



# CS3243: IoT Devices and Applications

*Module Introduction*

# What is the Internet of Things (IoT)

- *According to Wikipedia...*

“**The Internet of Things (IoT)**, also called **The Internet of Objects**, refers to a wireless network between objects, usually the network will be wireless and self-configuring, such as household appliances”

- *According to The World Summit on Internet Society...*

“By embedding short-range mobile transceivers into a wide array of additional gadgets and everyday items, **enabling new forms of communication between people and things, and between things themselves.**”

# Internet of Things: How it affects our lives

# Module outline

1. *Introduction to IoT systems and applications*
2. *IoT layered stack, concepts of edge processing, Fog layer and Cloud Layer*
3. *Edge device design and construction, sensors and transducers, interfacing techniques, device resource management*
4. *M2M communication systems and protocols – MQTT, XMPP, AMQP, ZigBee, Bluetooth, and WiFi*
5. *Common IoT cloud platforms – AWS, Alexa, Azure IoT Hub etc.*
6. *Using IoT prototyping platforms (Arduino, Node MCU, Raspberry PI, HackRF etc.*
7. *Debugging and testing IoT devices and applications*
8. *IoT application development project*

# Learning outcomes

- *After successful completion of this course module, students should be able to:*
  1. *Describe the architecture and fundamentals of IoT systems from an end-to-end basis*
  2. *Describe IoT system in terms of different aspects such as acquisition, actuation, communication, computing, and storage domains*
  3. *Evaluate and select suitable modules and processing layers for an IoT application*
  4. *Design, construct test IoT devices and applications or a given requirement*

# Lecture format & Assessment

- *Online / Physical lectures*
- *Demonstrations*
- *Self-study and some research work*
- *Continuous assessments and related work*
- *Project study*
- *Continuous Assessment: 30%*
- *Final assessment: 70%*
  - *50% on a design problem*
  - *10% directly from notes*
  - *10% class work related activities*

# Recommended reading

- *Designing the Internet of Things*  
*Adrian McEwen & Hakim Cassimally*
- *Internet-of-Things (IoT) Systems: Architecture, Algorithms, Methodologies*  
*Dimitrios Serpanos and Marilyn Wolf*
- *Introduction to Internet of Things: A definite guide to learn IoT – enabling technologies, connectivity, protocols and cloud*  
*Monisha Macharla Vasu*
- *Building Arduino Projects for the Internet of Things*  
*Adeel Javed*



**QUESTIONS ?**





# CS3243: IoT Devices and Applications

*Introduction to the IoT World*

# What is IoT

# IoT: Internet of Things



IoT is a new concept that is based on existing and matured technologies

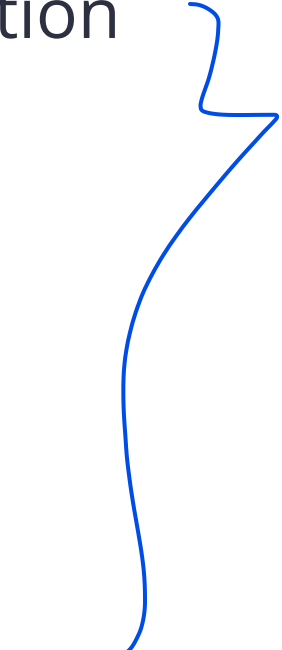
It focuses on the value addition in a network (Internet) of sensors and actuators (Things)  
Sometimes also referred to as a “System of Systems”

# A brief history of IoT development

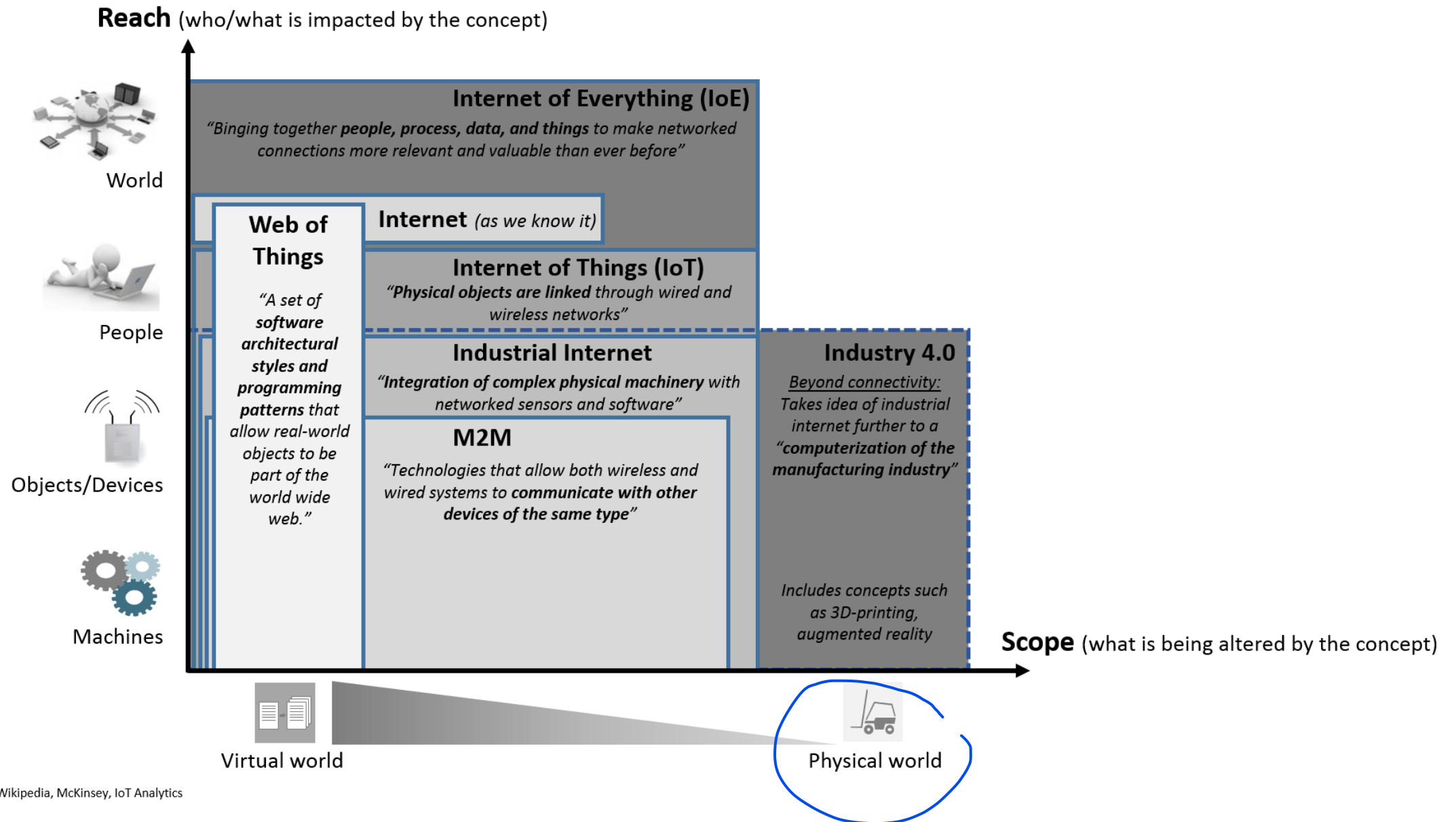
- Initial concepts on devices with embedded “smartness” started to appear even in the late seventies with the popularity of **microprocessors and microcontrollers**.
- In 1989 a modified **Coca-Cola vending machine** at Carnegie Mellon University became the first embedded system (Thing) to be connected to the **ARPANET (Internet)**.
- In early 2000, new technologies such as **RFID (Radio frequency identification)** and cellular communication networks provided momentum for IoT applications – by solving two key issues, continuous power and communication.
- The term “Internet of Things (**IoT**)” was introduced by **Kevin Ashton** 1999.
  - Initially, the Internet was for People-to-People communication. Later, when devices started to become more intelligent it became also a platform for “**Machine-to-Machine**” communication
- Availability of low-cost, high powered embedded processors make IoT to reach consumer device market in the second decade of the current century.
  - *Popularity and availability of mobile data communication networks too a contributing factor.*
- IOT has become a common household-thing, that we use everyday either knowingly or unknowingly.



# Other concepts similar to IoT

- Cisco has been driving the term **Internet of Everything** (IoE). Intel initially called it the “**embedded internet**”.
  - Other terms that have been proposed but don't mean exactly all the same are:
    - M2M (Machine to machine) communication
    - Web of Things
    - Industry 4.0
    - Industrial internet (of Things)
    - Smart systems
    - Pervasive computing
    - Intelligent systems
- 

# IoT And Other Concepts



# The world of IoT

Smart home

Smart Grid

Wearables

Industrial  
Internet

Smart City

Connected  
Vehicles

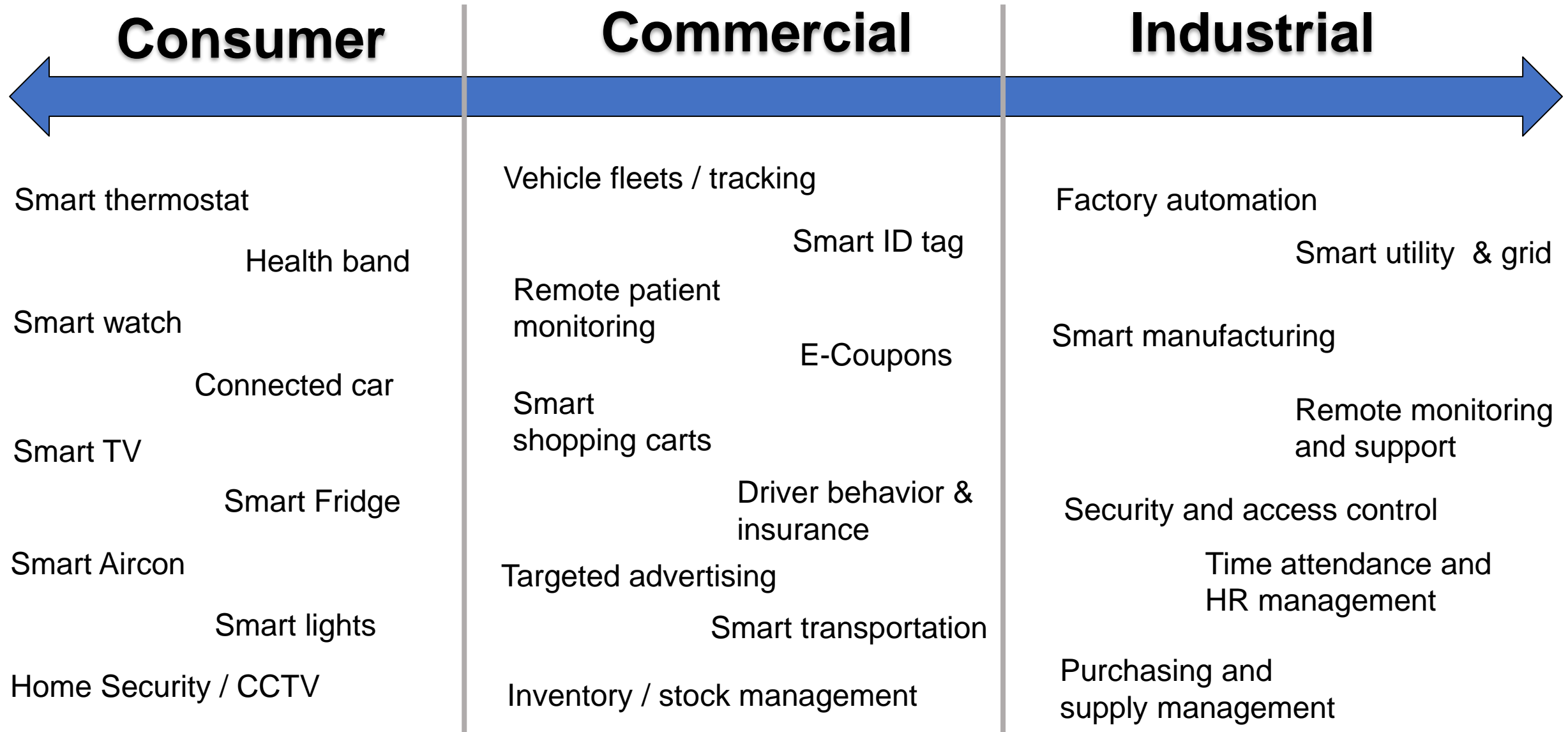
Connected  
Health

Smart Retail

Smart  
Supply  
Chain

Smart  
Farming

# IoT Sub domains





# Common IoT Applications

## Energy

- Smart meters
- Distribution automation
- EV charging Mgt
- Solar power
- Battery Storage Mgt

## Automotive

- Navigation
- Tolling
- Traffic management
- Parking systems
- Drive assistance systems
- OBD and assistance

## Cities & Communities

- Public transport
- Smart buildings
- Smart government
- Utility and power supply
- Healthcare and emergency assistance
- Public access services

## Healthcare

- Implants
- Vitals monitoring
- Telemedicine
- Remote diagnostics
- Equipment tracking
- Equipment status monitoring

## Industrial

- Asset utilization
- Inventory and stock management
- Just-in-time manufacturing
- Location aware safety
- Smart tags & production monitoring
- Smart pumps / valves etc.
- Energy management

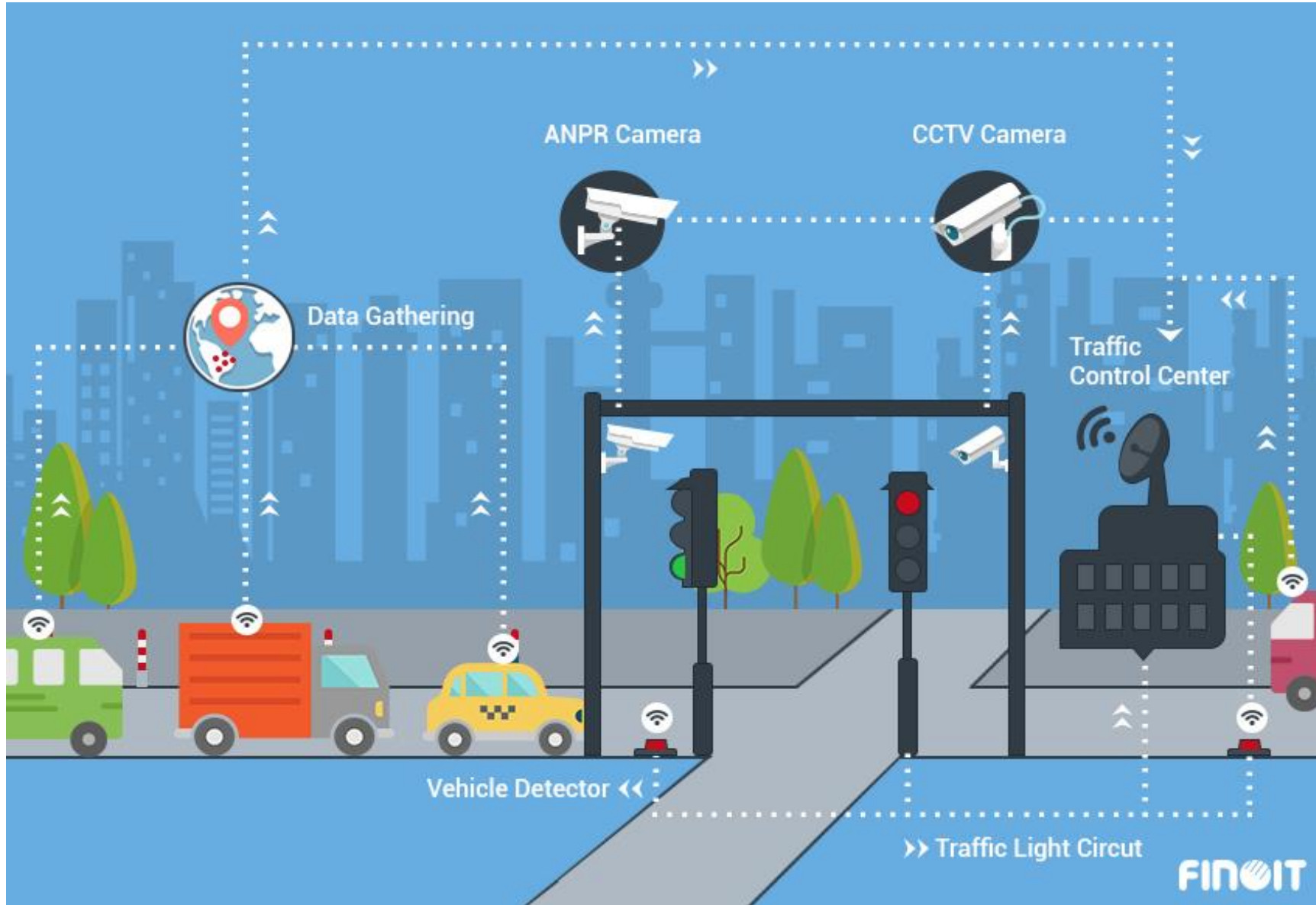
## Security

- Surveillance & tracking
- Access control
- Emergency services
- Environmental Monitoring
- Disaster management and response

## Retail & Finance

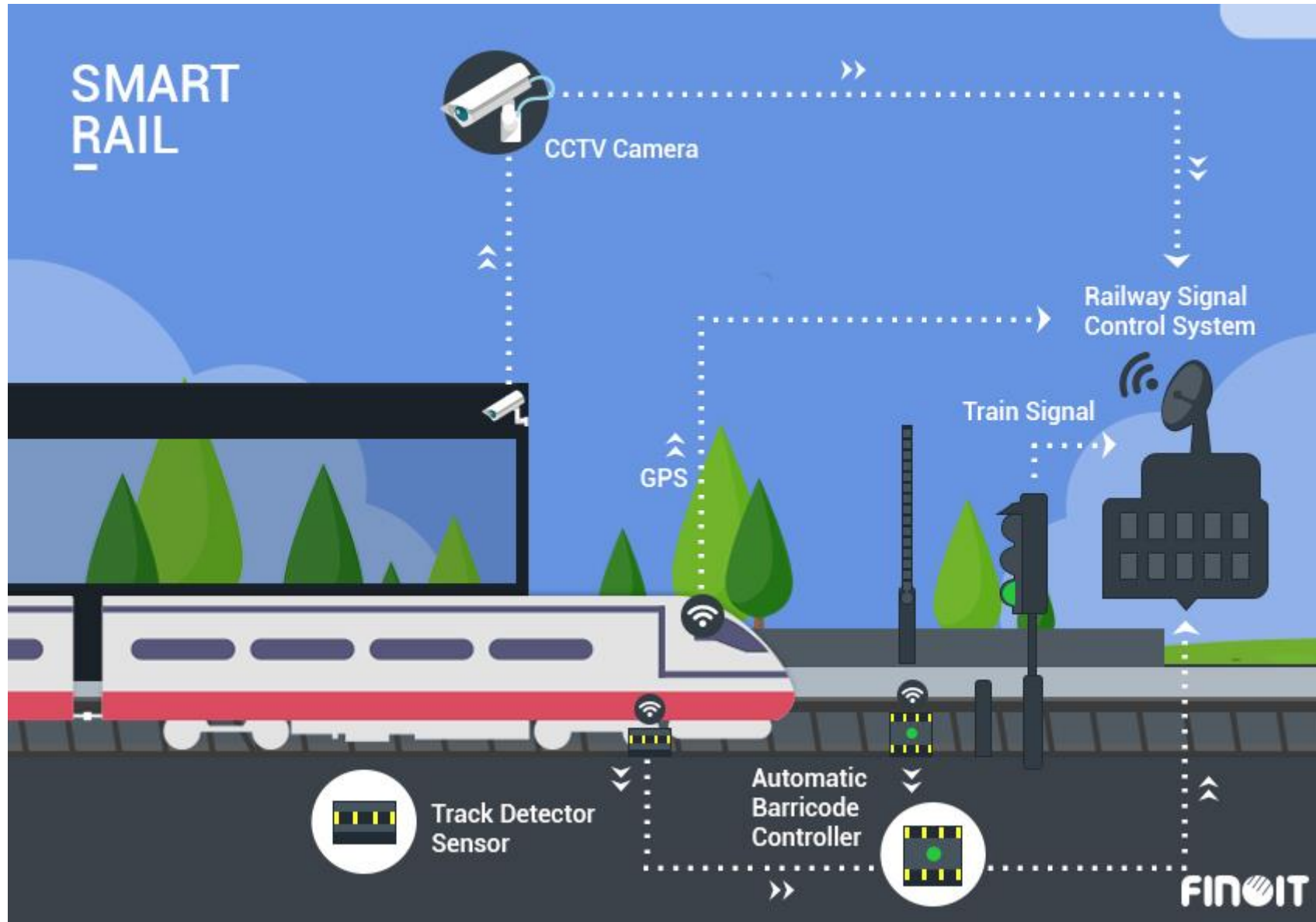
- Fuel stations
- Supermarkets
- Vending machine
- ATM/CDM/CRMs
- Self service checkouts
- Customer Relations Mgt

# IoT Examples: Smart Traffic Management



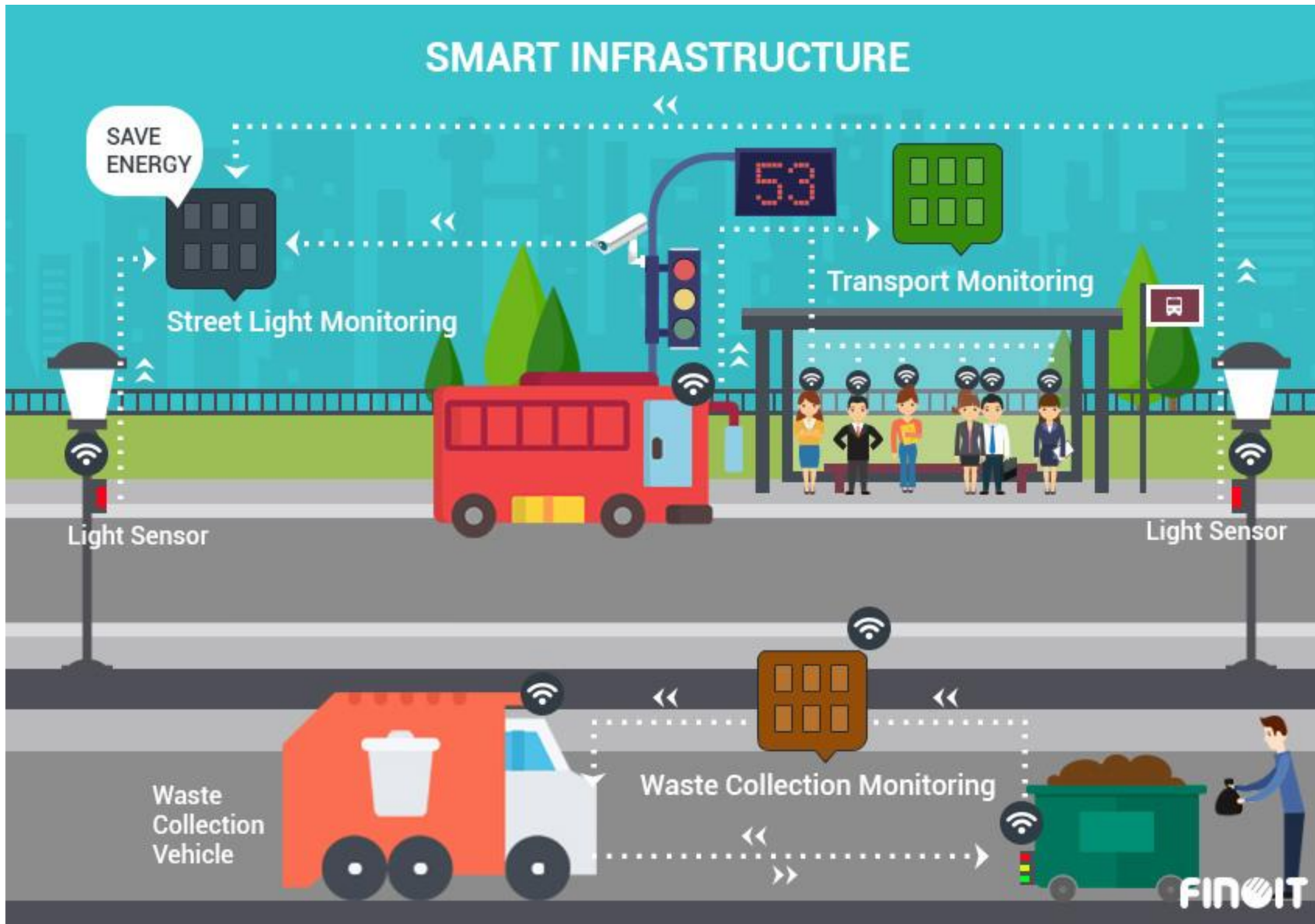
- Fewer traffic jams
- Less accidents
- Optimal road usage
- Reduced travel times
- Enhanced security
- Less pollution

# IoT Examples: Smart Railway



- Shorter passenger waiting times
- Safer journeys
- Less energy usage
- Increased security
- More revenue for the operator
- Convenience to the public

# IoT Examples: Smart Cities



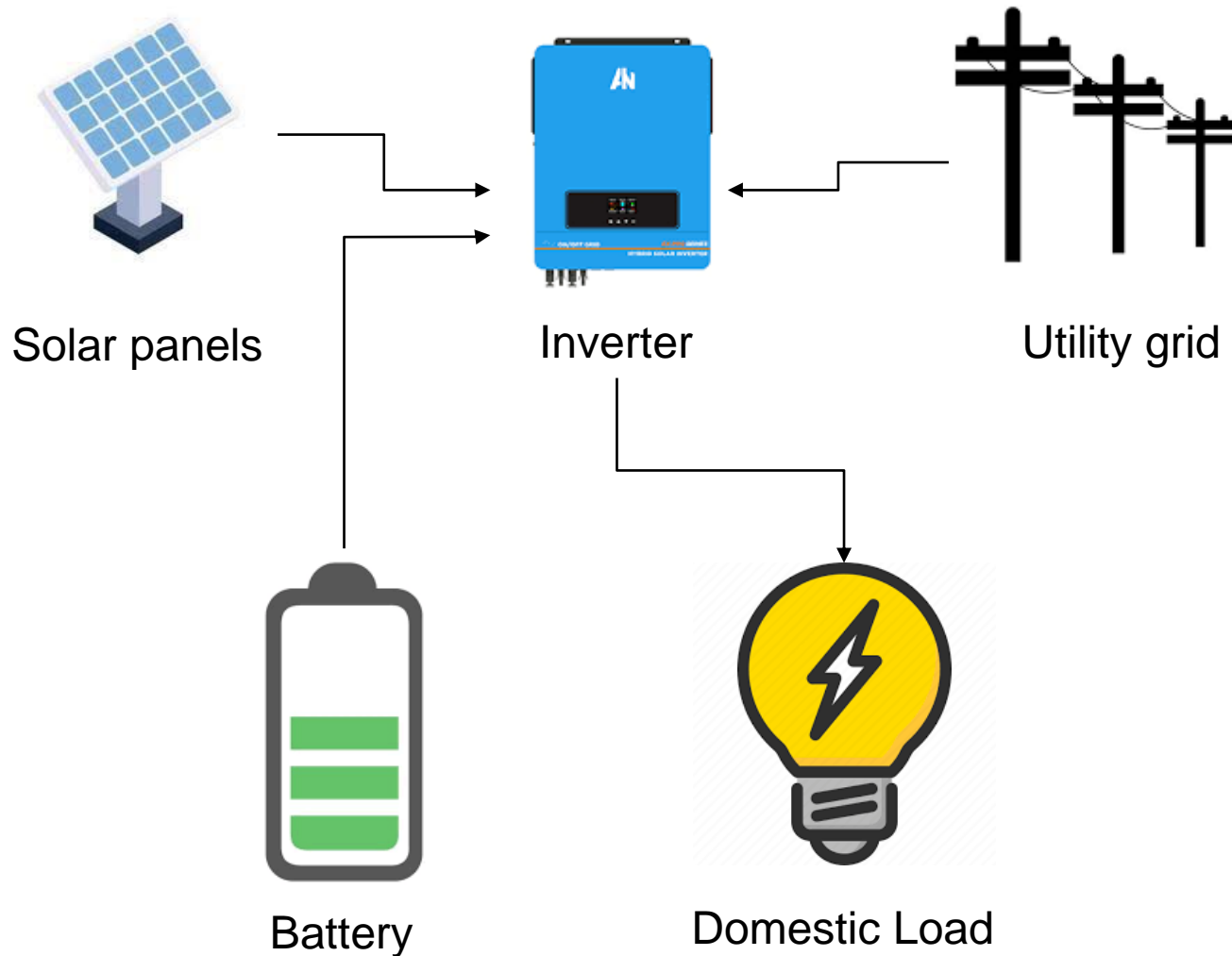
- Save energy
- Less pollution
- Optimal service usage
- Passenger convenience
- Enhanced security
- Less waiting time
- Less traffic



# IoT Examples: Smart Home

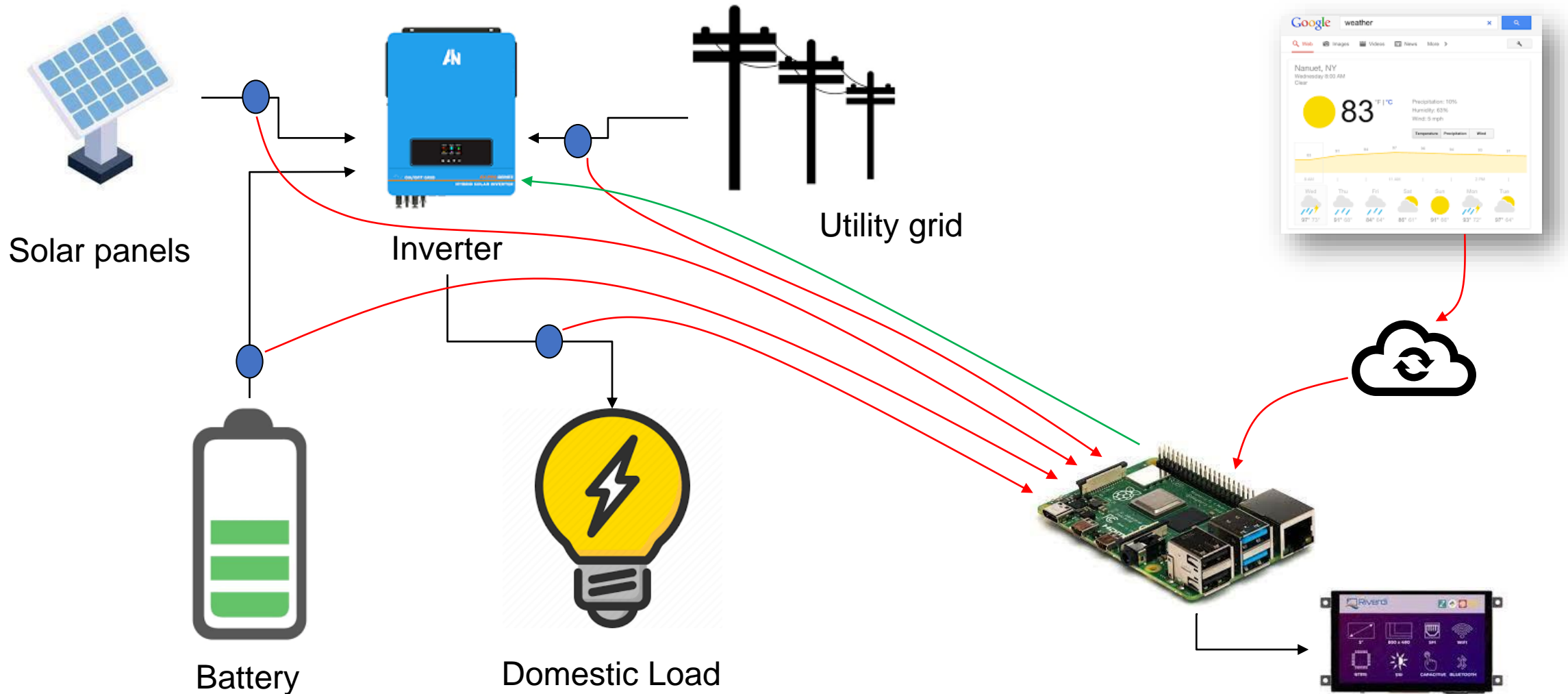


# IoT Example: Off-grid solar energy



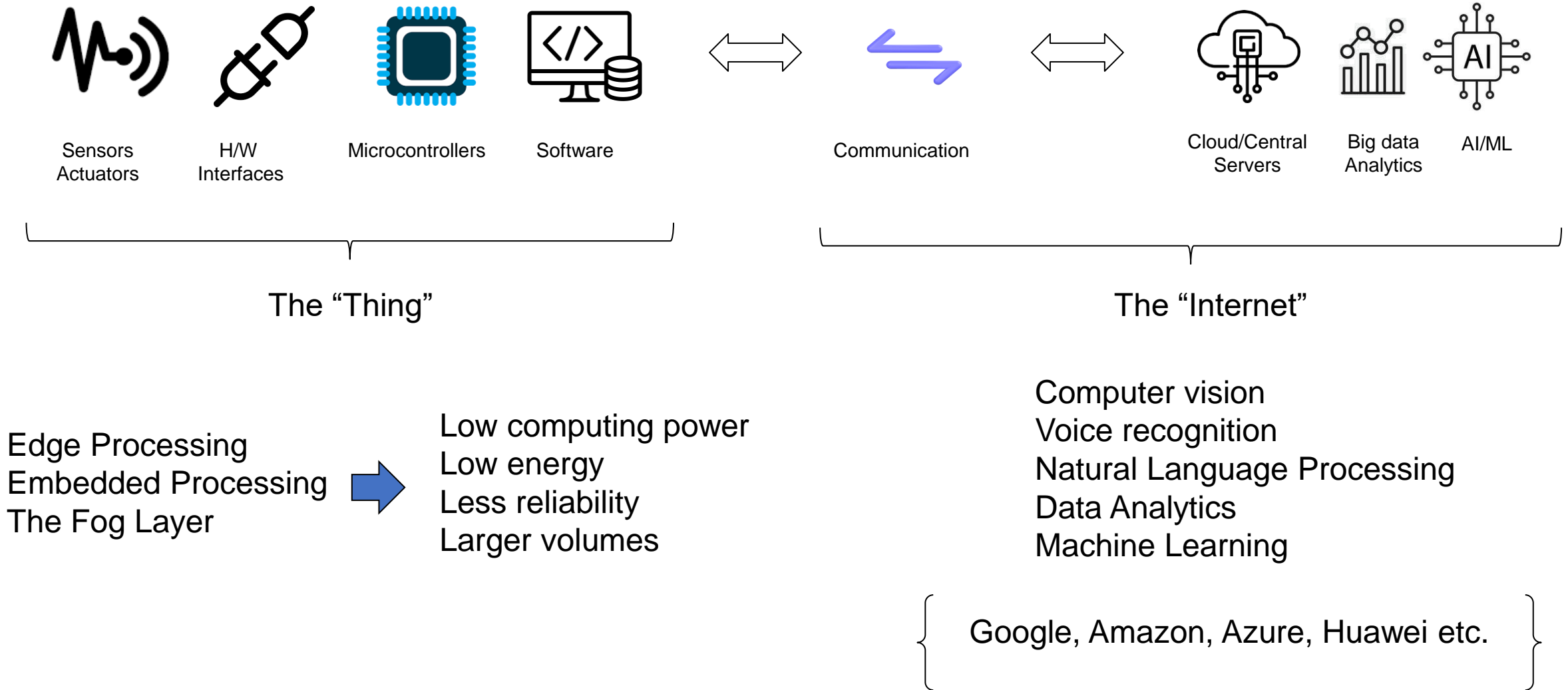
How do we manage limited battery capacity to ensure long battery life and the and minimum use of energy from the grid?

# IoT Example: Off-grid solar energy





# Components of an IoT solution



# IoT systems: Challenges and issues

## Security

- How to protect access to devices
- Physical security of devices
- Lack of visibility
- Lack of controllability
- Malware
- DDOS

## Privacy

- How to protect access by devices
- Data exposure
- Personal privacy
- Unauthorized access

## Technological issues

- Relatively young concepts and technologies
- Rapidly changing environments
- Dependency on external factors (power, network, cloud etc.)
- Infrastructure dependency

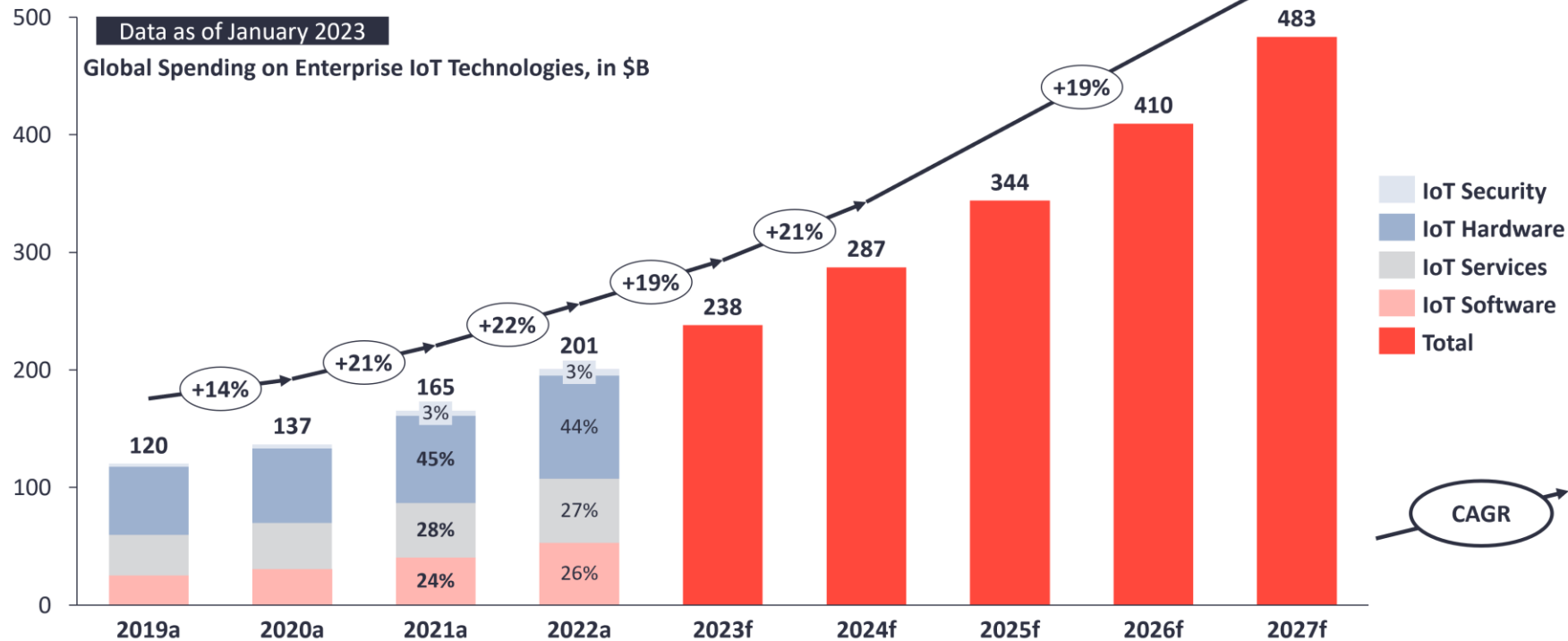
# IoT: The Commercial perspective



February 2023

Your Global IoT Market Research Partner

## Enterprise IoT market 2019–2027



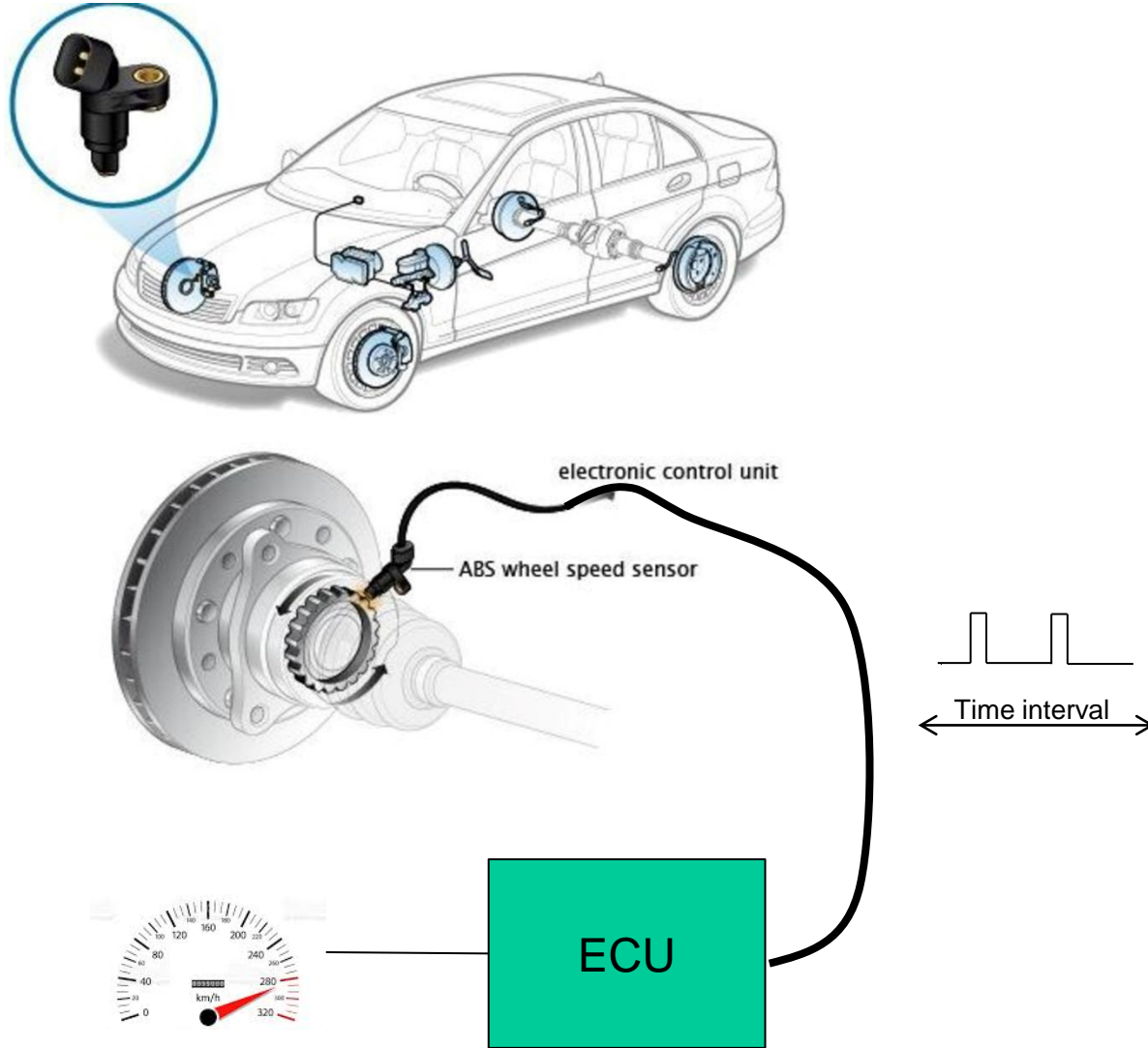
**Note:** IoT Analytics defines IoT as a network of internet-enabled physical objects. Objects that become internet-enabled (IoT devices) typically interact via embedded systems, some form of network communication, or a combination of edge and cloud computing. The data from IoT-connected devices is often used to create novel end-user applications. Connected personal computers, tablets, and smartphones are not considered IoT, although these may be part of the solution setup. Devices connected via extremely simple connectivity methods, such as radio frequency identification or quick response codes, are not considered IoT devices. a: Actuals, f: Forecast

**Source:** IoT Analytics Research 2023. We welcome republishing of images but ask for source citation with a link to the original post or company website.

**QUESTIONS?**

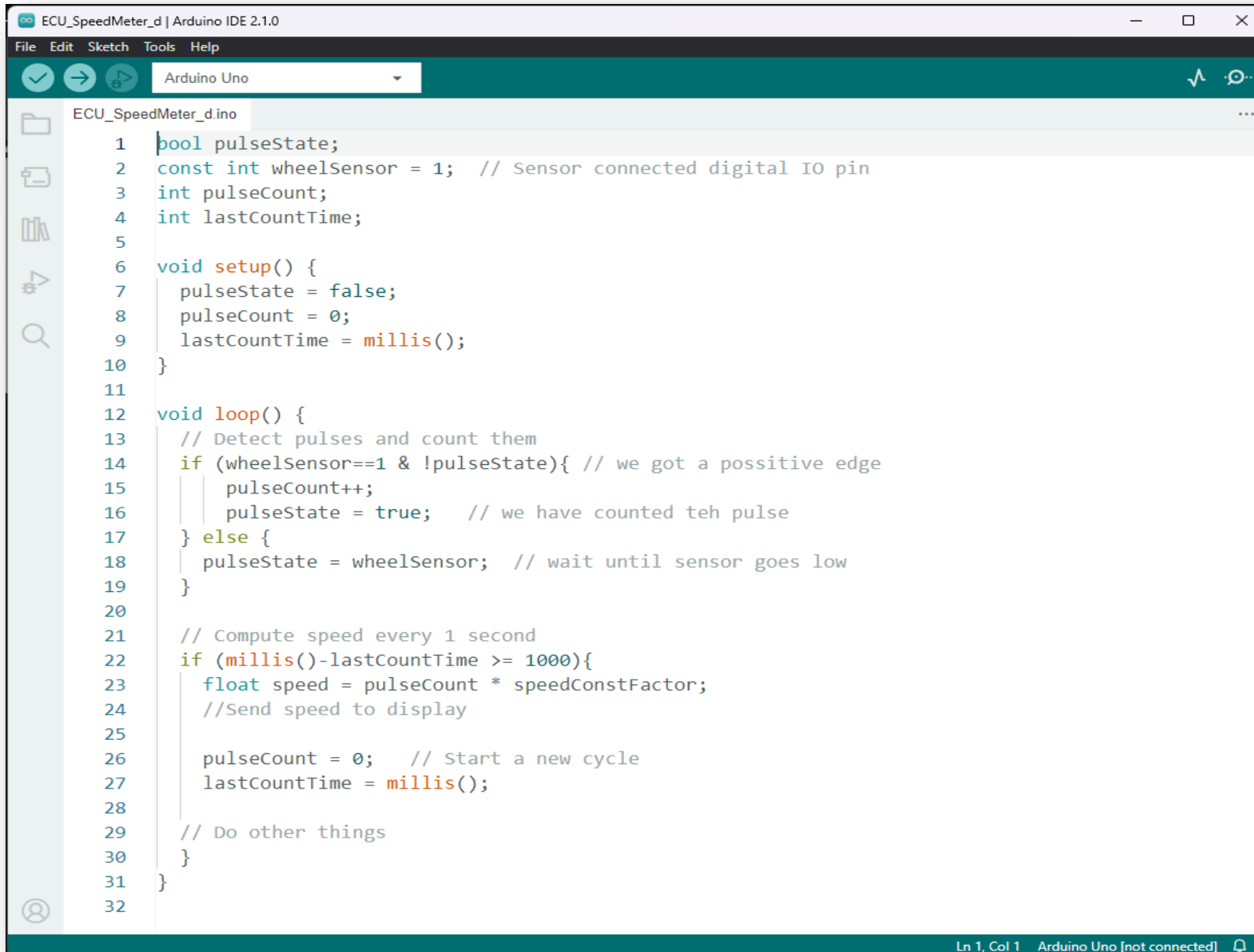
# PERIPHERAL MODULES

# Counting external events



- *The wheel sensor generates each time it rotates a specific angle.*
- *At the 1-second interval, pulse count represents the angular displacement of the wheel.*
- *Speed can be computed by multiplying angular displacement by a constant*
- *So, speed is updated at 1-second intervals*

# Complete software approach...



```
ECU_SpeedMeter_d | Arduino IDE 2.1.0
File Edit Sketch Tools Help
Arduino Uno
ECU_SpeedMeter_d.ino
1 bool pulseState;
2 const int wheelSensor = 1; // Sensor connected digital IO pin
3 int pulseCount;
4 int lastCountTime;
5
6 void setup() {
7   pulseState = false;
8   pulseCount = 0;
9   lastCountTime = millis();
10 }
11
12 void loop() {
13   // Detect pulses and count them
14   if (wheelSensor==1 & !pulseState){ // we got a possitive edge
15     pulseCount++;
16     pulseState = true; // we have counted teh pulse
17   } else {
18     pulseState = wheelSensor; // wait until sensor goes low
19   }
20
21   // Compute speed every 1 second
22   if (millis()-lastCountTime >= 1000){
23     float speed = pulseCount * speedConstFactor;
24     //Send speed to display
25
26     pulseCount = 0; // Start a new cycle
27     lastCountTime = millis();
28
29     // Do other things
30   }
31 }
32
```

Ln 1, Col 1 Arduino Uno [not connected]

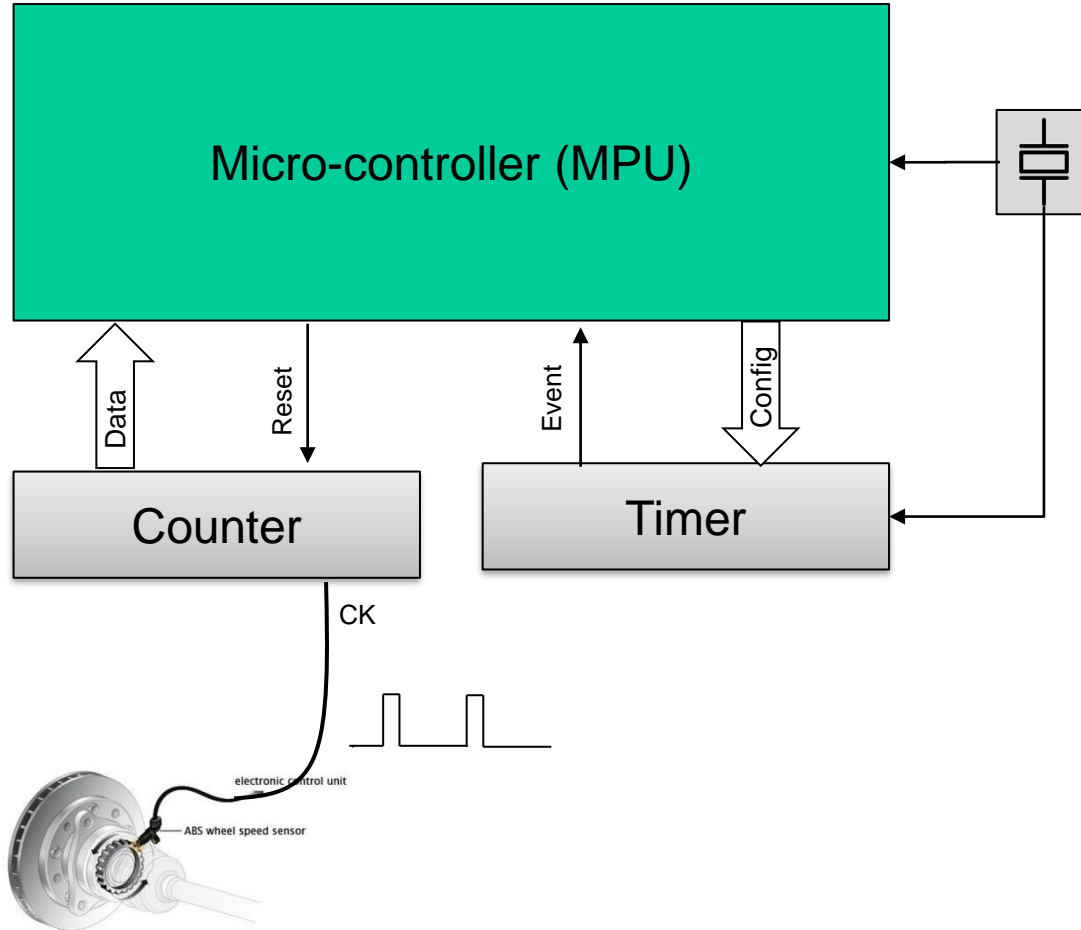
- *millis()* is an Arduino library function that returns the time since the last power cycle.
- There is another function called *micros()* which returns the same in microseconds

*However, the timing accuracy returned by these functions is not guaranteed.*

- The uC also needs to:
  - Manage the display
  - Handle de-bouncing etc.
  - Do multiple other tasks



# With peripheral module processing...

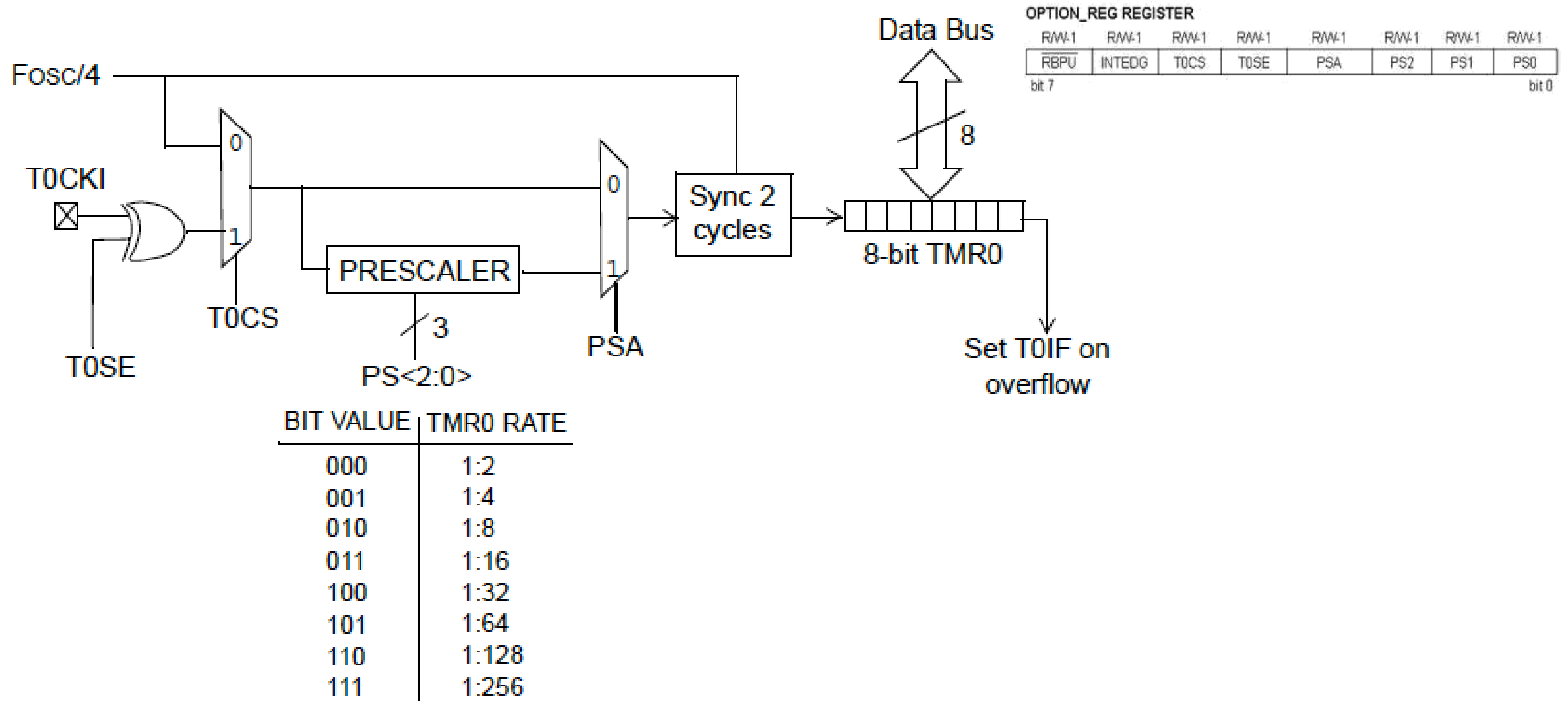


- *The counter runs independently and gets incremented each time a sensor outputs a pulse.*
- *The timer runs independently using an accurate oscillator input.*
- *The MPU configures the timer to generate an event (interrupt) every 1 second.*
- *When the timer event occurs, it simply reads the counter value and computes the speed*

# Timer / Counter Modules

- *Usually consist of a free running binary counter with clock sources that operates independent to processor execution logic*
  - *Can be used to measure time (in clock ticks), count external / internal events or to generate events at specific time intervals*
- *Often provided with number of accessory circuits for additional feature support*
  - *Clock-source / edge selection*
  - *Pre-scaler*
  - *Post-scaler*
  - *Interrupt generators*
  - *Pre-loading*

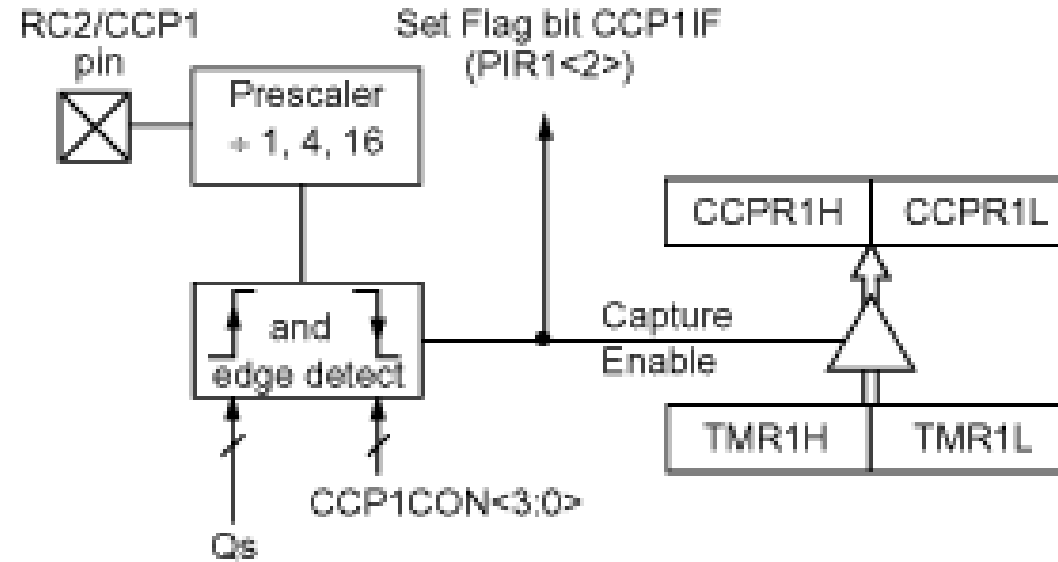
# Example: Timer/Counter in PIC16F877



# Capture / Compare Timer Modules

- *These are extended versions of timer modules that support additional functions*
  - *Capture*
    - *Capture and register the exact time (in clock tick count) of an external event*
  - *Compare*
    - *Generate an event exactly when the timer reaches a given tick count*
    - *Continuously compare timer register against a given value and generate either an internal interrupt or an external event once the match is detected*
- *Typically used when an event time is required to be recorded at high accuracy*

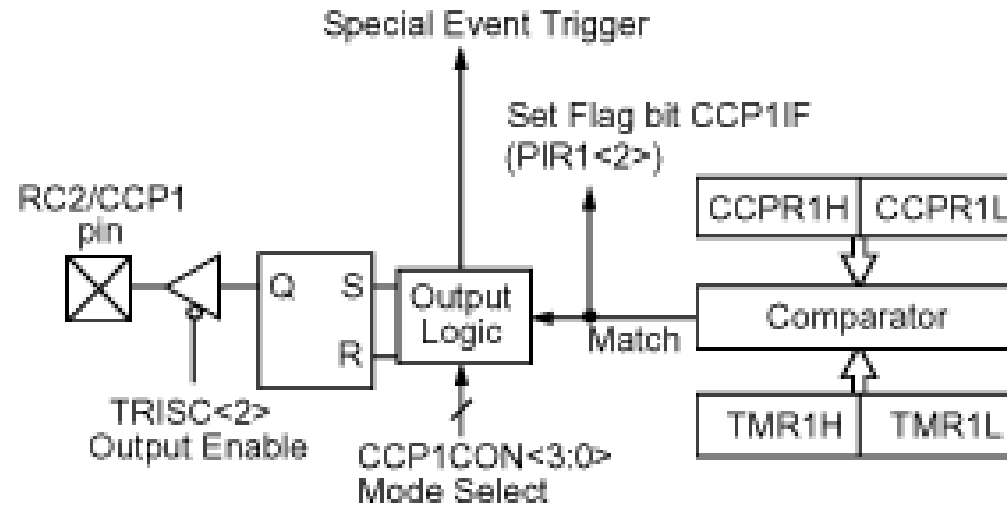
# Capture Mode Operation



- ⊕ In Capture mode, CCPR1H:CCPR1L captures the 16-bit value of the TMR1 register when an event occurs on pin RC2/CCP1.
- ⊕ An event is defined as one of the following:
  - ▶ Every falling edge
  - ▶ Every rising edge
  - ▶ Every 4th rising edge
  - ▶ Every 16th rising edge

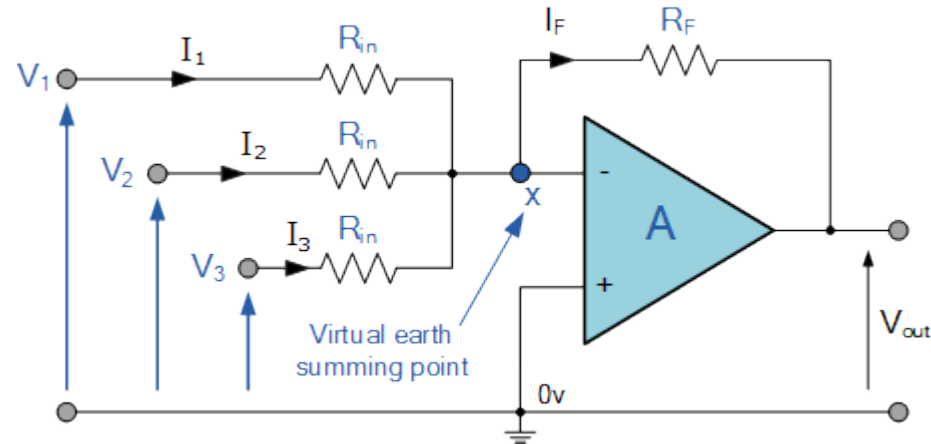
# Compare mode operation

Special event trigger will:  
reset Timer1, but not set interrupt flag bit TMR1IF (PIR1<0>),  
and set bit GO/DONE (ADCON0<2>).



- *The 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RC2/CCP1 pin is:*
  - *Driven high*
  - *Driven low*
  - *Remains unchanged*

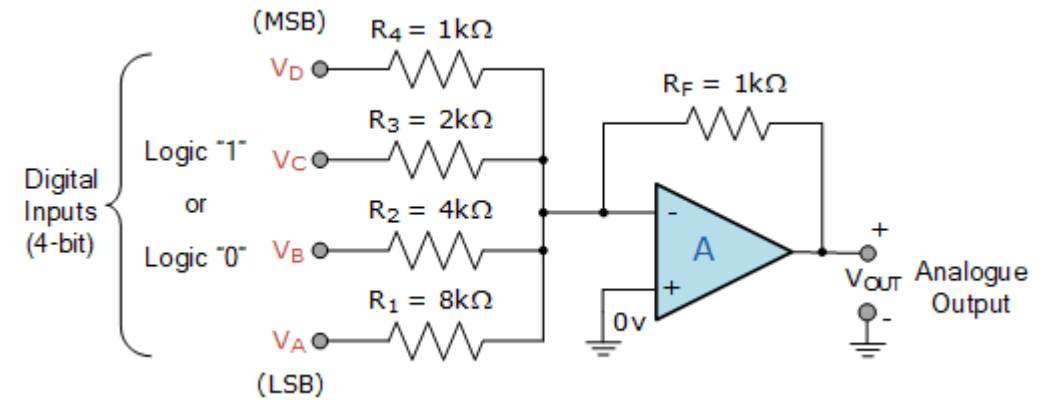
# Digital to Analog converters



$$I_F = I_1 + I_2 + I_3 = - \left[ \frac{V_1}{R_{in}} + \frac{V_2}{R_{in}} + \frac{V_3}{R_{in}} \right]$$

$$\text{Inverting Equation: } V_{out} = - \frac{R_f}{R_{in}} \times V_{in}$$

$$\text{then, } -V_{out} = \left[ \frac{R_F}{R_{in}} V_1 + \frac{R_F}{R_{in}} V_2 + \frac{R_F}{R_{in}} V_3 \right]$$



$$V_{OUT} = - \left[ \frac{R_F}{R_4} V_D + \frac{R_F}{R_3} V_C + \frac{R_F}{R_2} V_B + \frac{R_F}{R_1} V_A \right]$$

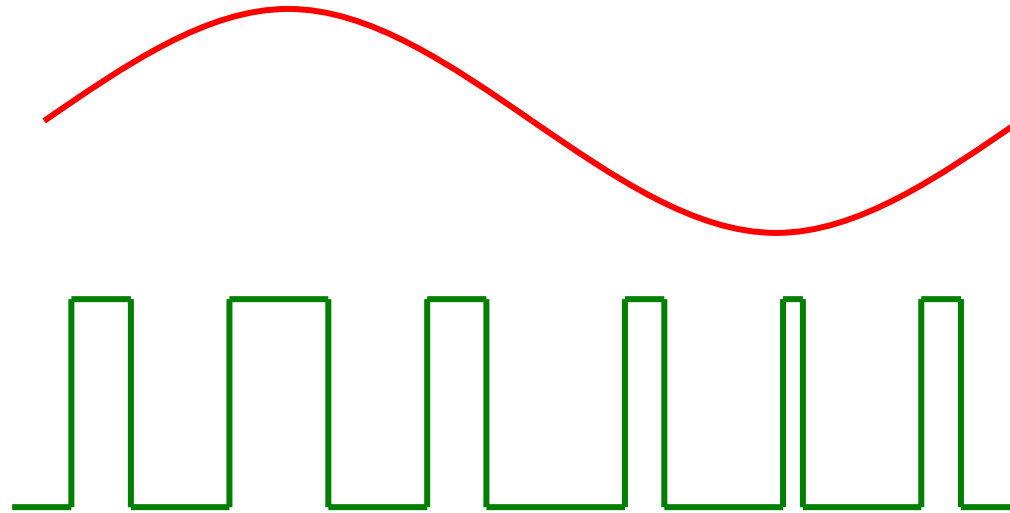
$$V_{OUT} = - \left[ \frac{1k\Omega}{1k\Omega} V_D + \frac{1k\Omega}{2k\Omega} V_C + \frac{1k\Omega}{4k\Omega} V_B + \frac{1k\Omega}{8k\Omega} V_A \right]$$

$$V_{OUT} = - \left[ 1V_D + \frac{1}{2} V_C + \frac{1}{4} V_B + \frac{1}{8} V_A \right]$$

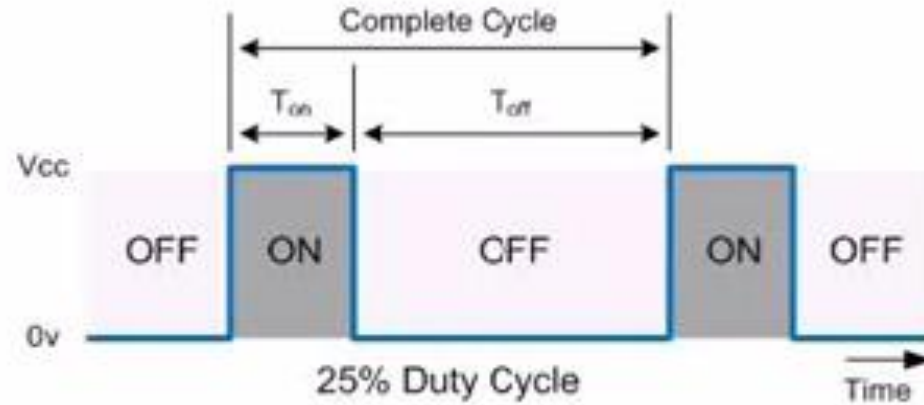


# Pulse Width Modulation

- The amplitude is maintained constant but the width of each pulse is varied in accordance with instantaneous value of the analog signal.*



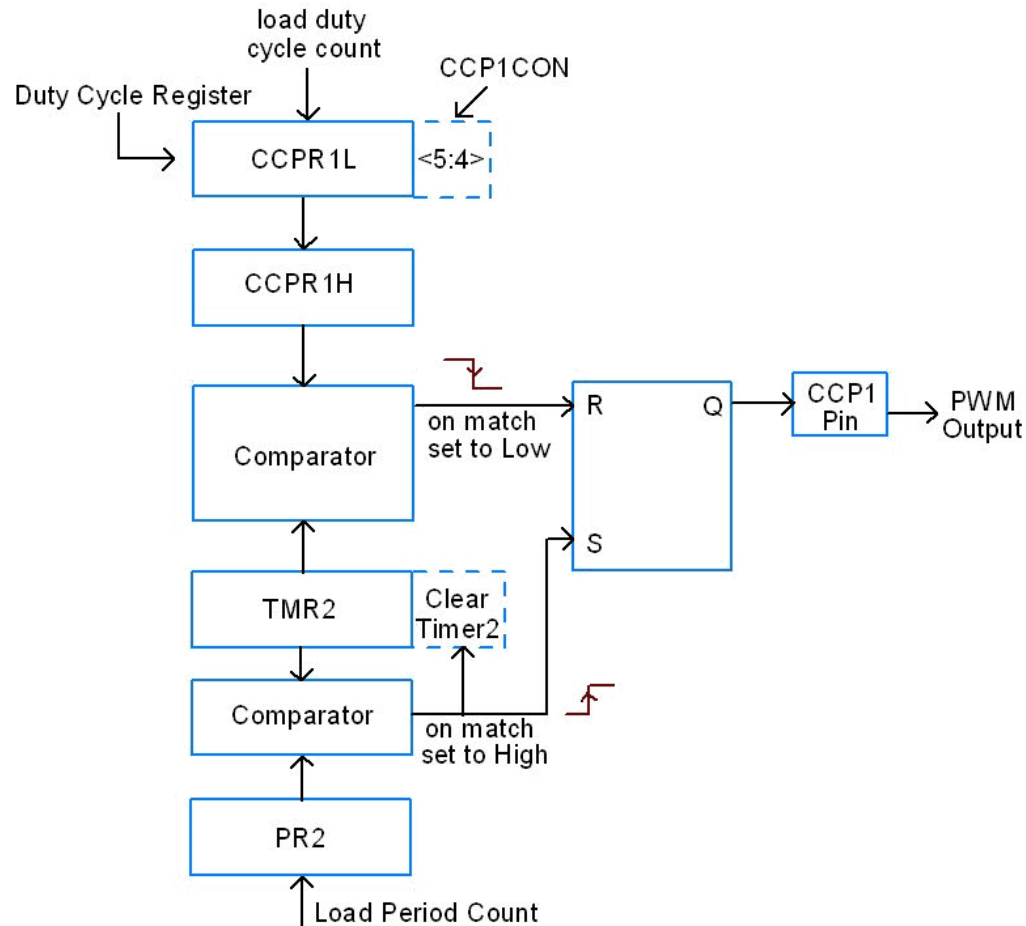
# Pulse Width Modulation explained



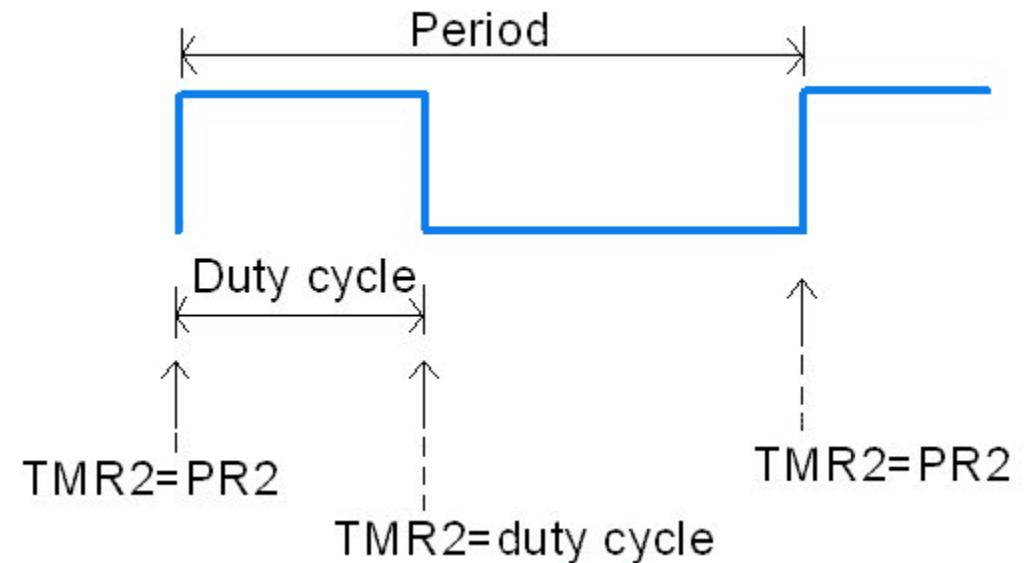
Part 4:  
**Using Pulse-Width-Modulation  
(PWM) with Arduino**

**by Lewis Loflin**

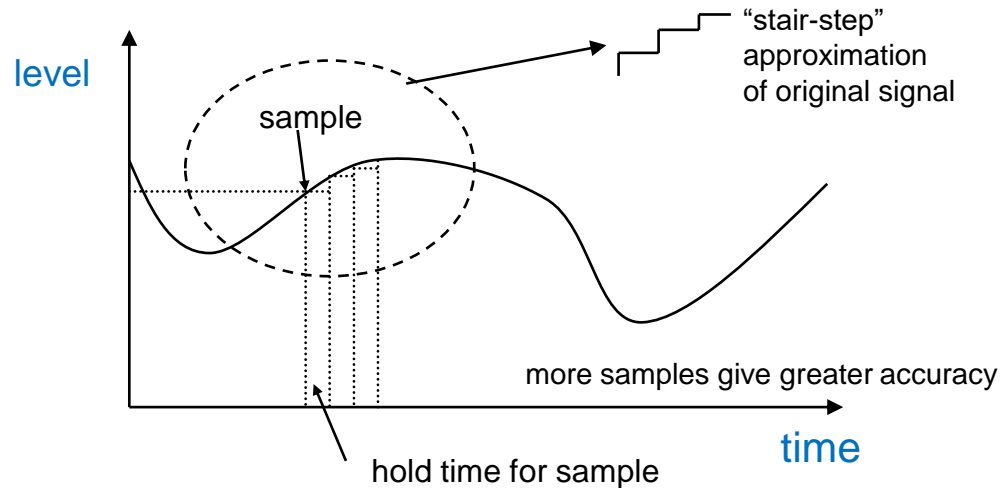
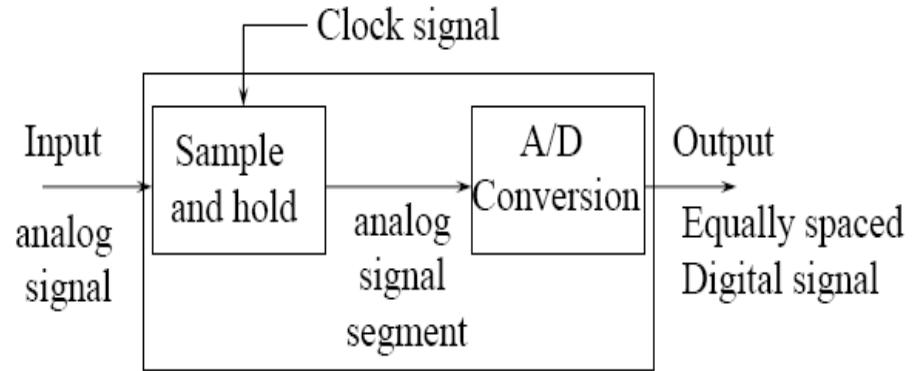
# PWM module in PIC microcontroller



Register	Description
CCPxCON	This register is used to Configure the CCP module for Capture/Compare/PWM opertaion.
CCPRxL	This register holds the 8-Msb bits of PWM, lower 2-bits will be part of CCPxCON register.
TMR2	Free running counter which will be compared with CCPR1L and PR2 for generating the PWM output.



# Analog to Digital Conversions



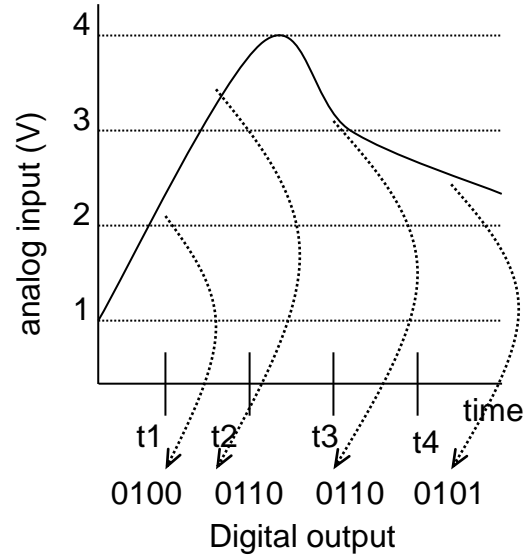
- *Converts analog signals into binary words*
- *Involves a 3-step process*
  - *Sample and Hold*
    - *Freeze ever changing analog voltage until conversion is done*
  - *A/D conversion*
    - *Quantization*
    - *Encoding*

# Quantization and Resolution

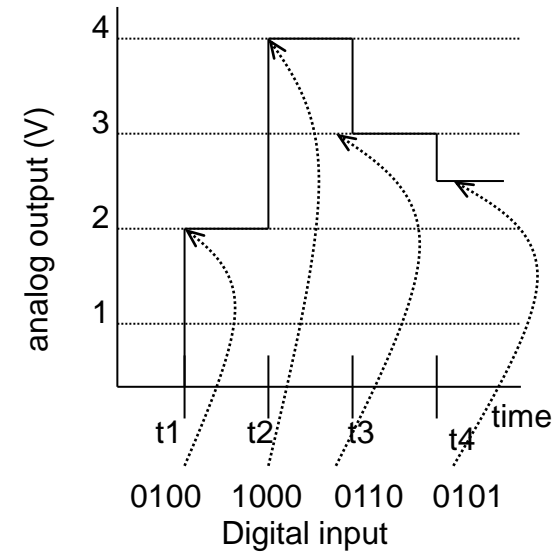
$V_{\max} = 7.5\text{V}$

7.5V	1111
7.0V	1110
6.5V	1101
6.0V	1100
5.5V	1011
5.0V	1010
4.5V	1001
4.0V	1000
3.5V	0111
3.0V	0110
2.5V	0101
2.0V	0100
1.5V	0011
1.0V	0010
0.5V	0001
0V	0000

proportionality



analog to digital



digital to analog

# Quantizing and Resolution

*Example:*

*You have 0-10V signals. Separate them into a set of discrete states with **1.25V** increments.*

*The number of possible states that the converter can output is:*

$$N=2^n$$

*where  $n$  is the number of bits in the AD converter*

*Example: For a 3 bit A/D converter,  $N=2^3=8$ .*

**Analog quantization size:**

$$Q=(V_{max}-V_{min})/N = (10V - 0V)/8 = 1.25V$$

Output States	Discrete Voltage Ranges (V)
0	0.00-1.25
1	1.25-2.50
2	2.50-3.75
3	3.75-5.00
4	5.00-6.25
5	6.25-7.50
6	7.50-8.75
7	8.75-10.0

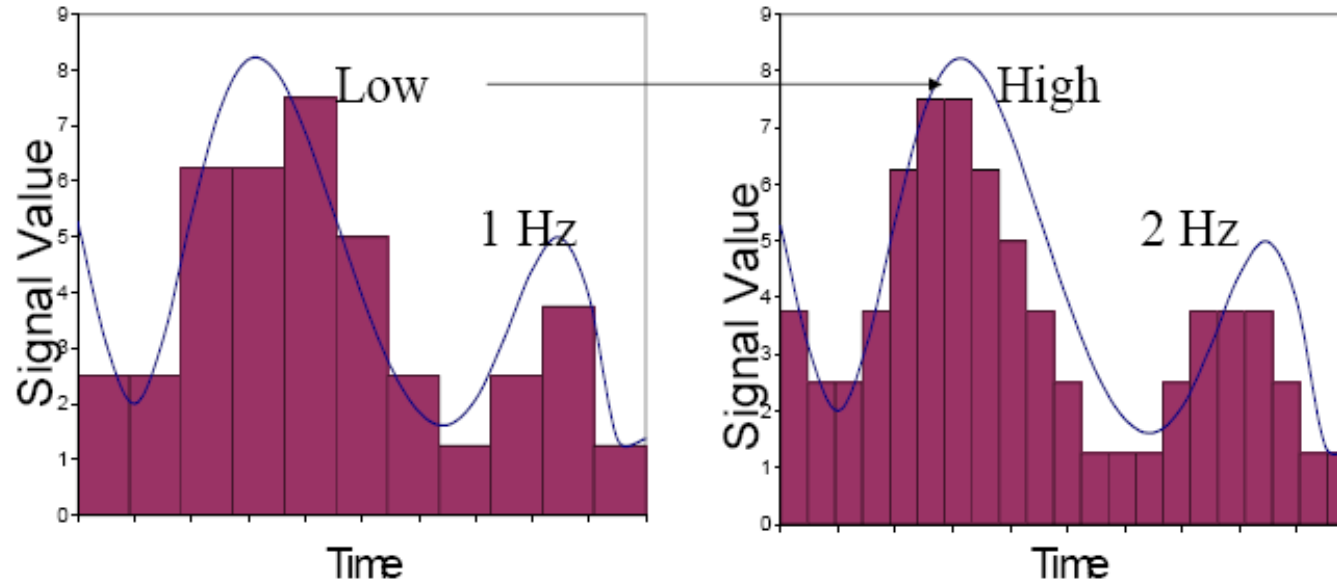
# Quantization and Resolution

- *Resolution*
  - *Number of discrete values the converter can produce = Analog Quantization size (Q)*  
 *$(Q) = V_{range} / 2^n$ , where  $V_{range}$  is the range of analog voltages which can be represented*
  - *Often limited by signal-to-noise ratio*
- *In our previous example:  $Q = 1.25V$ , this is a high resolution. A lower resolution would be if we used a 2-bit converter, then the resolution would be*

$$10 / 2^2 = 2.50 \text{ Volts}$$



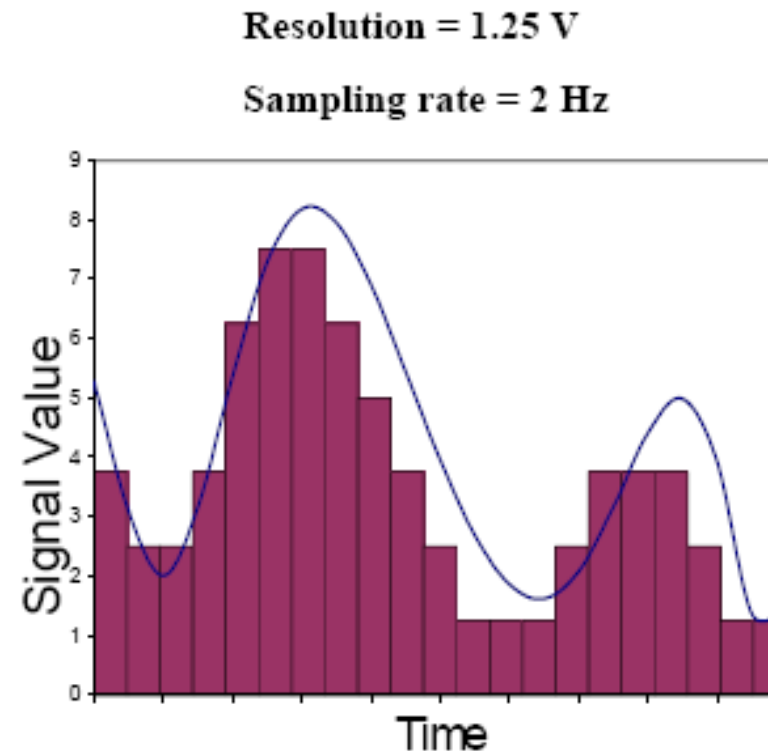
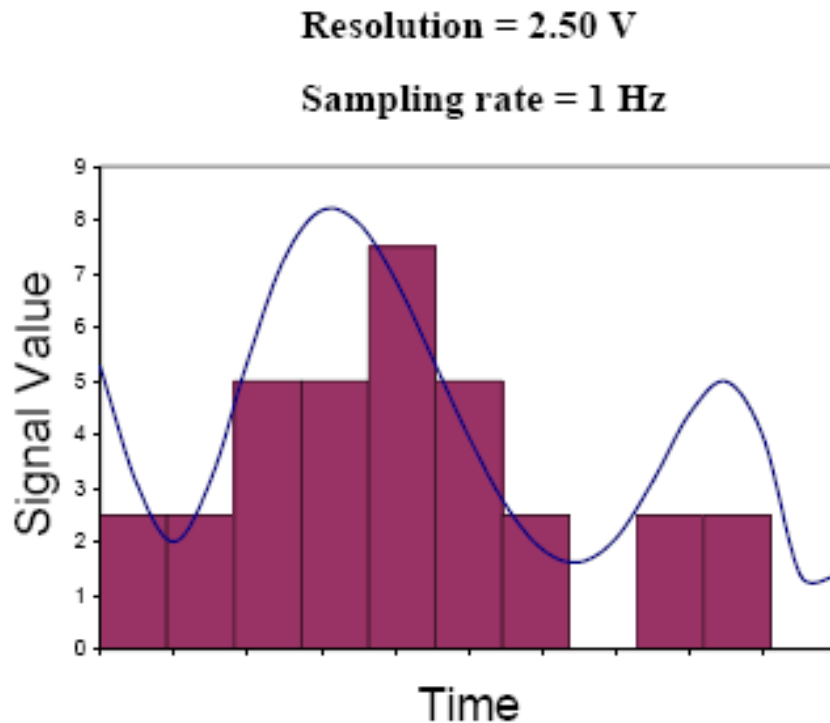
# Sampling Rate



Frequency at which ADC evaluates analog signal. As we see in the second picture, evaluating the signal more often more accurately depicts the ADC signal.

# Overall Better Accuracy

- Increasing both the sampling rate and the resolution you can obtain better accuracy in your AD signals.*



# Encoding

- *Here a digital (binary) value is assigned to each quantized level of the input signal*
- *Assigned binary value will be a direct representation of the quantized analog input*

Output States	Output Binary Equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

# AtoD converter types

- *Conversion required input voltage to be compared against available quantization levels to determine the appropriate mapping*
- *Different converters use different ways to perform this conversion*
  - *Parallel (Direct) conversion*
    - *Input signal is compared against all  $(2^n)$  quantization levels at the same time*
  - *Series conversion*
    - *Input signal is compared against each quantization level, one at a time*

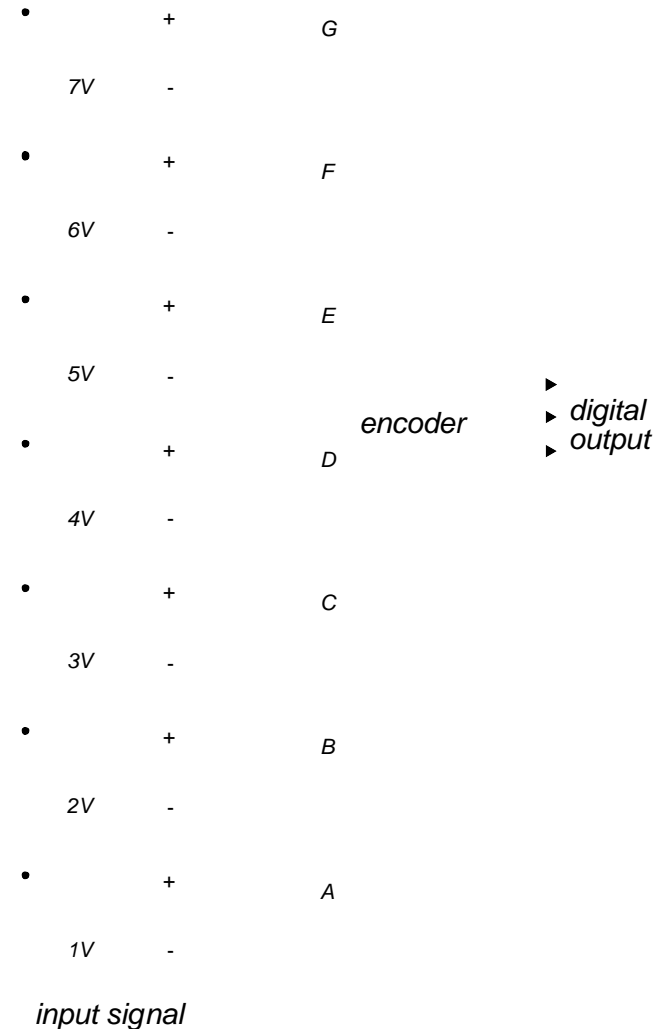
analogue input ——— +

reference voltage ——— -

- *A comparator compares 2 signals A and B*
  - *if  $A > B$  the comparator output is in one logic state (0, say)*
  - *if  $B > A$  then it is in the opposite state (1, say)*
- *A comparator can be built using an op amp with no feedback*

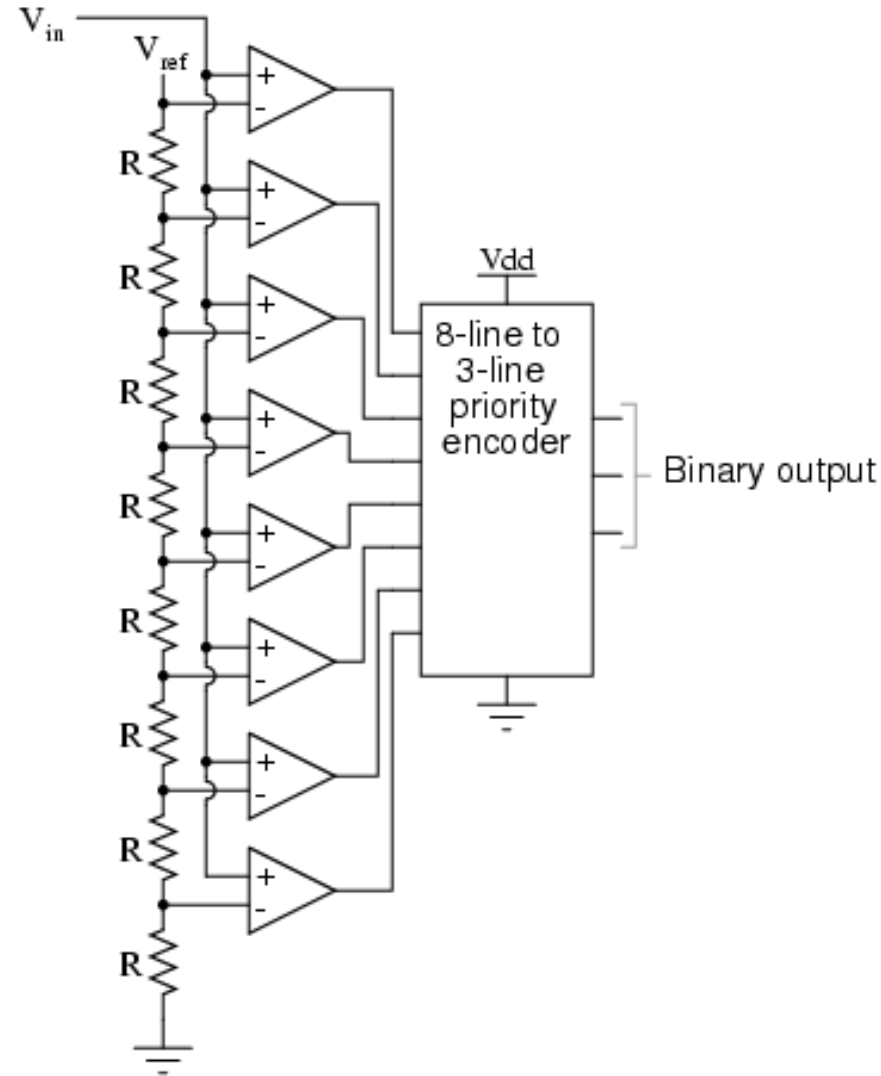
# Parallel (Direct) A to D Conversion

- Use number of comparator to compare input signal against all  $2^n$  quantization levels at the same time
  - Need  $2^n$  comparators for a  $n$ -bit conversion
- The speed of the converter is limited only by the speeds of the comparators and the logic network.
  - Speeds in excess of 20 to 30 MHz are common, and speeds  $> 100$  MHz are available.
- The cost stems from the circuit complexity since the number of comparators required increases rapidly.
  - 3-bit converter need 7 comparators
  - 6-bits would require 63
  - 8-bits would need 256



# Example: 3-bit flash converter

- The resistor network is a precision voltage divider, dividing  $V_{ref}$  (8 volts in the sample) into equal voltage increments (1.0 volt) to one input of the comparator. The other comparator input is the input voltage
- Each comparator switches immediately when  $V_{in}$  exceeds  $V_{ref}$ . Comparators whose input does not exceed  $V_{ref}$  do not switch.
- A decoder circuit (a 74148 8-to-3 priority decoder) converts the comparator outputs to a useful output

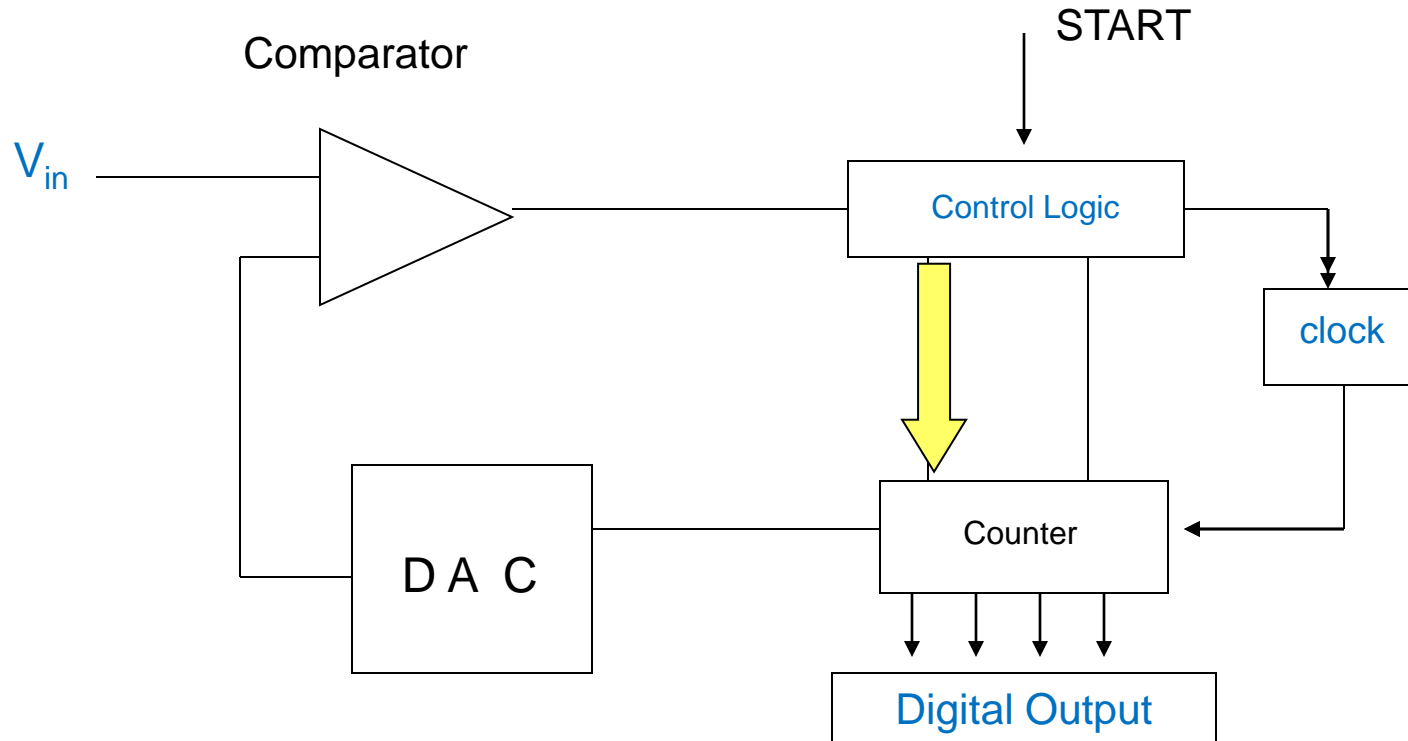


# Counter type converters

- *Counter type converters eliminate the need for large number of analog comparators*
  - *Use a single comparator and change the reference voltage serially to compare against each quantization level*
  - *Employ a digital to analog (D to A) converter to generate quantization reference levels directly from the encoded binary output*
  - *Process is sequential and require more conversion time*
  - *Three main types exist based on the logic used to search and generate quantization reference levels*
    - *Ladder (ramp) converter*
    - *Tracking (servo) converter*
    - *Successive Approximation Register (SAR)*



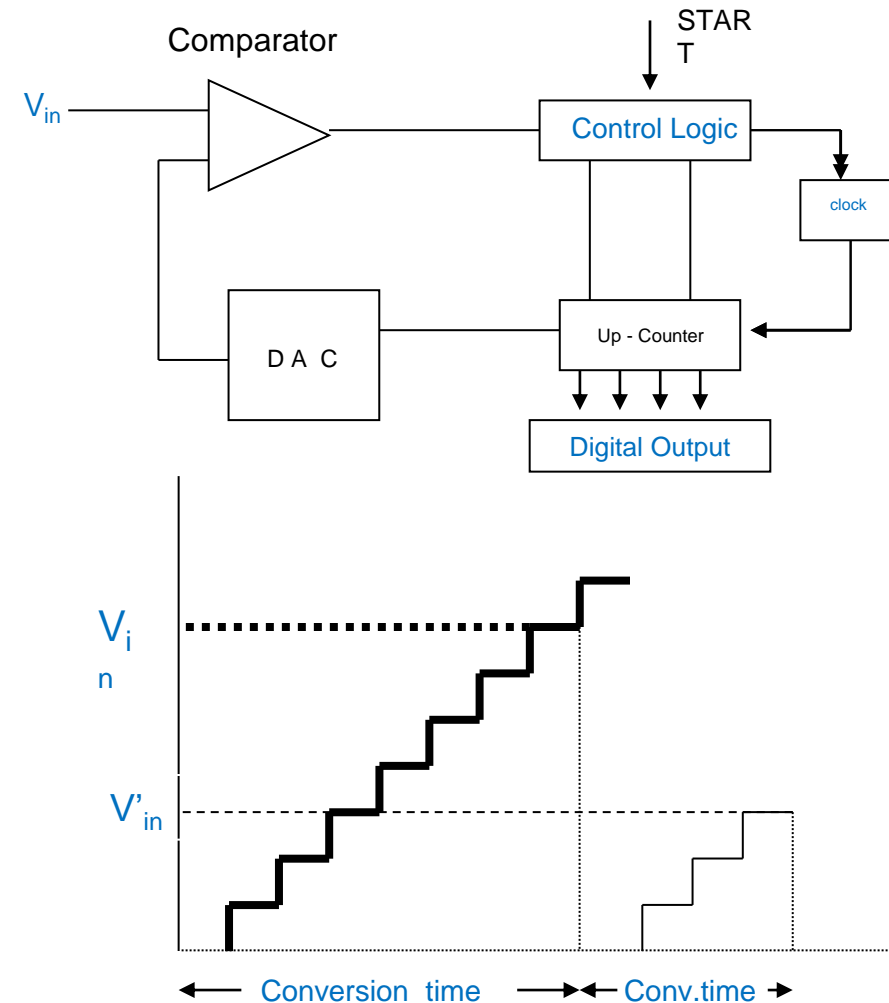
# Counter type converters



- When *START* is received, control logic initializes the system, (sets counter) and modify counter value in steps so that DAC generates the corresponding analog voltage reference for compactor.
- Process stops when a match is found

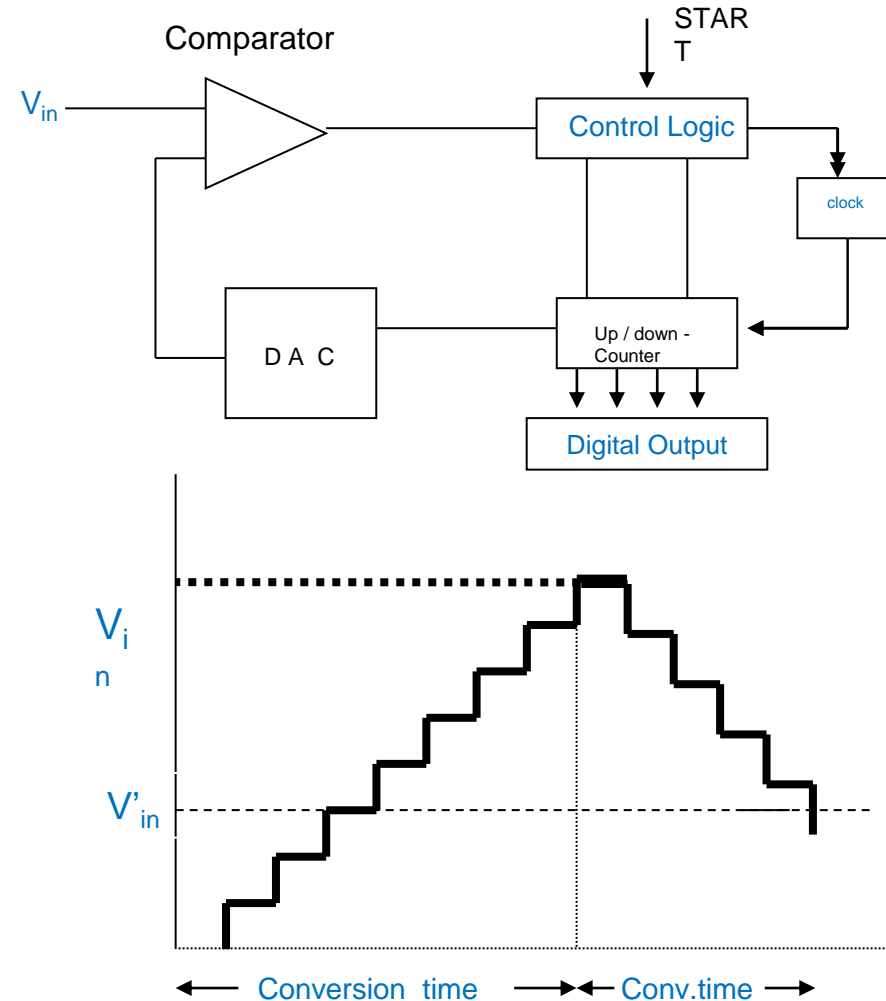
# Ladder (Ramp) Converter

- *Conversion process*
  - When *START* signal is received, conversion starts by initializing up-counter to zero
  - On each clock tick the counter is incremented by 1, gradually increasing the quantization reference level in a ramp pattern
  - Conversion completes at the point where DAC output meets  $V_{in}$
- *Each conversion cycle starts from the same initial, zero value*
- *Conversion time is variable and depends on absolute value of  $V_{in}$*



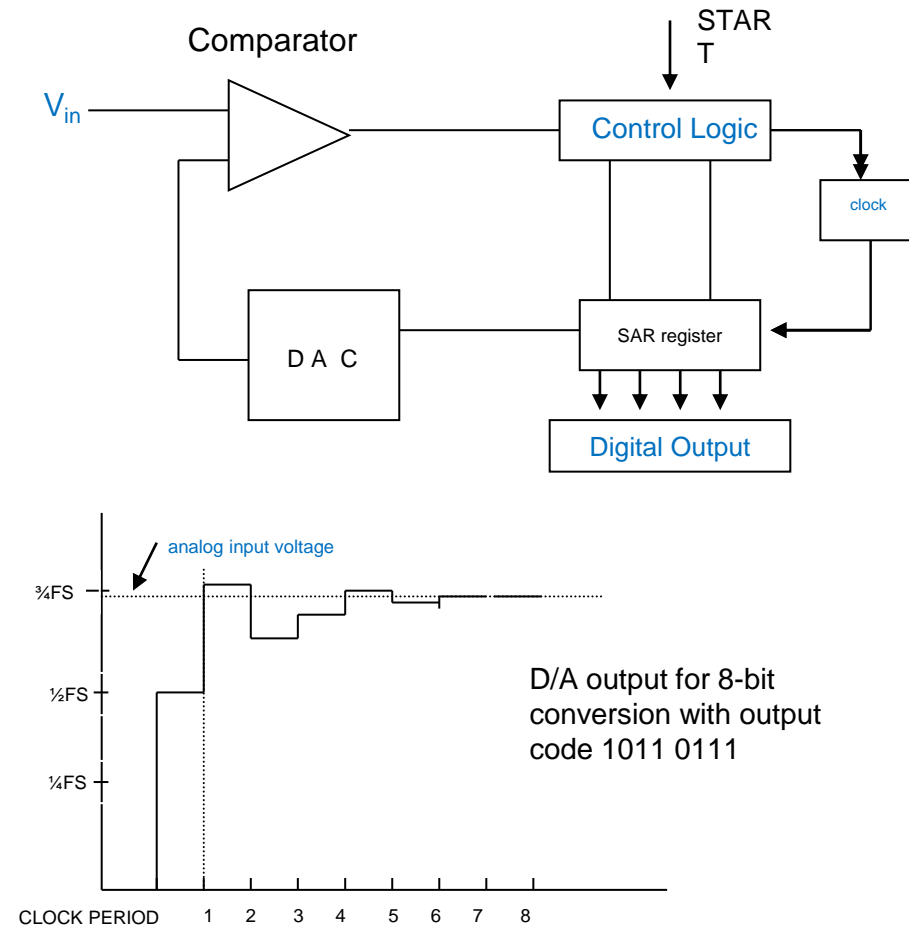
# Tracking (Servo) Converter

- *Conversion process*
  - When *START* signal is received, conversion starts from the last matching value in the counter
  - On each clock tick the counter is either incremented or decremented by 1, gradually moving the quantization reference level in a ramp pattern towards the input voltage
  - Conversion completes at the point where DAC output meets  $V_{in}$
- Each conversion cycle starts from the last matching reference value
- Conversion time is variable and depends on the change in  $V_{in}$

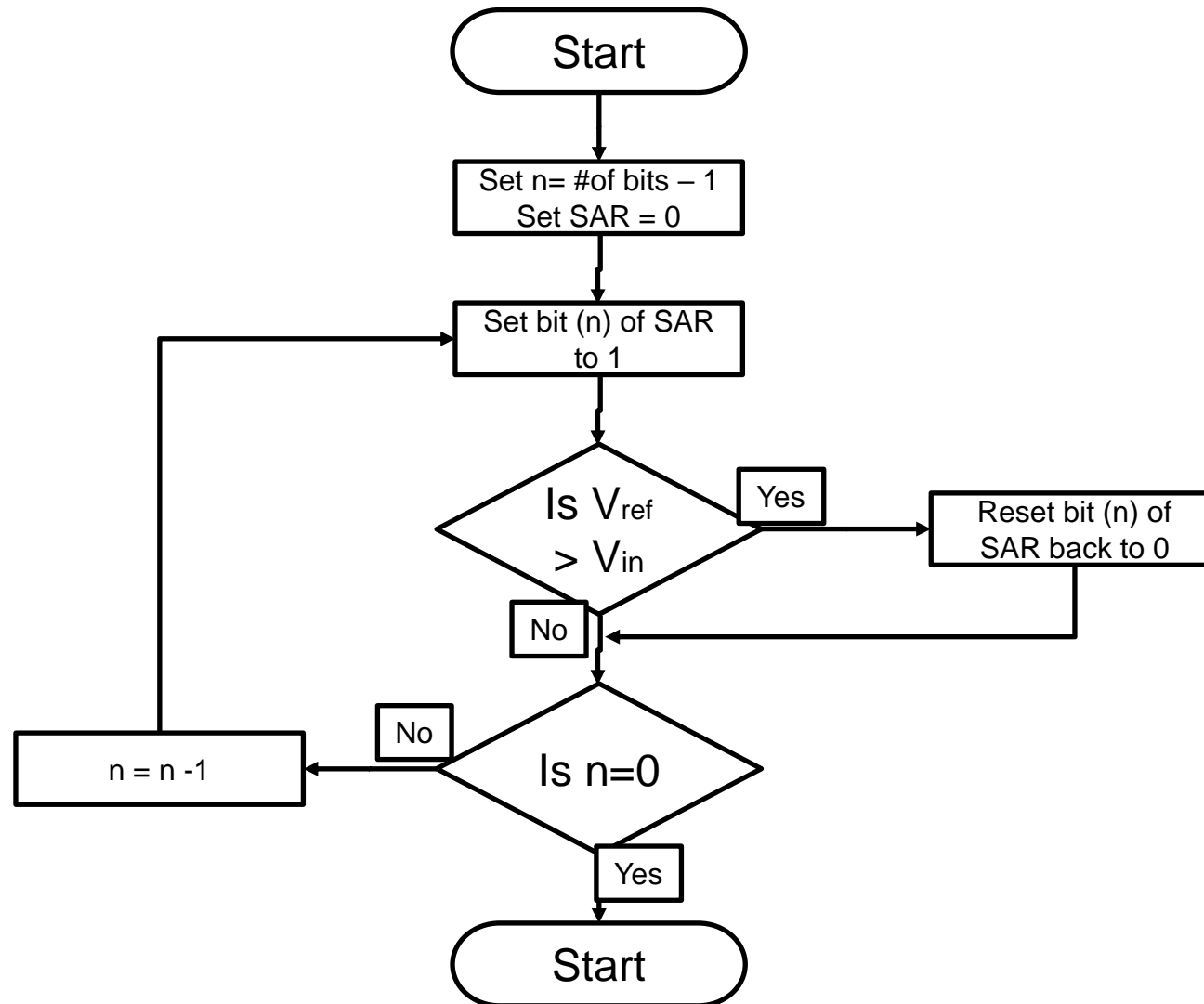


# Successive Approximation Register Converter

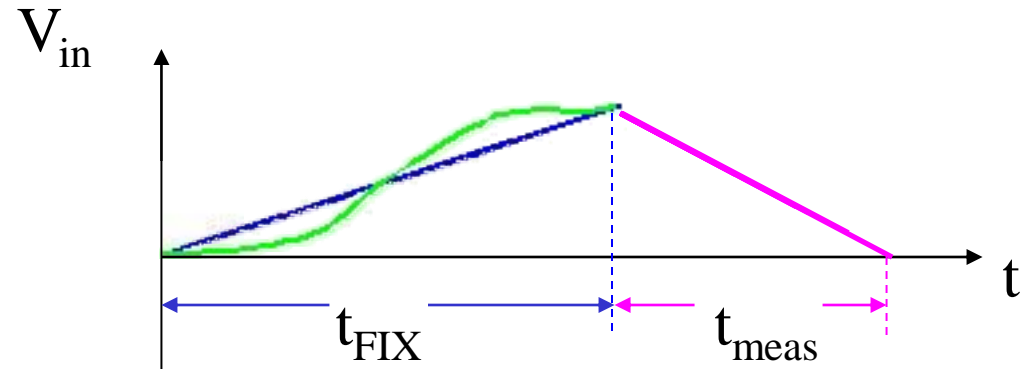
- *Use a binary search for the matching quantization level*
- *Manipulate individual bits in order rather than counting*
  - *Start with the MSB of the output word*
  - *Progressively set / reset each bit moving the quantization level reference towards the input signal*
- *Each conversion cycle starts from zero for each conversion*
- *Conversion time is fixed and independent from  $V_{in}$ , approximately  $n$ -cycles for a  $n$ -bit conversion*



# SAR Algorithm

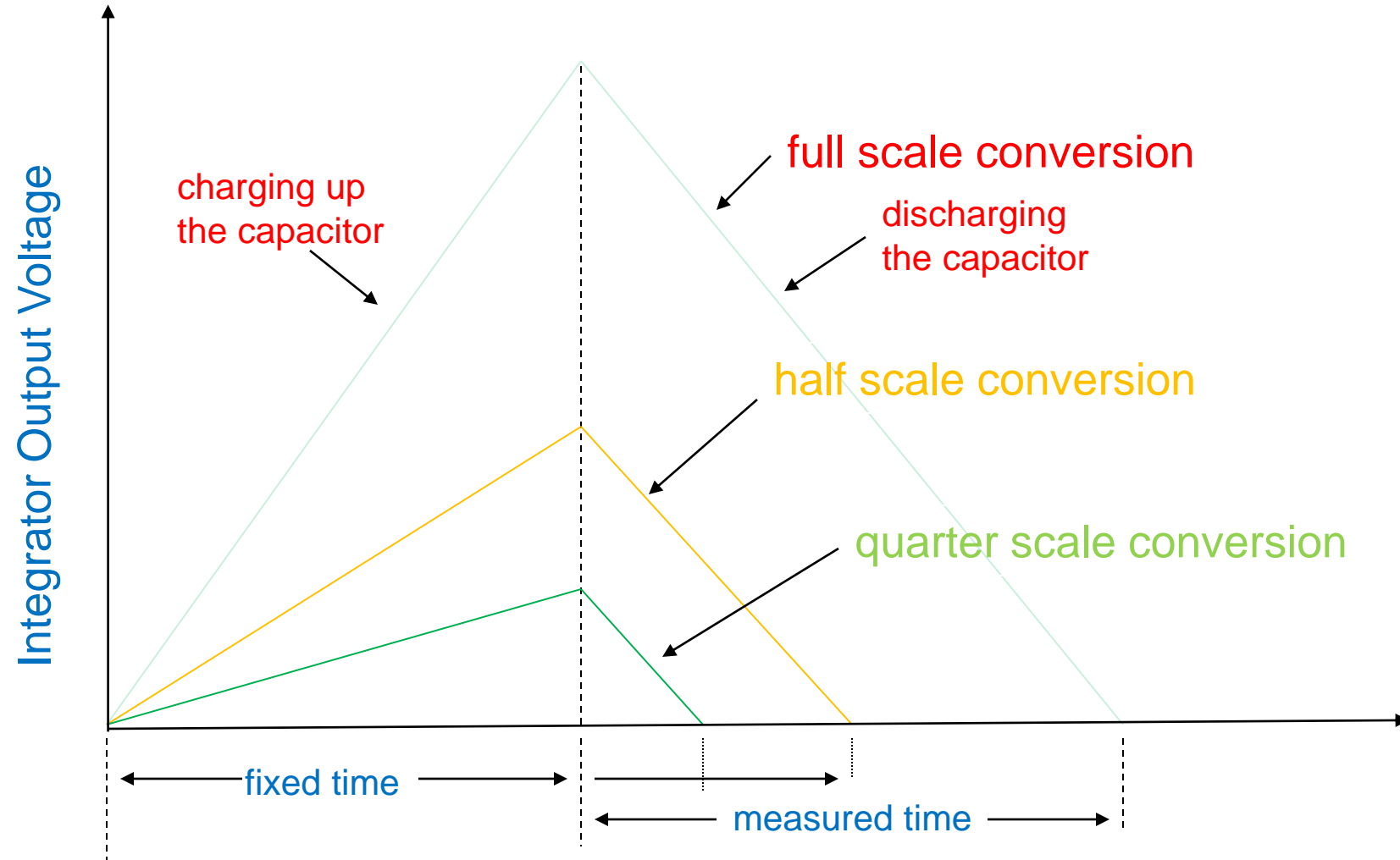


# Dual Slope Converter



- *The sampled signal charges a capacitor for a fixed amount of time*
  - *By integrating over time, noise integrates out of the conversion*
- *Then the ADC discharges the capacitor at a fixed rate with the counter counts the ADC's output bits. A longer discharge time results in a higher count*

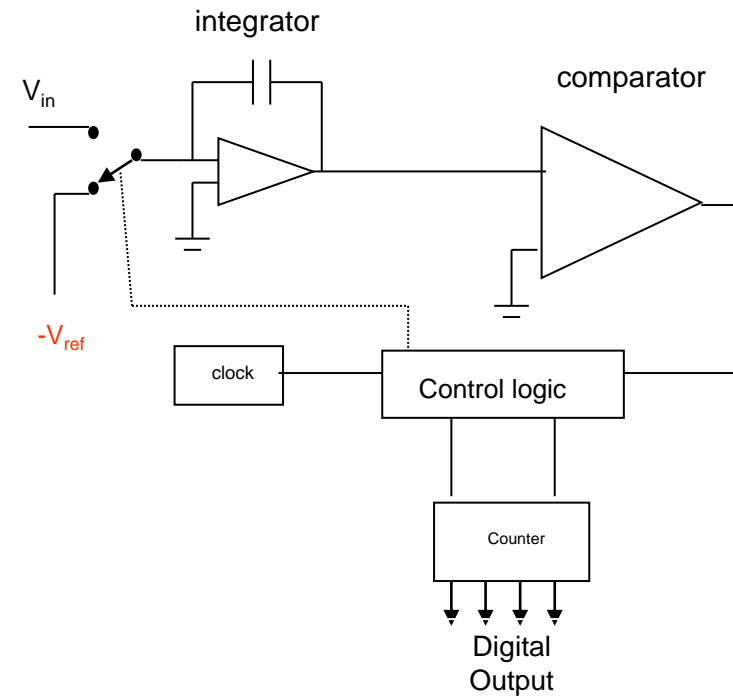
The operation of this A/D requires 2 voltage slopes, hence the common name DUAL-SLOPE.





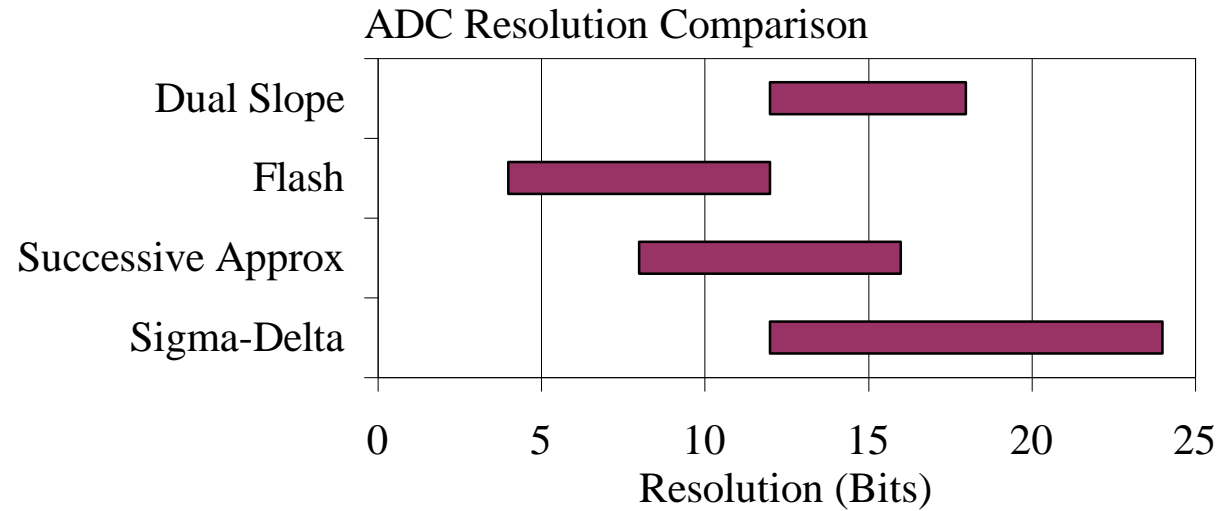
# Dual slope conversion

- When conversion is initialized, the switch is connected to  $V_{in}$  which is applied to the OP amp integrator.
  - Control logic allows input signal to be integrated, charging the capacitor for a fixed amount of time.
- At the end of the charging period, control logic activates the input switch from  $V_{in}$  to  $-V_{ref}$ , applying a negative reference voltage to the integrator
  - The negative reference voltage removes the charge stored in the integrator at a constant rate until the charge becomes zero.
  - At this point, the comparator switches states producing a signal that disables the clock and freezes the counter reading.
- The total number of counts on the counter (determined by the time it took the fixed voltage  $V_{ref}$  to cancel  $V_{in}$ ) is proportional to the input voltage, and thus is a measure of the unknown input voltage.



Since this A/D integrates the input as part of the measuring process, any random noise present in the signal will tend to integrate to zero, resulting in a reduction in noise.

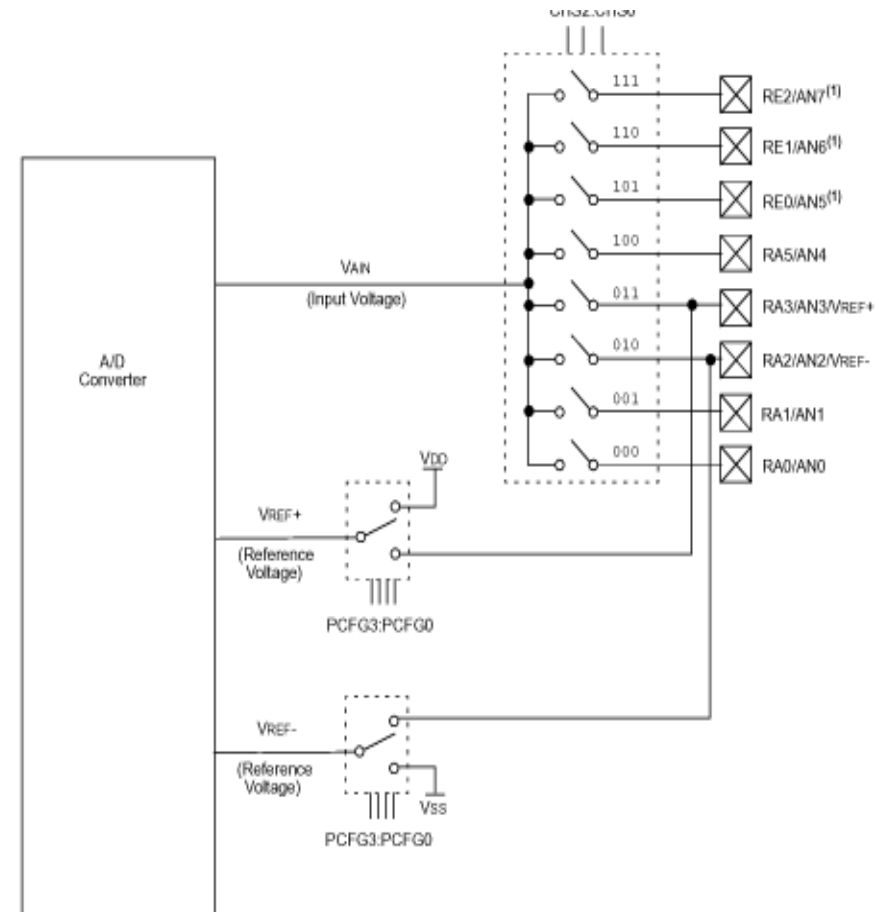
# ADC Types Comparison



Type	Speed (relative)	Cost (relative)
Dual Slope	Slow	Med
Flash	Very Fast	High
Successive Appox	Medium – Fast	Low
Sigma-Delta	Slow	Low

# Example: A/D converter in PIC 16F87xx

- *10 bit Analog to Digital converter with
  - 8 channels for 40 pin devices
  - 5 channels for 28 pin devices*
- *Built in sample and hold circuit.*
- *Programmable conversion clock*
- *Preset or external voltage references for analog input.*

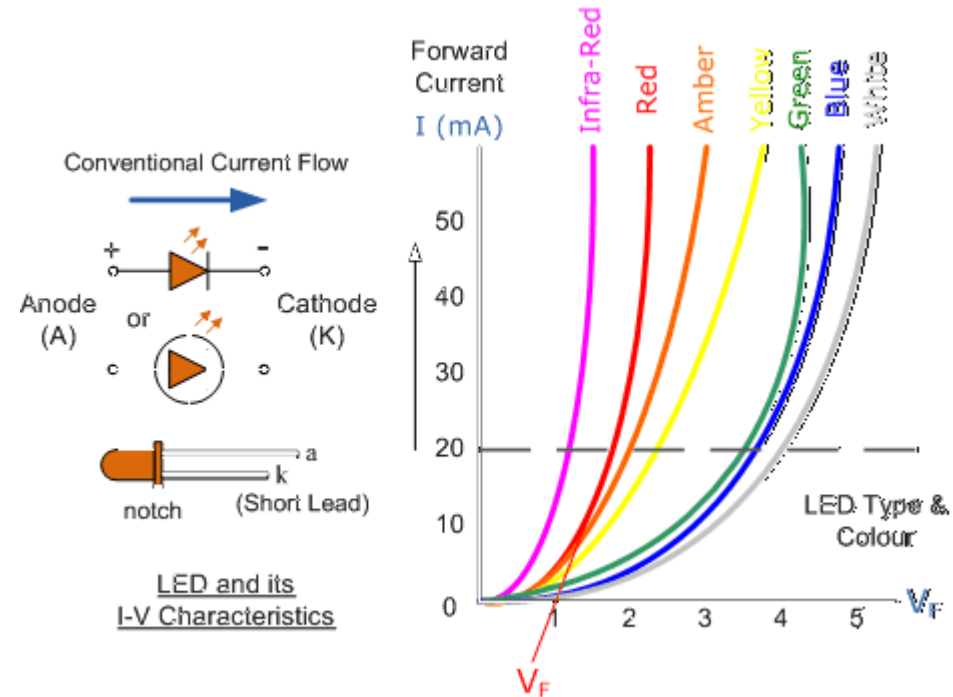


**Note 1:** Not available on PIC16F873/876 devices.

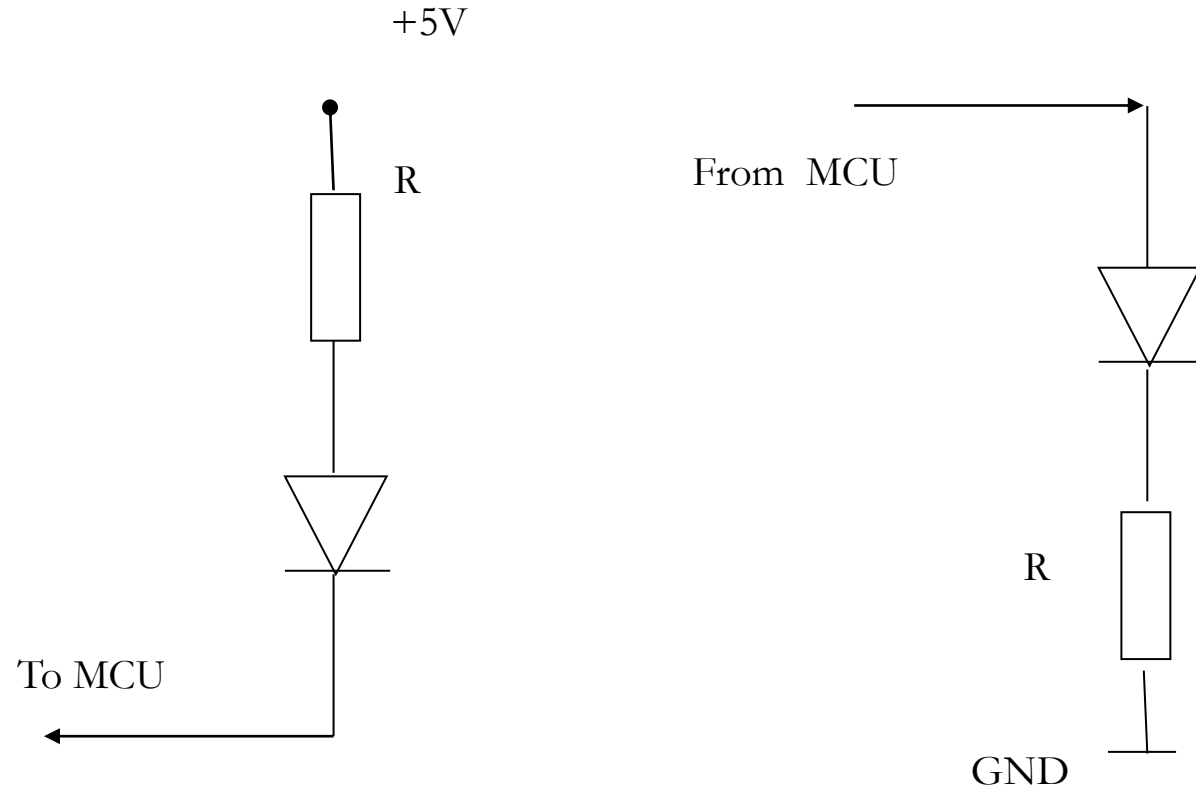
# PHYSICAL INTERFACING

# Driving LED's

- *LED's are some of the most commonly interfaced device with MCU's.*
- *Low current consumption of LED's allow them to be driven directly with most types of MCUs.*
- *However a typical LED consumes about 15-25mA. Thus care must be taken to avoid overloading of the MCU pin as well as the MCU itself.*



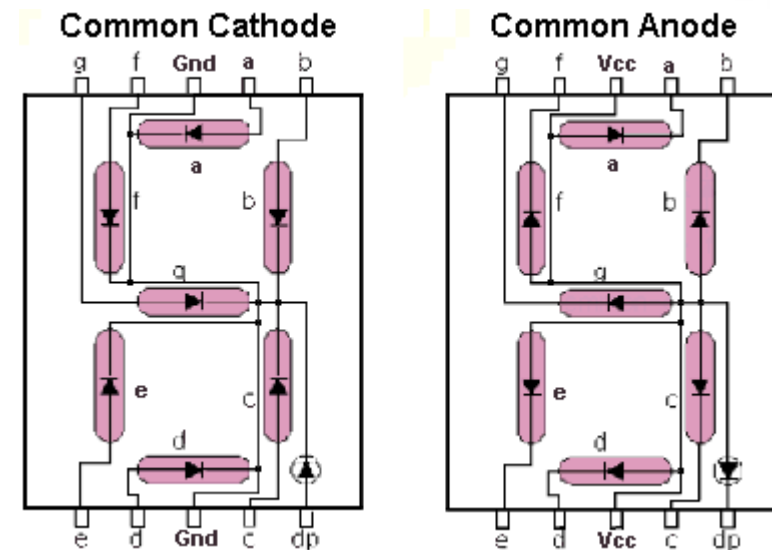
# Driving LED's



- *Resister  $R$  must be selected to limit the current through the LED thereby preventing the MCU from overloading*

# 7-Segment Displays and Multiplexed driving

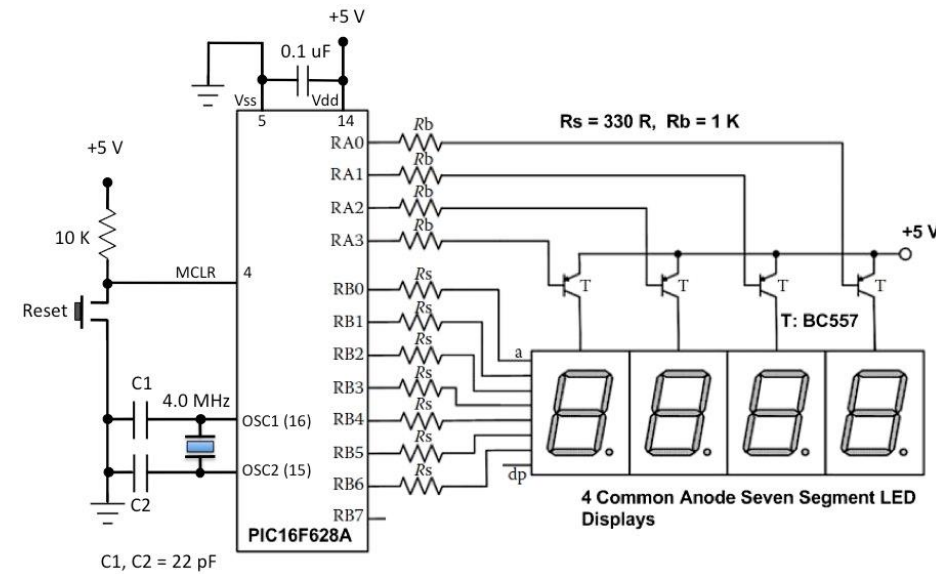
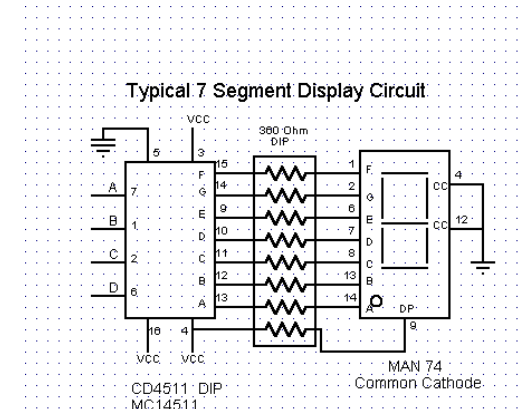
- Consist of 7 rectangular LEDs arranged in the formation of a decimal digit (8<sup>th</sup> LED for decimal point)
- Available in two types
  - Common cathode
  - Common anode
- Each LED must be driven individual with current limiting for uniform brightness
- Multi-digit displays are arranged in a matrix pattern with corresponding segments connected together and driven by multiplexing on the common node





# Direct and Multiplexed driving

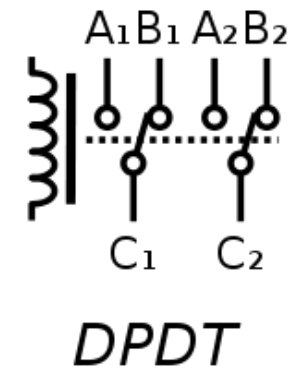
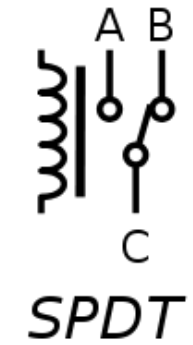
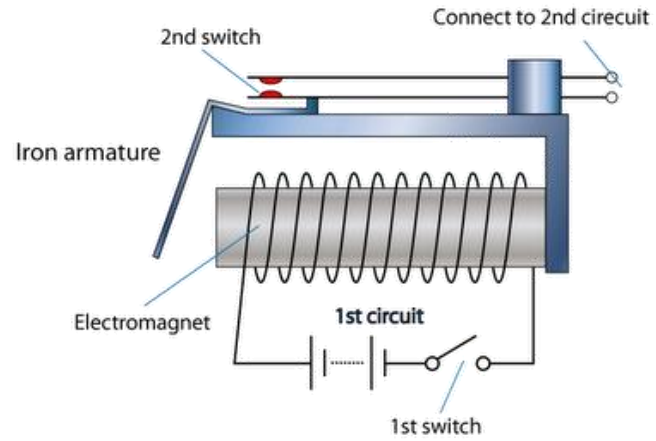
- *A resistor in between the microprocessor and display acts as a current limiter*
- *When using a multi-digit segmented display, the output is driven by scanning across digits*
  - *Use the POV of human vision to create a nonflickering display*



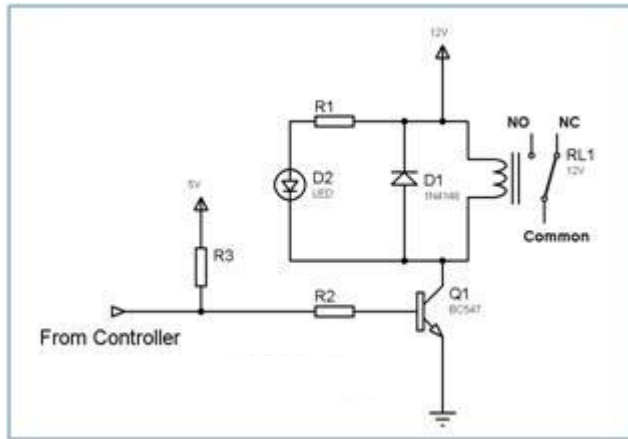
# Driving Inductive Loads

- *Special care must be taken when driving inductive loads directly as well as through buffer stages.*
- *Large transient currents and voltages involved in inductive loads are likely to cause noise and other related fluctuations in the supply voltage.*
- *These must be filtered to prevent un-predictable behavior of the system.*

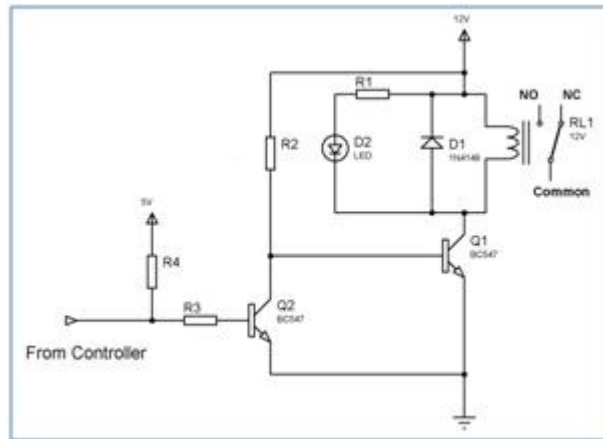
# Electromagnetic relays



# Driving Relays

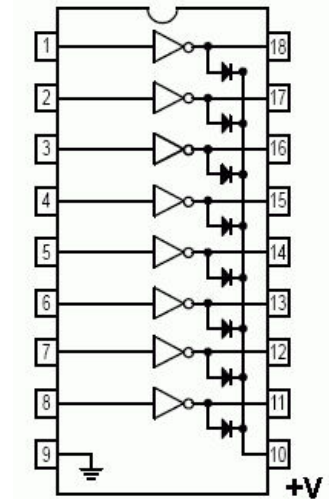


Single Transistor Driver



Twin Transistor Driver

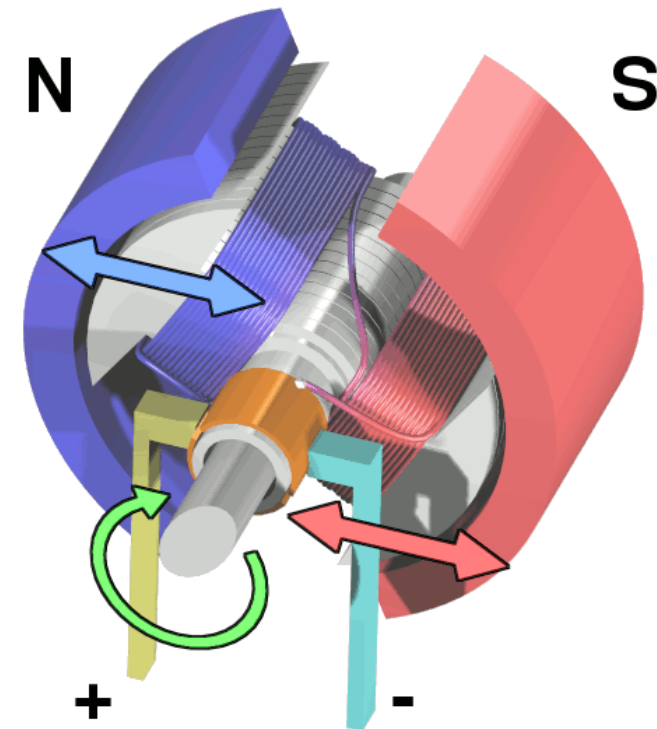
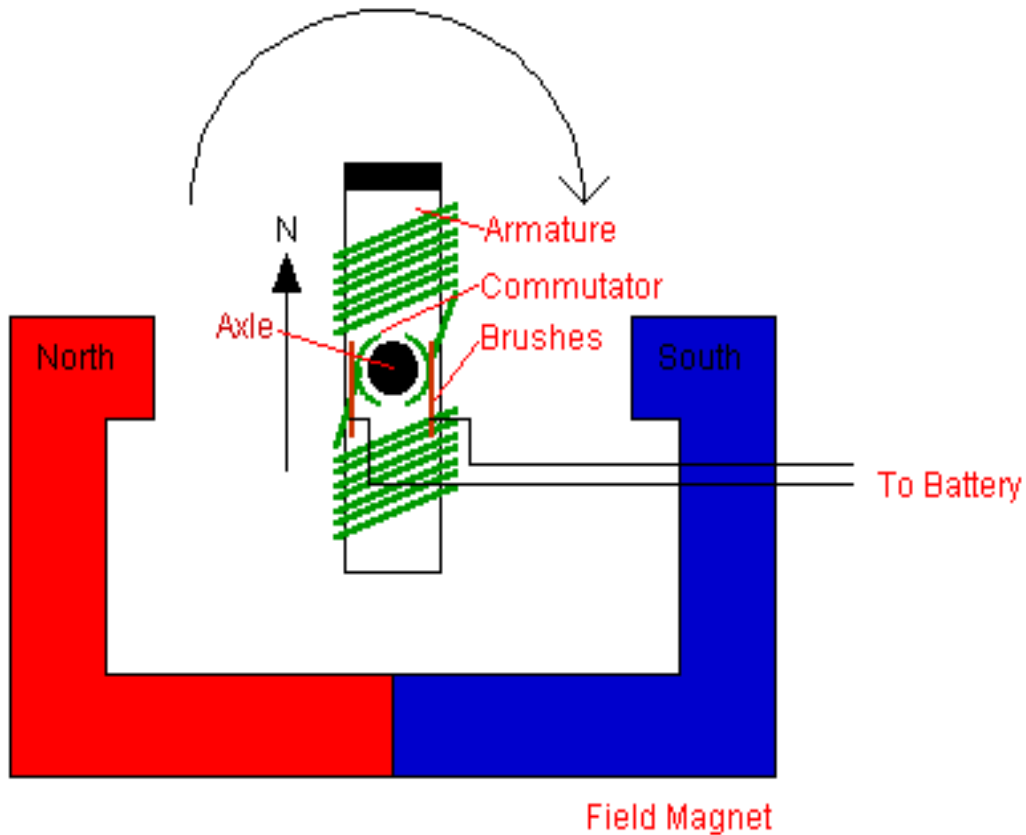
- A shunt diode is required to provide a conducting path for the back EMF generated when power to the relay is switched off.*



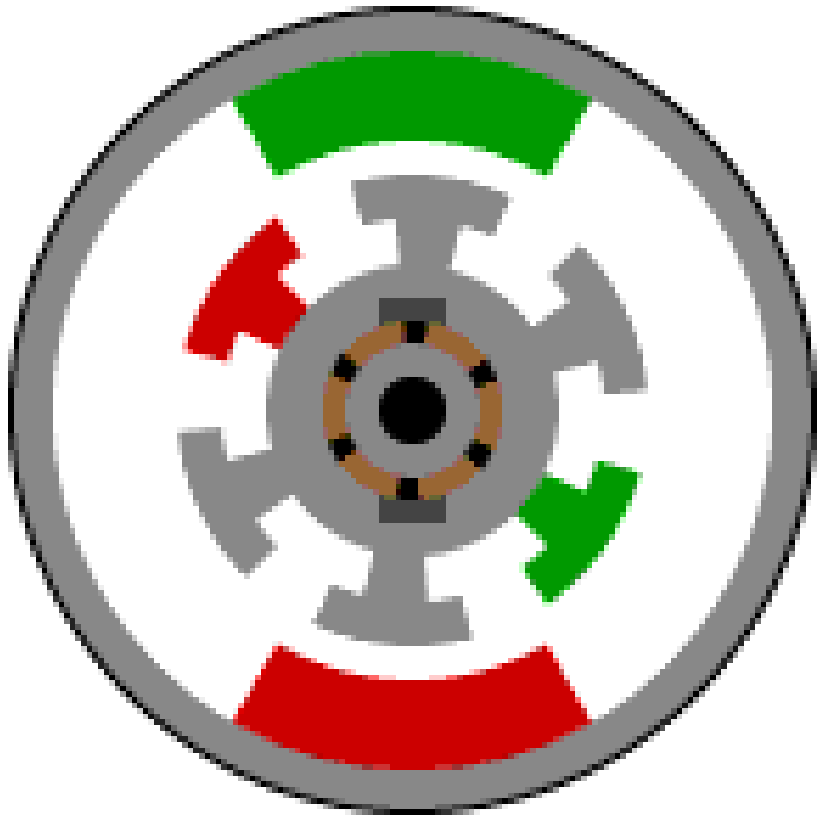
Relay drivers- e.g. ULN2803

# Direct Current Motors

- *overall plan of a simple 2-pole DC electric motor*
- *A simple motor has 6 parts, as shown in the diagram*

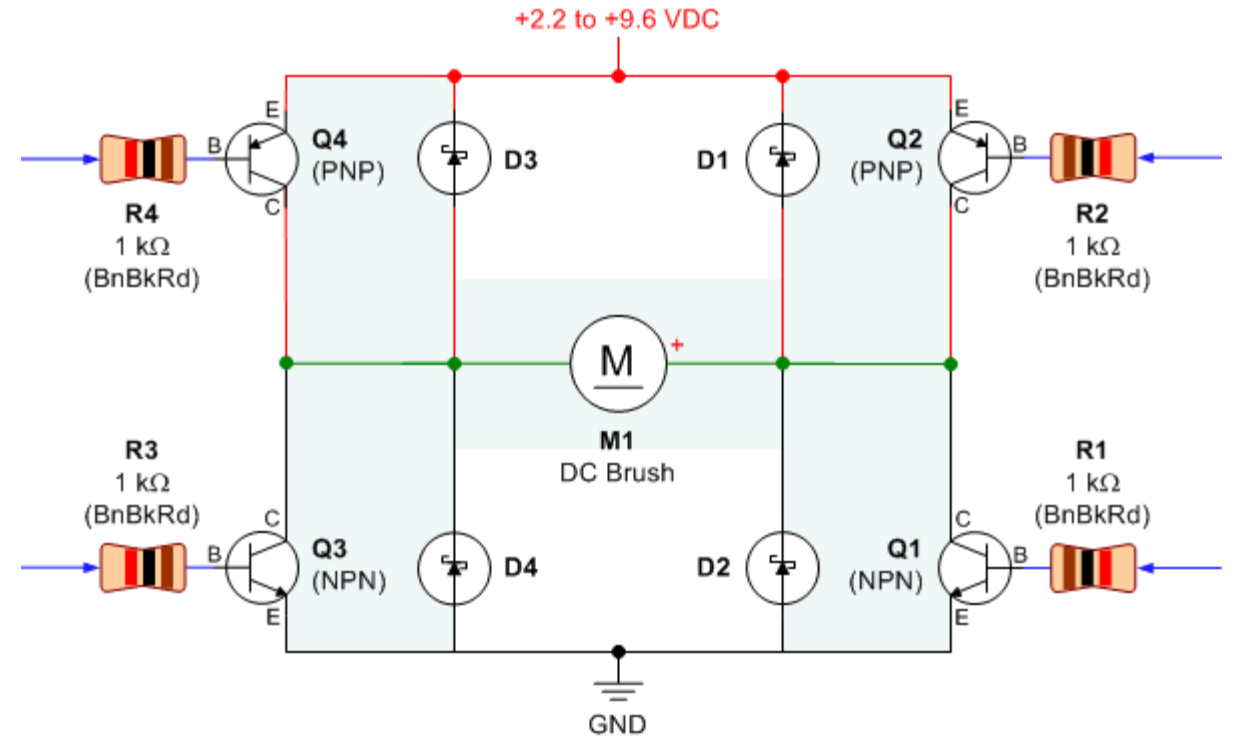
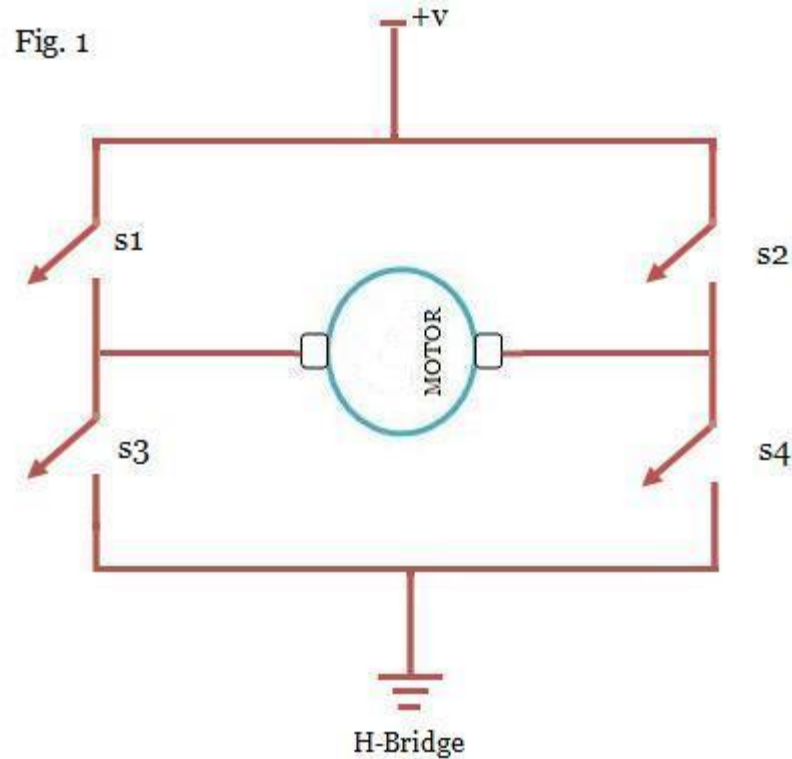


# Brushed DC Motor



- *The brushed DC motor is one of the earliest electric motor designs*
- *Easy to understand design*
- *Easy to control speed*

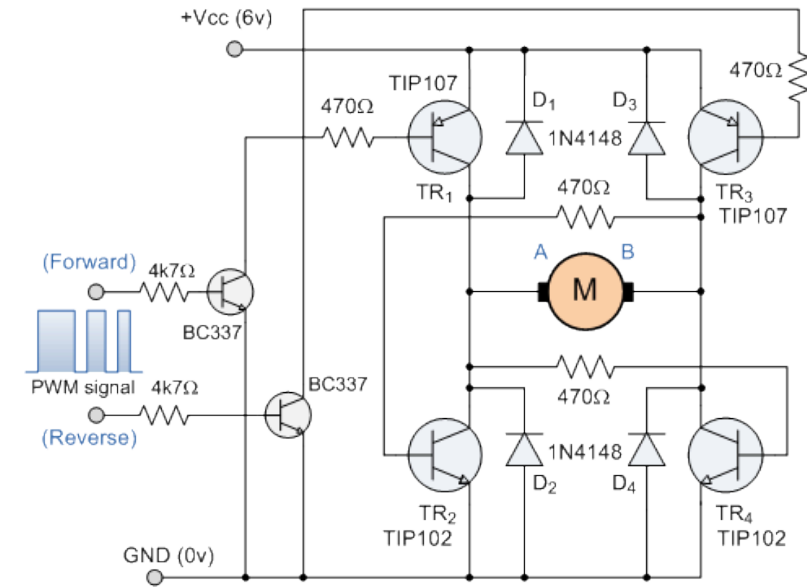
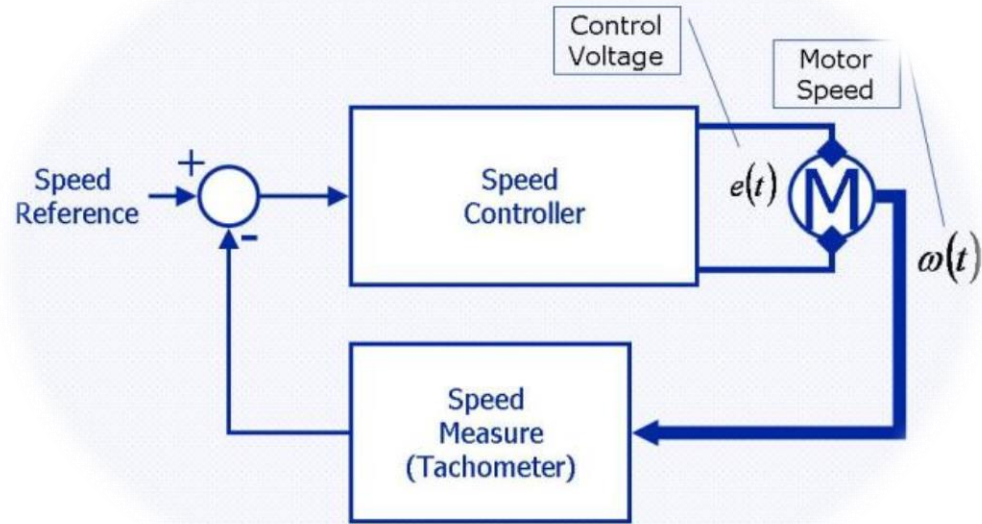
# DC Motors – Direction control



- *Switches are needed to control the direction*

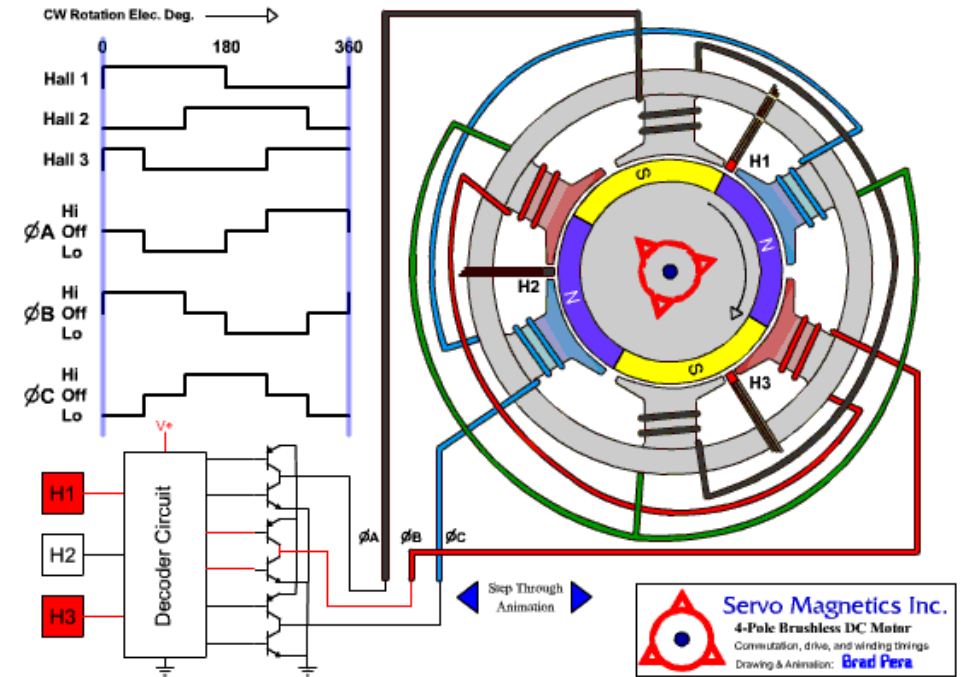
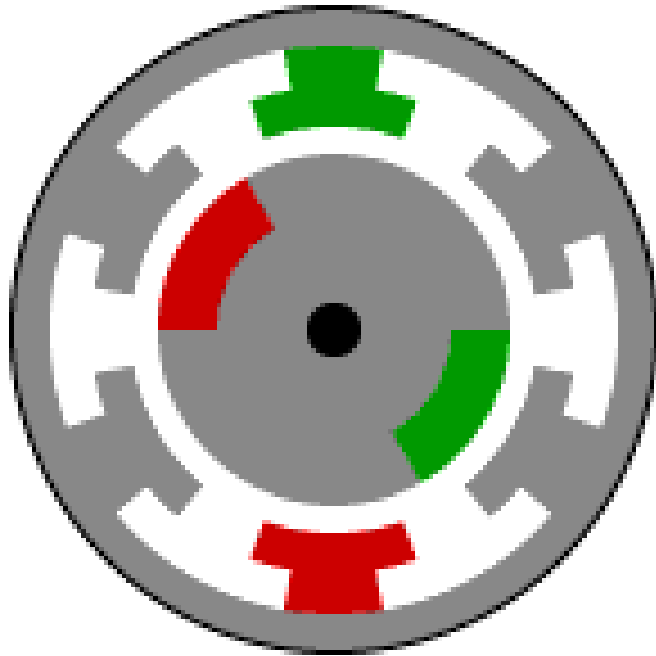
*Switches replaced with transistors*

# DC Motor speed controller





# DC STEPPER MOTORS



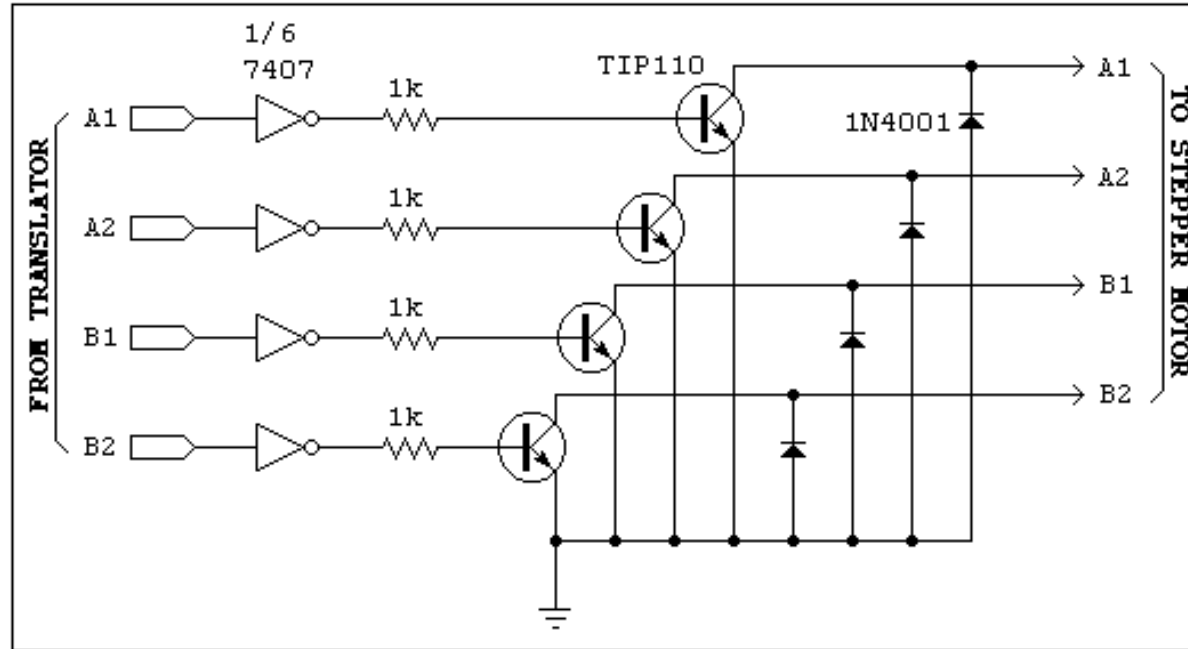
- *Stepping motors are electric motors without commutators*
- *Commutation is handled externally by the motor controller*
- *Controller charges opposite coils attracting the center rotor magnets*

# DC STEPPER MOTORS

- *Voltage Rating*
  - *provides desired torque*
- *Resistance-per-winding determines*
  - *the current draw of the motor*
  - *Maximum operating speed*
- *Degrees per Step*
  - *Sets the number of degrees the shaft will rotate for each full step*

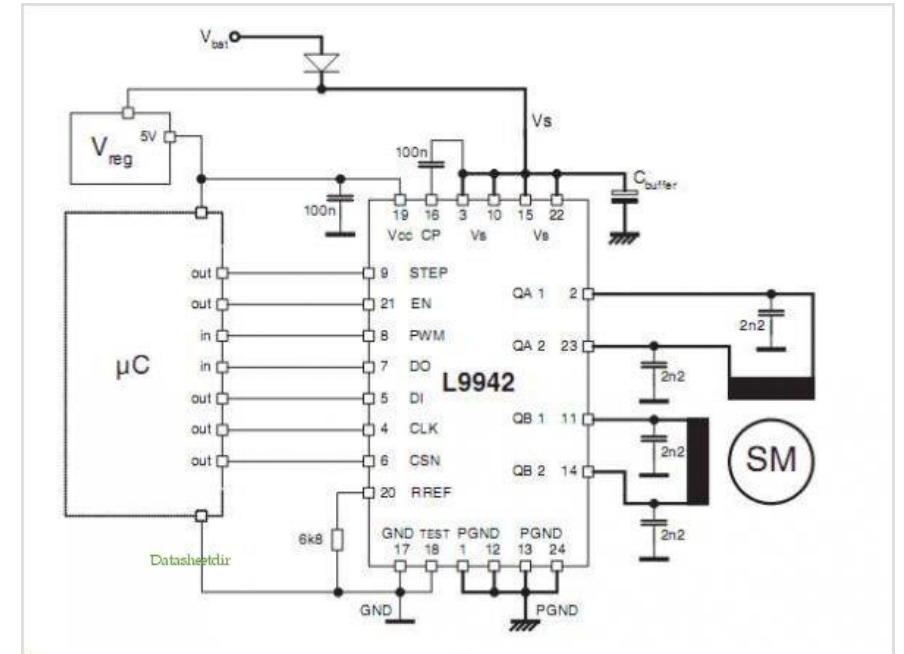
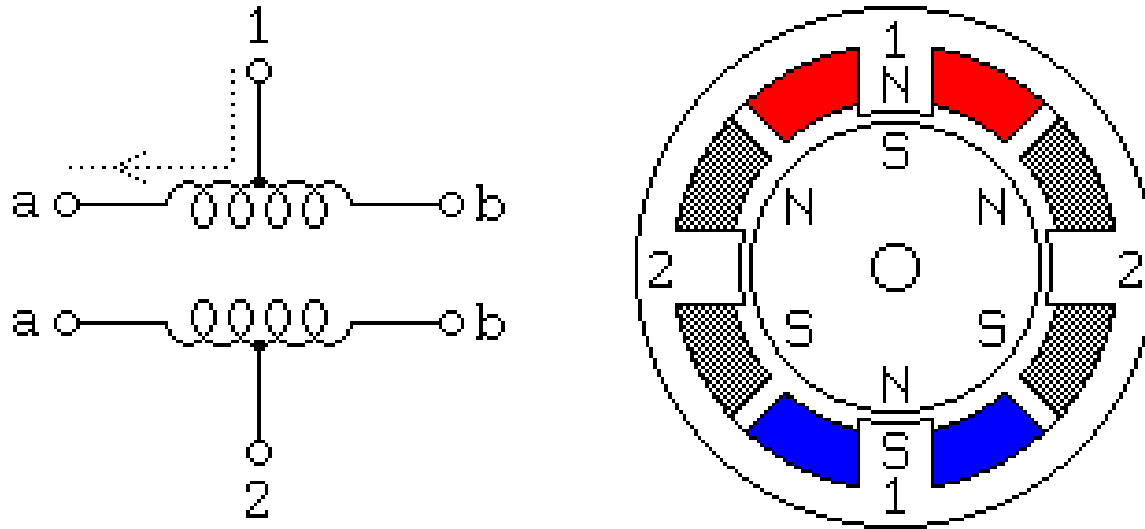


# UNIPOLAR STEPPER MOTOR



- *Relatively easy to control*
- *simple 1-of-'n' counter circuit can generate the proper stepping sequence*
- *1 transistor per winding*

# UNIPOLAR STEPPER MOTOR



- *6 wires with a center tap on each of two coils*

**QUESTIONS?**