University of Nottingham
UK | CHINA | MALAYSIA

Lecture 1

Introduction: Overview; introduction to the Arduino

Mechatronics
MMME3085

Module Convenor – Abdelkhalick Mohammad

University of Nottingham
UK | CHINA | MALAYSIA

Know Your Teacher!

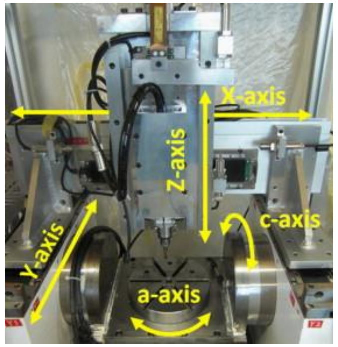
- 1984 — Birth, Egypt
- 2006 — B.Sc. AUN, Egypt
- 2007 — TA, AUN, Egypt
- 2010 — M.Sc. TUT, Japan
- 2013 — Ph.D. TUT, Japan
- 2014 — Postdoc, NTU, Singapore
- 2017 — Assist. Prof. AUN, Egypt
- 2018 — Postdoc, RR-UTC, UK
- 2020 — Assist. Prof. UoN, UK
- 2023 — Associate Prof. UoN, UK

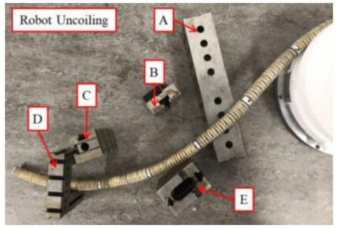
豊橋技術科学大学
Toyohashi University of Technology

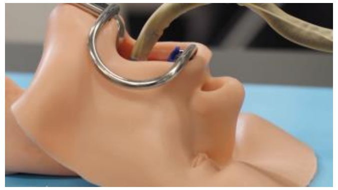
NANYANG TECHNOLOGICAL UNIVERSITY

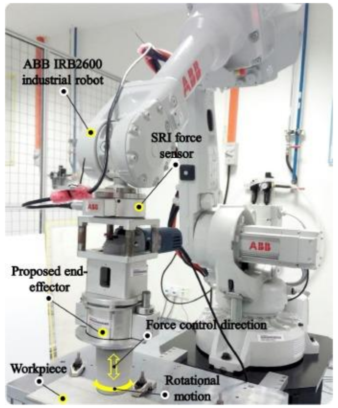
University of Nottingham
UK | CHINA | MALAYSIA

ROLLS ROYCE









Introduction

Module overview

University of Nottingham UK | CHINA | MALAYSIA

Description of the Module

Mechatronics:

Controller: (1) Microcontroller: Computer architecture, digital or analog input & output; timer-counters. (2) Field programmable gate arrays (FPGA) and programmable logic controller (PLC)

Interfacing: Digital-to-analog and analoge-to-digital, serial, parallel communication

Actuators & Sensors: DC & stepper motors, encoders, LVDT, thermocouples, drivers, etc.

Computer engineering:

1. Software design; planning a program
2. Programming in the C language
3. Version control, documentation etc.

Objectives of the Module

- **Understand** and **select** the **hardware** and methods used for **data conversion** and transmission in **mechatronic systems**.
- **Control** the **electronic and electromechanical hardware** (sensors, transducers, actuators and motion control hardware) involved in **interfacing** electromechanical systems to computers.
- To apply the principles of **software engineering** via the use of sound program design, development, version control, documentation and testing
- Attain a reasonable proficiency in using **high level programming languages** to create a **solution** to an **instrumentation** or **mechatronics problem**.
- To have a basic appreciation of **objects** and **classes** with reference to driver objects for specific **interfaces for microcontrollers**

How teach in this Module?

- Convener, **Mechatronics**:
  - Dr Abdelkhalick Mohammad
  - Room B37, Advanced Manufacturing Building

- Co-teacher, **C programming**:
  - Dr Louise Brown,
  - Room C18, Advanced Manufacturing Building

- Co-teacher, **Laboratory**:
  - Dr Surojit Sen
  - Room B90 Coates Building

Module Assessment

- Assessment:
  - 20 credits
  - Final Exam: **55%** (January)
    - Section A (Programming, via software)
    - Section B (Mechatronics, written exam)
  - Laboratory exercises
    - Preparatory programming work: **5%** (Lab 1)
    - Comprehension Quiz Lab 1: **7.5%**
    - Comprehension Quiz Lab 2: **7.5%**
  - Software-based project
    - Software Project preparatory work:**5%**
    - Final Software Project: **20%**

Module Plan

	Week		Assessment		Programming		Mechatronics					
w/c ↓	University	Teaching			Lecture	Lab	Lecture	Seminar	Lab-1	Lab-2	Lab-5	Lab-6
			Room →		*Chemistry C15*	*Coates C19*	*Psychology A1*	*Psychology A1*	*JC AMB C09/10*			
			Time →		*Mon 13-15*	*Tues 11-13*	*Thurs 9-11*	*Fri 13-14*	*Wed 9-11*	*Wed 11-13*	*Fri 14-16*	*Fri 16-18*
25-Sep	1				*No teaching*							
02-Oct	2	*1*			Design Principles C part 1: VSCode and Hello World	Getting started with C	Laying the Foundations	Laying the Foundations				
09-Oct	3	*2*	Lab 1 programming intro (5%)		C part 2: Operators, printf/scanf and conditional statements	C part 1 & 2	Comp architecture; digital signals (parallel); digital i/o;	Comp architecture; digital signals (parallel); digital i/o;			*Collect kit (group-3)*	*Collect kit (group-4)*
16-Oct	4	*3*			C part 3: Loops, arrays and functions	C part 2	Counter-timers; digital signals: serial protocols	Counter-timers; digital signals: serial protocols				
23-Oct	5	*4*		Lab 1 programming submission Thurs 26 Oct (5%)	C part 4: Memory and pointers	C part 3	Sequences, state tables, finite state machines	Sequences, state tables, finite state machines				
30-Oct	6	*5*			C part 5: functions using pointers	C part 4	Analog signals, data acquisition: aliasing, grounding	Analog signals, data acquisition: aliasing, grounding				
06-Nov	7	*6*	Software project prep intro (5%)		C part 6: structures; projects	C part 5	Data conversion including PWM; sensors	Data conversion including PWM; sensors	Lab-1 (group-1)	Lab-1 (group-2)	Lab-1 (group-3)	Lab-1 (group-4)
13-Nov	8	*7*		Lab 1 comprehension quiz Thurs 16th Nov (7.5%)	C part 7: numbers, enums and conditional compilation	C part 7; project	Motion Control: Servo Motors, closing the loop	Motion Control: Servo Motors, closing the loop				
20-Nov	9	*8*		Software project prep submission Tues 21st Nov (5%)	Command line arguments and code optimisation	C part 8; project	Stepper motors; drivers; Bresenham and ramping	Stepper motors; drivers; Bresenham and ramping				
27-Nov	10	*9*			Software best practice	Project	Stepper motor dynamics. Solenoids, pneumatics, hydraulics.	Stepper motor dynamics. Solenoids, pneumatics, hydraulics.	Lab-2 (group-1)	Lab-2 (group-2)	Lab-2 (group-3)	Lab-2 (group-4)
04-Dec	11	*10*		Lab 2 comprehension quiz Thurs 7th Dec (7.5%)		Project	Interrupts and real-time issues; FPGAs	Interrupts and real-time issues; FPGAs	Robot Testing (15 min slots)	Robot Testing (15 min slots)	Robot Testing (15 min slots)	Robot Testing (15 min slots)
11-Dec	12	*11*		Software project submission Thurs 14th Dec (20%)	Consolidation and revision	Project	Consolidation and revision	Consolidation and revision	Robot Testing (15 min slots)	Robot Testing (15 min slots)		
18-Dec	13											
25-Dec	14											
01-Jan	15											
08-Jan	16											
15-Jan	17	Final Exam 55%										
22-Jan	18											

University of Nottingham
UK | CHINA | MALAYSIA

Introduction

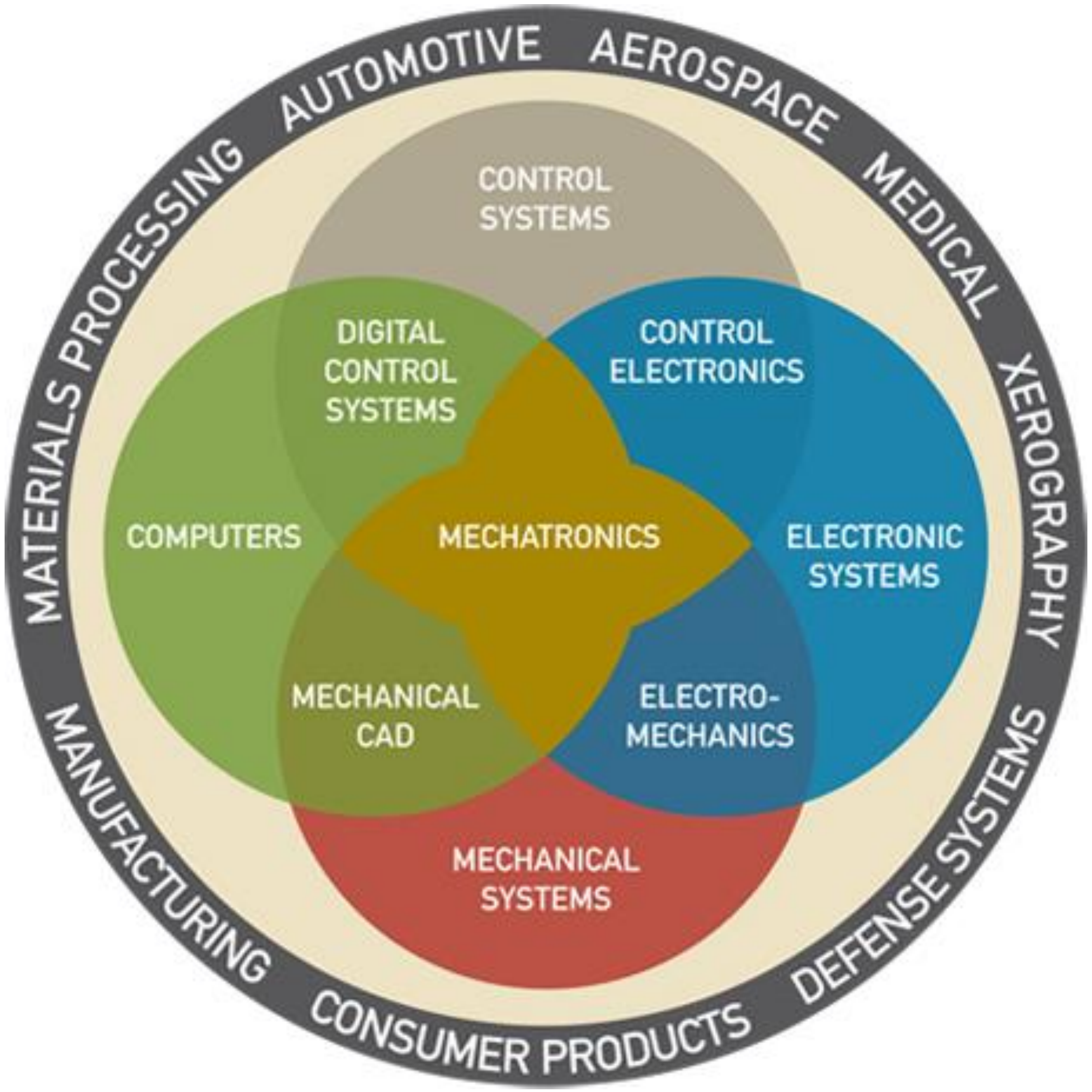
Mechatronics

What is Mechatronics?!

- "Mechatronics is the synergistic combination of precision **mechanical engineering**, **electronic control** and **systems thinking** in the design of products and manufacturing processes. It relates to the design of **systems**, **devices** and **products** aimed at achieving an optimal balance between basic mechanical structure and its overall control". *Journal of Mechatronics*.

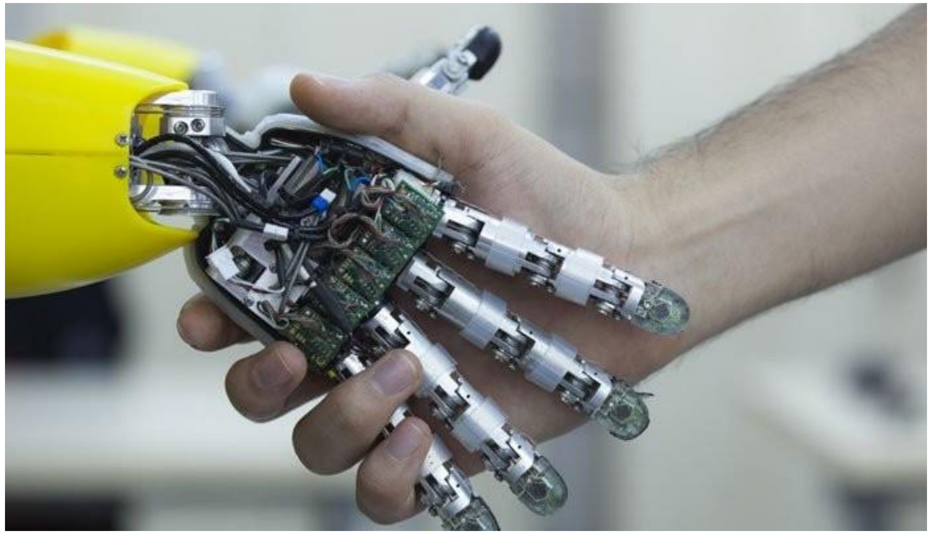
- "Mechatronics is a technology which combines **mechanics** with **electronics** and **information technology** to form both functional interaction and spatial integration in components, modules, products and systems" (Buur, J: A Theoretical Approach to Mechatronics Design. Dissertation, Technical University of Denmark, Lyngby 1990)

What is Mechatronics?!



Want To Become A Mechatronics Engineer?

- To be a Mechatronics Engineer you need to be able to work across the boundaries of constituent disciplines to identify and use the right combination of technologies which will provide the optimum solution to the problem in hand.
- You should also be a good communicator and able to work in and lead a design team which may consist of specialist engineers as well as generalists.



www.howtoabroad.com

www.imeche.org

Mechatronics System Examples

- Computer Numerical Controlled (CNC) Machines
- Robots (e.g., industrial, mobile, soft, human-like, etc)
- 3D printers
- Automatic driving cars/vehicles
- Single-lens Reflex (SLR) digital camera
- Hard drive
- Writing robot or desktop plotter - your job will be to program this!

Mechatronics System Examples

3-Axis CNC Machine

5-Axis CNC Machine

Mechatronics System Examples

Industrial Robots

www.robots.com

3D printer

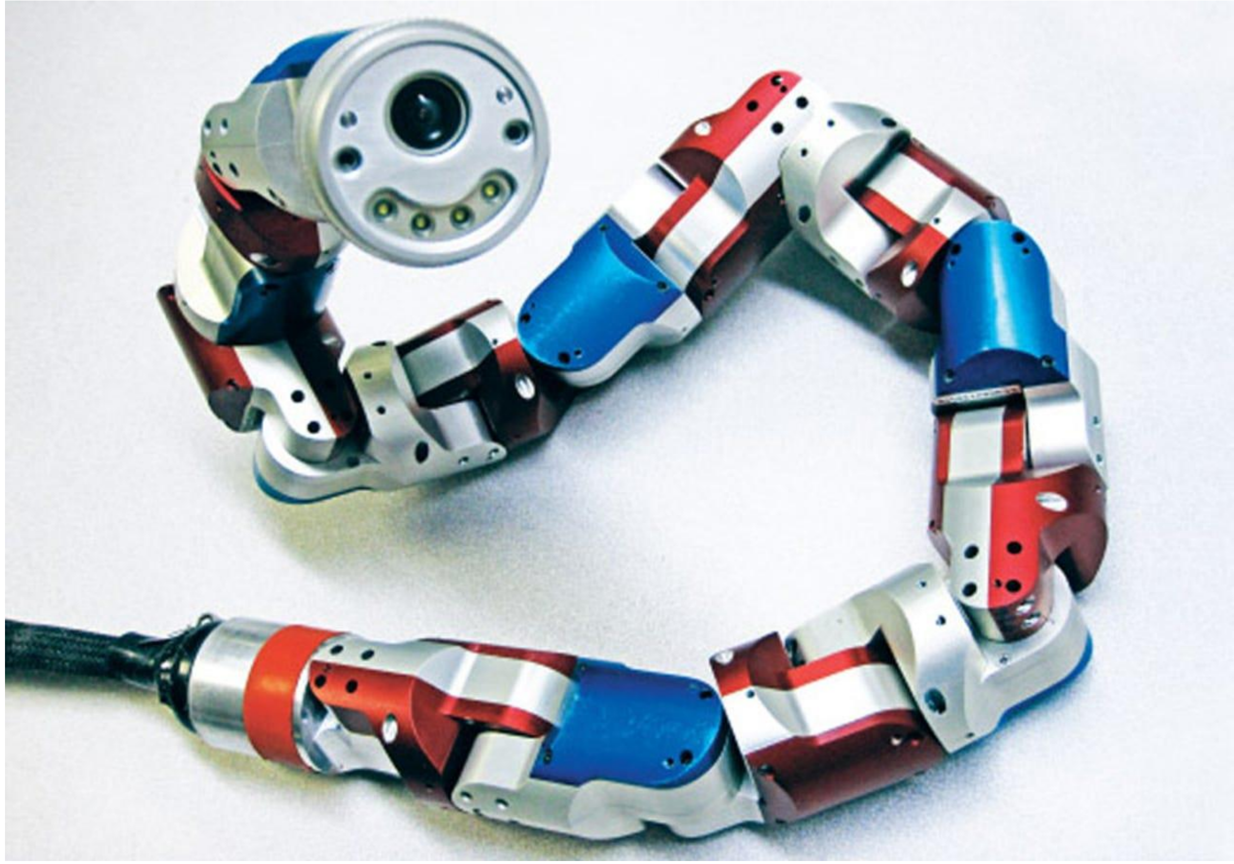
www.matterhackers.com

Mechatronics System Examples

Nature-inspired Robots

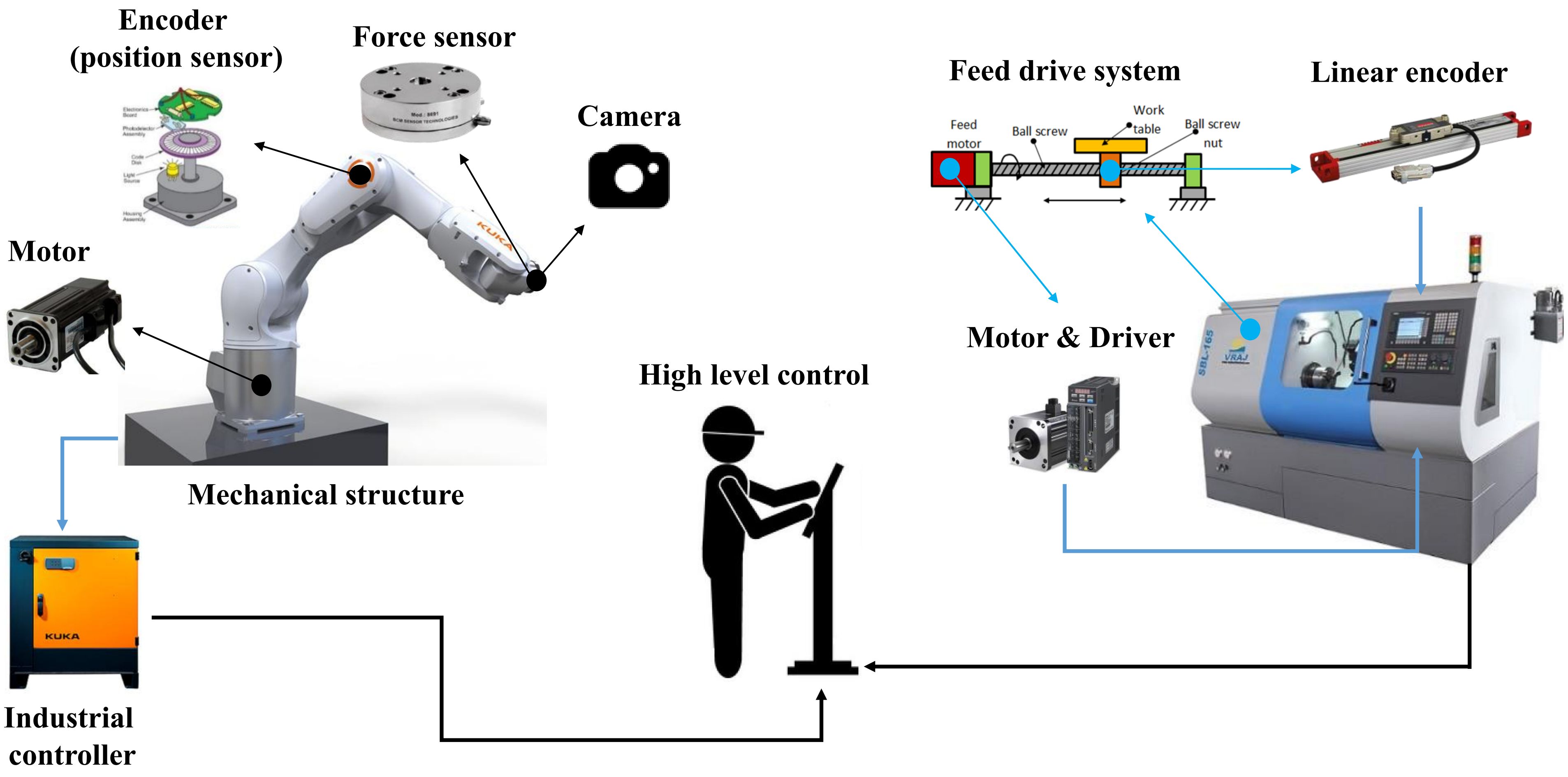
Mechatronics System Examples

Nature-inspired Robots



Mechatronics System Examples

Advanced Manufacturing Systems



Mechatronics System Examples

Desktop Plotter or Writing Robot

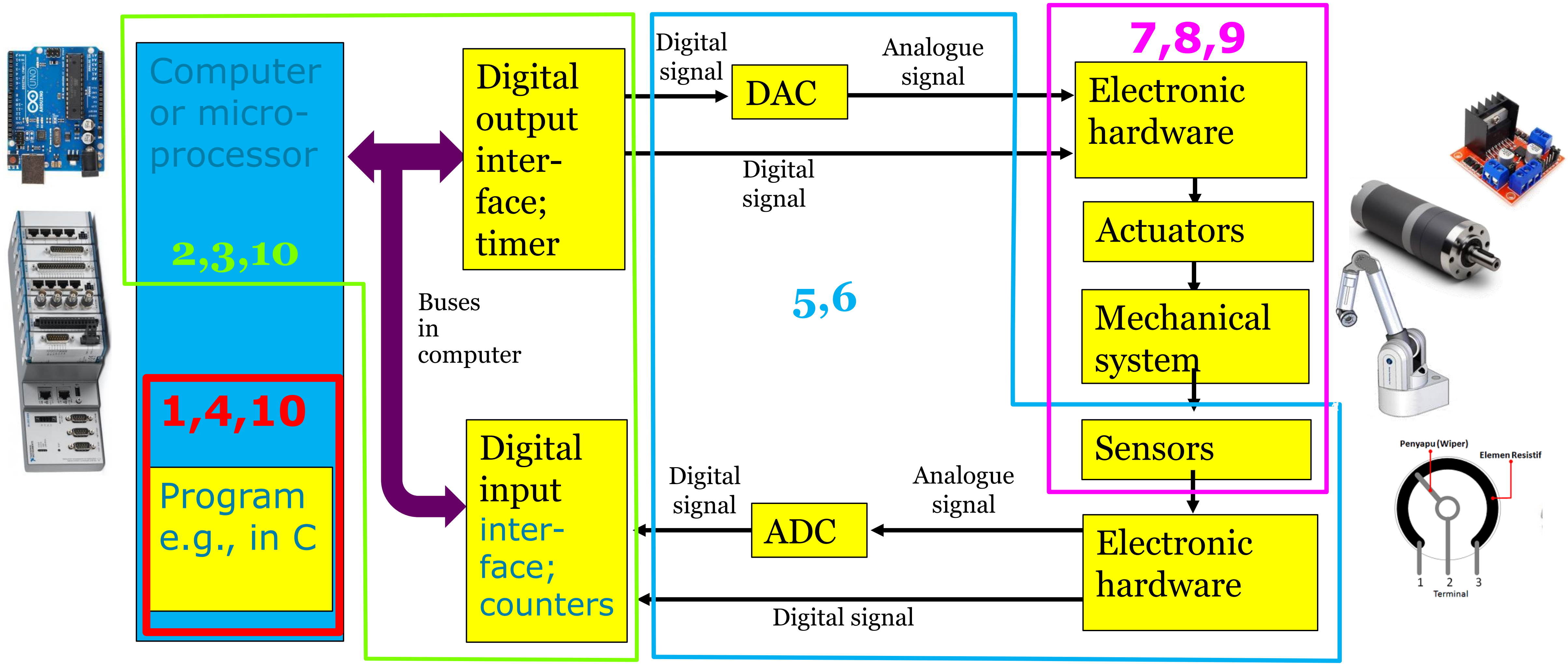
www.tomtop.com

Your job will be to program this!

A Typical Mechatronics System

Basic Elements

A typical Mechatronics System



Revision

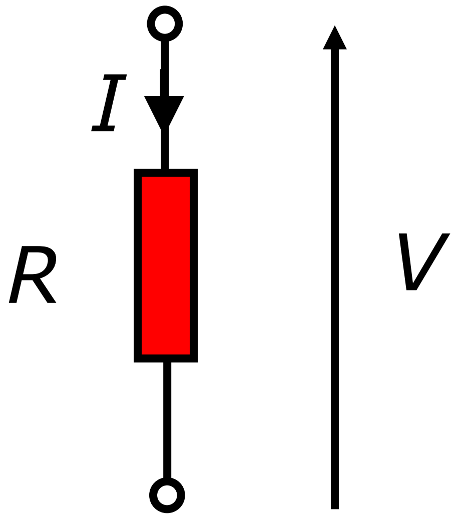
Revision of basic electronics

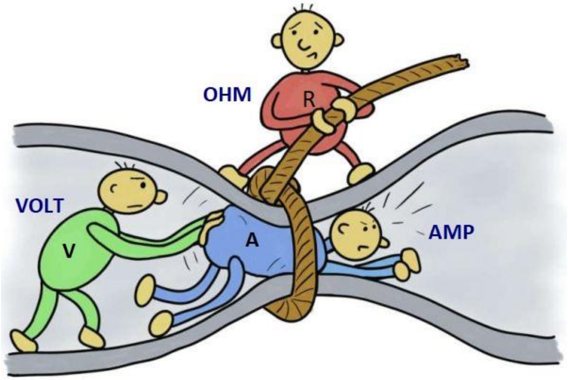
Revision of basic electronics

- **The resistor**: if a current I passes through a resistor of value R, a voltage V will appear across it (Ohms law)
- Similarly, V will cause a current I to flow

$V = IR$

$I = V/R$ etc.

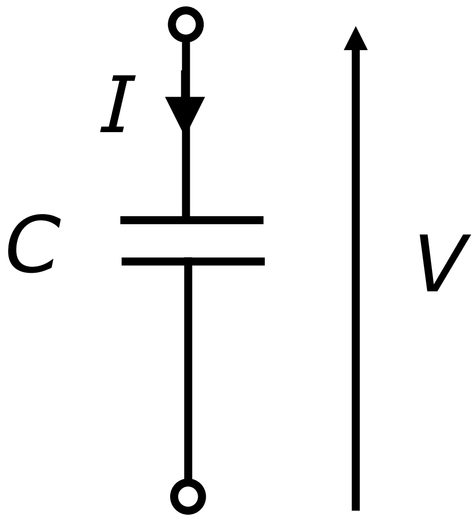




Revision of basic electronics

- **The capacitor:** if a voltage *V is* applied across a capacitor with capacitance *C*, a charge *Q* flows into the capacitor.
- In general: charge = current × time
  (actually, an integral)

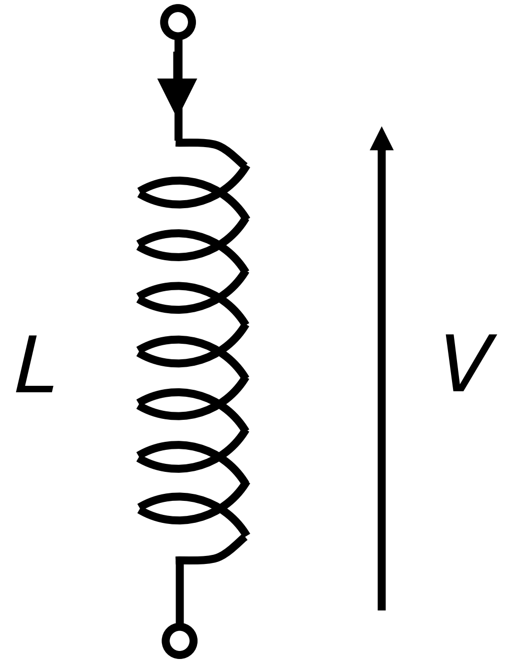
$Q = CV$ or $V = Q/C$
where $Q = \int I \, dt$



Revision of basic electronics

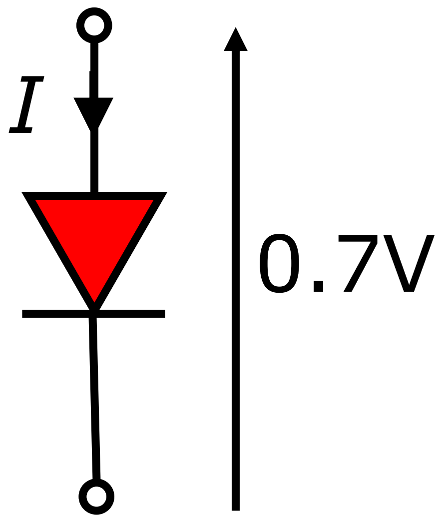
- **The inductor:** if the current through an inductor L changes at rate dI/dt a voltage V will appear across the inductor

$$V = L\,\mathrm{d}I/\mathrm{d}t$$



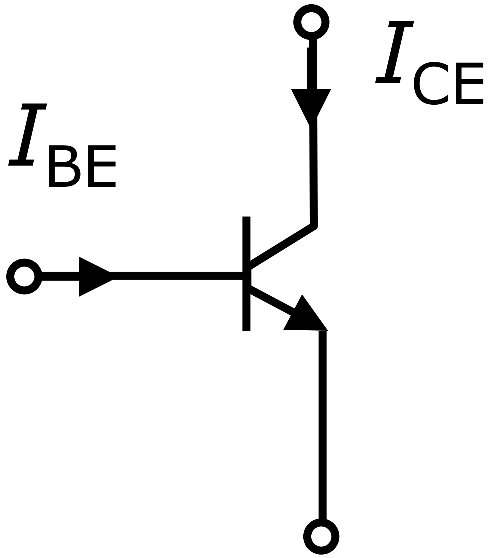
Revision of basic electronics

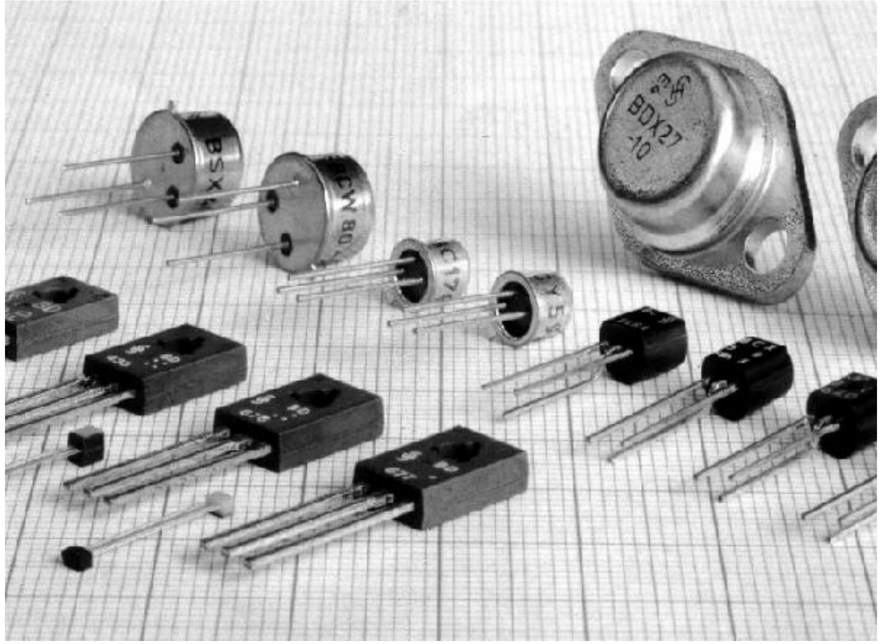
- **The diode:** acts like a one-way valve or check valve (current can flow one way only)
- Not a perfect forward conductor: silicon diode voltage drop typically 0.7 V (for any non-zero current)



Revision of basic electronics

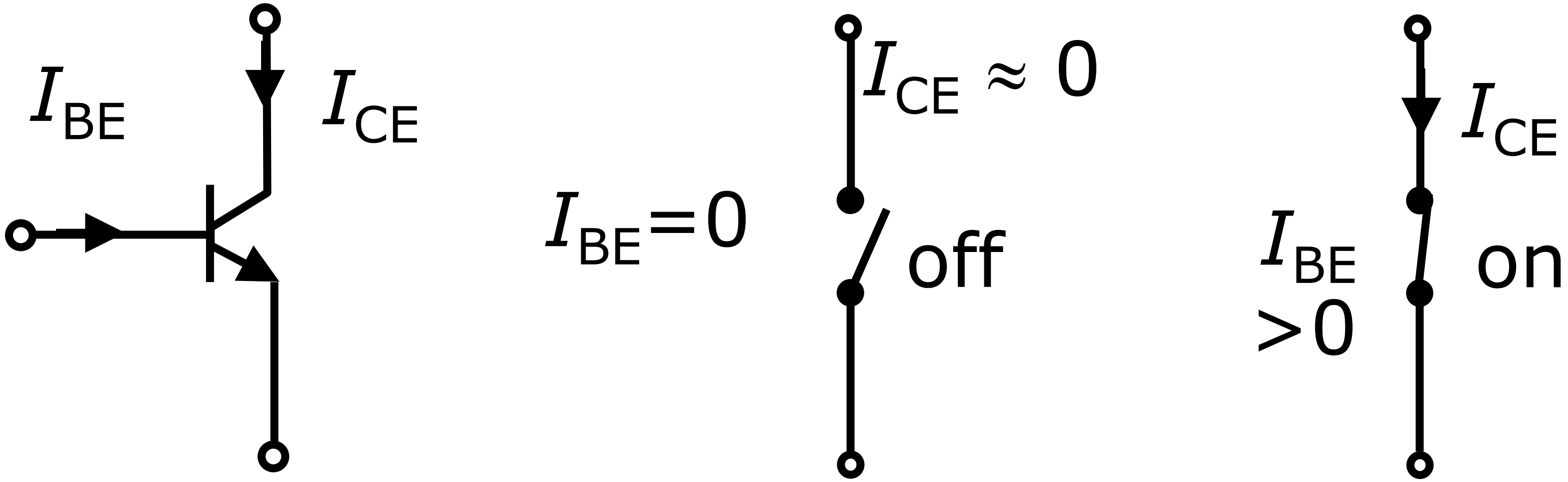
- **The transistor:** a current between base and emitter causes the transistor (with gain h_{FE}) to conduct between collector and emitter





Revision of basic electronics

- For zero I_{BE} , acts like an open (off) switch
- For large I_{BE} , acts like a closed (on) switch

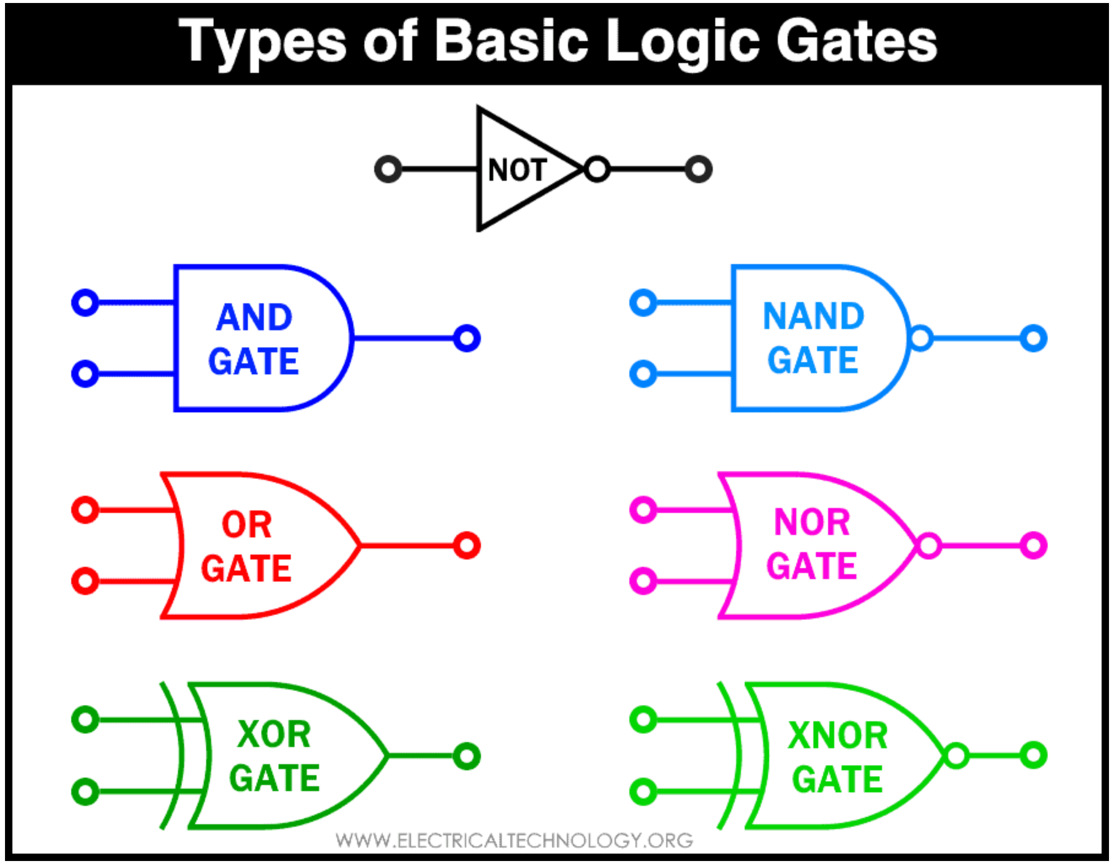


Revision

Digital electronics & Boolean logic

Digital electronics & Boolean logic

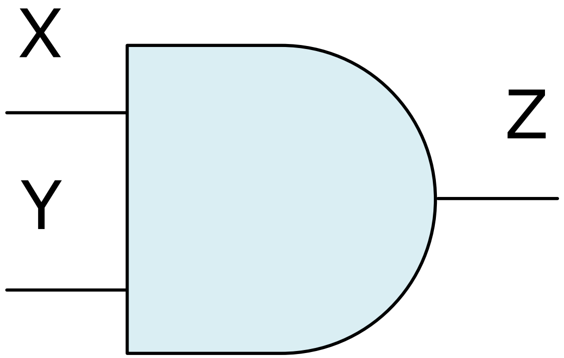
- Computers are made up of very large numbers of **logic gates**
- We will revisit this in due course in much more detail
- But first we need to revise:
  - Boolean logic
  - Truth tables
  - Logic gates (physical and conceptual)



- The main Boolean operations we will consider are:
  - NOT (inverter)
  - AND
  - OR
- Behaviour represented using truth tables
- Can also be illustrated via **timing diagrams**

Digital electronics & Boolean logic

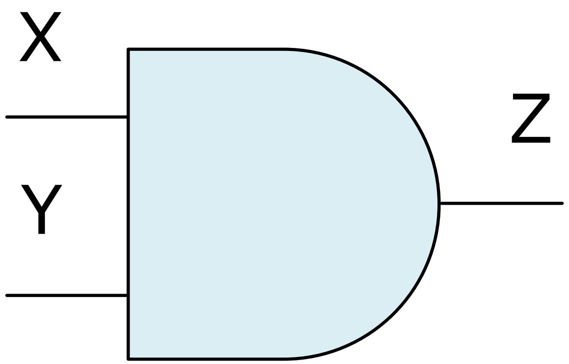
Input		Output
X	Y	Z



Usual symbol for AND gate

Digital electronics & Boolean logic

Input		Output
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

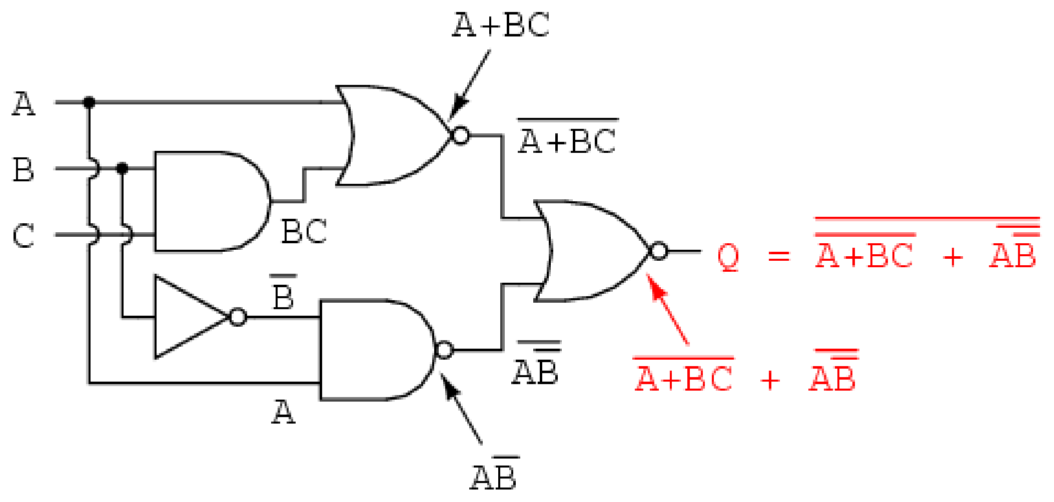


Usual symbol for AND gate

Digital electronics & Boolean logic

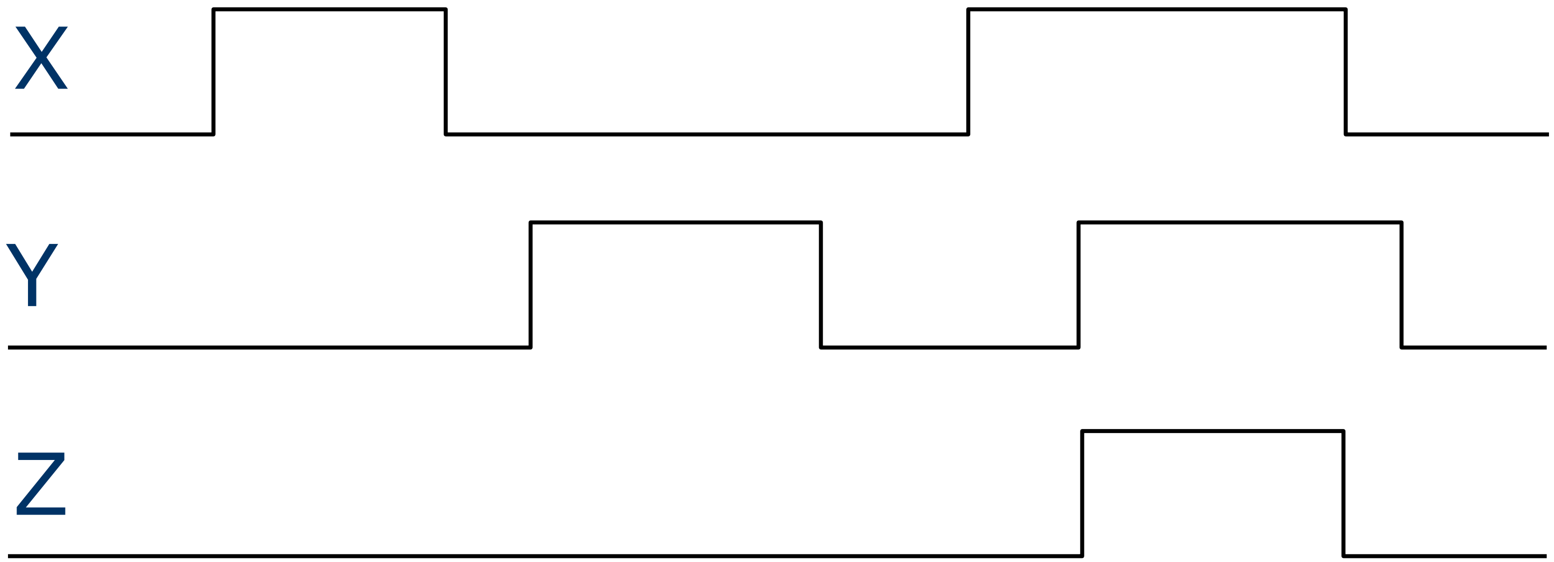
- Boolean algebra (or logic) involves operations on TRUE/FALSE states

- In digital electronics:
  - **Boolean logic** is performed by logic gates
  - Typically: **TRUE** is represented by a wire at around **5v**
  - **FALSE** is represented by a wire at around **0v** (earth potential)



Digital electronics & Boolean logic

- May be considered to represent ’scope trace
- Timing diagram representation of an AND gate:



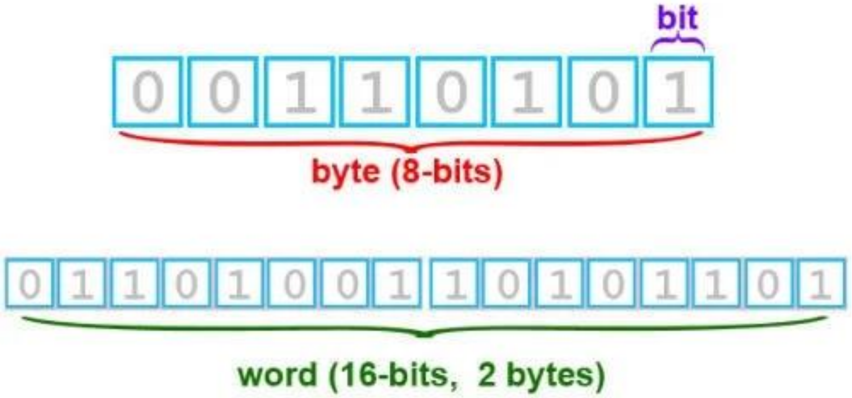
University of Nottingham
UK | CHINA | MALAYSIA

Revision

Numbers in computers

Binary numbers and logic

- In practice, logic signals may be either:
  - **single** items of Boolean data or signals
  - **groups** (usually in multiples of 8 “bits”) of Boolean data representing a number in binary form



- As well as representing **data**, binary numbers are used to represent **instructions for a computer or microprocessor**

Hexadecimal notation (“hex”)

- Writing binary numbers e.g.

  1111101000001010000010011010110011

  is awkward and error-prone.

- Introduce “hex” notation: 0-9 take usual meaning, A=ten, B=eleven, C=twelve etc. up to F = fifteen

Hexadecimal notation (“hex”)

Dec	Bin	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

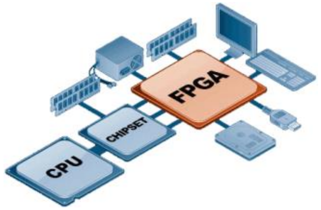
Dec	Bin	Hex
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

A Typical Mechatronics System

Controller: Hardware

Computers and microprocessors

- All these examples involve mechanical equipment being controlled by a computer
- In practice the computer may be:
  - A PC
  - An embedded microprocessor (microcontroller) running code directly e.g., Arduino
  - A system running a real-time operating system (e.g., the Compact RIO) – no distractions e.g., mouse or anti-virus
  - An FPGA – not really a computer at all!

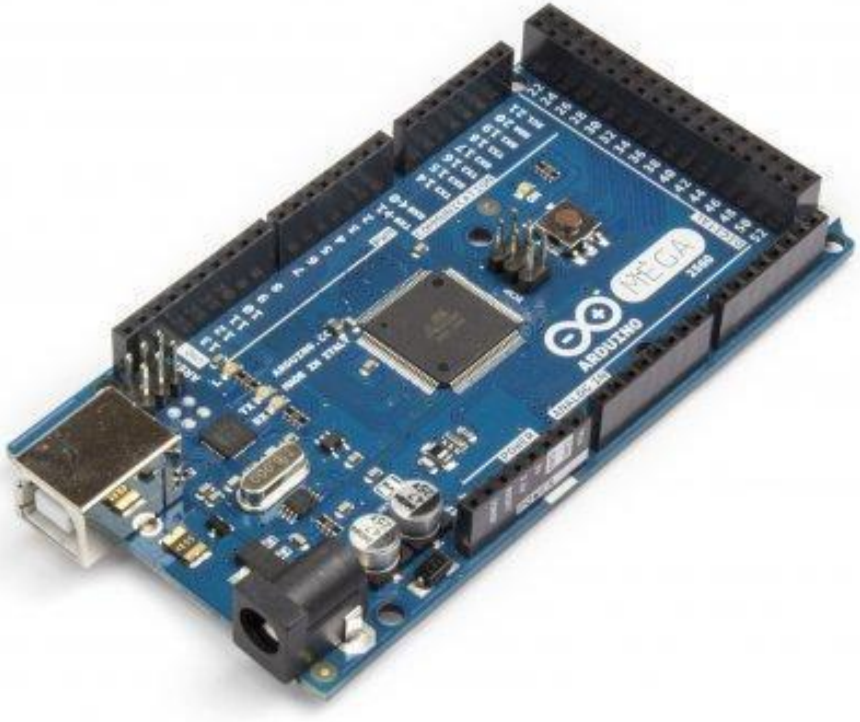


Computers and microprocessors

- Arduino microcontroller (Mega, Uno and Nano)

Arduino Mega 2560

- The Arduino hardware we'll be using
- Really just a **microcontroller** chip on a board
- An **AVR Atmega 2560** microcontroller (computer on a chip)
- Mounted on a circuit board with clearly labelled connections



Arduino Mega 2560

- Only enhancements to basic Atmega chip are:
  - A **USB** interface to allow programming and communication
  - Some special code (“**bootloader**”) to load code
  - **A reset** button and an **LED** on one output line
  - **Voltage regulator**
- The Atmega 2560 incorporates:
  - Digital input and output
  - Analog input, “Analog output” (or something equivalent i.e., PWM – pulse width modulation)
  - Numerous programmable features for pulse generation etc.

A Typical Mechatronics System

Controller: Software

Microprocessors Programming

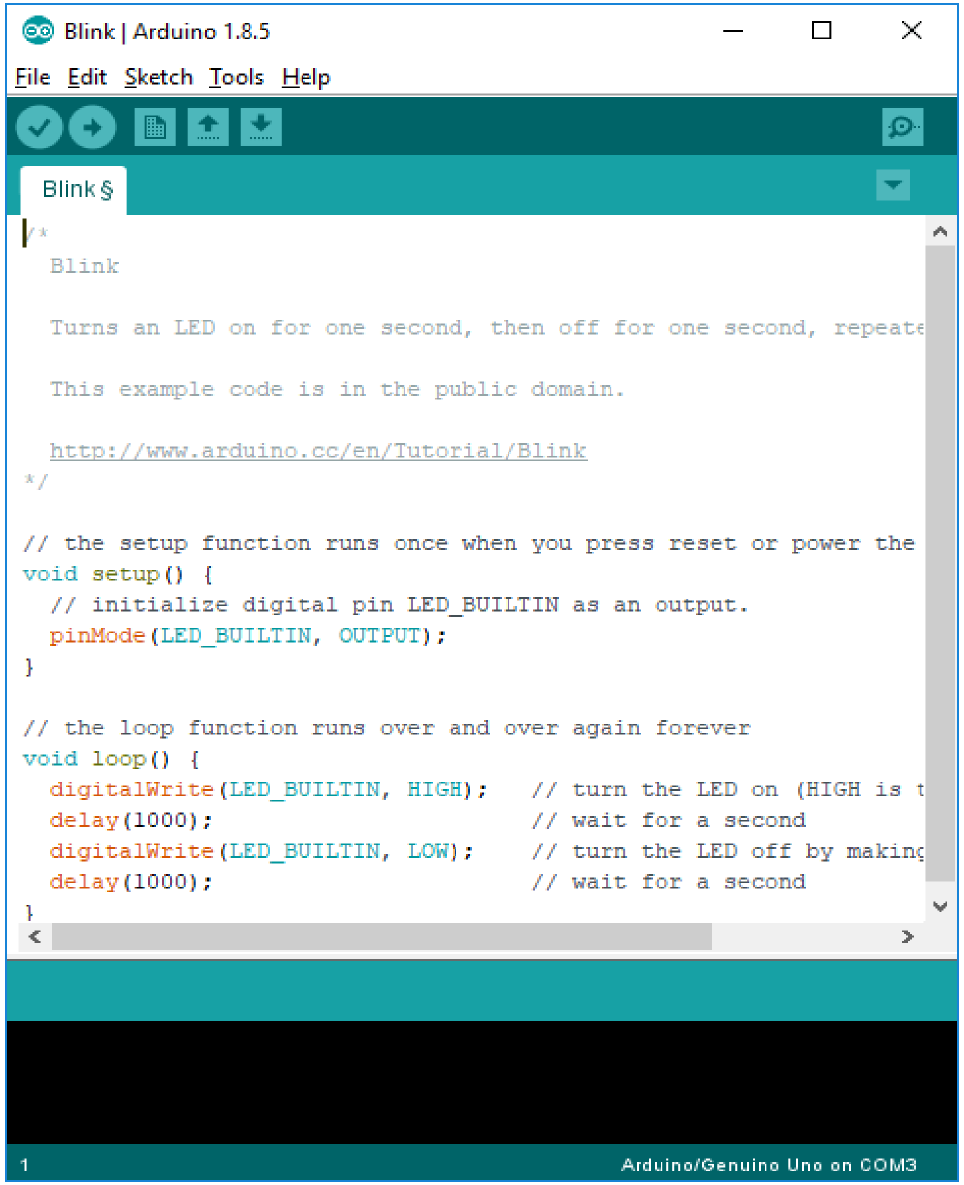
- Usually program a microprocessor via some form of language, traditionally text-based:
  - Assembly language: “mnemonics” which contain very detailed instructions each of which is translated into a single instruction machine code using an **assembler**.
  - High-level language e.g., MATLAB, Python, C, C++: language is human readable and oriented to the **problem** not the **programming task,** translated into machine code using **compiler**
  - **We’ll use the Arduino variant of C/C++**

Microprocessors Programming

- Used to program the Arduino series of microcontrollers including the Mega 2560
- As we've already said, it is essentially the C language
- Written and compiled within the Arduino integrated development environment (IDE)
- An easy-to-use programming interface which does roughly same job as VSCode (Louise Brown) but for Arduino

Arduino IDE

- **Text** editor
- **Menus** to select files, board etc.
- **Buttons** to compile & upload code
- **Serial monitor**: the nearest we have to text input and output on the PC



A Typical Mechatronics System

Controller: Software – Simple Example

The Arduino language

- A conventional Arduino program to "Hello World"

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // Opens serial port at 9600 baud (~bits/s)
}

void loop()
{
  // put your main code here, to run repeatedly:
  Serial.println("Hello world!"); // Write with new line
  delay(1000);
}
```

The Arduino language

- Mostly the same syntax
- Different top-level structure – no **`main()`**
- and different input/output statements:

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // Opens serial port at 9600 baud (~bits/s)
}

void loop()
{
  // put your main code here, to run repeatedly:
  Serial.println("Hello world!"); // Write with new line
  delay(1000);
}
```

A Typical Mechatronics System

Controller: Software – Setup & Loop

Let’s pull the sketch apart…

- We have **two function** definitions

```
void setup()
void loop()
```

- In each case they are trivially simple:
  - Don’t take any parameters so () are empty
  - Don’t return any value so declared as void()
- The code they contain between {…} is executed each time the function is called
- Nearly every statement finishes with a semicolon ; except function definitions & {…}

Let's pull the sketch apart…

- We have one simple function call

```
delay(1000);
```

  - Takes one function parameter, delay in milliseconds
  - It doesn't return any value (it's a **`void`** function)
- We also have an "object", **`Serial`**, for which we call two of its functions or "methods":

```
Serial.begin(9600); // Opens serial port at 9600 baud (~bits/s)
Serial.println("Hello world!"); // Write with new line
```

- Note: "pure C" doesn't have objects, they are a C++ feature we'll use occasionally.

Let’s put the sketch back together…

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // Opens serial port at 9600 baud (~bits/s)
}

void loop()
{
  // put your main code here, to run repeatedly:
  Serial.println("Hello world!"); // Write with new line
  delay(1000);
}
```

- Comments are preceded by `//`
- Also, long comments can be enclosed with

  `/* … */`

Another easy Arduino sketch: "Blink"

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

- Pre-loaded on every Arduino you buy

A Typical Mechatronics System

Controller: Software – More

More about the Arduino language

- In principle, that is all I should teach you
- Louise will do the C language itself with you, starting soon!
- To keep us going, we need to learn enough to “get by in C”

Variable types in the Arduino

- Unlike MATLAB, variables in C and Arduino must be declared up-front to have specific types, for example, signed and unsigned integers:

```
int a;     // -32768 to 32767
long b;    // -2147483648 to 2147483647
unsigned int c;      // 0 to 65535
unsigned long d;     // 0 to 4294967295
```

Variable types in the Arduino

- Similarly, we can have floating point numbers (always signed, of course):

```
float f; // -3.40282×10^-38 to 3.40282×10^38
double g; // same as above in Arduino only
```

- Single characters are stored as an integer number representing ASCII code e.g. A is 65:

```
char h; // -128 to 127
```

Variable types in the Arduino

- Sometimes we wish to store only an 8-bit unsigned number, known as a byte:

```
byte a;   // 0 to 255
```

- And sometimes we just wish to store a value which is either true or false:

```
bool a;  // true or false
```

Declaring & initialising at same time

- Quite often we want to set the value of a variable at the same time as we declare it:

```
int a = 42;     // the meaning of life
long b = 0;     // initial position
byte mask = 0xF0 // Hex: means 11110000
```

- And sometimes we want the value to stay constant and to be impossible to change:

```
const float e = 2.7182818; //log base
```

Variable scope

- Variables only exist in the “scope” in which they are declared
- In practice, a variable within a function only exists in that function
- To be visible anywhere else it must be passed as a function parameter. But **`setup`** and **`loop`** don’t have parameters so how can they share data?

Variable scope

- Any variable declared OUTSIDE of any functions is visible EVERYWHERE!
- It is a **global variable**
- (Just don’t do this for Louise…)

```
int counter = 0;   // A global variable

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // Opens serial port
}

void loop()
{
  // main code here, to run repeatedly:
  Serial.println(counter); // Disp on monitor
  delay(1000);
  counter = counter + 1; // Or: counter++;
}
```

Manipulating variables: arithmetic

- More or less the same as in MATLAB:

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Division	`/`
Assignment e.g.	`a = b + c;`

- And remember the big trap: equality test uses == not = e.g. `if(a==b)`

Manipulating Boolean quantities

- When we are dealing with Boolean variables, we use the operators `&&`, `||`, `!`, for example

```
bool a=true, b=false;
```

`a && b` represents a AND b

`a || b` represents a OR b

`!a` represents NOT a i.e., the opposite of a

Manipulating bits: bitwise operations

- But sometimes we want to treat each bit (binary digit) of an integer as a Boolean: this is called **bitwise operations**

- Just does the operation one bit at a time

X	10110101
Y	00101110
X bitwise-AND Y	00100100

Manipulating bits: bitwise operations

Bitwise operations in C:

Bitwise-AND: `z = x & y;`

Bitwise-OR: `z = x | y;`

Bitwise-XOR: `z = x ^ y;`

Bitwise-NOT: `z = ~x;`

(single symbol c.f. `&&` and `||` for operations applying to whole Boolean variables)

Manipulating bits: shifting of bits

- It is very often useful to shift a binary number left (or right) by a certain number of bits
- Equivalent to multiplying or dividing by a power of 2 (throwing away any remainder or any bits that "fall off the end" or overflow). New bits are 0. For example:

`1 << 3` takes the number 00000001 and shifts it left by 3 places to give 00001000 (equivalent to multiplying it by 2^3 i.e. 8)

Summary of lecture

- Module objectives and learning outcomes presented
- Mechatronics defined and illustrated
- Basic electronics etc. revised
- Arduino language (simplified C) introduced