University of Nottingham
UK | CHINA | MALAYSIA

Lecture 2

Computer Architecture, Digital input/output, and Timer/counters

Mechatronics

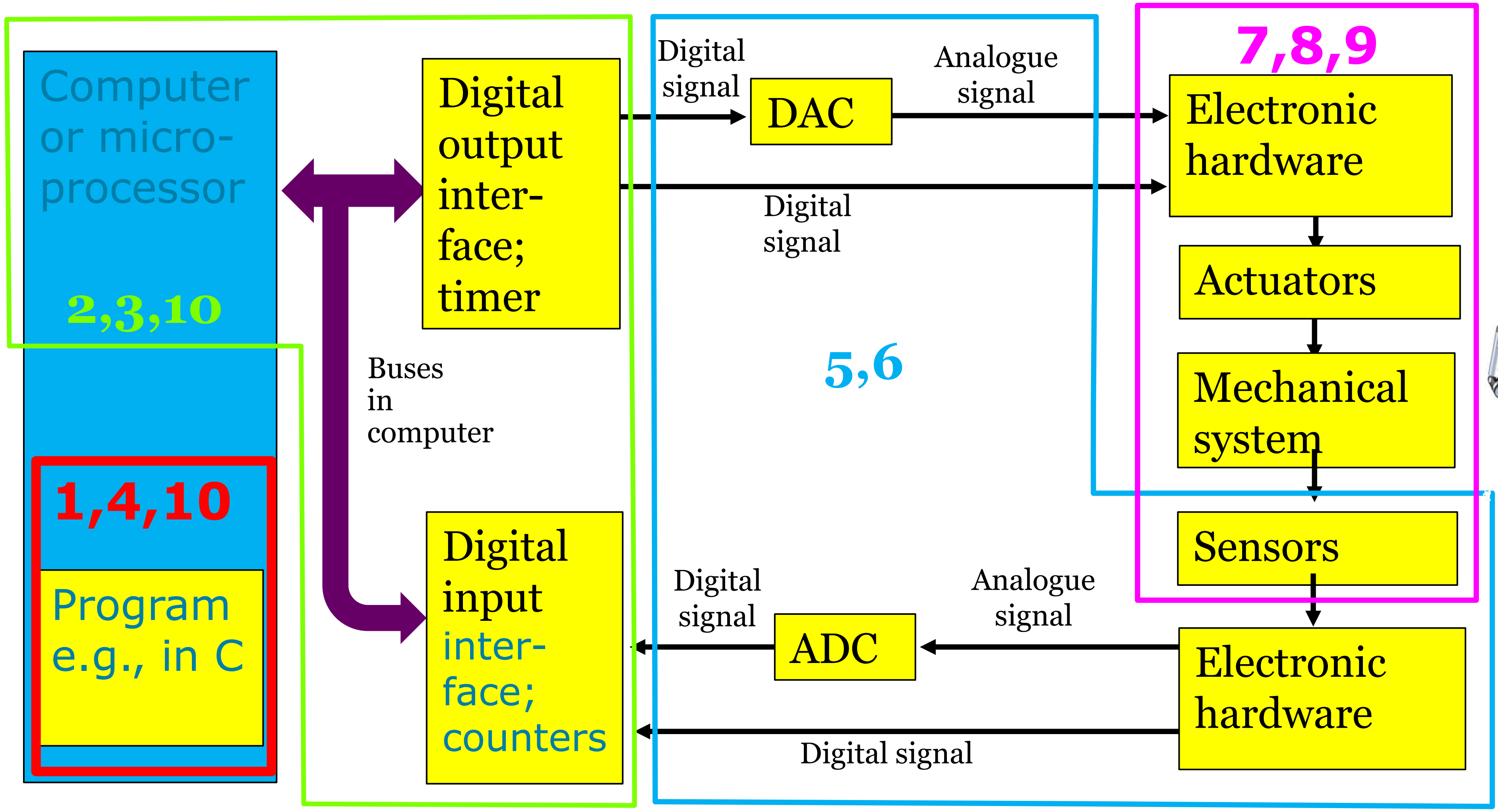
MMME3085

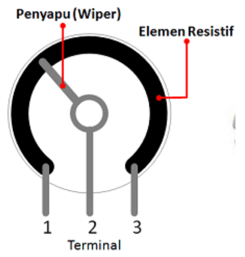
Module Convenor – Abdelkhalick Mohammad

Objectives of lecture

- To introduce computer architecture and understand what goes on in a computer
- To introduce the hardware and implications for software required to interface a computer to electromechanical systems
- To demonstrate the way that a simple parallel interface is implemented in Arduino
- To introduce the concept of timer/counters and their uses in the Arduino

A typical Mechatronics System





University of Nottingham
UK | CHINA | MALAYSIA

Introduction

Introduction

- Want to:
  - transfer data from program to an output device
  - take data from external source (input device) into the computer
- Data can be:
  - Serial (series of pulses - high or low for 1 or 0, in principle only 1 or 2 wires plus ground needed)
  - Parallel (1 wire per bit, usually 8 per "port")
- Concentrate on parallel data at present time
- Need to know what we can connect to

University of Nottingham
UK | CHINA | MALAYSIA

Types of Signals

Types of signal

- **Analogue**: signal proportional to physical value
  - Varying voltage or current
  - *Infinitely* variable, resolution limited by:
    - Resolution of measurement device e.g., ADC (analog-to-digital converter)
    - Noise
- **Digital**: binary number repr'g physical value
  - Two states (0/1 represented as 0v/5v etc)
  - Serial or parallel (consider only parallel)
  - Resolution limited by number of bits

Types of signal

- **Trains of pulses**:
  - Voltage alternates between two levels (square wave).
  - Frequency of the pulse train or the number of pulses transmitted
  - To have an up/down count we need two sets of pulses 1/4 phase out of step (quadrature pulses) and a suitable decoder
  - Usually defined as a digital signal (interested in “rate” rather than “state”)

University of Nottingham
UK | CHINA | MALAYSIA

Digital Signal

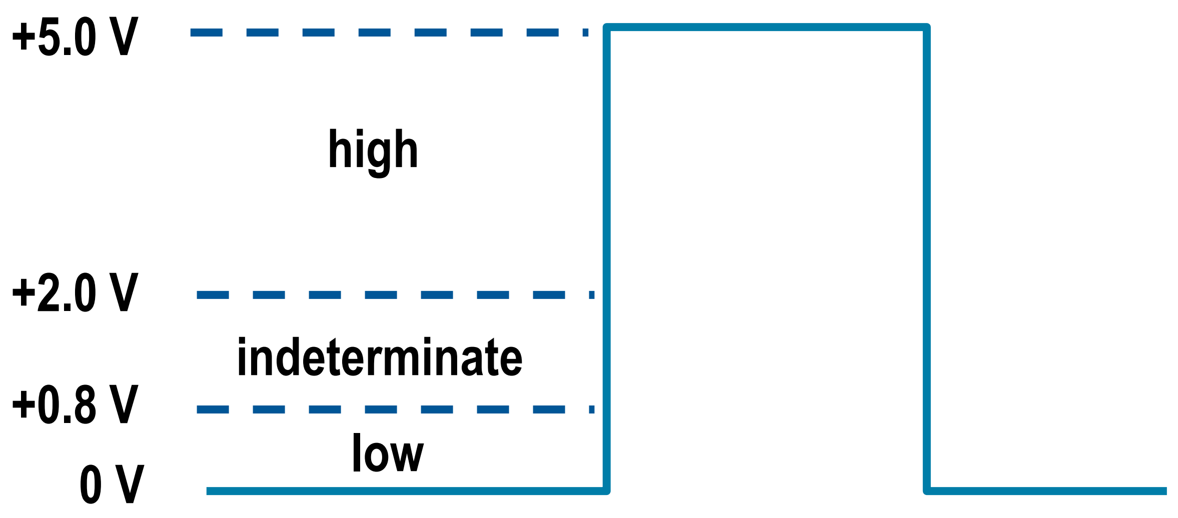
Digital signals

- So far, we have talked about digitals signals having:
  - A certain number of bits (8, 16, 24 etc.)
  - Only two levels: 0v or 5v

- This last point is not strictly true...
- It will work but isn't the full story

Digital signals

- Digital lines on a Data Acquisition (DAQ) device accept and generate *Transistor-Transistor Logics* (TTL) compatible signals

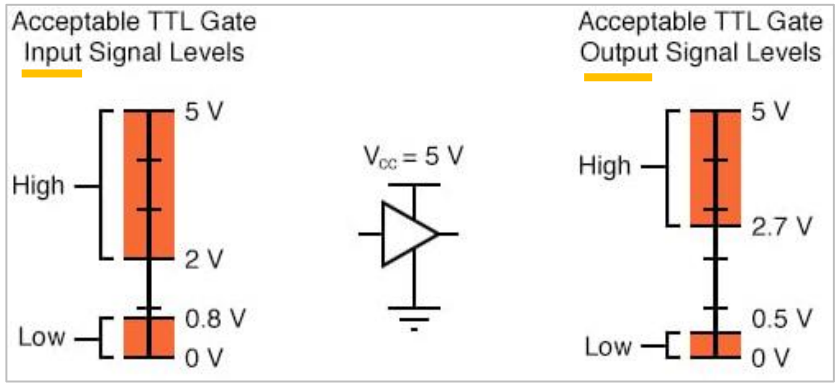


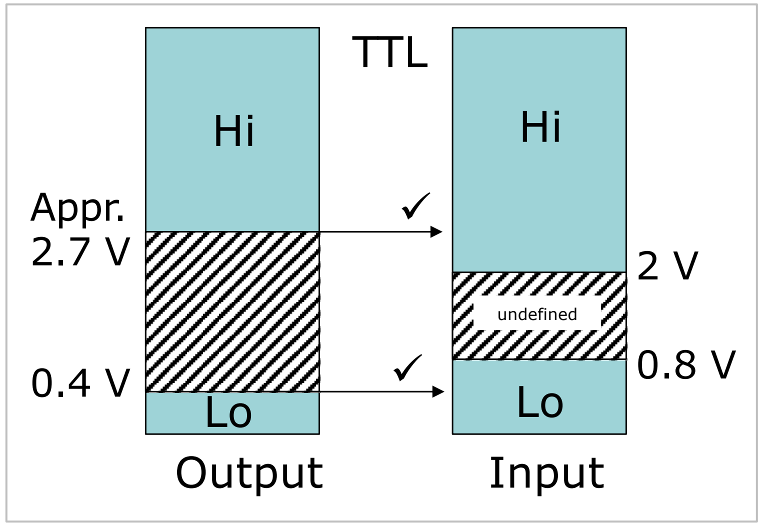
Definition of a TTL Signal

Digital signals: TTL

- Actual voltage levels depend on the circuitry we use to generate the digital signal
- Most common (and the one used in your labs) is Transistor-Transistor Logic (TTL)
- Traditional, obsolete for actual circuits
- Still used as a standard for i/o levels

Description	Voltage level
Low input	< 0.8v
High input	> 2v
Low output	< 0.4v
High output	> 2.7-3.4v (typical)

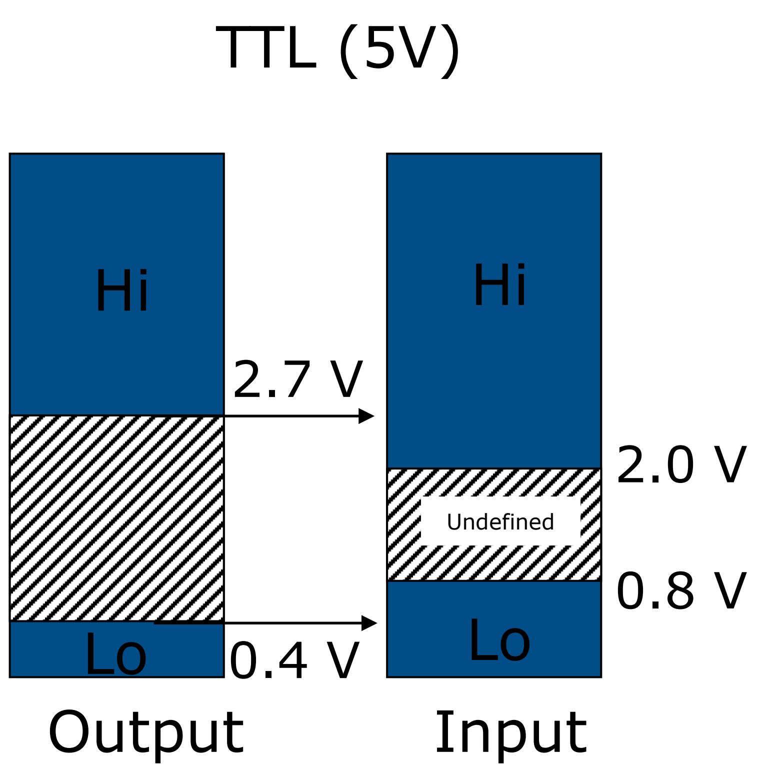

Acceptable TTL Gate Input Signal Levels
High
Low
5 V
2 V
0.8 V
0 V
V_{cc} = 5 V
Acceptable TTL Gate Output Signal Levels
High
Low
5 V
2.7 V
0.5 V
0 V

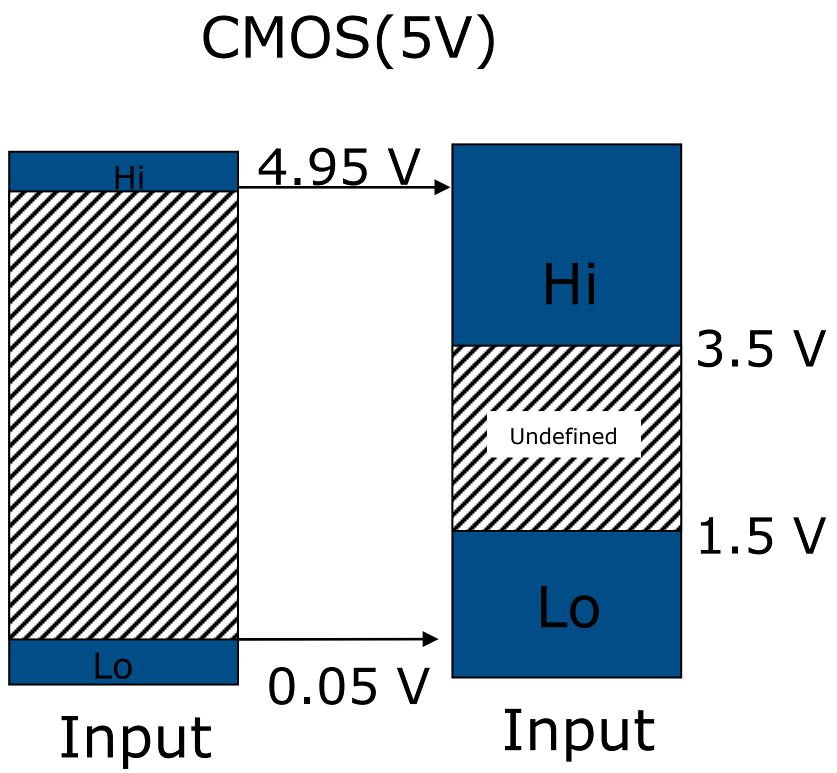


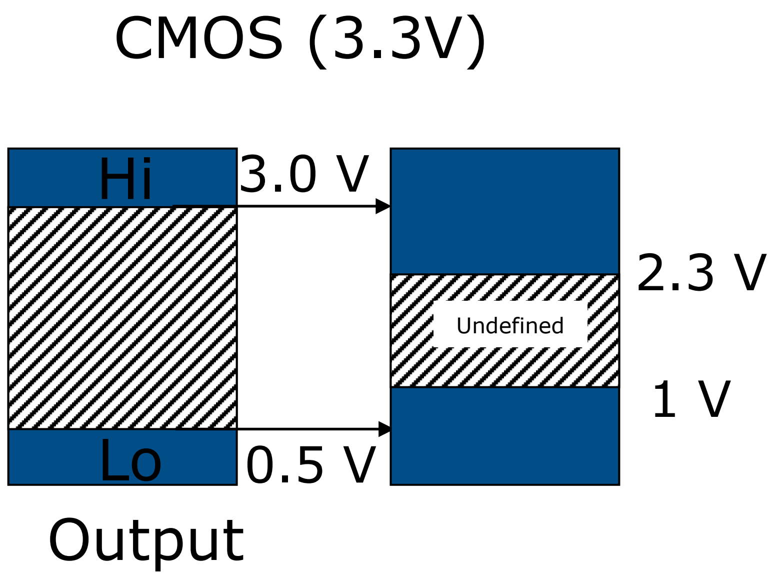
Digital signals: Other standards

- Be aware that there are various other standards for logic levels, relating to other types of circuitry
- e.g., CMOS (complementary metal oxide semiconductor) now standard approach: some High-speed CMOS (HC) use significantly different levels
- Fortunately, can get CMOS circuits that accept TTL voltages for input (HC Transistor series)
- Other levels sometimes used e.g., 0v/24v
- 3.3v is being increasingly widely used

Digital signals: Other standards



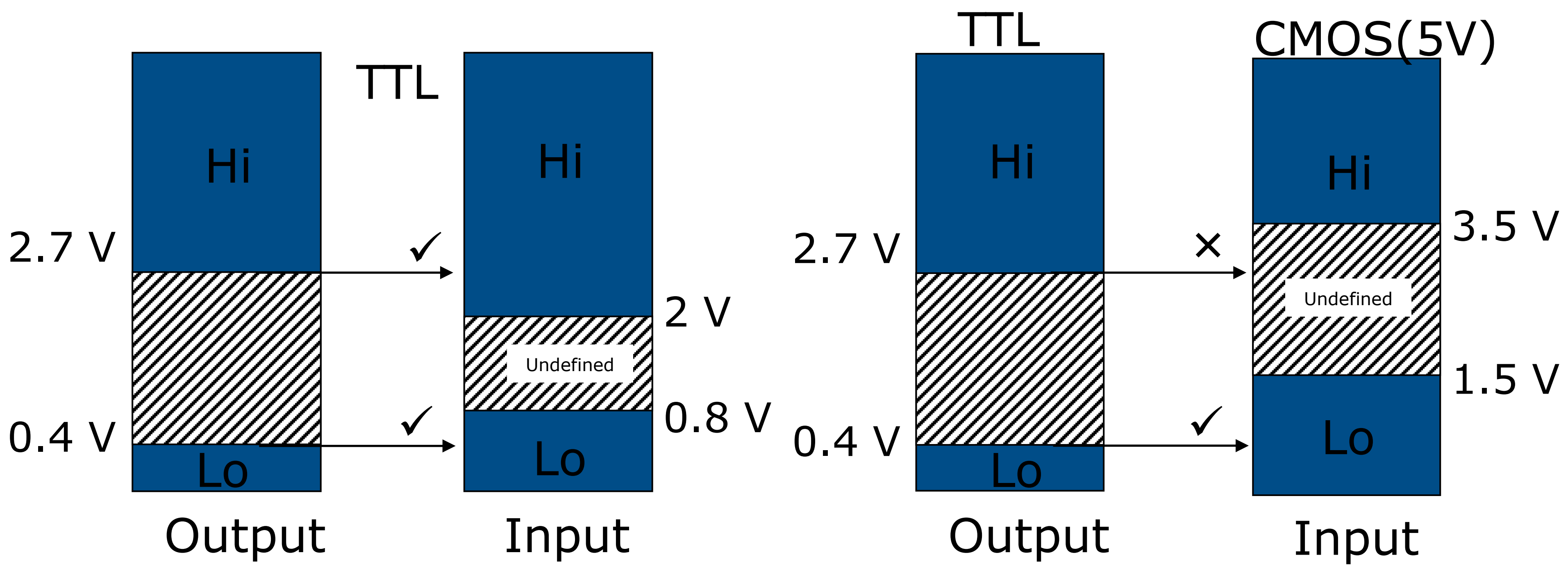




University of Nottingham UK | CHINA | MALAYSIA

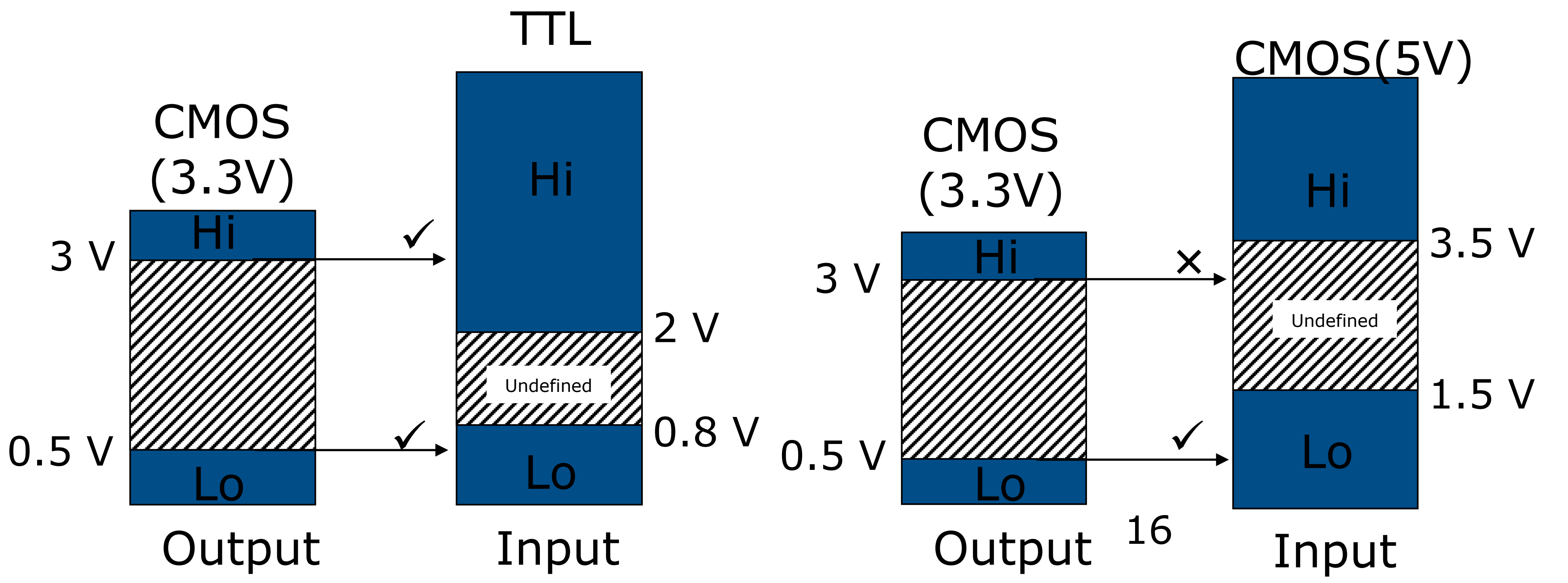
Digital signals: Other standards

- There can be incompatibilities between different logic families even if supplied by same voltage…



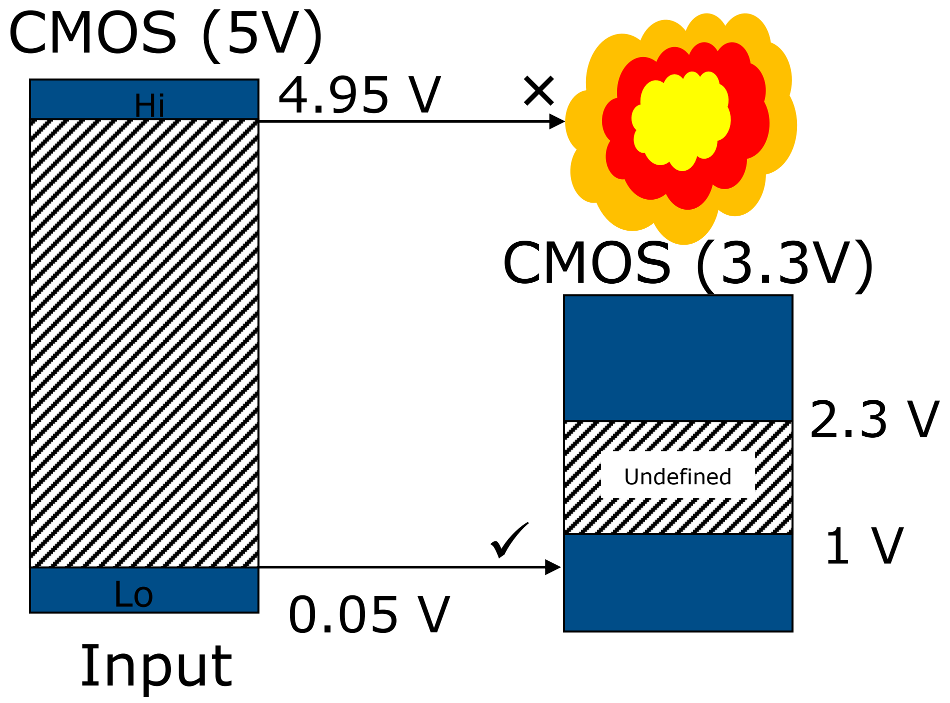
Digital signals: Converting between standards

- Not directly compatible! Will probably work for driving low power TTL input from 3.3V logic, but won't work for driving some 5V CMOS:



Digital signals: Converting between standards

- And driving 3.3V logic input from 5V logic (CMOS or TTL) may well burn it out!



University of Nottingham
UK | CHINA | MALAYSIA

Digital Signal

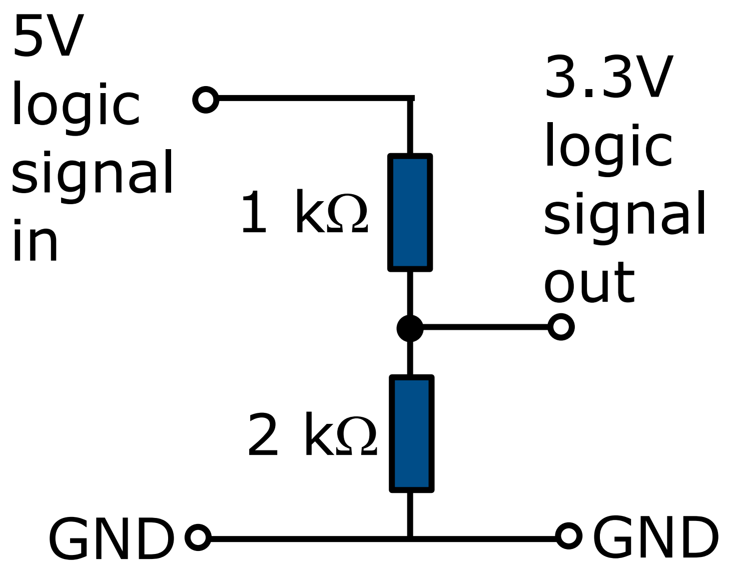
Converting between standards

Digital signals: Converting between standards

- Example: the Arduino Due is based on different technology (ARM) from the Atmega 328 and 2560
- It is based on 3.3 V logic
- But you may want to use it with 5V devices

Digital signals: Converting between standards

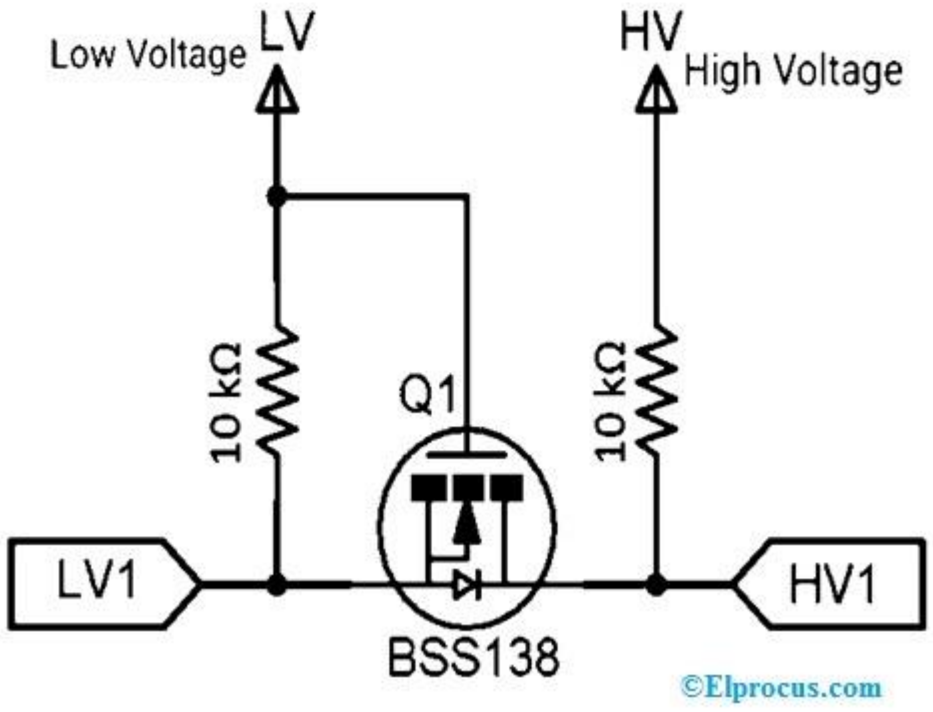
- Potential divider: only for slow signals, 1 direction
- Or use a level converter – safe bet, really clever, same device works in both directions!



Digital signals: Converting between standards

- Potential divider: only for slow signals, 1 direction
- Or use a level converter – safe bet, really clever, same device works in both directions!

Learn concept not circuit!



Digital signals: Converting between standards

- Fortunately, it seems that 3.3 V logic WILL be correctly interpreted by a 5 V Arduino (but not of course the other way around)
- In general, the inputs to an Arduino are fairly “forgiving” and will accept TTL (under typical conditions) and CMOS levels
- And (unusually for a logic gate) they will supply quite a lot of current, enough to operate an LED via a resistor without damage.

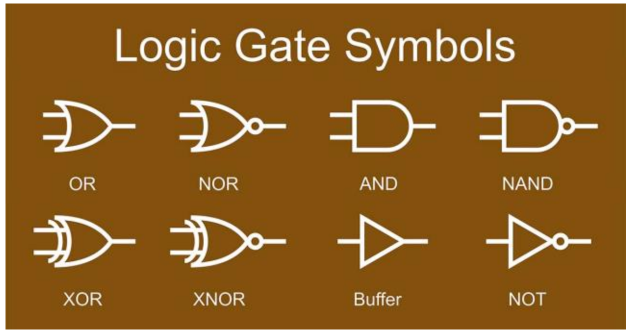
University of Nottingham
UK | CHINA | MALAYSIA

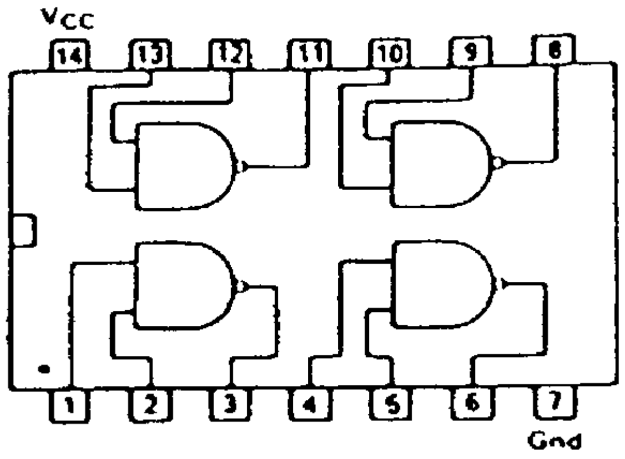
Digital Signal

Logic gates in practice

Digital signals: logic gates in practice

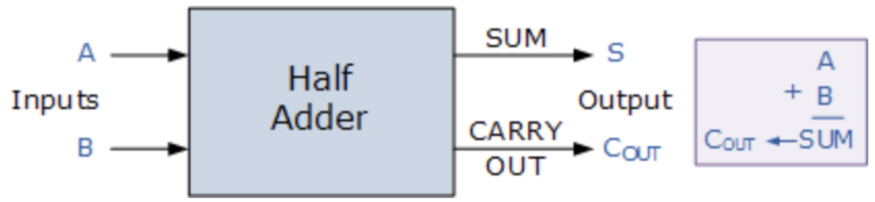
- Logic signals are generated by, and interpreted by, logic gates
- Very large-scale integration (VLSI) chips, microprocessors, microcontrollers (including Atmega!), FPGAs
- Can buy logic gates separately if you wish e.g., 7400 series logic gates based on TTL
- By default, the outputs from these are "totem pole", but can be "open collector"



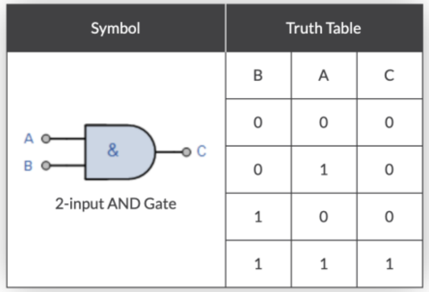


Digital signals: logic gates in practice

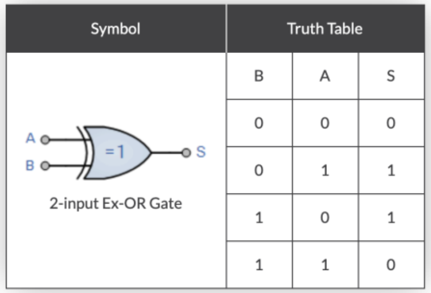
123	A
+ 789	B
912	SUM



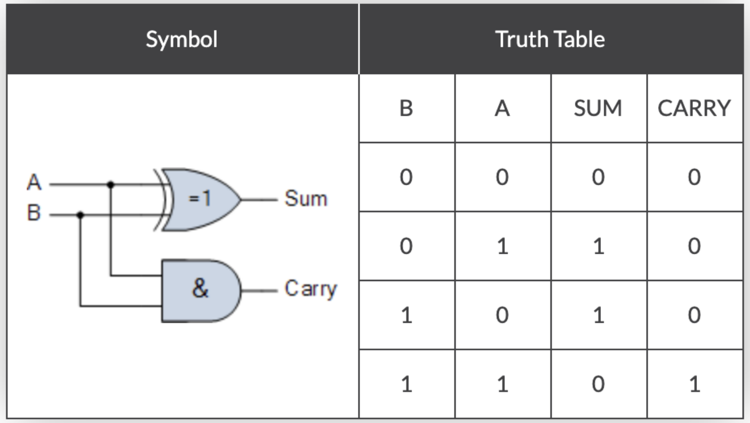
0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	(carry) 1←0



Symbol	Truth Table		
	B	A	C
2-input AND Gate	0	0	0
	0	1	0
	1	0	0
	1	1	1

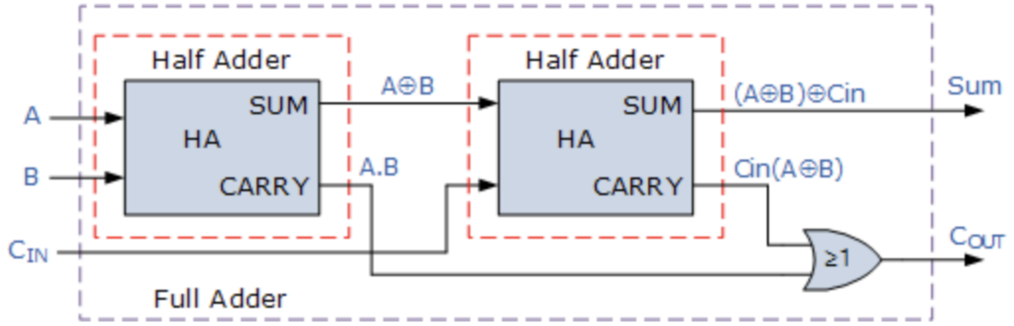


Symbol	Truth Table		
	B	A	S
2-input Ex-OR Gate	0	0	0
	0	1	1
	1	0	1
	1	1	0



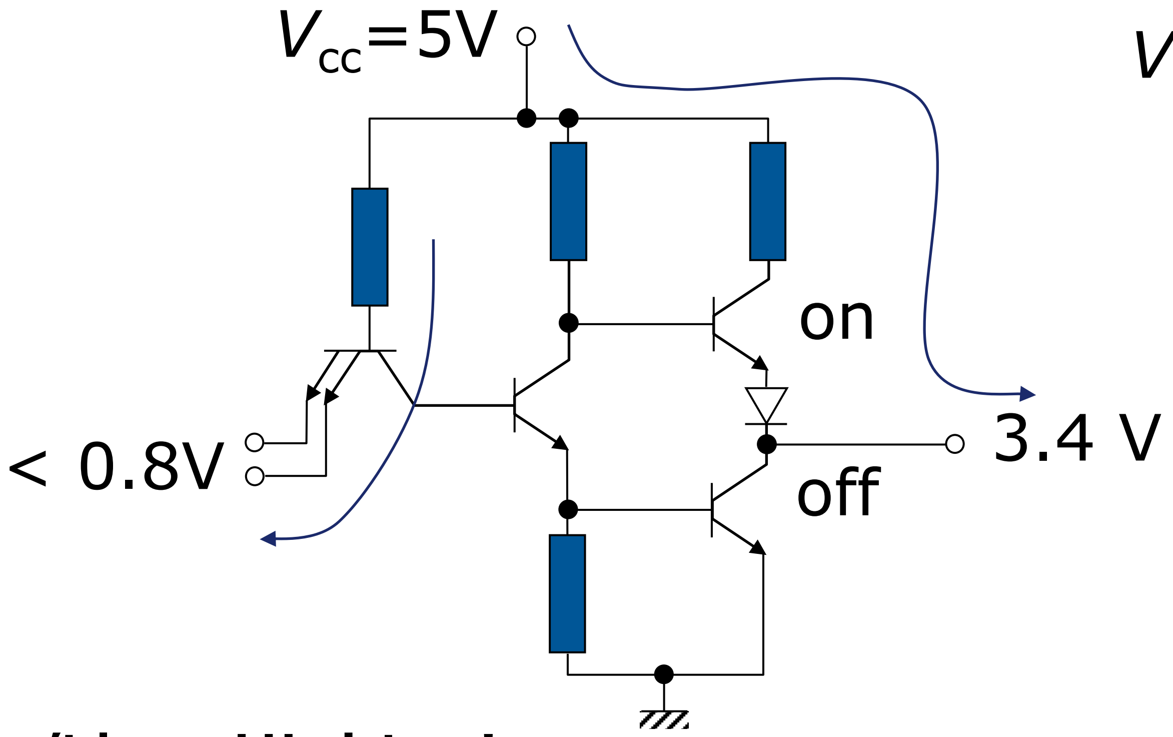
Symbol	Truth Table			
	B	A	SUM	CARRY
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1

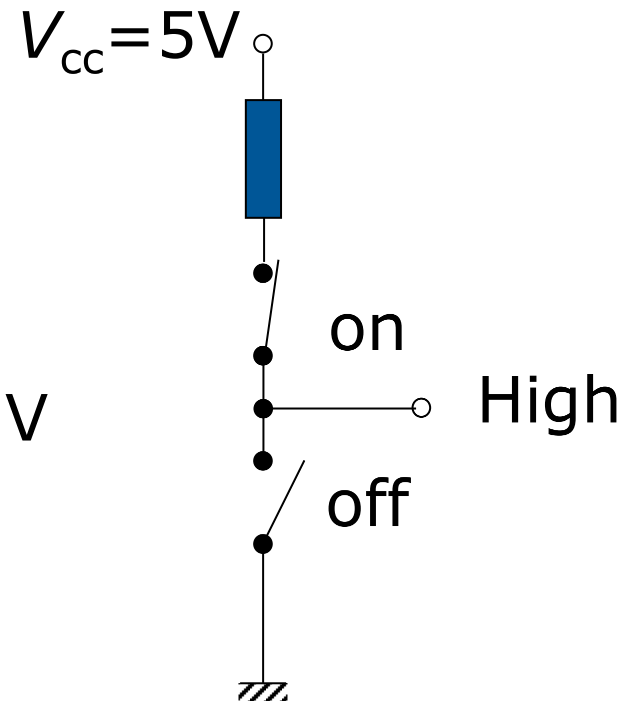
A
B
=1
Sum
&
Carry

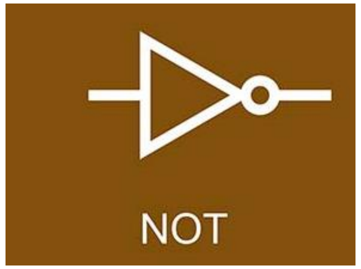


TTL totem-pole (push-pull) output

- Output involves two transistors which act as switches



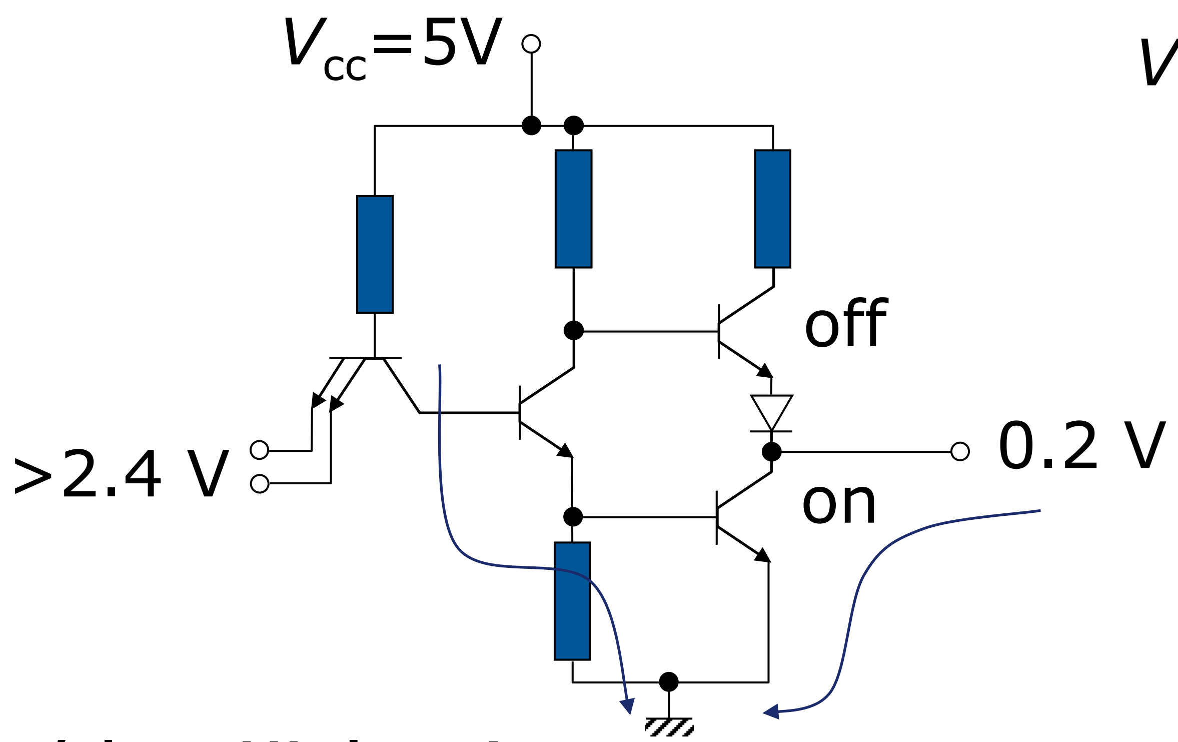


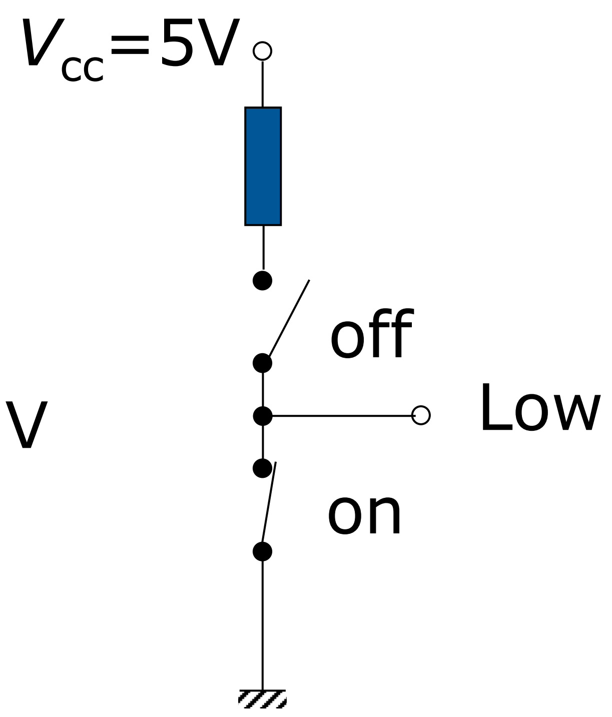


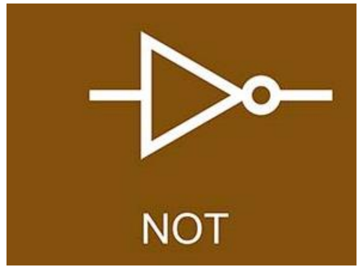
Don’t learn LH picture!

TTL totem-pole (push-pull) output

- Always runs from 5v – output voltages fixed





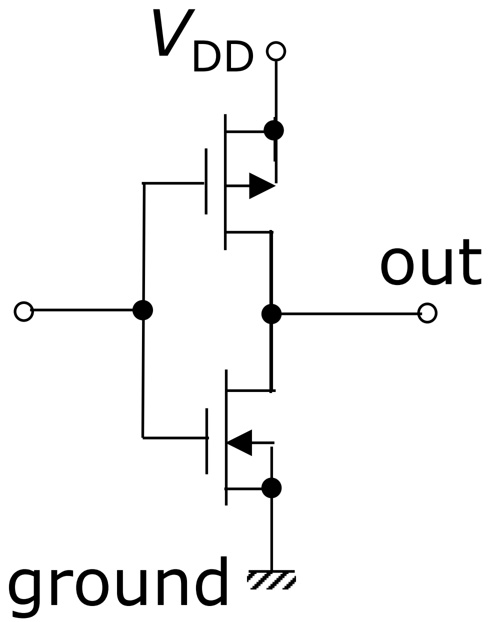


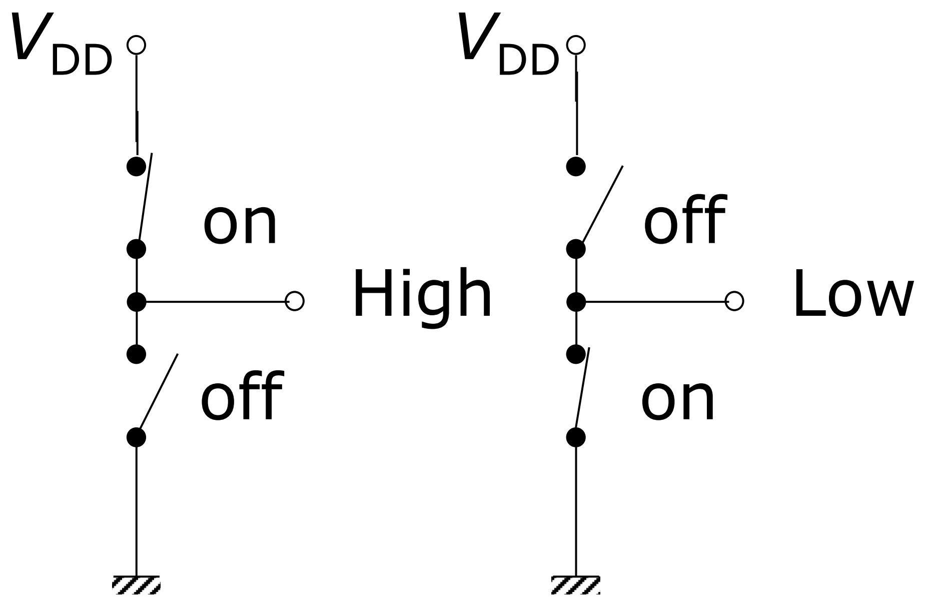
Don’t learn LH picture!

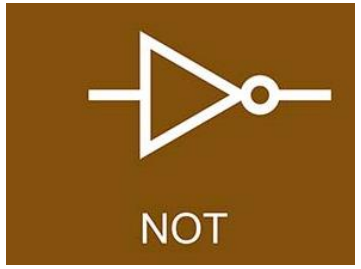
University of Nottingham UK | CHINA | MALAYSIA

CMOS push-pull output

- Can use different voltage ranges but output limited to same voltages as rest of circuit



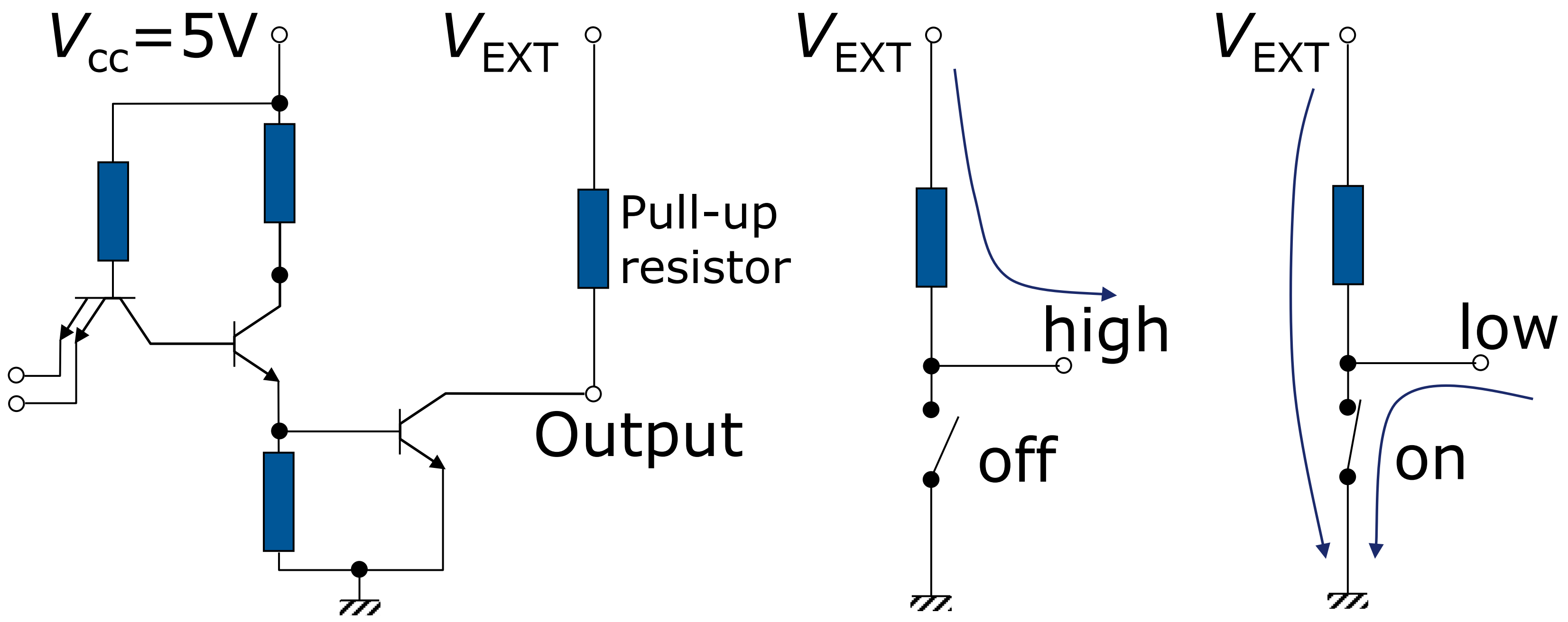


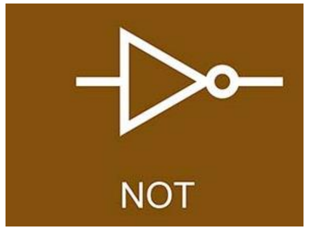


Don’t learn LH picture, and RH pictured are more or less as prev slide
Operation same as for TTL totem pole so nothing new to learn

TTL open-collector

- Output is either connected to ground (low) or not connected to ground
- V_{EXT} is any reasonable voltage (+12V, +15V)





Don’t learn LH picture!

TTL open-collector

- More versatile than totem-pole in the sense that output can be in any reasonable desired range not just 0-5v.
- Normal situation relates to totem-pole
- But outputs from interfaces or other devices are often defined as being open-collector
- Good example is output from optical isolators – to be discussed shortly

But we still have two important questions!

- How do we get logic signals *into* the computer so that we can use the data within our program?
- How do we get data in our program *out* of the computer in the form of logic signals so that we can do something real with it such as switching on a light?

To answer these, we need to know how a computer works!

University of Nottingham
UK | CHINA | MALAYSIA

How a computer works!

University of Nottingham UK | CHINA | MALAYSIA

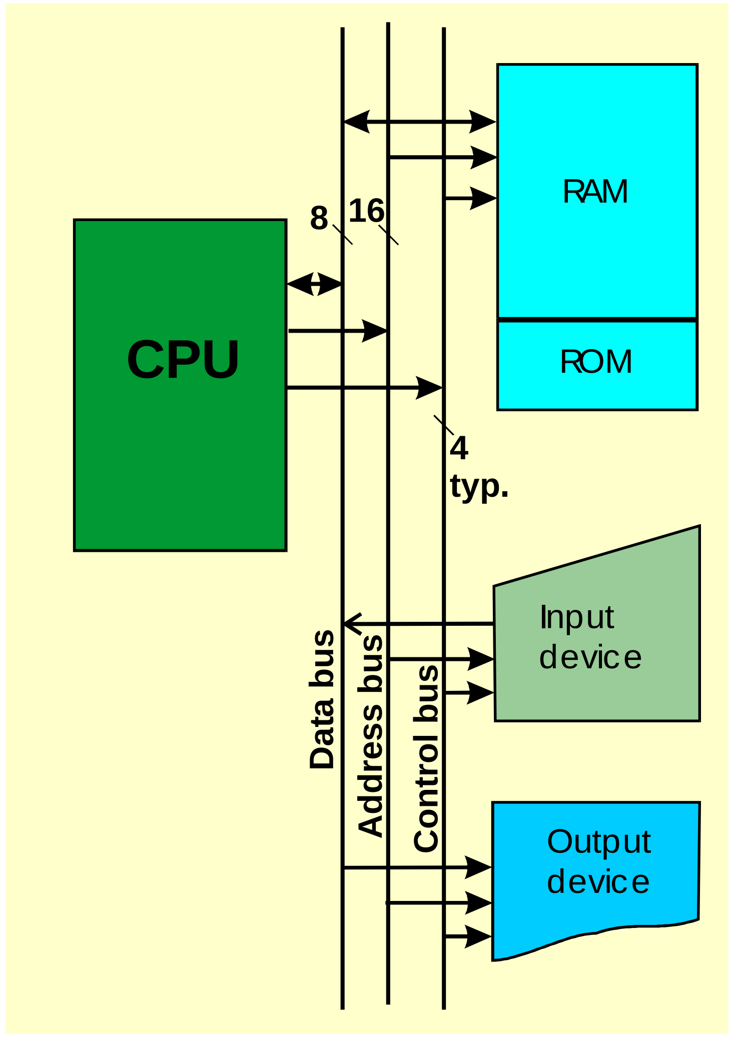
Some terms (by popular request!)

- Central Processing Unit (CPU) - the part of a computer which executes the commands in the program
- Microprocessor – an integrated circuit (chip) that contains a CPU e.g., of a laptop
- Microcontroller: complete computer on a chip for control applications e.g., ATMega 328 in Arduino
- Real time – used to describe a system which responds within an *accurately-known time*
- Buses – communication routes in computer
- Interrupt – an event causing special function (ISR, *interrupt service routine*) to take over computer temporarily

University of Nottingham
UK | CHINA | MALAYSIA

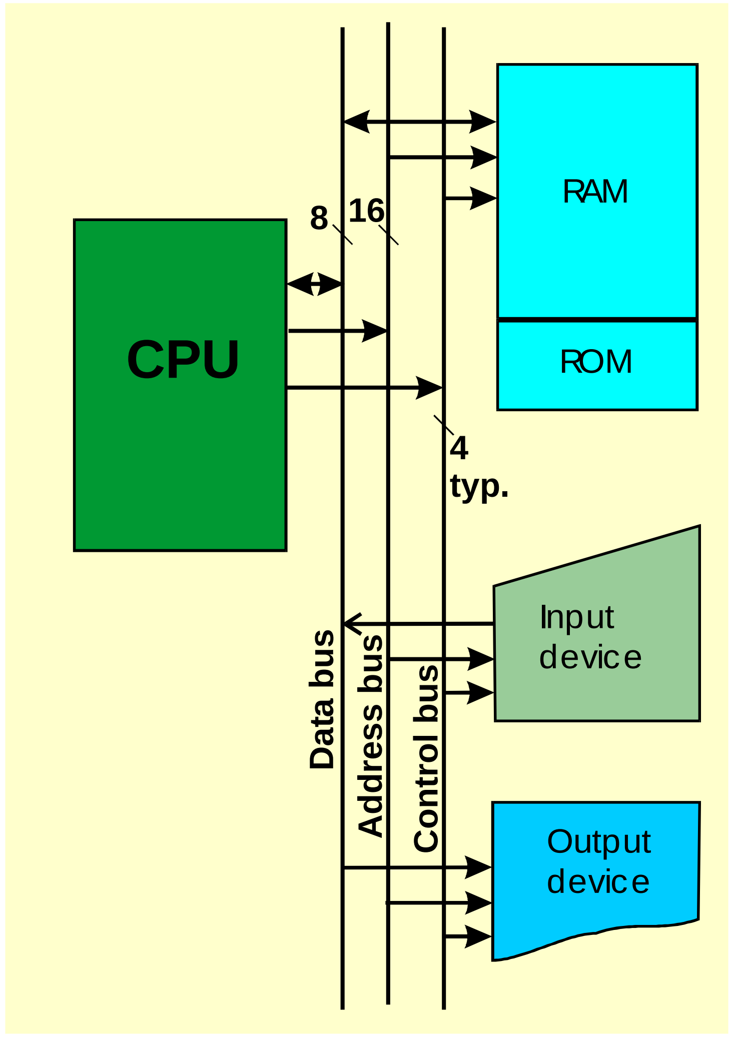
Architecture of simple computer

- Central Processor Unit (CPU), contains local memory (registers)
- Read only memory (ROM)
- Random access memory (RAM)
- Input and output
- Buses



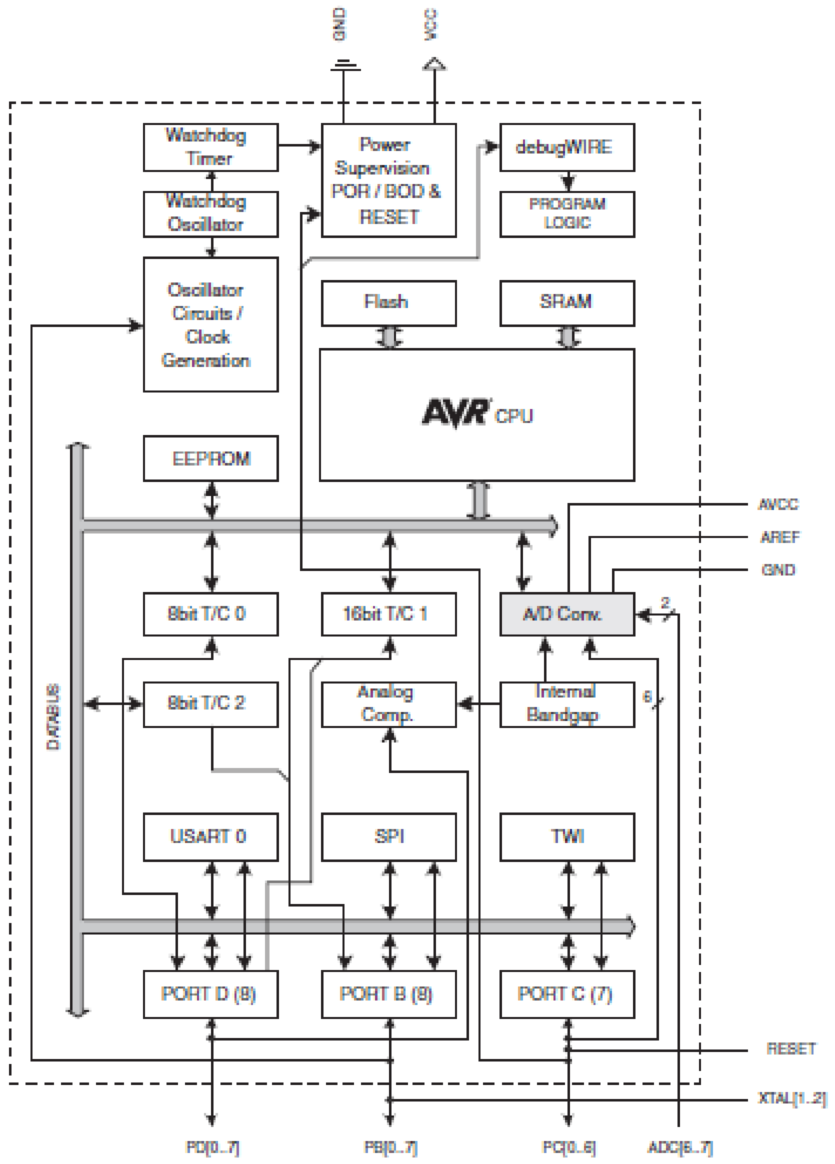
Architecture of simple computer

1. The CPU places the address of the I/O device that it wants to communicate with on the address bus.
2. The CPU sends a control signal to the I/O device over the control bus to indicate whether it wants to read or write data.
3. If the CPU wants to read data from the I/O device, the I/O device places the data on the data bus.
4. If the CPU wants to write data to the I/O device, it places the data on the data bus.
5. The CPU sends a control signal to the I/O device over the control bus to indicate that the data transfer is complete.



Architecture of simple computer

- Central Processor Unit (CPU), contains local memory (registers)
- Read only memory (ROM)
- Random access memory (RAM)
- Input and output
- Buses



So how do we connect into this?

- Needs interface to connect data bus to peripheral device (i.e., sensor/actuator)
- Will consider input/output on the Arduino
- In most cases, the interface device is “memory mapped”
  - perceived by computer as a memory location (or something very similar – “port mapped”, e.g., PORTA, PORTB. Etc.)
  - identified by an address on the address bus
- Assume:
  - digital "high" (1) ≈ 5V
  - digital "low" (0) ≈ 0V

University of Nottingham
UK | CHINA | MALAYSIA

How a computer works!

Problems when data get *into* a computer!

Simple parallel I/O on a PC or microprocessor

- Often need to get data into the PC’s data bus and into program from external device
- External device will put either 5V or 0V on (typically) eight wires (parallel data)
- Need to transfer it to the *data bus* for access by CPU.
- Problem:
  - When bus used for other purposes, external signals would *swamp/corrupt* data!

Simple parallel I/O on a PC or microprocessor

- Solution: use three-state (tri-state) buffer, has the states:
  - high
  - low
  - “high impedance” (i.e., open-circuit)

- When enabled:
  - if input is high, puts high signal onto bus
  - if input is low, puts low signal onto bus

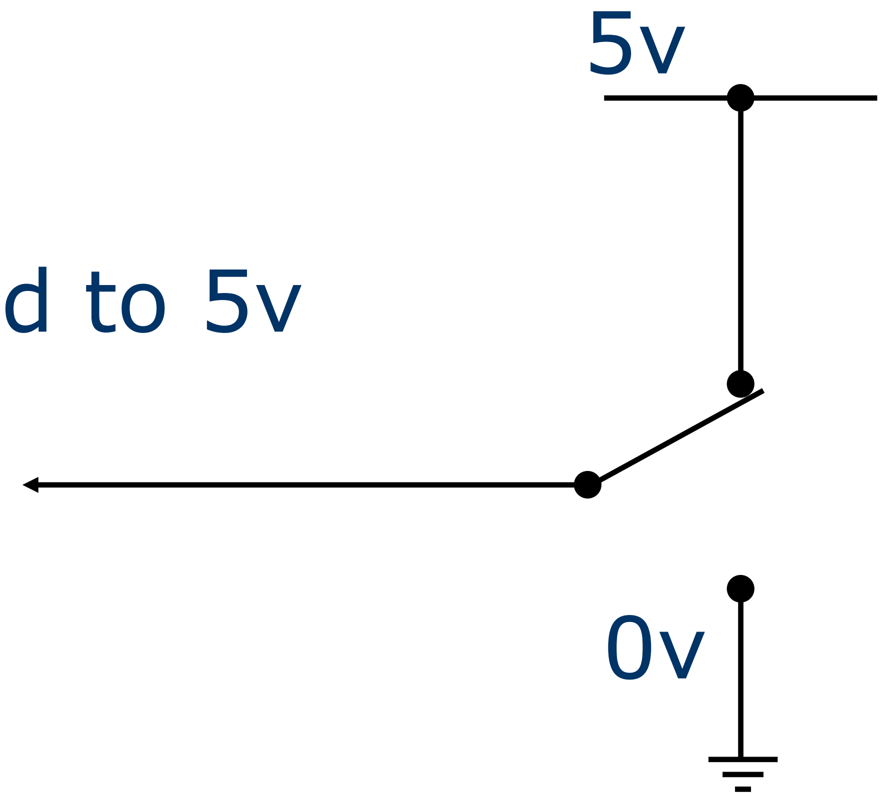
Simple parallel I/O on a PC or microprocessor

- But when disabled:
  - connection to data bus is broken ("high impedance state") like an open switch

- Enable/disable depends upon whether
  - the address on the address bus corresponds to the address identifying the interface
  - whether the data at the address is to be read or written (status of read/write line)

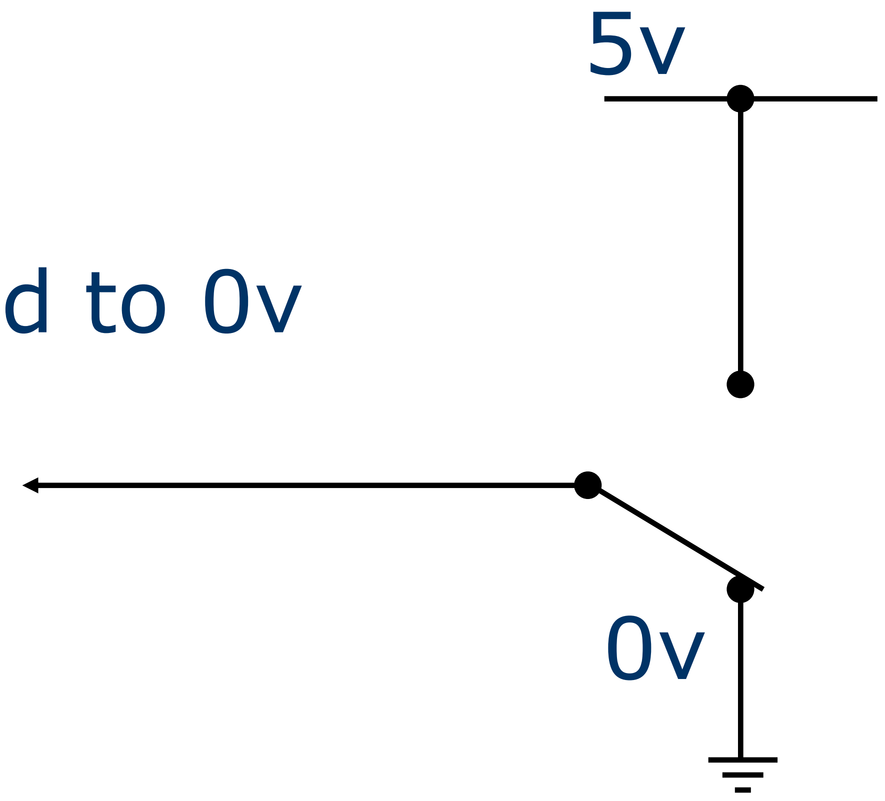
Equivalent behaviour of tri-state buffer output

Output connected to 5v
(high or logic 1)



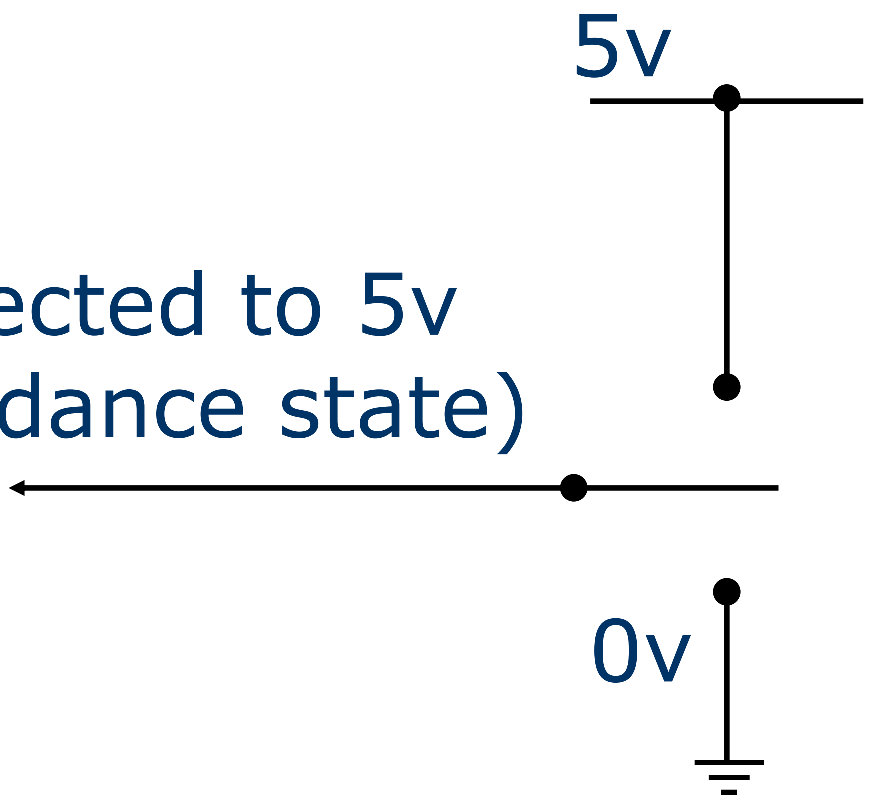
Equivalent behaviour of tri-state buffer output

Output connected to 0v
(low or logic 0)

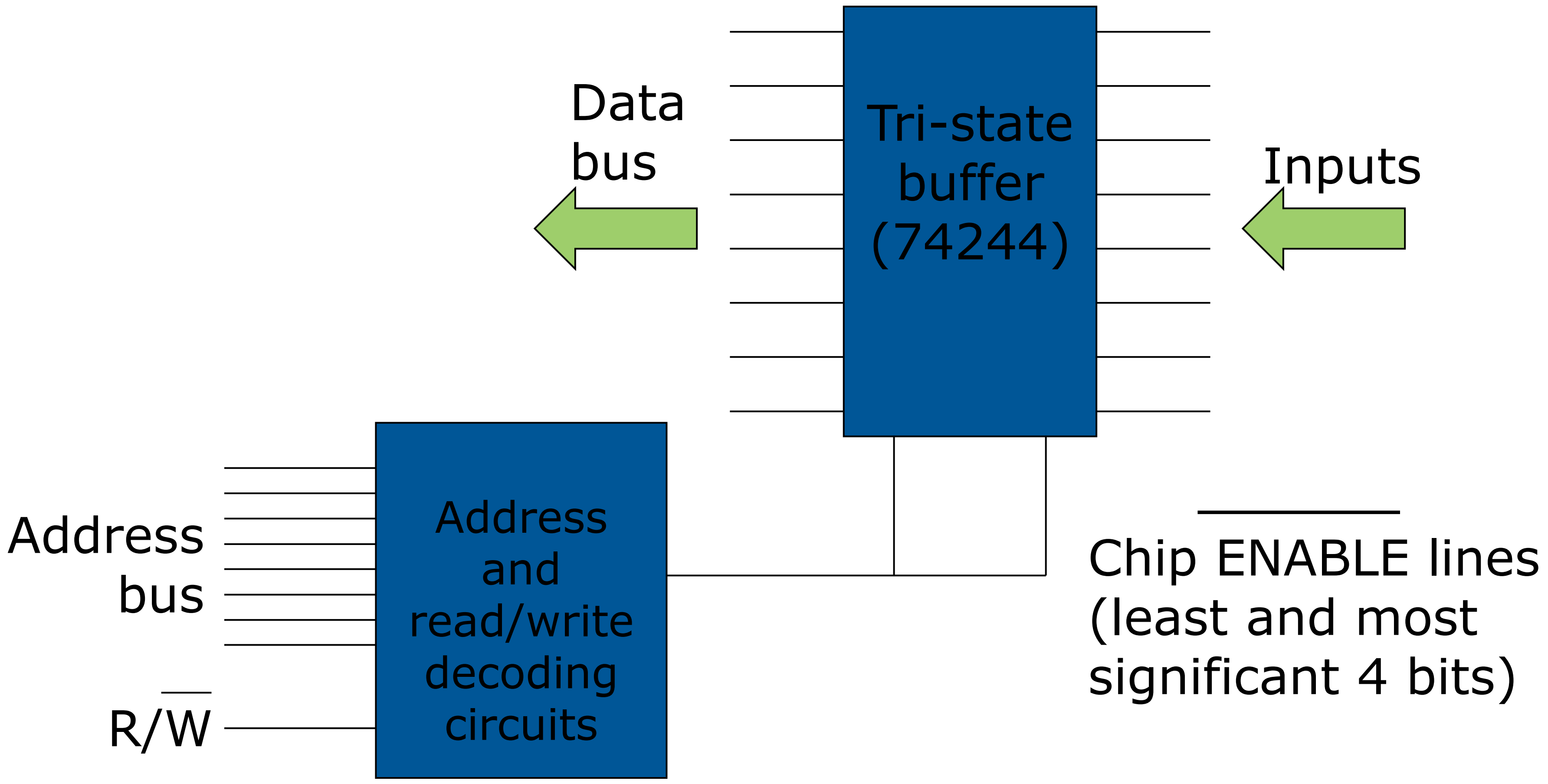


Equivalent behaviour of tri-state buffer output

Output not connected to 5v
or 0v (high impedance state)



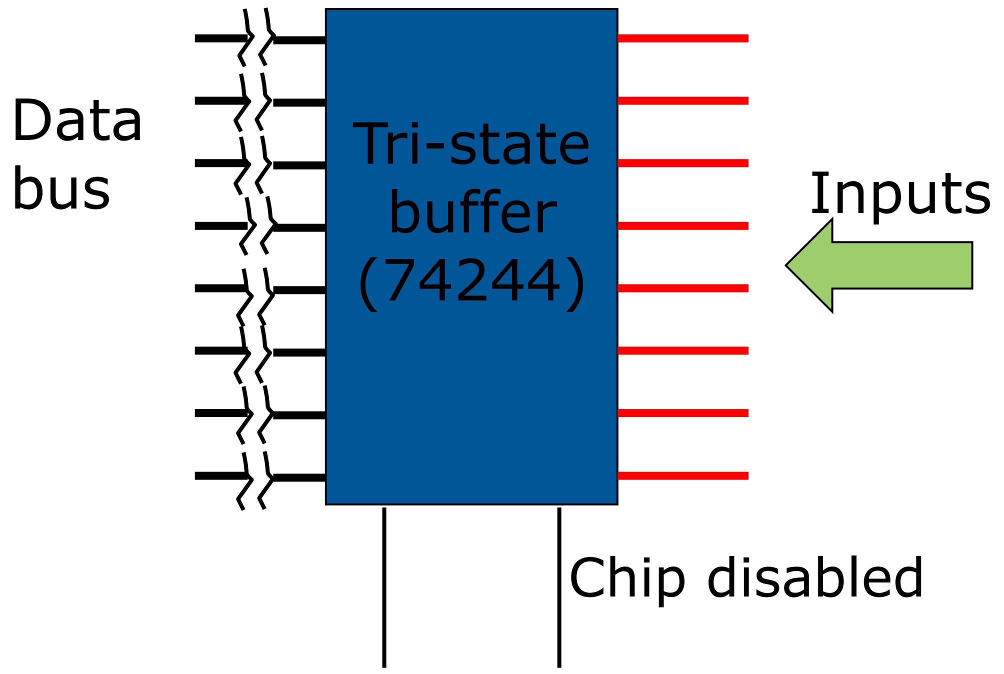
Simple digital input using tri-state buffer (e.g., 74244)


Data bus
Tri-state buffer (74244)
Inputs
Address bus
$R/\overline{W}$
Address and read/write decoding circuits
$\overline{\text{Chip ENABLE}}$ lines (least and most significant 4 bits)

University of Nottingham UK | CHINA | MALAYSIA

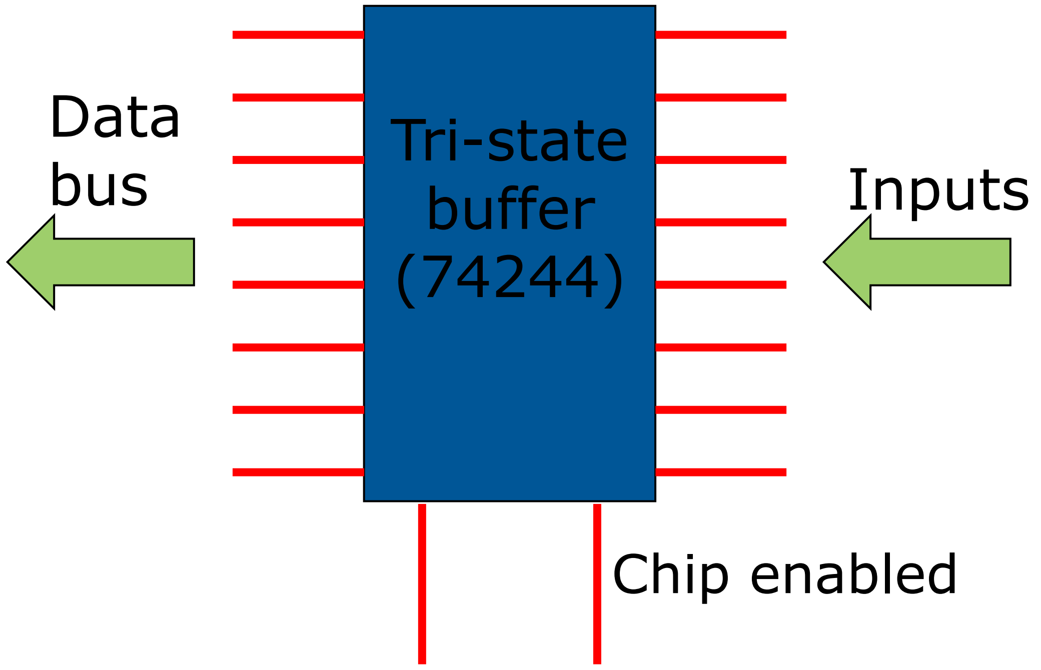
Simple digital input using tri-state buffer (e.g., 74244)

- Initially, all outputs form buffer are high impedance and data bus can "do its own thing" without interference from it.



Simple digital input using tri-state buffer (e.g., 74244)

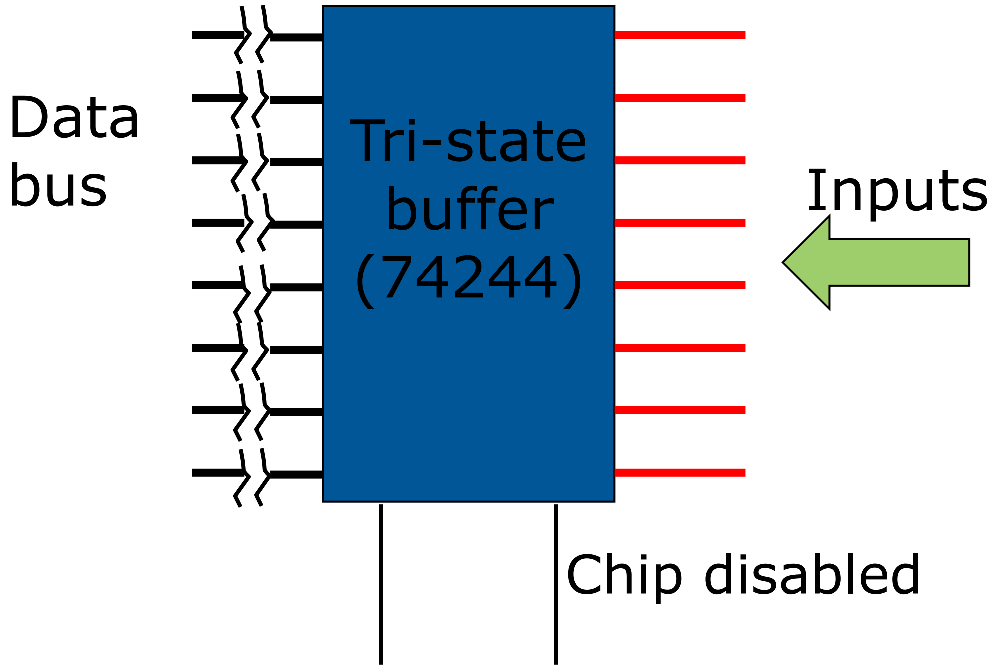
- When:
  - correct address is selected and
  - READ line is set
- data bus is set to the same values (high or low) as the input lines
- data are transferred to the bus



University of Nottingham
UK | CHINA | MALAYSIA

Simple digital input using tri-state buffer (e.g., 74244)

- When:
  - data have been read by (for example) the CPU and
  - address is no longer selected
- the 74244 outputs go "high impedance" (open-circuit) again
- disconnects from data bus.



University of Nottingham
UK | CHINA | MALAYSIA

How a computer works!

Problems when data get *out* of a computer!

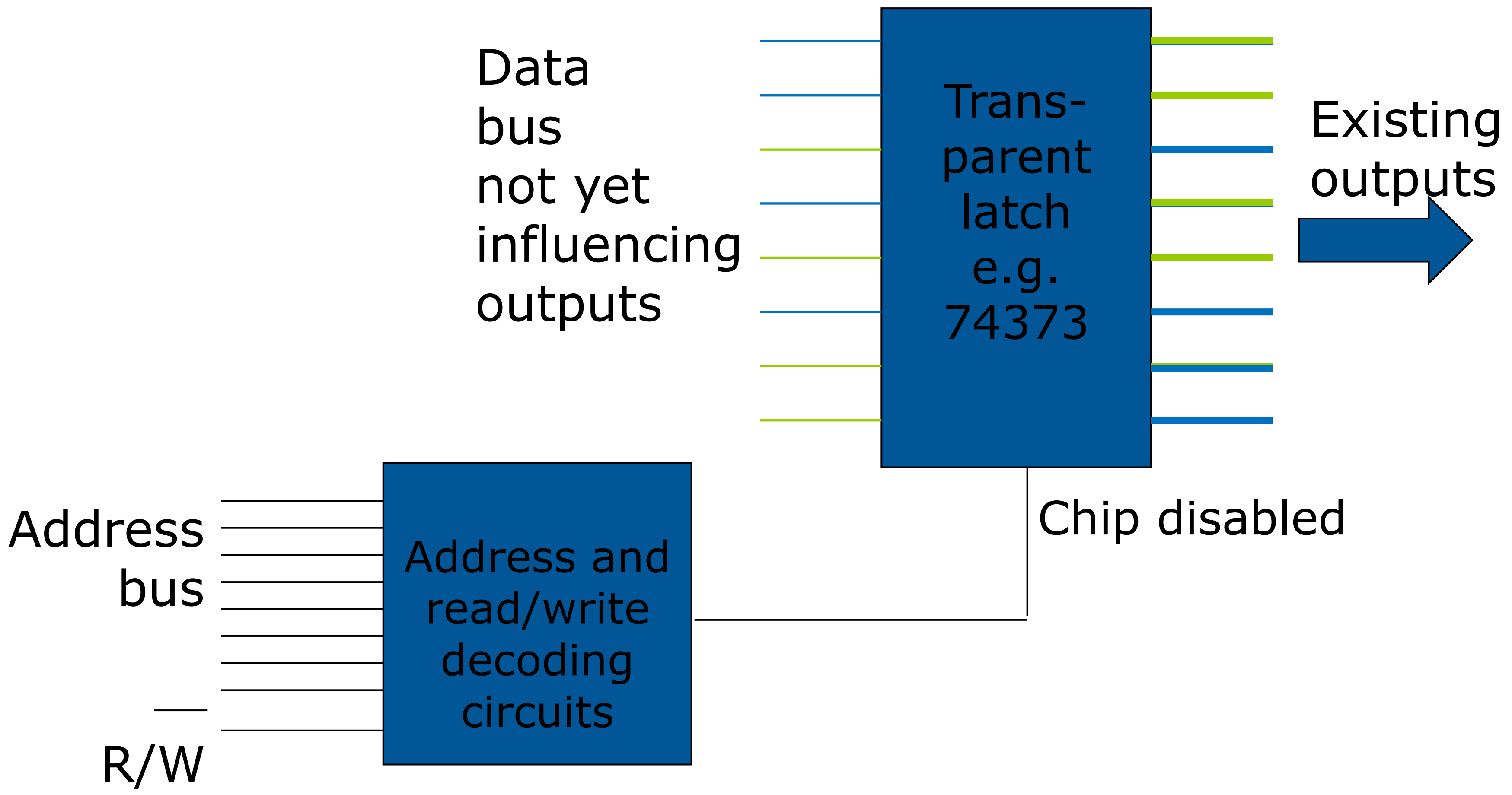
Simple digital output using a latch (e.g. 74373)

- When putting data out from PC we have a different problem
- Data only present on data bus for **a tiny fraction of a second**
- But we need data to persist on output port for as long as we wish

Simple digital output using a latch (e.g. 74373)

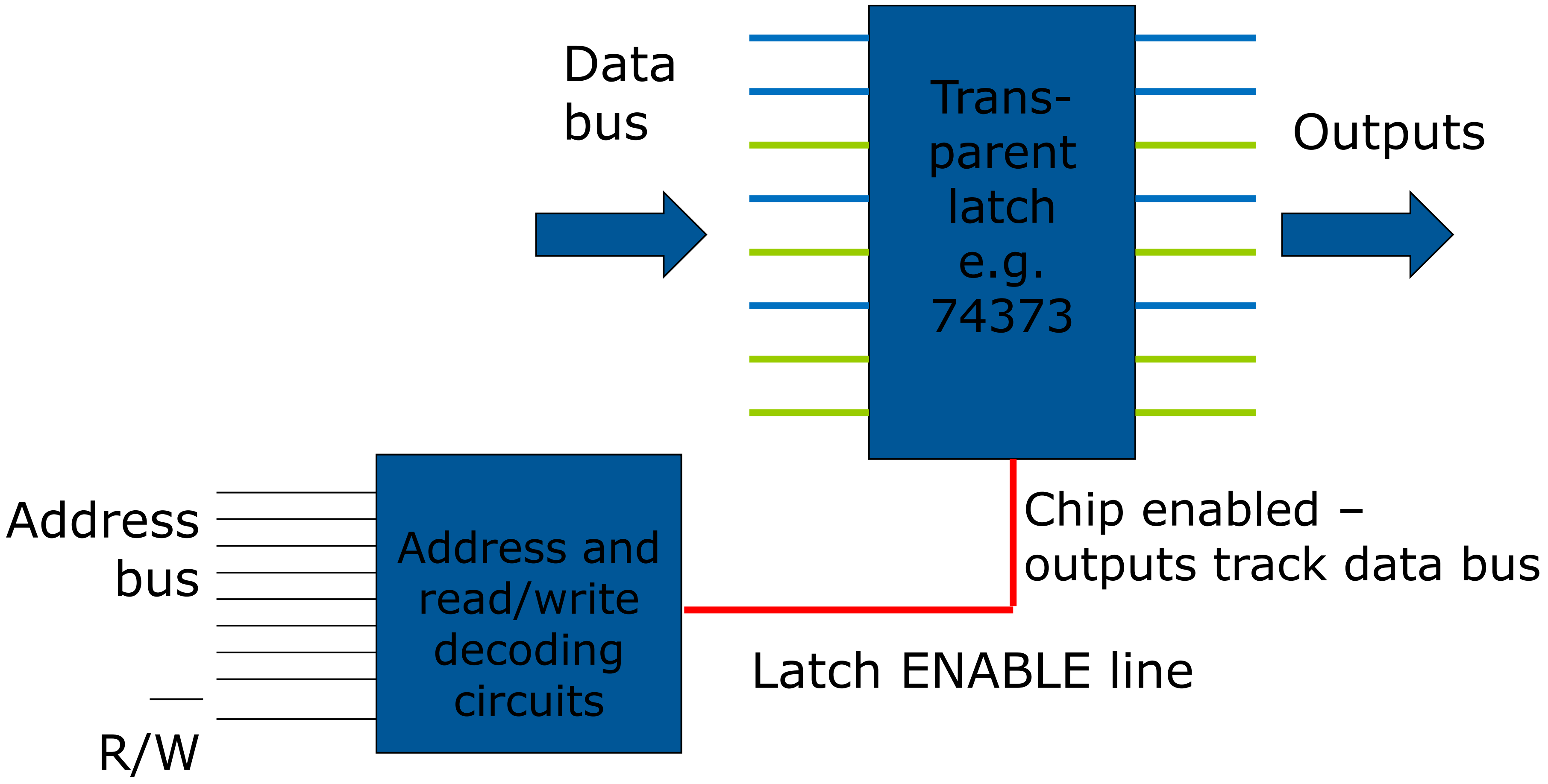
- Necessary to **"latch" the values on the data bus** so they persist on the output lines after chip ceases to be addressed
- Latch outputs follow data inputs (from the bus) when the latch ENABLE line is high.
- When the latch ENABLE line goes low, the outputs are stable (latched) – “freezes” value that was on bus when ENABLE went low

Simple digital output using a latch (e.g., 74373)

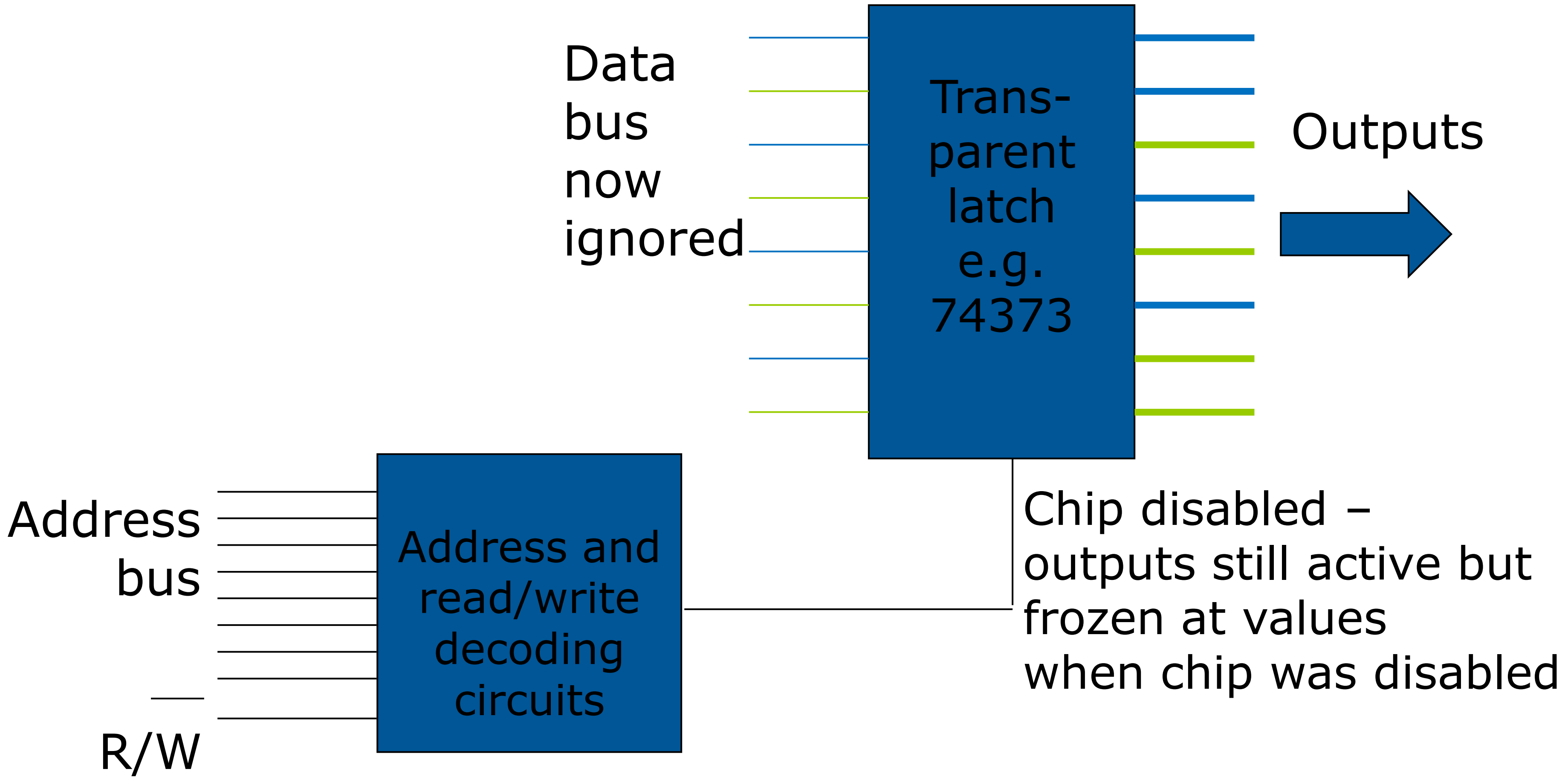

Data bus not yet influencing outputs
Trans-parent latch e.g. 74373
Existing outputs
Chip disabled
Address bus
$\overline{R/W}$
Address and read/write decoding circuits

University of Nottingham
UK | CHINA | MALAYSIA

Simple digital output using a latch (e.g., 74373)


Data bus
Trans-parent latch e.g. 74373
Outputs
Address bus
$\overline{R/W}$
Address and read/write decoding circuits
Chip enabled – outputs track data bus
Latch ENABLE line

Simple digital output using a latch (e.g., 74373)


Data bus now ignored
Trans-parent latch e.g. 74373
Outputs
Address bus
$\overline{\text{R/W}}$
Address and read/write decoding circuits
Chip disabled – outputs still active but frozen at values when chip was disabled

University of Nottingham
UK | CHINA | MALAYSIA

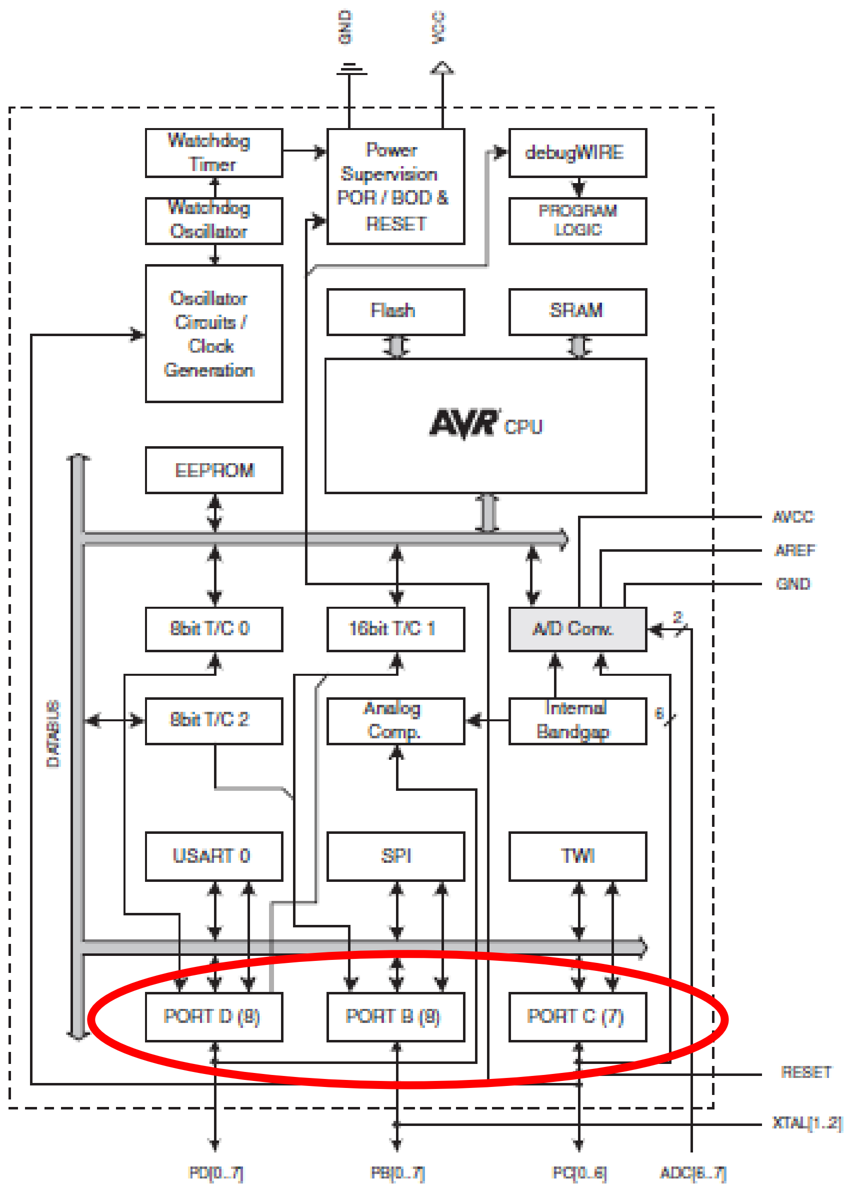
Practical input/output devices

Practical input/output devices: Arduino

- In reality, we tend to use multi-purpose, programmable input/output devices
- This is true for the Arduino, but i/o device are built-in
- In the Arduino all pins can be configured separately for digital input or output

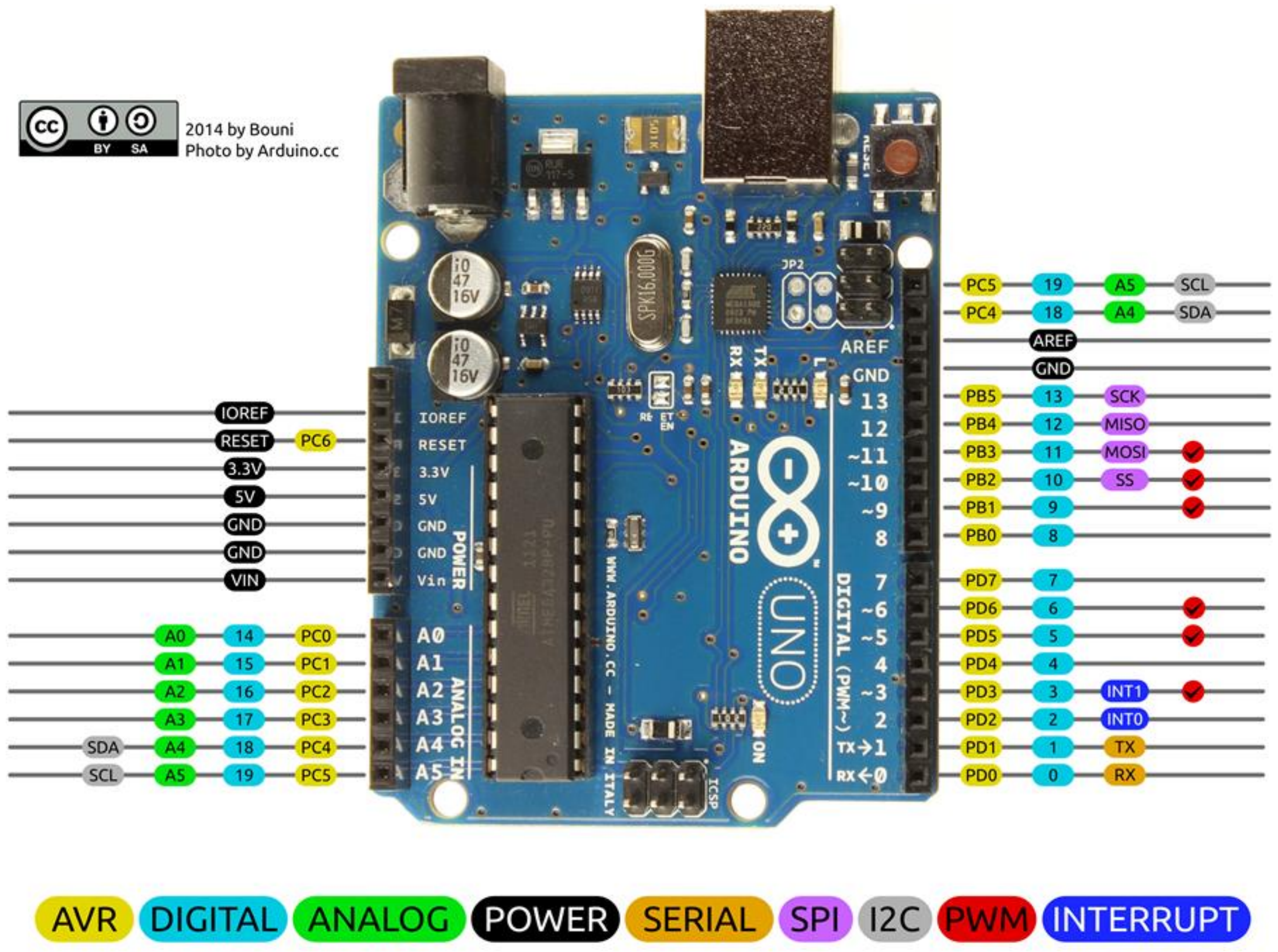
Input/output ports on the Arduino

- The input and output pins are grouped into ports (termed B, C and D on the Uno)
- Each port (8 pins) has an address and behaves exactly like memory



Practical input/output devices: Arduino Uno

- Port B: pins 8-13
- Port C: pins A0-A5 + reset
- Port D: pins 0-7

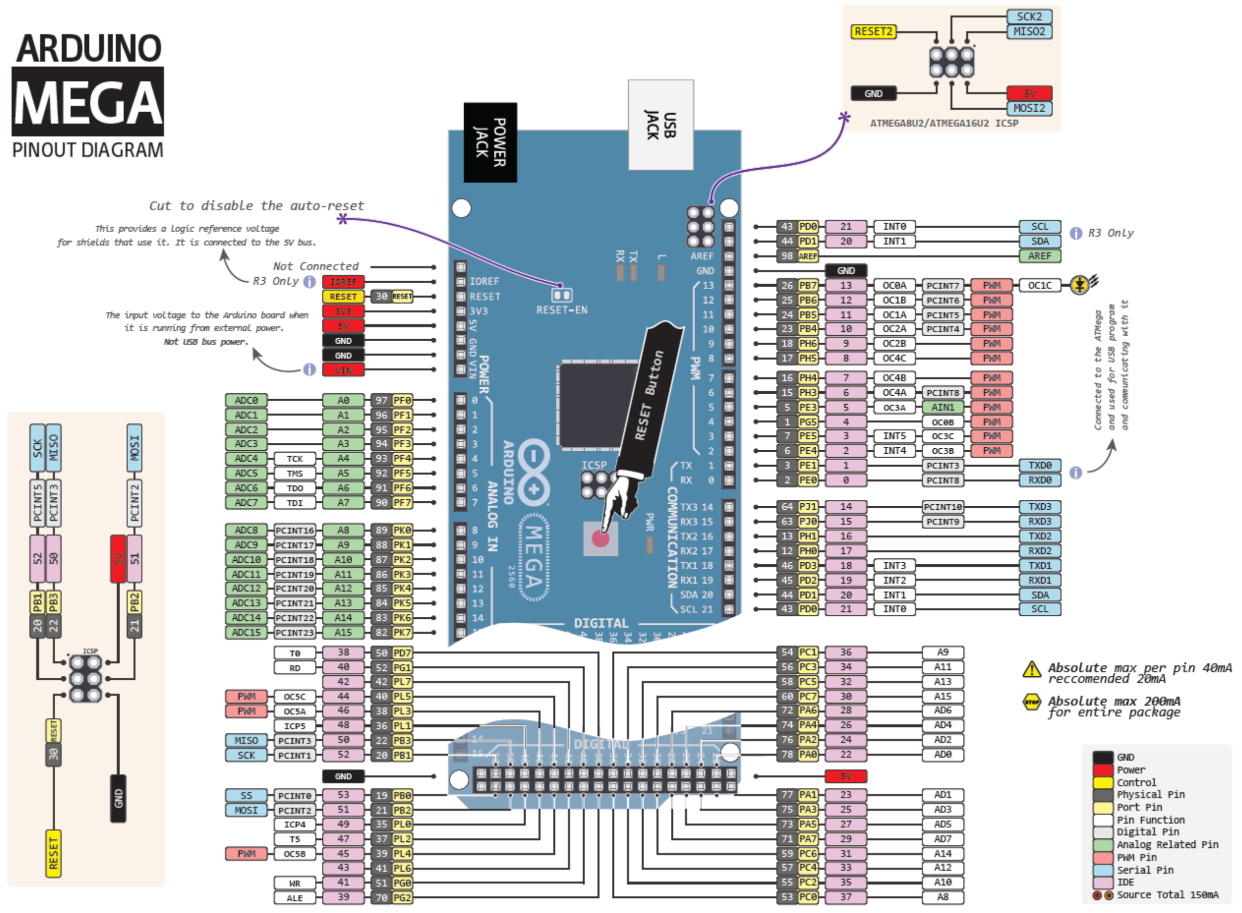


https://github.com/Bouni/Arduino-Pinout

University of Nottingham UK | CHINA | MALAYSIA

Practical i/o devices: Arduino Mega

- Ports A-L (not I), not all connected to pins on Arduino



https://lynx2015.files.wordpress.com/2015/08/arduino-mega-pinout-diagram.png

University of Nottingham
UK | CHINA | MALAYSIA

Simple digital input on the Arduino

By Arduino language

Simple *digital input* on the Arduino

- In principle we need a function which will handle processes we’ve talked about e.g.,
  - Take a value of address or port location on the data bus
  - Generate the necessary signals on the control bus (i.e., read)
  - Give out an integer signal (*to the user*) equal in value to the binary value of the hi/lo combination present on the input port

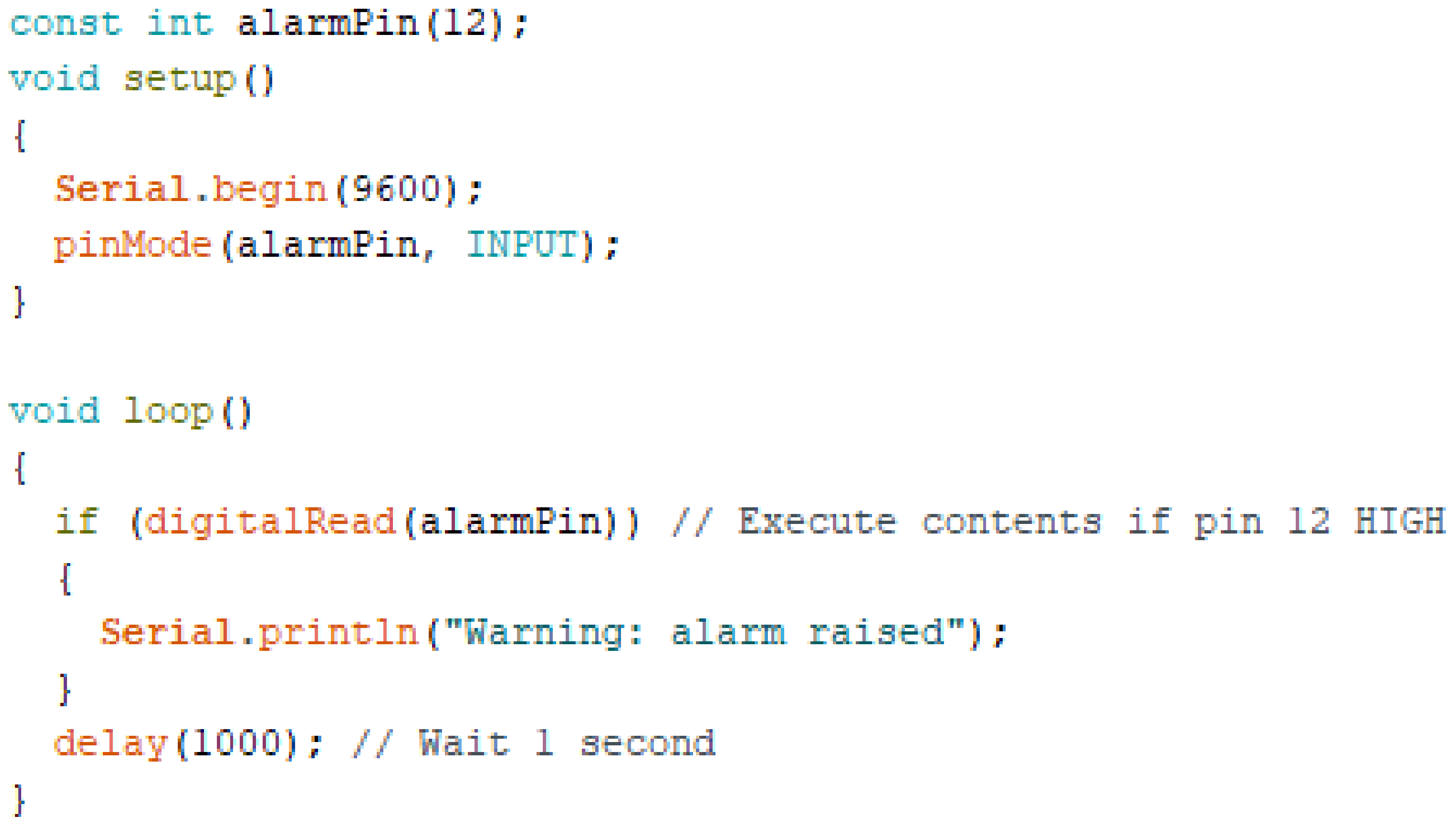
Simple *digital input* on the Arduino

- Most Arduino users are not interested in the address of a port, and are only interested in the pin number on the Arduino e.g., pin 1-13 or A0-A5 on Uno
- The "easy" way is to do input one pin at a time using the **`digitalRead()`** function:

```
int digitalRead(pin);
```

- Returns **`HIGH`** (1) or **`LOW`** (0) depending on whether input line is low or high

Example: simple *digital input*



```
const int alarmPin(12);
void setup()
{
  Serial.begin(9600);
  pinMode(alarmPin, INPUT);
}

void loop()
{
  if (digitalRead(alarmPin)) // Execute contents if pin 12 HIGH
  {
    Serial.println("Warning: alarm raised");
  }
  delay(1000); // Wait 1 second
}
```

Simple *digital output* on the Arduino

- The “easy” way is to output one pin at a time using the **`digitalWrite()`** function:

```
void digitalWrite(pin, value);
```

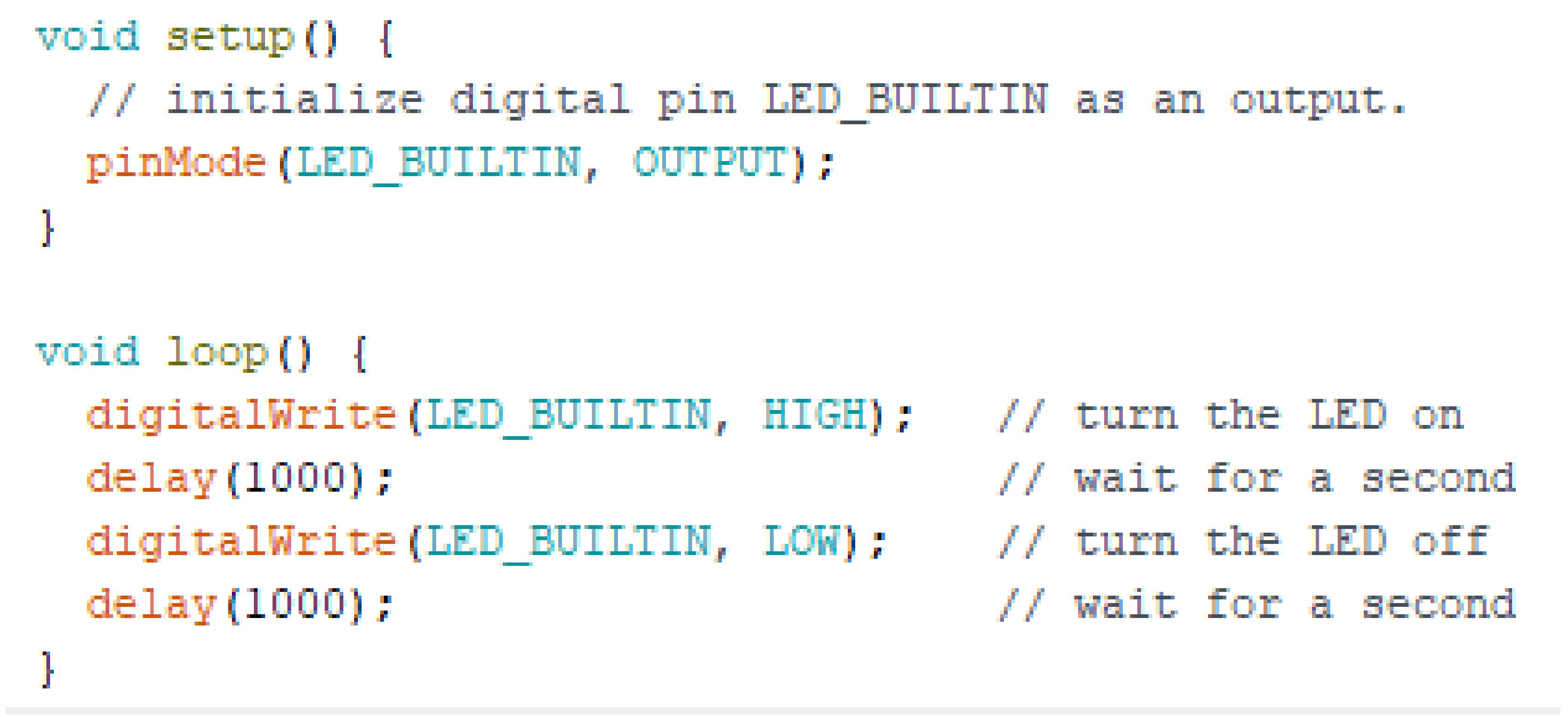
- Before we can use digitalWrite we need to set the pin as output:

```
    void pinMode(pin, mode);
```

where mode is **`INPUT, OUTPUT`**

University of Nottingham
UK | CHINA | MALAYSIA

Example: our old friend “Blink”



```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

More advanced input/output

- Above approach works fine but is slow
- There is a MUCH quicker way which involves bypassing the easy-to-use API for Arduino
- Makes use of low-level facilities for programming the Atmega chip:

More advanced input/output

- Above approach works fine but is slow
- There is a MUCH quicker way which involves bypassing the easy-to-use API for Arduino
- Makes use of low-level facilities for programming the Atmega chip:

Registers!

Simple digital input on the Arduino

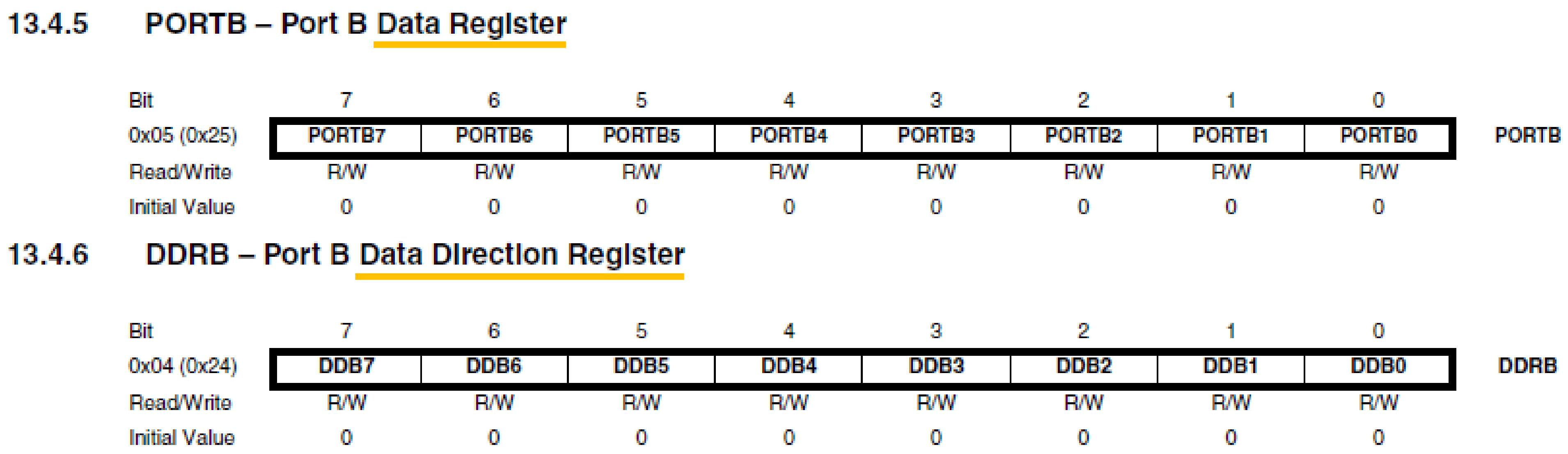
Registers

What are registers?

- *Memory-mapped* locations containing:
  - Digital input or output ports (as we described earlier)
  - Control registers of bits (flags) for configuring port e.g., direction of bits for input and output
  - Other control registers for all kinds of in-depth functionality.

What are registers?

- All this is described in the datasheet (the manual for the Atmega2560 chip)



13.4.5 PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

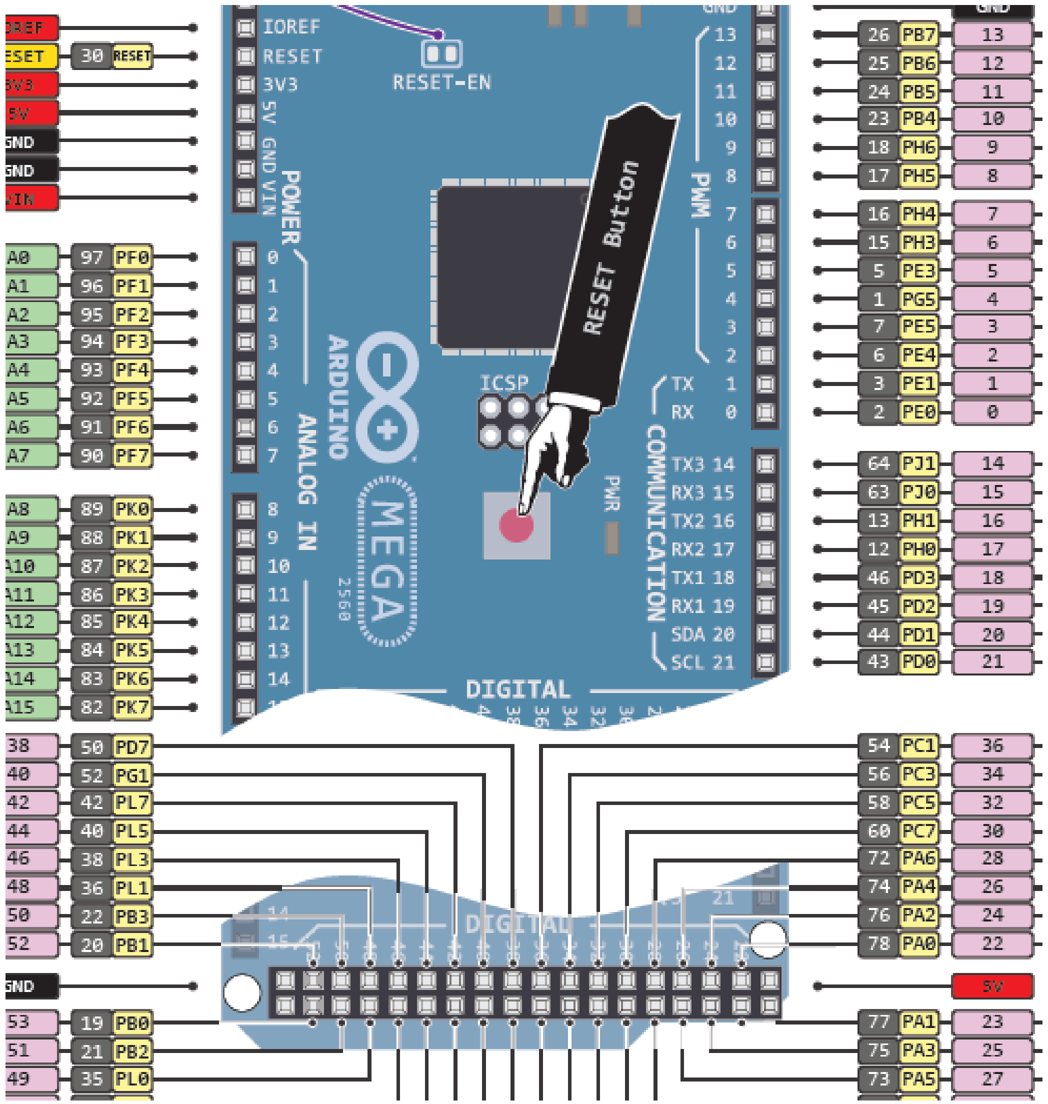
13.4.6 DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Any information you need on the chip is there – trust me!

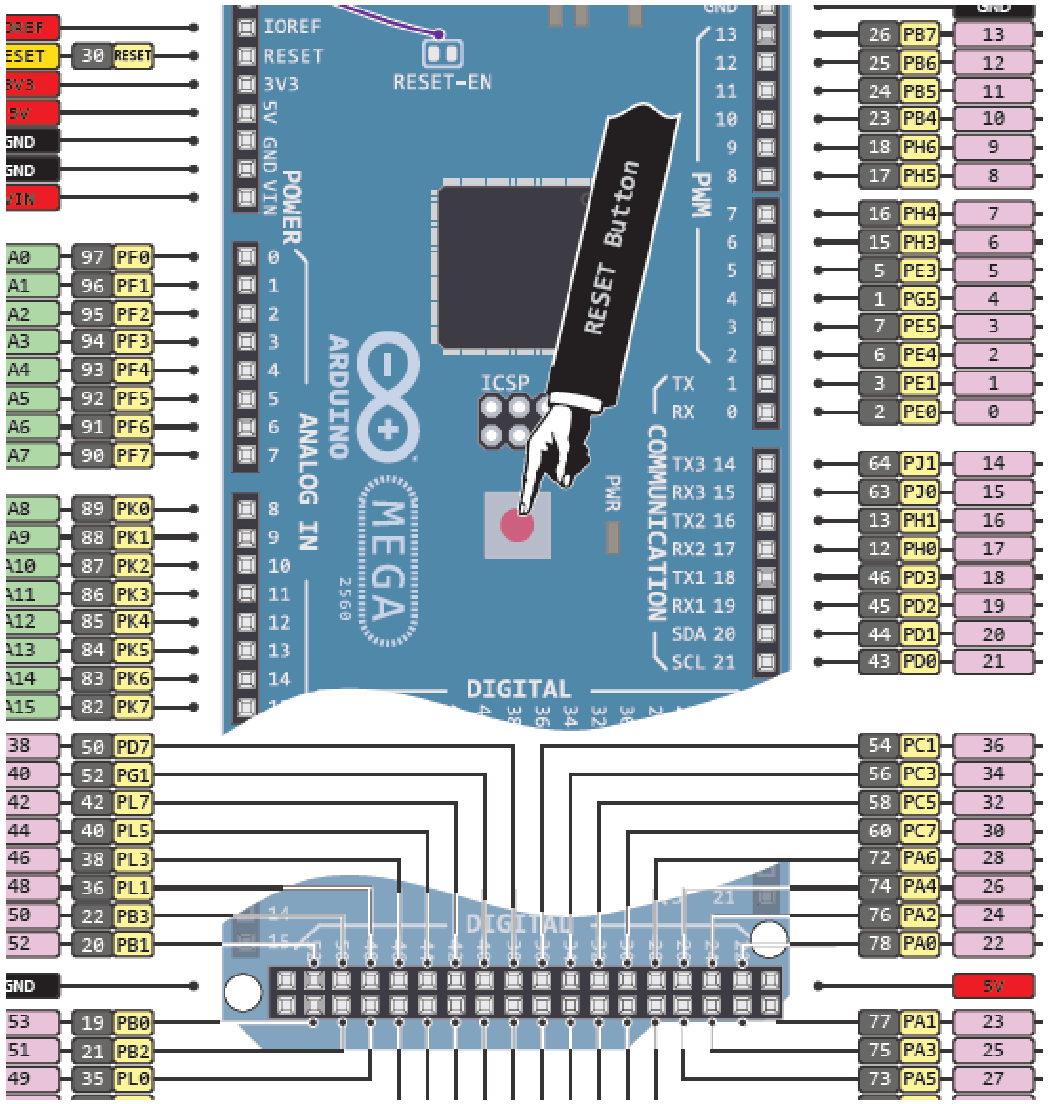
Example: Blink using registers

- We'll use two registers on the Arduino Mega:
- The port itself (the Data Register):
  - **Port B** of Mega covers pins 53-50 & 10-13
  - So, Pin 13 (the LED pin) is Port B bit 7
  - Can switch this on by writing binary 10000000 (0x80) to the address of Port B which is 0x25, given the code PORTB

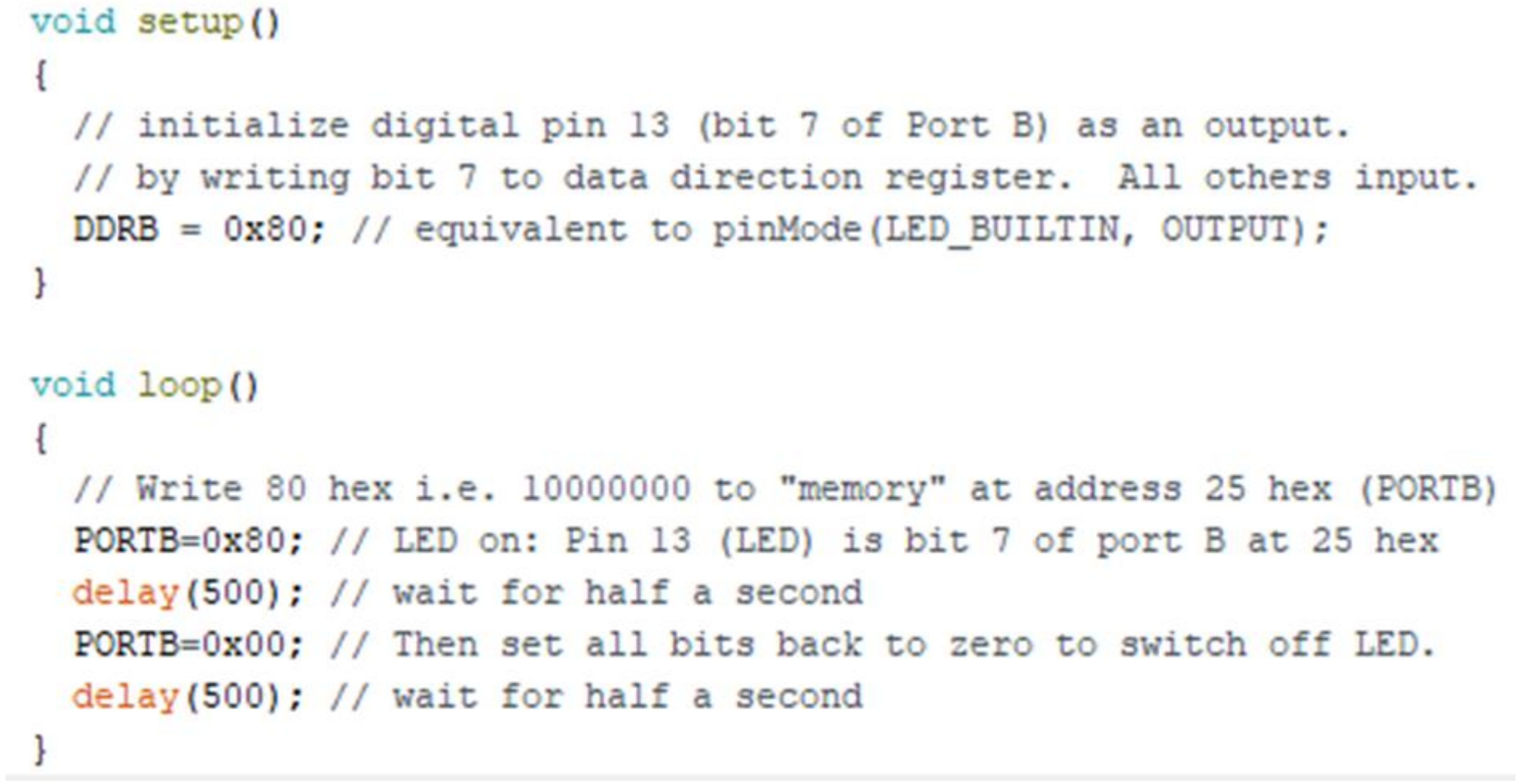


Example: Blink using registers

- We'll use two registers on the Arduino Mega:
- Before we do this, we set the pin directions (0=input, 1=output) using the Data Direction Register, given the code DDRB=0x80



Example: Blink using registers

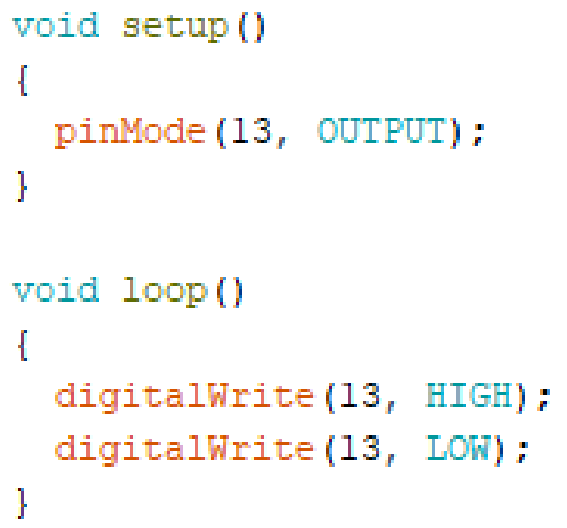


```
void setup()
{
  // initialize digital pin 13 (bit 7 of Port B) as an output.
  // by writing bit 7 to data direction register.  All others input.
  DDRB = 0x80; // equivalent to pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  // Write 80 hex i.e. 10000000 to "memory" at address 25 hex (PORTB)
  PORTB=0x80; // LED on: Pin 13 (LED) is bit 7 of port B at 25 hex
  delay(500); // wait for half a second
  PORTB=0x00; // Then set all bits back to zero to switch off LED.
  delay(500); // wait for half a second
}
```

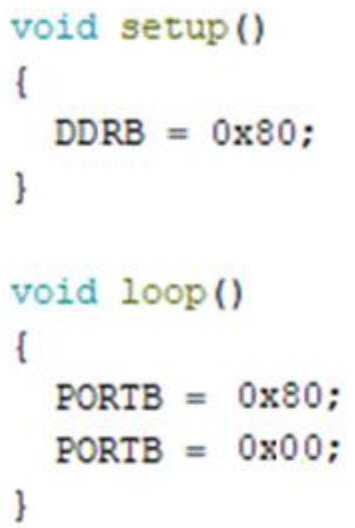
Does it really matter?

- Many of the clever things we want to illustrate can only be done well with registers
- Register-level programming can run *much faster*, important for microprocessors.
- Compare these "high speed Blink" programs:



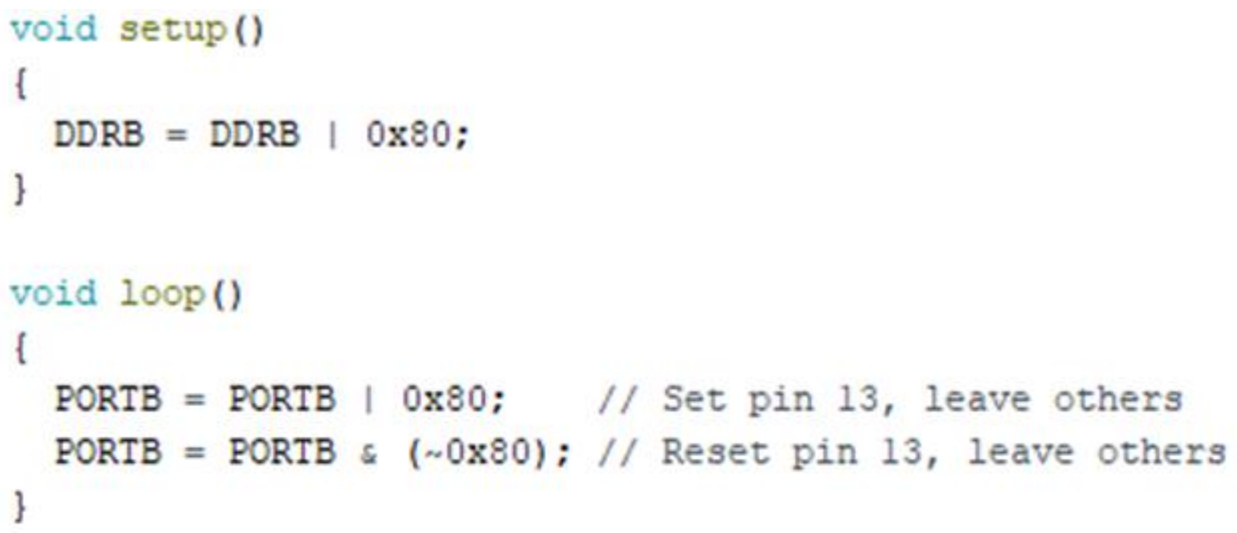
```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  digitalWrite(13, LOW);
}
```



```
void setup()
{
  DDRB = 0x80;
}

void loop()
{
  PORTB = 0x80;
  PORTB = 0x00;
}
```



```
void setup()
{
  DDRB = DDRB | 0x80;
}

void loop()
{
  PORTB = PORTB | 0x80;     // Set pin 13, leave others
  PORTB = PORTB & (~0x80); // Reset pin 13, leave others
}
```

Simple digital input on the Arduino

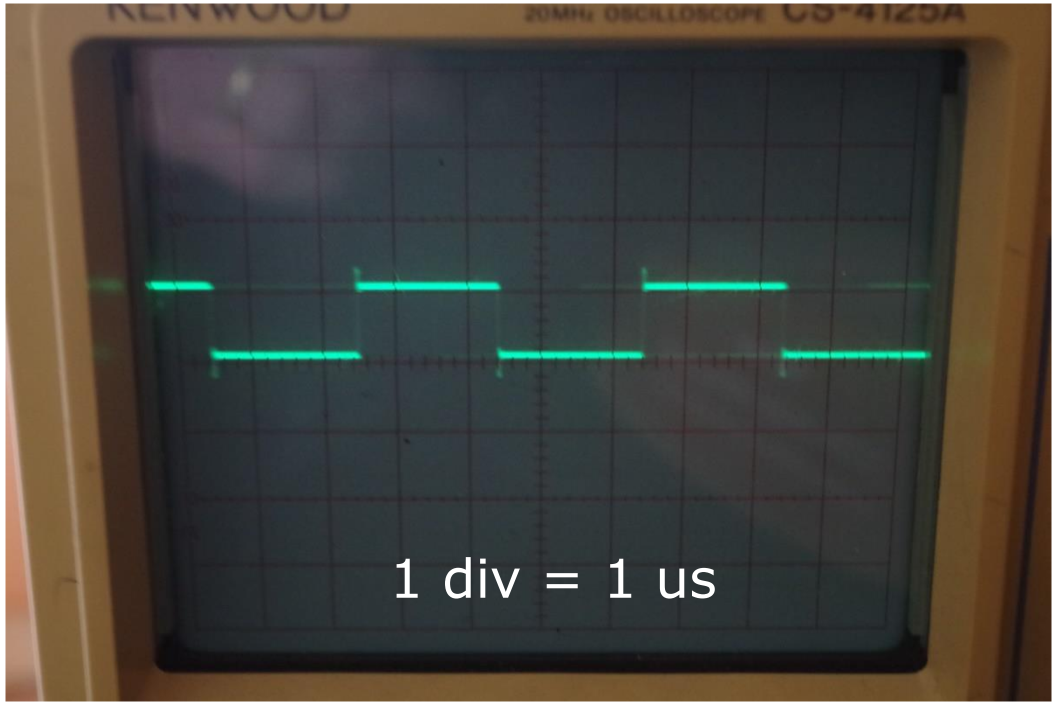
Let us compare!

First, the “easy” way to blink as fast as possible

- The fastest we can “blink” with **`digitalWrite()`** is about 0.25 MHz (250 kHz) as the function is quite slow to run

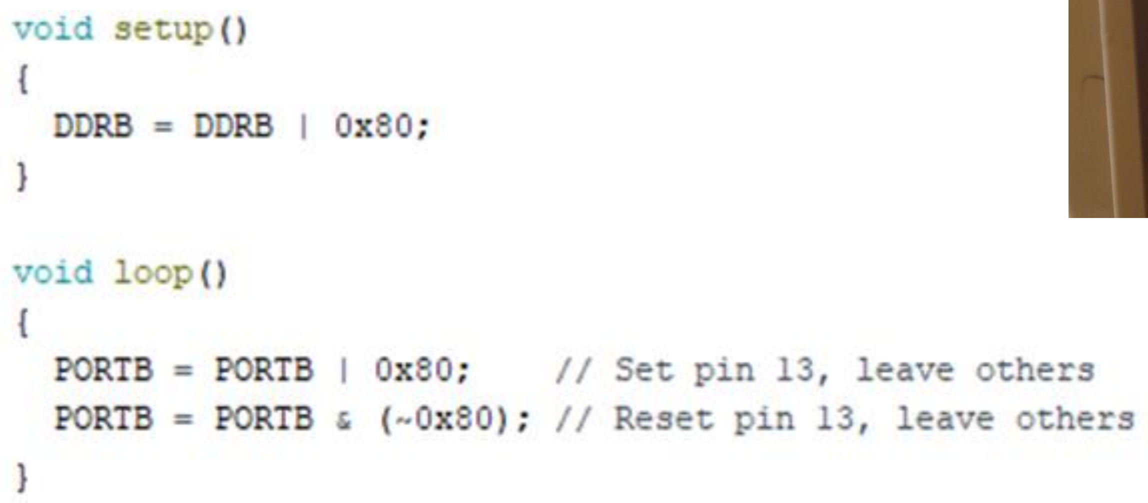
```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  digitalWrite(13, LOW);
}
```



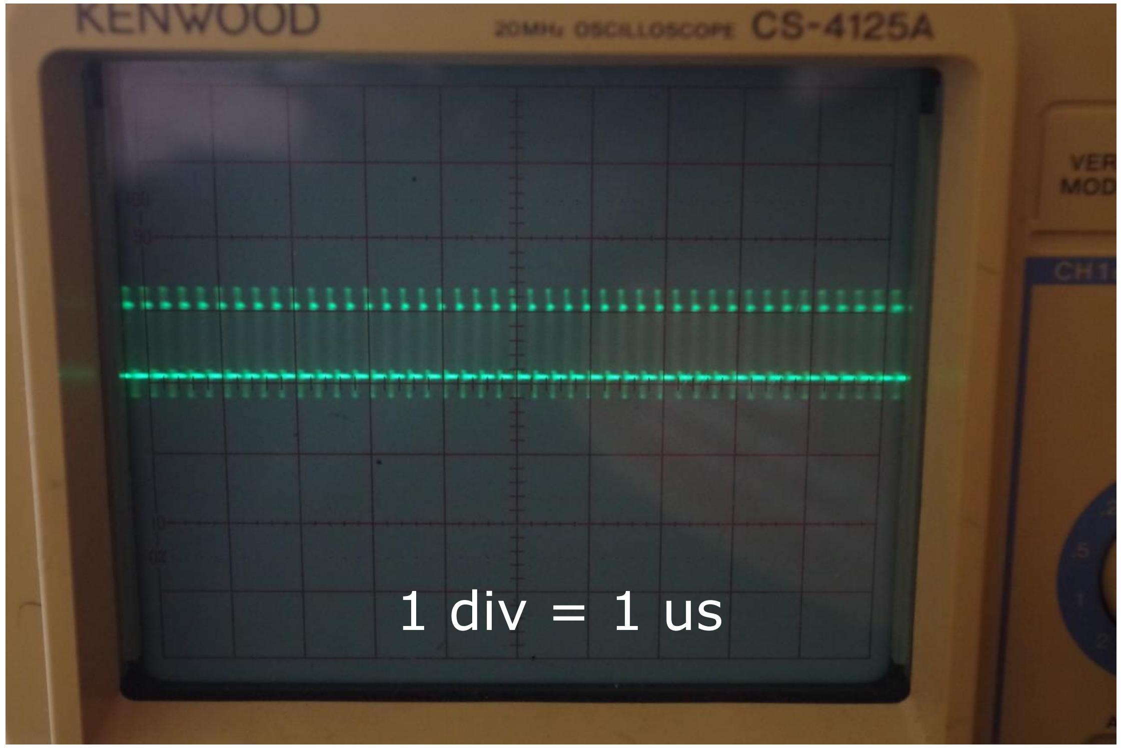
Now using direct register access

- With direct register access fastest is about 4 MHz



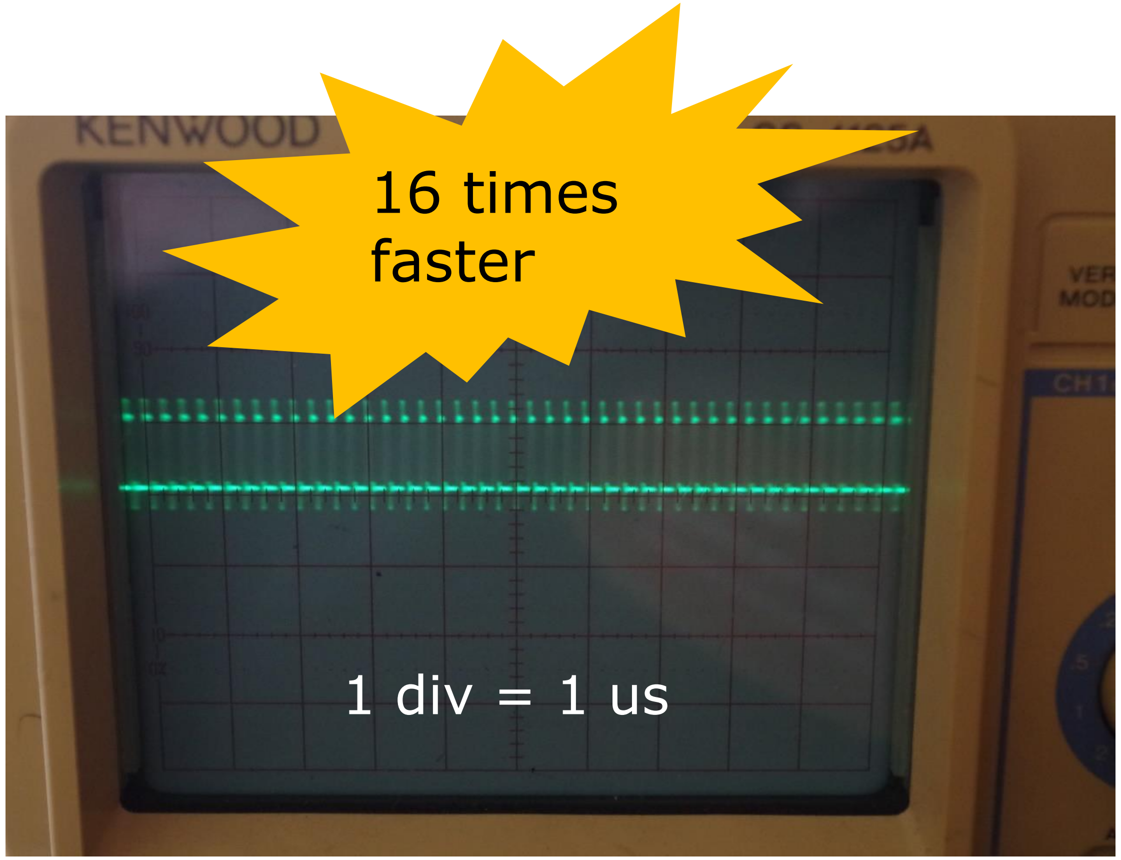
```
void setup()
{
  DDRB = DDRB | 0x80;
}

void loop()
{
  PORTB = PORTB | 0x80;    // Set pin 13, leave others
  PORTB = PORTB & (~0x80); // Reset pin 13, leave others
}
```



Now using direct register access

- With direct register access fastest is about 4 MHz



```
void setup()
{
  DDRB = DDRB | 0x80;
}

void loop()
{
  PORTB = PORTB | 0x80;     // Set pin 13, leave others
  PORTB = PORTB & (~0x80); // Reset pin 13, leave others
}
```

University of Nottingham
UK | CHINA | MALAYSIA

Hardware timer/counters

Introduction

Real-time input/output on non-RT system: hardware timer/counters

- Often need interfacing which requires:
  - Detection of specific events, including counting of pulses or measuring rates of pulses
  - Generation of timed events such as pulses for driving a stepper motor
  - Signals with specific waveforms e.g., pulse width modulation (PWM) to give an effective value of voltage.

Shortcomings of software-controlled input/output

- In principle:
  - Can use repeated “polling” of inputs to monitor for input events
  - Can create pulses using i/o and software – write high output, wait, write low output, etc.
- Not advisable with general-purpose operating systems e.g., Windows
- Pulse generation for stepper motor - slowed down when mouse was moved!
- Even in Arduino, may not be fast enough

Time-critical input/output: hardware timer/counters

- Timer/Counters: low cost and fairly easy
- Useful for real time generation of pulses and frequency counting on PC
- Several built into the Arduino Uno, Mega etc., can be configured for different jobs
- As the 8254, available on boards that simply slot into back of PC, but are slightly different from those on Arduino
- Other more specialist counters are available e.g. HCTL 2022, LS7366R

Time-critical input/output: hardware timer/counters

- We will talk about timer-counter chips in more detail next time
- They underlie several jobs we will do on the Arduino e.g., generation “analogue” (PWM) outputs for driving motors at variable speed
- Will use a specialist timer counter (LS7366R) to measure and control position of a rotating shaft.

Summary

- Examined briefly how computer works
- Explored circuitry used in digital logic: TTL, CMOS, push-pull, open collector
- Examined the need for interfaces to get digital data onto and off the data bus
- Examined simple input/output interfaces
- Explained why simple interface no good for counting events or measuring frequencies, and hence the need for timer-counters