagram
The **clock** keeps all parts synchronized.
Memory

Memory is like a tall stack mailtrays. In the case of the mbed, each holds 32 bits.

- Each location is identified by its **address**, a serial number starting from 0.
- The address is said to **point to** a memory location.

When a communication is made with memory to read or write a value, both the address and the data stored in the address must be communicated.

There may be distinct “stacks” for different types of memory.

- A memory location may be called a register. It may just be a set of 32 D-type flip-flops.
Communication with memory

Data is transferred between memory and the rest of the system on buses. These are shared sets of wires that join the components, like a multi-lane highway.

- There are 32 parallel wires on the 32 bit μCon the mbed.

Several sets of these parallel wires are required:

- Address bus – carrier address (serial number) of the “mailtray”
- Data bus – carries the 32 bits either from the memory (read) or to the memory (write).
- Control lines are also needed to synchronise timing, select read/write, ensure that only one device tries to use the bus at once.
Memory Types

Microcontrollers have 2 types of memory

RAM - random access memory
  It is volatile, which means that the contents are lost when the power is removed. e.g., SRAM (static RAM – 6 transistors)

ROM - read only memory.
  It is non-volatile, the contents are retained even when power is removed. Modern ROM is flash memory.

Almost all memory in a PC is RAM. Each program must be read into RAM from non-volatile memory (hard disk) whenever it is needed. Similarly, the operating system is loaded into RAM when the system, is booted.

In contrast, microcontrollers execute only one program, which can be stored in ROM and is therefore available instantly.
Memory Architectures

The two types of memory:
non-volatile ROM for program and volatile RAM for variables

Can be treated in two general ways
First, two completely separate memory systems, each has its own data and address bus. This is the Harvard Architecture.

The second is single memory system, called the von Neumann or Princeton architecture.
The NXP LPC 1768 Microcontroller Block Diagram
Definitions for mbed microprocessor

**Master Clock**
fastest clock on the microprocessor, which keeps everything synchronised - 100 MHz for the mbed

**CPU (Central Processing Unit)**
the heart of the micro – the ARM Cortex-M core in the mbed

**Instruction decoder**
controls the chip, carries out actions for each instruction

**Input/output (I/O) ports**
The types on the mbed are serial, digital, analog, PWM (Pulse Width Modulation).

**Memory**
stores the program (Flash) and data (SRAM)
Input and Output Pins on the mbed
## Key Features of the NXP LPC 1768

### LPC1768 features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| ARM Cortex-M3 core       | • 100 MHz operation  
                          • Nested Vectored Interrupt Controller for fast deterministic interrupts  
                          • Wakeup Interrupt Controller allows automatic wake from any priority interrupt  
                          • Memory Protection Unit  
                          • Four reduced-power modes: sleep, deep sleep, power-down and deep power-down                                                                 |
| Memories                 | • 512 KB of Flash memory  
                          • 64 KB of SRAM                                                                                                                           |
| Serial peripherals       | • 10/100 Ethernet MAC  
                          • USB 2.0 full-speed device/Host/ OTG controller with on-chip PHY  
                          • Four UARTs with fractional baud rate generation, RS-48, modem control, and IrDA  
                          • Two CAN 2.0B controllers  
                          • Three SSP/SPI controllers  
                          • Three I²C-bus interfaces with one supporting Fast Mode Plus (1-Mbit/s data rates)  
                          • I²S interface for digital audio                                                                                                         |
| Analog peripherals       | • 12-bit ADC with eight channels  
                          • 10-bit DAC                                                                                                                             |
| Other peripherals        | • Ultra-low-power (< 1 uA) RTC  
                          • General-purpose DMA controller with eight channels  
                          • Up to 70 GPIO  
                          • Motor control PWM and Quadrature Encoder Interface to support three-phase motors  
                          • Four 32-bit general-purpose timers/counters                                                                                              |
| Package                  | • 100-pin LQFP (14 x 14 x 1.4 mm)                                                                                                         |
The mbed Development Board Pinout
The mbed development board – very flexible

pins 5-30 can be used as digital in and digital out

pins 15-20 can also be configured for analogue input

pin 18 can be configured for analogue output

pins 21-26 can be configured for pulse width modulated output

pins 5-7 and 11-13 can be configured as Serial Peripheral Interfaces (SPI) – a simple serial communication interface used by many microprocessors and microcontroller peripherals. Full duplex (signals can go in both directions simultaneously) can support up to 10 Mbps communication.
The mbed development board – very flexible

Pins 9/10, 13/14 and 27/28 can be configured as serial interfaces.

Pins 33-36 can be configured as an ethernet port

Pins 31,32 can be configured a USB data

Pins 9/10 and 27/28 can also be configured as Inter-Integrated Circuit, which is shortened to I2C (pronounced I-squared-C), and allows communication of data between I2C devices over two wires. It sends information serially using one line for data (SDA) and one for clock (SCL)

Pins 29/30 can be configured as a CAN (controller area network) interface. Originally developed by automotive manufacturers, the CAN bus is designed to allow microcontrollers and devices to communicate with each other within a vehicle without the need for a host computer.
Interfacing

Driving a Light-Emitting Diode

Switching an Output Pin between ‘1’ and ‘0’
## Digital Input and Output

<table>
<thead>
<tr>
<th>0V</th>
<th>3.3V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Closed</td>
</tr>
<tr>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Clear</td>
<td>Set</td>
</tr>
<tr>
<td>logic 0</td>
<td>logic 1</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

### Voltage Levels
- **Logic 1** (3.3 V)
- **Logic 0** (0.0 V)
- **Undefined logic level** (2.3 V, 1.0 V)
Digital Output

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```
Light-Emitting Diode

An LED is a diode, also known as a p-n junction.  
• It has two terminals, an anode and cathode.  
• It conducts current when the voltage measured from the anode to the cathode is positive.

When conducting current, the diode is called ‘forward biased’.  
• When forward biased, it will emit light as long as the temperature is within the specified range and the maximum current is not exceeded.  
  • Read the datasheet to find the allowed temperature and current range.
Remember that LEDs are **diodes**, so we must connect them the correct way round! This can be done in two ways.

1. Suppose that we want the LED to illuminate when the logical value of the output is high (1) — **active high**.

Thus the LED should light when the pin is at $V_{DD}$ (supply, positive, logic 1) but not when the pin is at $V_{SS}$ (ground, negative, logic 0).

We therefore connect the LED between the pin and $V_{SS}$.

Conventional current flows from the pin to $V_{SS}$ with this connection.

The resistor $R$ is to limit the current — we’ll calculate its value shortly.
It might seem obvious that you would always want to connect an LED like this but there is a second, complementary way in which this can be done.

2. Suppose that we want the LED to illuminate when the logical value of the output is low (0) — **active low**.

Thus the LED should light when the pin is at $V_{SS}$ (ground, negative, logic 0) but not when the pin is at $V_{DD}$ (supply, positive, logic 1).

We therefore connect the LED between the pin and $V_{DD}$.

Conventional current flows from $V_{DD}$ to the pin with this connection.

Note that the LED is the other way around!
Remember how the output works for each bit of the port:

logical 0 causes the μC to drive the pin low, to $V_{SS}$, ground, negative

logical 1 causes the μC to drive the pin high, to $V_{DD}$, supply, positive

Both ends of the LED are at $V_{SS}$ (0 V) so no current flows

The anode of the LED is at $V_{DD}$ and the cathode is at a lower voltage.

A forward current can flow in the direction of the arrow so the LED emits light.

(What would happen if the LED was connected the other way around?)
To control brightness or limit current

Add a resistor in series with LED between the mbed and ground. One reason to do this is to increase battery lifetime.

Suppose you would want to limit current through red LED connected to mbed DigitalOut pin to 5 mA. What value of resistor should be used?

Current doesn’t flow through the red LED until the voltage drop from the anode to the cathode reaches what value?

How much more voltage is needed to force a current of 5 mA through the red LED?
Connecting switches to the mbed

Suppose that we want to connect a simple on-off switch or pushbutton as an input to a $\mu$C. How should this be done?

The input should be either:

- $V_{DD}$ (supply, positive) for logic 1
- $V_{SS}$ (ground, negative) for logic 0

This may be the obvious approach:

However, this is not used in practice, for several reasons:

- it needs an expensive switch (3 terminals instead of 2, not just a simple pushbutton!)
- it needs 2 or 3 wires
- what happens while the switch is being operated?
What is actually done in practice

SPST (Single Pole Single Throw) Switch with pull down resistor

Switch open – Vi is connected to ground
Switch closed – Vi is connected to $V_{DD}$

When switch is closed, there is a wasted current flowing through resistor $R$, so resistor is usually large to limit this current.

On the mbed, the resistor is available within the microcontroller.
- The default when a pin is configured as DigitalOut, is the pull-down configuration so the default of an output pin is that it will be at 0 V.
- This can be changed via the “mode” function of the DigitalIn command.
How much current is wasted?

Suppose the supply voltage is 3 V and pullup resistors is 10 kΩ.

How much current is wasted through the pullup when the button is pressed? For comparison, the microcontroller draws about 4 mA in normal operation.

Ohm's law tells us that current, voltage, and resistance are related by:

\[ I = \frac{V}{R} = \frac{3V}{10k\Omega} = 0.3mA \]

The normal operating current of the microcontroller is about 4 mA. So, the pullup draws 7.5%. This sounds small, but will drain a battery while the microprocessor is sitting idle, which is why larger resistors are more commonly used.

Of course, the current flows only while the button is pressed.
Other Impacts

The pull-down resistor also affects the circuit that is connected to the output pin. It contributed to the equivalent impedance of the circuit seen by $V_{DD}$.

What is the difference in current flowing through a resistor that is connected between the output pin and ground with and without the pull-down resistor when:

(a) $R_{out}$ is a 1kΩ resistor?
(b) $R_{out}$ is a 10 kΩ resistor?
(c) $R_{out}$ is a 100 kΩ resistor?