L1

Tuesday, March 7, 2023 8:17 AM

Introduction to the World of 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"

- Here we consider a network between the **internet** and **things**
- Connects small systems with other systems into a larger one – aka "System of Systems"
- But in IoT – internet refers to a network of networks (and not only www)

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.*

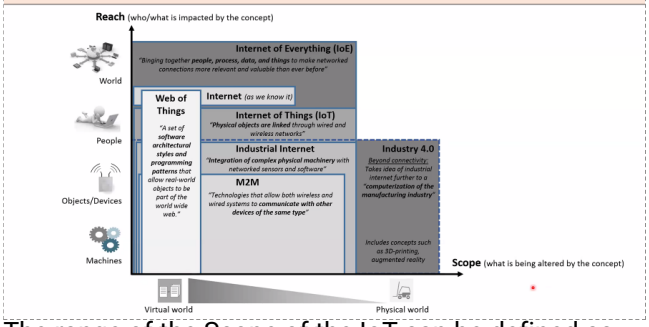
- IoT came to fore front with the development of smart devices
  - Simply a device being able to connect to the internet/network does not make it smart
  - A smart device should be able to communicate with another device to share input and output signals
  - Such smart devices may have to sense various features from the environment (or other devices) and then will change their functionality
  - Traditional devices may have limited predefined functions
    - But Smart devices directly/indirectly can support multiple functions
  - The "smartness" comes from the functionality of the software input – the functions are defined by the software (*software defined networks*)
- So Smart devices have improved the traditional devices to provide better functionality
  - But initially these devices acted alone independently
- But with popularity of internet, computers/devices began to communicate with each other to share information
  - Got easier with RFID and cellular networks
  - Allowed intelligent devices to do "Machine-to-Machine" communication
- A necessity to the growth/development of IoT is
  - Availability of low cost
  - High powered embedded processors
  - Development of mobile data communication networks

Concepts similar to IoT

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



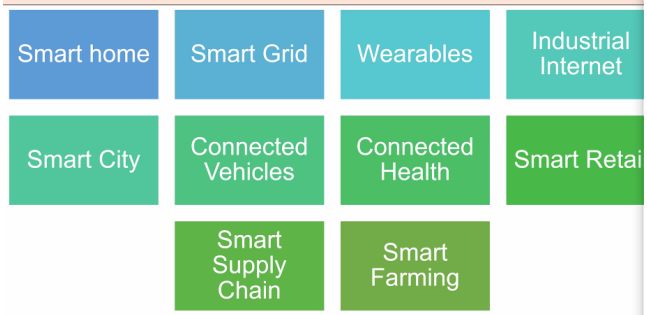
The range of the Scope of the IoT can be defined as

- Virtual world – Only virtually and online
- Physical world – there is a tangible aspect of it

The Reach defines the type of devices they connect to

- Web of Things
  - Focuses on reaching a large amount of users (reach)
  - But only in the virtual aspect
- Machine-to-Machine
  - Focused only on devices and machines
  - But handled a higher range of virtual & physical
- Industrial 4.0
  - Mostly applied in the physical world
  - For devices and machines mainly
- **IoT**
  - Focuses on reaching and communicating for People
  - But opens and communicates in both online/physical domain

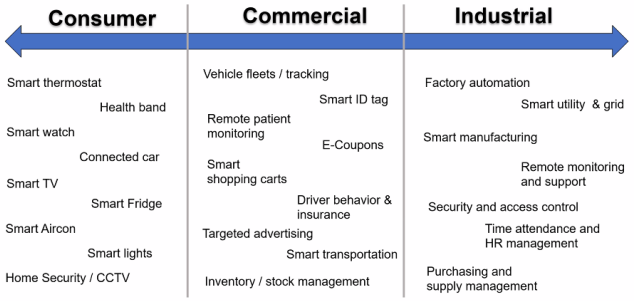
The world of IoT



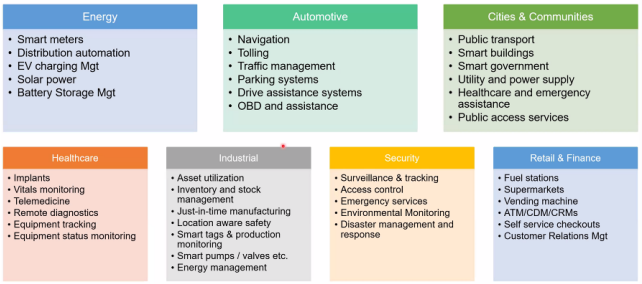
- Connected Devices are those which communicate with another device
  - Rather than connected to the internet to just get updates or so

IoT Sub domains

IoT Sub domains

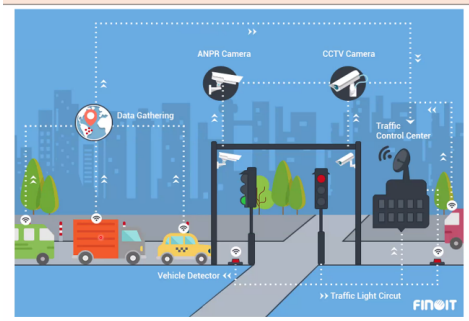


Common IoT Applications



Examples:

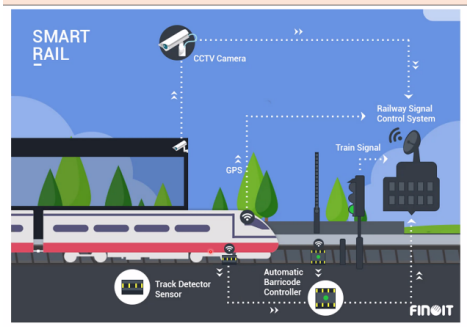
IoT Examples: Smart Traffic Management



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

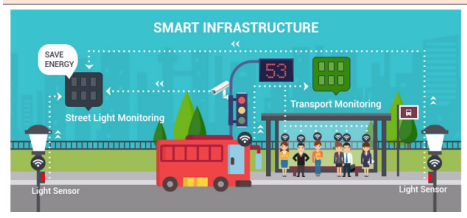
- A control center will collect data from the sensors (traffic cams) around
- The control center can adjust based on the data received and send appropriate to the traffic lights and etc.

IoT Examples: Smart Railway



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

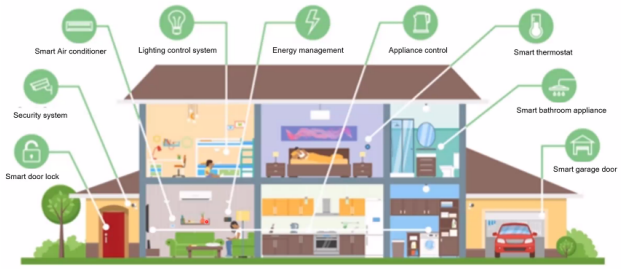
IoT Examples: Smart Cities



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

- Less traffic

IoT Examples: Smart Home



- Such devices can help to collect data as well about usage and etc.

Components of IOT Systems

Sensor & Actuators

H/W Interfaces

Micro-controllers

Software

} Thing

Communication

} Internet-

Cloud Central Services

Big data Analytics

AI / ML

* Most of the IOT devices are Real-Time.

→ Doesn't work with the Cloud.

⇒ Any device which sends data to a cloud and receive from it is not a Real-Time System.

L2

Tuesday, March 21, 2023 8:15 AM

- In today's world there are machines/systems that can be employed to do simple tasks
    - But with the rise of Ai and ML machines may come to an extent that they can think and do tasks which cause concerns of security of our information
- There a lot of unresolved social issues with IoT and usage
    - Like driverless cars unable to take the blame for accidents
- It is important to make IoT fail-safe from the beginning
    - Referring if it makes a mistake can the damage be minimized without consequential damage

Solar Harvesting

- Most cases power is consumed highly at night time
- But off grid systems can't get power at night and need to save the power from sunlight at peak light
    - This requires batteries
    - Like Li-ion, car batteries
    - Their characteristics depend on the lifetime, usage and charging cycles
- The battery capacity can defined by the average household usage
    - But it should be planned to work optimal
        - Without overcharging and discharging fully
    - An inverter will direct power from all sources
        - But doesn't know how much it should allocate especially with unexpected events

- For this purpose one approach is to use a Raspberry Pi board to monitor and sense the batteries
    - Software can be written to collect weather info and power cut schedules
    - Using these it can decide the most optimal form of allocating power to minimize utility usage

and power cut issues even with small batteries
- <<Check the CAN bus>>

The Components of IoT solutions

- It is everything
- Two parts
  - Higher focus on this components type
  - **Things** like Sensors, Hardware Microcontrollers software, actuators
    - There interact with users
    - Provides commands with actuators
    - And sends input to others
  - Hardware interfaces are needed to provide interface between sensors and microcontrollers
    - Useful to match the analog signals between them into matching ranges and convert to digital form
  - Software may process
  - This complete process is known as **The Thing**.

Technologies are
  - Embedding Processing
  - Fog Layer
  - Edge Processing – some level of the processing can be done on the device – like services which don't need internet to do and locally applicable
    - Hence IoT is not purely internet reliable
      - Eg:
        - Driverless cars managing on roads
        - A/C's changing temp
    - But there are constraints
      - Like low power and low memory and low computing processing power
      - Solutions include sleep mode to save energy
      - Edge computing can be less reliable -

        - Hence should be failsafe
        - And fault tolerant – even if a fault occurs some level services should be able to do
            - Like a lift must stop at closest floor when lift failure
    - Since there are needed to make at larger quantities, consider their economics

---

- **Internet** include Cloud central services, Big Data analytics, AI/ML
    - For example complex computations may need to do on cloud computing
    - The internet allows to do complex analysis & logic for the complete solution
- In the Internet, we need to handle many types of data like audio, video
    - Such technologies to handle them are
        - Computer Vision, Voice recog, NLP, Analytics, ML

- Between the two there should be **communication**.
    - Difficult compared to PC's to connect
    - Mainly due to low power consumption – battery powered
        - Like Wi-Fi can't take-up a lot of power as some devices need to move around
        - Also complex services AWS, cloud etc can't be used
            - Hence need to go for simpler efficient versions

Challenges and Issues of IoT

- Security
  - Protecting the access to those devices – both physical aspects and logical/virtual/remote protection
  - In general, we may not be knowing there are IoT devices without knowing (**known as visibility issues**)
  - Lack of controllability – unlike a physical device, a user may be unavailable to make changes
  - Malware & DDOS
    - Or even attacking it
- Privacy
  - Protecting the person from IoT
  - Data Exposure to wrong hands
  - Personal privacy of information
  - Unauthorized access through the device
- Tech issues
  - Built on mature tech but IoT is a fairly new concept where it combines them
  - Especially with rapidly changing environments there are economical risks of manufacturing
  - Highly dependent on external factors like power and network
  - And also on their infrastructure is dependent

Market Capacity for IoT

- High potential and exponential growth for IoT devices
- Relatively IoT security is low market
- The sectors of IoT which will grow
  - IoT Security
  - IoT Hardware
  - IoT Services
  - IoT Software

L3 – I – Anatomy of a Thing

Tuesday, March 21, 2023 9:35 AM

- A thing is divided to 2 parts

Devices

- Sensors
- Actuators
  - Giving some signal output for the device to do something
- Interfaces
  - Have to connect with internet and also how devices connect between each other
  - Like connecting between sensors and actuators directly if no computation is required in between
- Local communication
  - Controller and sensor communication
    - Note some sensors can be iot devices even
- Compute
  - The processing power needed to run
- Remote Communication
  - Over the internet communication
- Power Management
  - Need to do components to work at low power
    - Or even remote power
      - Like ID cards with microchip which gets powered from RF radiation when used at the place
  - Limited power and need to manage it all

And the others are

Operations

- Realtime Response
  - When input is given it should change
  - Event driven operations where an event occurs and a signal is given to do a command
  - Some cases Realtime responses and some with real time guarantee too
- How to operate with Low power operation

    - Eg: When connecting online it is inefficient to be online continuously
    - Bluetooth in cases will require reconnection when not used and (and special version goes to sleep state)
    - Though it is easy for IoT device to send signals to others with low power protocols
        - But more challenging to know when other devices want ot communicate when it is sleeping
- Harsh env
    - Handle difficult env
- Failsafe operation
- Unreliable communication
    - When moving about need to reconnect to a new station seamlessly
- Limited resources
- Self-configurable
    - Need to able to connect to internet without passwords easily
    - Should be transparent

Example : Smart Watch

- Sensors
    - Heart Rate Monitor
    - Oxygenator – Blood oxygen
        - measuring with light sent and light refracted/reflected back
    - Step counter
    - These are measured in indirect ways to get estimates of measurements
    - A GPS receiver can be there
        - But unreliable - like in a building
            - So needs to store the last known location
    - Wi-Fi and Bluetooth connection
- All of them have to be powered by a small battery – maybe wireless charging in some cases
- And also need to be cheap to produce them
- Few of computing is done on the device
- Rest need to connect to internet to get the services
- We may send signals via the phone or directly to device

    - Between those connections can include Wifi or Bluetooth
- By connecting the watch to the phone we can get an IoT I/O terminal which to setup and give inputs and read outputs
- If the phone is offline the watch may need to reconnect to internet to do its tasks
- The challenge is making all these aspects are working together seamlessly (check recording)

Comparing Traditional and IoT

Traditional

- Designed for data intensive
    - Large memory
        - By Von Neumann all required memory is needed to be present for computation
    - Powerful processors
    - Faster connecting
    - And have large devices like screens and keyboards
- I/O Requirements
    - LIMITED and general
    - The processor can be used for wide variety of tasks
    - The I/O ports such as USBs are generic and allowed for multiple type of devices to be connected
        - These ports also can be used for purposes other than data transfer

IoT

- Not intensive
    - Specialized to communicate with environment
    - Just limited to what is required to do its task
- I/O Capabilities
    - So much more I/O requirements
        - Like analog devices, interfaces, character modes, outputs, high energy devices
        - It will get some data from the env and then process it

        - And repeat again for another task
        - Hence its not needed to have a high memory as it highly depends on the env for input to work on
- Specialized processes for the task
    - If the Iot device is for mainly sensing – then it would more catered for I/O
    - For video processing more emphasis for graphics
    - This helps to make more low energy processes
    - The I/O processes are highly specific
        - Like some record the time a connection was formed etc

In IoT

- Smaller in size and weight
- Smaller processing
- Smaller storage
- Smaller memory
- Small display
- Lower battery capacity (20x smaller)
    - But needs to run for a longer period
    - So longer endurance
- Have GPS sensor and low energy Bluetooth connection
- And maybe large complex sensors
- High device complexions
    - But highly restricted with resources
    -

L3 – II – Anatomy of a Thing

Tuesday, March 28, 2023 8:11 AM

- When generating input signals there can be unwanted noise that comes with the data
- The main focus is how the data transfers between the components

- IoT controllers act as the brain of the system while Peripheral systems are connected to it

In a IoT system there are 2 aspects to look from

- Devices connected
  - There are multiple things such as sensors, actuators, interfaces, communications etc.
- And the operations
  - And how are they facilitated
  - Here we expect the operations to work under some conditions such as real-time, limited resources such....

The heart of an IoT device is a computer processor

Moving from computers to IoT things

- In General Computer Architecture
  - There is a time factor
    - Due to multiple buses operating at different speeds that connect between multiple types of devices
      - This is to handle different speeds
    - Then there should be a buffered I/O to handle them
      - To prevent collisions from changing speeds of transfer
    - But this makes the data relatively slowed from real-time
  - Memory is large and hierarchical in nature
    - And connected to multiple peripheral memory objects

    - Generally it is assumed that computers have sufficient to do high processing
    - Reliable but *not necessarily fail-safe*
        - Fail-safe doesn't refer to whether the device will fail or not
        - It refers to if the device fails there won't be bad consequences
            - The reason is normally because the computers are usually monitored by a person at all times
    - Has many expansion features for the users
    - Designed to facilitate multiple purposes and software
        - Covers a wide range
        - BUT not in depth of a particular field
    - Optimized for CISC Archi processes

- Moving to an IoT thing
    - In I/O devices, may be slower but memory access may be faster sometimes
    - Handles a few I/O devices,
        - But they work in real-time
        - No use of buffers for data transfer to ensure the data is ready as soon as possible
    - Focus on energy saving & reliable operations
    - Uses minimal devices & components
        - More devices ==> more probability of failure
            - Unless there are designed to avoid failure
        - Hence it is better to reduce the number of devices
    - Built with fail-safe features
        - Provides stats such as mean time to fail
        - Some IoT devices have supervisory circuits/modules
            - They check if there is an error and will notify

and take some suitable action to restart or to fix it up
        - These modules work independent of the system and only monitor
    - Designed only to support a few function
        - Like using RISC
            - Which are highly efficient and optimized to do the task
    - Fault tolerance
        - Even when it fails there is another device/module to standby and continue the operations
        - There are different ways to apply such active-active
        - IoT devices are **not necessary** to be fail tolerant
            - But in critical operations such as aircraft then it is needed
        - Some cases it may needed to have the standby in another system or module implementation to avoid duplication of issues

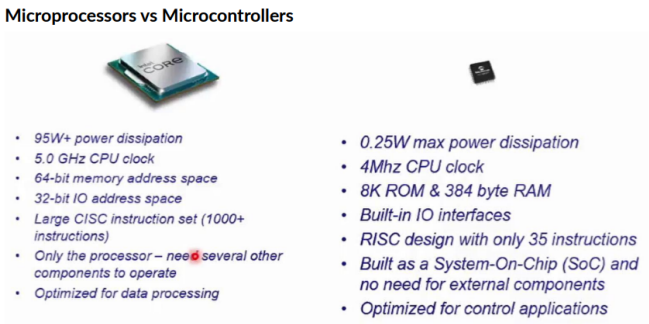
<<< **Paste picture of highlevel diagrams** >>>

- In the Computer Motherboard
    - The fastest buses are found to be at
        - Connecting the screen (Graphics)
        - And the memory
    - There is a bus to connect the memory controller and I/O controller and these will branch out to the relevant devices and components
    - So the true speed of the CPU is only realized by a few devices
        - And the other devices gets a slowed down speed

- In a microcontroller IC
    - One the components in ICs are Power management

        - The power management system is connected to the clock
        - This allows the IC into hibernation by slowing the clock or putting it to sleep
            - Saves power when not in use and can restart when needed back
    - Supervisory circuits
        - Sometimes when there is a problem it can be slowed down to check if there is an error and restarts to go back to working state
        - But they should be properly programmed to avoid repeated and infinite loops due to incorrect conditions
        - A solution to this, is using a watchdog timer
            - The program logic we should ensure if the limit is not reached when working at normal operation
                - So have to reset the timer
            - But if there is an error the computer hardware will go into hard reset and then a fault can be properly identified

Microprocessors vs Microcontrollers

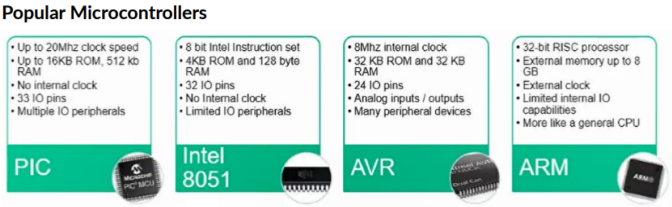


- In Microprocessors
    - Very high power/heat dissipation
    - High clock speed (5GHz)
    - 64-bit memory (2^64)
    - 32-bit I/O - not built in

- Microcontrollers

- There is a limited memory available
- Also the I/O is normally built in into the systems

<<Picture of popular microcontrollers>>



PIC – Peripheral interface controller

- PIC has built in memory

In the Intel 8051

- It has the option to reconfigure part of the IO pins into another memory bus to connect a larger external memory device
- But this makes the no of I/O devices to connected is limited
- But can only take in digital devices

AVR

- Used for Arduino
- It has analog inputs/outputs
- At face more memory, more processing and may seem better than other options
  - Therefore, important to compare and decide depending on the purpose of the use of the microcontroller

ARM

- Found on Raspberry Pi
- Limited internal IO capabilities and need to connect
- In general its features seem more and more like a CPU chip instead

History

Microcontroller as a System on a Chip (SoC)

- It is important to have a software to run the system needed even if the chips are available
- Note it is difficult to debug the IoT since there is limited ways to check the outputs
  - Hence most would code on a separate computer and then insert it into the chips

- For this purpose, now it has moved into building devices to insert the entire computer processing into a single chip itself

In the Parallax Basic Stamp

- Tiny device – connect to bread board and power and can run some processes and working
    - Could handle 10,000 basic instructions per second
- One serial port is needed to connect for the development purposes to check the outputs and another one is also provided
- It uses EEPROM which can be reprogrammed using electrical signals
    - At present, it can be reprogrammed while it is being used itself
        - Like Arduino

With the development of this, it lead to the development of microcontrollers

Microcontroller as a SoC Examples
AVR

- Only a small part of RAM is needed into the microcontroller chip
- Outside the chip there are multiple peripheral devices
- The bus is internal and cannot be accessed from outside
- The analog comparator is there to check is analog data or digital data
- Also have analog/digital convertors
    - This helps to connect analog devices directly

- There is Special Serial Peripheral Interface (with a 3-wire In/out) which allows to connect to the EEPROM
    - This allows to reprogram incircuit as well
- JTAG interface helps to control the CPU clock
    - It can check the contents of the registers and help decide whether to control the clock or not

Esp8266

- A soft microcontroller

- More optimized for low power
  - Can disable the modules not necessary to avoid power being fed to it
  - Unlike the AVR – those modules which may not disable and let them run at sleep mode like
- Additional had a RADIO receiver (Wi-Fi module) and can connect wirelessly as well
- Has a lot of peripheral devices
  - Because they can be easily disabled as well
- Many I/O options
- The improvement is the Esp32 chips with higher complexity
  - Powerful Wi-Fi adapter and Bluetooth devices
  - Large Core memory
  - Convertors
  - Infared
  - Other parts include
    - Radio
    - WiFi
    - Cryptographic hardware acceleration
    - Core and Memory
    - RTC and low power management subsystem
    - Peripheral interfaces

- There is IoT device language called "Lua"
  - With a higher disciplined and highly structured
    - A descendent of pascal
  - Specifically designed for IoT development and design

L3 – III – Anatomy of a Thing / Microcontroller

Tuesday, April 4, 2023 8:14 AM

- Microcontroller were created with the basic components of the computer
  - These ones should also be able to take in I/O though not needed in a computer
  - In-built battery needed to power the RAM to not lose the RAM memory
- Now Microcontrollers have all inbuilt in a silicon chip

- Microcontroller with WiFi modules can help build peer-2-peer networks and even act as an access point for multiple devices to connect
  - But since they are wifi connected they may have issues with battery and power as well
  - The Esp8266 was designed as a plan only and allowed for other manufacturers to update and improve them as they required

Software of IoT Things

DATA DRIVEN	*IO DRIVEN*
• *Lots of data-in-memory processing. Memory loaded with data in advance*	• *Most data is in the environment (i.e. sensors) and available only when a change has occurred*
• *Large, fast accessible memory space*	• *Limited data in memory. Speed and size not critical*
• *Complex instruction set supporting different types of number crunching procedures*	• *Asynchronous data use: Data arrived when the environment reacts, not when the algorithm needs them*
• *Synchronous data use. Data is available in memory whenever instructions need them*	• *Simple, but fast instruction set*
• *Internal events. Responses are mostly to internal changes*	• *Need to provide real-time response to external event*
• *Ideally suitable for stored-program serial processing*	• *Need to handle multiple external events at the same time. Most requiring some immediate processing*

There are 2 types of Software on Things

1. **Data Driven** (mainly for computers and programs)
   - High data processing. And has memory in pre-loaded
   - Uses CISC
   - **Synchronous Data Use –** Data is sent to memory whenever the instructions need them
   - Responds to events occurring internally
   - Best for stored program & serial processing

<<<<Complete this>>>>>

2. **IO Driven (**mainly for Microcontrollers)
    - Unlike computers – need to collect data from the environment
    - Limited memory capacity
    - **Asynchronous data use –** Data may not be able to be sent to memory when required
        - Only when it receives reaction from environment
    - Real time response to env
    - Handle multiple external events

<<<<< Not completed >>>>>>

Software vs Hardware Implementation

- Computers use the software to do the processing

- But microcontrollers & IoT needs more processing to be saved for its main tasks
    - So it uses hardware and software to do the processing
        - Like filtering the noise from input from env
    - Hence IoT has hardware that is mostly specialized for the device
    - And it is generally good to do some of the processing through the hardware
        - For example, to draw a circle
            - In a software side it is easy to give the command to do the task
            - But in the hardware level side (like the graphic card) needs to do much more computing
            - The reason the graphic card can do it fast is because that is the main/only task to be done by the card (as the software may focus on lot of the other parts)
        - Also since there is hardware level processing

– multiple tasks can be done at the same time parallelly
    - Like to press a button – on the software side, a computer will have to periodically check if the button is pressed which is inefficient
    - But it is better to have a hardware module (that is dedicated) which will inform computer when it is pressed
        - While the software can focus on another task
    - In certain cases, the edge processing can handle a few of the tasks too
- Edge Processing - Does some processing at the edge
    - So the data that is passed through is conditioned as well
    - Edge Processing for the input is known as **interface**
        - Like noise reduction
    - Edge Processing for the output is known as **Driver** (driving the output)
        - This is also required if the output is not compatible to the device needed
    - For all this it needs to communicate the multiple devices and requires internet/wifi and a suitable common language/protocol
        - Examples of languages include like positive logic (1-signal), or negative-logic (0-signal)
        - For protocol communication we use HTTP, TCP

- For IoT devices have specific comm protocols
    - Examples: _____
    - These are designed to work in unreliable environments

        - Like has a "last-will" command in case of failure
    - And mostly handles only smaller sized packets

What is required to Develop a Thing

- **Hardware**
    - Development Board
        - Even though we write the program on the microcontroller, there should be a way to insert the program as well
        - The development board helps include all the requirements for the microcontroller chip
            - Like power, clock, debugging, programming port(USB), terminal interface (GUI/Text), Reset Circuitry
                - Terminal Interface
                    - There are 2 types
                        - Board with a head
                        - Board without a head
                        - <<<<Recheck on this>>>>
                - Debugging
                    - It requires in-circuit debugging
                    - Allows to run in the environment itself without removal by bringing a PC to it nearby to do the checking
                    - It may have less features to de debugging but practically easy
    - Sensors & Actuators

        - Some maybe Digital I/O modules
        - But most of them are analog and they need to converted at the edge level or in the I/O module
    - Other Electronic Components
        - Capacitors, Transistors, wires, protection
    - To capture the output during debugging or such we can send the output to a serial port and have the data be displayed using a computer running a terminal software

- **Software**
    - Development Framework
        - Collection of different software components like
            - Programming Languages
                - There are now languages which are dedicated for microcontrollers
                    - Like C++ special version
                    - C sharp
                    - Lua
            - Embedded OS and firmware
            - IDE env
                - The program is written here and tested to be run
            - Compilers, linkers, debuggers
            - Firmware upload tools
                - Like the hardware components need a software to control them (like support libraries)
    - <<<< **Incomplete** >>>>>

- Another platform like Arduino is NodeMCU
    - This has an inbuilt Terminal in the software side
- Also Raspberry Pi

L4 – I – Microcontroller Internals

Tuesday, April 4, 2023 9:26 AM

- The core difference is covered by the differences between Data Driven and IO Driven applications
  <<<< Paste Slide Image >>>>>

The different views of Hardware & Programmers Views of microprocessors

In both the hardware and Programmers view

- Connects both the hardware and programmers views and important to know how to convert in between each other
  - Interfacing
  - Program upload
  - Interrupt Processing
    - A lot of interrupts are needed in microcontrollers
  - Peripheral devices
  - Supervisory Features
    - As told earlier – IoT should be failsafe
    - They look and monitor the activity
    - Improper programming of supervisors will impact negatively the program

In the Hardware aspect

- Doesn't care about the instructions the hardware works on
  - Power supply
  - Clock requirements
  - Memory types
    - Is it flash memory/EEPROM
  - Architecture
  - Bus timing
    - When do the signals need to be active and when

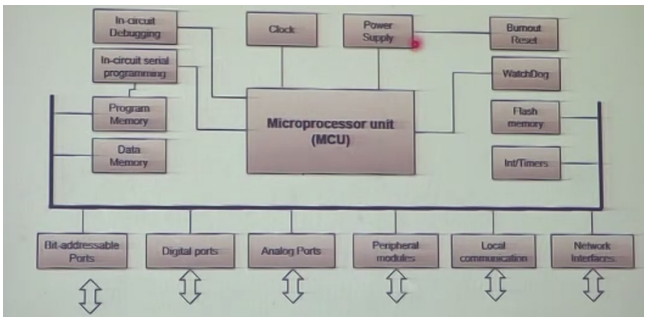
In programmers model

- Instruction Set – what instructions it does
- Memory Org
- Instruction timing
- ALU capabilities
  - Floating point calculations or not

<<<<< Need to be completed >>>>>>>

- Earlier hardware and software could function independently and not needed to give emphasis on the other
- But in microcontrollers need to integrate with both at the same time and need to collaborate
    - Which is why the intersection in of hardware and programmers model is a large and important section.

Components of Micro Controller

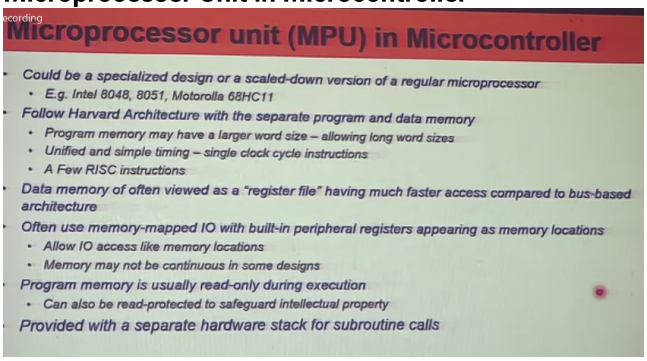


- Power Supply tends to cause some issues in the functioning in most cases
- Also due to resistance in wires (due to length) may be cause the required voltage to drop down
- Causing to behavior to be unexpected
    - Hence there Burnout supervisor and senses it and then it will reset
        - Which is better than freezing and causing erratic behavior
- The Watch Dog circuit
    - Microcontrollers work as event driven programs
    - Sometimes it may cause spend too much time in a single task and miss the other tasks
    - And sometimes, they are maybe software bugs in the loops (for example) which makes the device to keep running continuously due to faulty conditions
    - The watch Dog (another supervisory) handles that
- The In-Circuit programming
    - Connects with the MCU and the main bus connecting with the other devices/ports
- Digital ports
    - Handles in 8 bits at a time generally
    - Sometimes will have to send serially in a transaction
    - But most peripherals need only 1 or 2 bits to send data which is

wasteful
    - Hence we use Bit-Addressable ports
        - There will be convertor to make it into a byte addressable form
- Analog Ports
    - Need convertors that match the speed of the data received as well
    - Note for some outputs need analog and hence need digital to analog convertors
    - Also there Pulse Width Modulation
        - Which are digital but have features of analog
- Peripheral modules
    - Can handle some higher level of processing
    - Like recording time as well and monitors continuously as well
    - Capture and compare modules
    - Timer modules
- There are two types of communication modules
    - Local
        - Short range between devices
        - Like Inter IC communication
        - Eg: CAN bus
    - Network
        - To the internet
- Flash memory
    - Used to keep data for archival usage
    - Like a peripheral

- All these are built in the same chip – and hence known as System on Chip (SoC)

Microprocesser Unit in Microcontroller

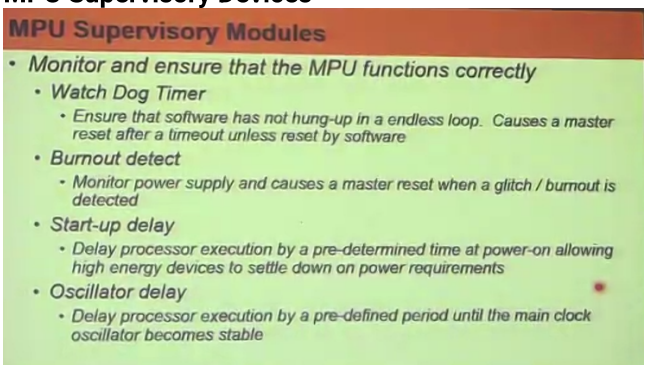


- Ones like Intel 8048, 8051 are specially designed as a microprocessor itself
- Most Microcontrollers use Harvard Archi

    - With the limited memory available for IoT it may be better for Harvard
    - We can have different sized busses as well for instruct and data separately as needed without waste
    - There may be fewer type of instructions executable – but highly optimized using RISC architecture
- Since there is a few data transfer locations, - a data is not needed and the data can be sent straight to ALU for processing
    - Makes it faster
- Using Memory mapped IO mostly
    - Where the IO device is given a memory address identifier
    - Then no need separate instructions to handle them and can consider the IO as an device itself
- Timing is important and need to be same

- Drawbacks
    - There may not be continuous memory locations
    - Program Memory is only read-only execution
        - For protection of IP in the software
        - But maybe difficult to debug
            - Can be handled by an override bit
    - There are separate stacks for the software and hardware
        - But there may be a limited number of subroutine calls allowed into the those stacks

MPU Supervisory Devices



- Most cases Power supply will not be steady when turned out
    - So a startup delay is given to the microprocessor so the env around (the other components)

will be steady and ready to function
- Oscillator Delay
  - When there is oscillator is not steady it will be reset

Forms of In-Circuit Programming
- ICSP – In Circuit Serial Programming
  - Writes object code directly to memory
- ICD – In Circuit Debugging
  - Microcontrollers are headless – no visual output – no input
  - The computer connected to it will just act as input and see intermediate outputs – while the execution occurs on SoC
  - Works with MPU to provide insight into program execution
  - ICD came to the forefront when it was necessary to check the microcontroller without using expensive special devices to reprogram

  - Here in these 2 parts we use an external computer to do debugging
    - Cross compilation
    - Developing the software on a different host computer and then download to the microcontroller to run it
  - Only text data can be transferred through the serial links

- The idea of in-circuit debugging is to debug while being on the system itself
- **In-circuit serial programming module** is connected to the Program Memory to allow to re-write the program

- <<<< Check the other types here >>>>
- OTA (Over the Air)
  - Was designed to minimize the overhead of having a serial port just for debugging in wireless devices
  - Allows firmware to uploaded/updated by Wi-Fi
  - Not done at hardware level but with a bootstrap software module
  - But limited on debugging features

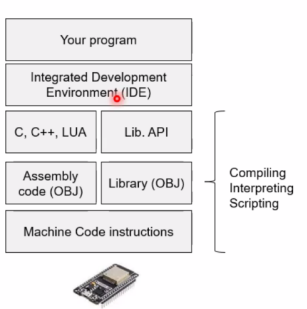
- Easy to deploy the updates and changes around the world
  - But easy to fall into cyber security risks due to hacking
  - Hence need extra effort to protect them

25th April 2023

- Understand the concept and logic and general implementation
  - Be able to figure out how to understand the schematic diagram given

Microcontroller Programming

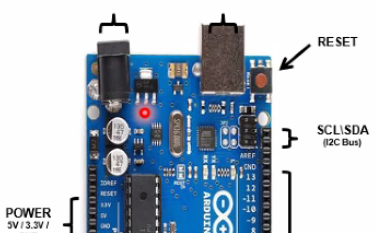
Microcontroller programming

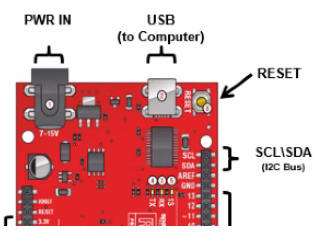


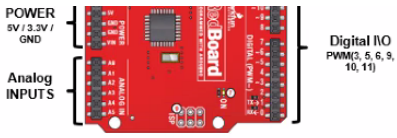
- *What would you need?*
  - *A microcontroller with a development board*
  - *A computer with an IDE installed*
  - *Program download cable/adapter*
  - *A power supply*
  - *Input / Output devices*

- The development board provides the connections to the computer for debugging and programming
- Microcontrollers can only follow machine code
  - Hence need a high level language to code with the IDE on the computer
    - Eg: MicroPython (developing), C, C#, Lua
- Even a simple task needs large number of steps to be done
  - So libraries can be used to do the more common trivial tasks
  - IDE's can handle and maintain them with Library Manager

Getting Started with Arduino: The UNO board







- I/O header board gives the connections to external devices
  - A – Analog
  - D – Digital pins
- Supports Arduino in circuit programmer which can be connected to the I/O header pins
- Arduino shields can be used to stack on top of each other for expansion

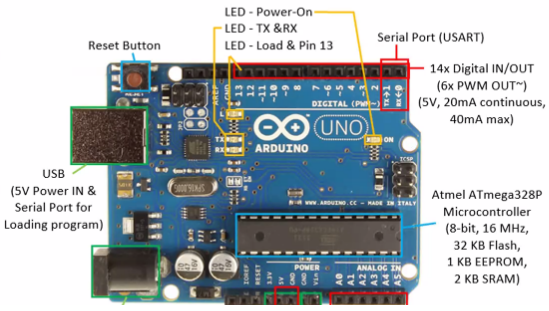
- Arduino IDEs have two buttons for Verifying (check the code only) and for uploading to the board
- The board and serial port should be checked if identified correctly or not

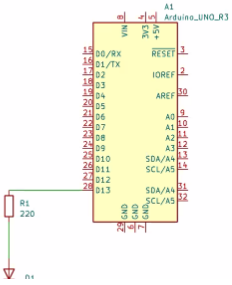
- Arduino uses a variation of C language with a different syntax

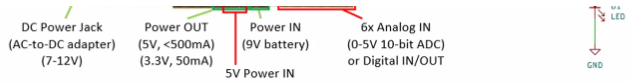
- The device must react to changes in the environment (or the controls) and remain unchanged and constant if no changes in the environment
  - Hence we need to loop that action (or standby)
  - Using the **loop()** function
  - This also should do conditional checking too (repeatedly checking the environment)
  - Also each task should not spend too much time (like reading the sensors)
    - Hence the loop must quickly implement and finish through
    - So it can return again to do env checking

- When the device is started for the 1st time, we need the **setup()** function to be run once and initialize the necessary components
  - This should be a very short process which can be done quickly

Blinking the in-built LED

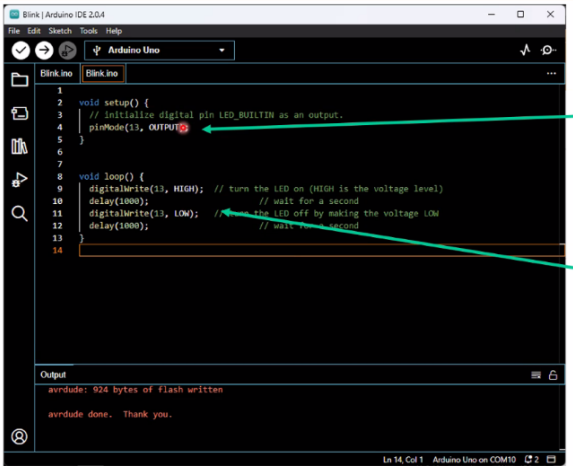






- TX&RX light used to indicate if connected to serial input/output
  - These two ports can also be used as a digital port even
- D13 is connected to an LED
  - If D13 is at logic 1 the LED will light up and vice-versa
  - A resistor is placed to protect from very high current flowing through the microcontroller and LED
- Note that before using any of the ports/pins they should be configured for Input/Output/Both
  - Should be done in the setup() function

Blinking the in-built LED



By default, all pins are set to input mode. So set pin# D13 to output mode at startup

Loop that executes continuously setting pin# D13 to high/low and waiting for 1000ms (1s) in between

Let's see a live demo!

The microcontroller advantage…

- *Suppose we want to do the same with Z80 microcontroller. Out LED will be connected to bit 0 of a port having address 0x378*

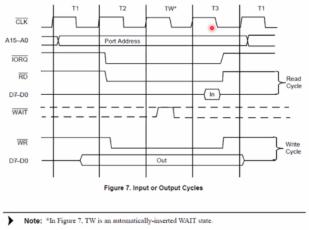
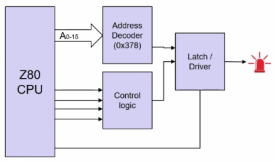


Figure 7. Input or Output Cycles

Note: *In Figure 7, TW is an automatically-inserted WAIT state.



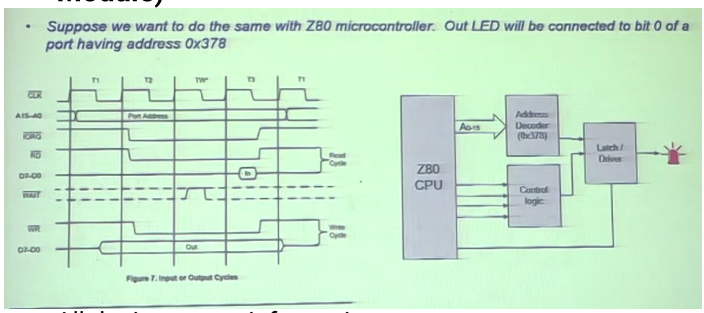
- Refer the Computer Architecture / Computer Organization knowledge here
- Latch (Flip-Flop) is used to capture the data passed out

L4 – II – Microcontroller Internals

Tuesday, May 02, 2023 8:24 AM

Recording for 2nd May Lec
https://dms.uom.lk/s/e8GJfibBefE2FAd

- How does a Z80 processor work **(Similar info from Computer Org. Module)**



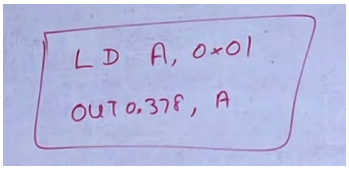
- All the important information goes through the bus
- All the signals are timed according to the Clock signal
- A15-A0 is the 16bit address bus and the port address is loaded into the bus to be sent to the address decoder
  - The cross of the address bus indicate that the change in the address bus
- This shares the I/O request and memory request on the same the bus
  - Hence it has a IORQ and MEMRQ signals are present to indicate if IO or Mem Address is sent or not respectively
  - IORQ is active low signal and hence the when signal is low only the voltage is high
  - Similar to MEMRQ
- RD indicates if the processor is reading data or not (active low)
  - A WR signal is also present
  - Both will never be active at the same time. But both can be inactive at the same time
- The D7-0 data bus will always contain some value. But can be invalid, Then the timing diagram it shows as a flat line

- We have to deal with address, data bus and the data to be written
- When required to turn on the LED connected to bit 0 of a port having address 0x0378

    - Port Address of LED is sent to address bus
    - If we want to switch the LED, we send 1 to the D0 bit line and all the other bit lines in the Data bus can be anything
        - Technically any odd binary number will turn on the LED
    - Then the IORQ and WR are sent to active low state
        - Then the relevant IO device with address knows it will have data rewritten
    - The Latch will store that value which should be sent to the LED
- For reading a value, the data bus is updated near the end of the active read signal

- There is a wait signal to indicate the processor to slow down and wait
    - It is present to allow the IO devices to catch up to the processor

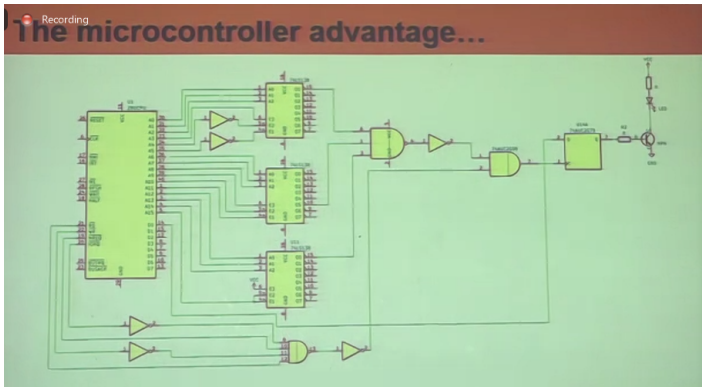
- **No need to memorize the timing diagram**

- Address Decoder will detect the relevant address of the IO device and output signal 1 to the device
- The Control Logic
    - Indicates to the device if IO process or Memory Access process is taking place
    - And also if it is reading or writing
    - Therefore the control signals (all are active low) the device should pay attention to are
        - MEMRQ – will 1 if using data from memory
        - IORQ
        - WR
        - RD – will 1 if reading data to device/memory
    - Z80 instructions for the LED turning on
        - 

- We have to build the address decoder and control logic decoder
- Using Sum of Products expressions and logic circuits

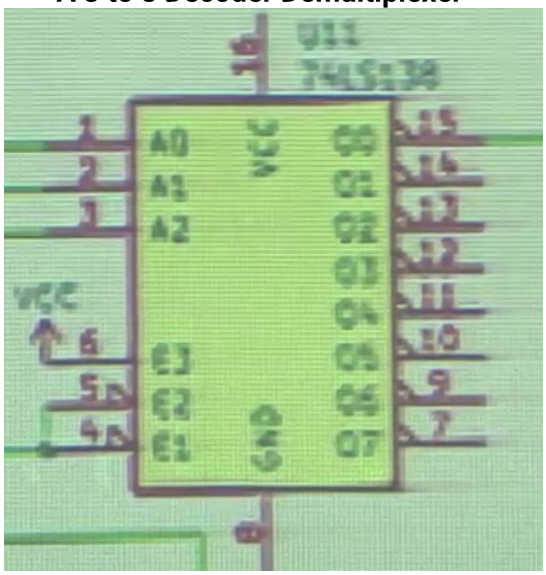
- Practically only 4 signals can be put to a single logic gate
- And we would need a a lot of IC's to handle them

Microcontroller Advantage...



74138 Decoder

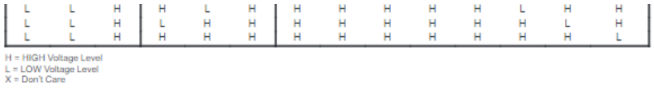
A 3 to 8 Decoder Demultiplexer



- Besides the signals it should provide the power through the V_{cc} and GND terminals are present
- Y and Z terminals are used for outputs
- A, B, E, G(Gates), Q, X are used for inputs
- There are 3 inputs A0, A1, A2
- There are 8 outputs O0- O7
- There are 3 other inputs E1, E2, (which are active LOW) and E3 (which is active HIGH) (also referred as G or gates)

TRUTH TABLE

INPUTS						OUTPUTS							
E_1	E_2	E_3	A_0	A_1	A_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H



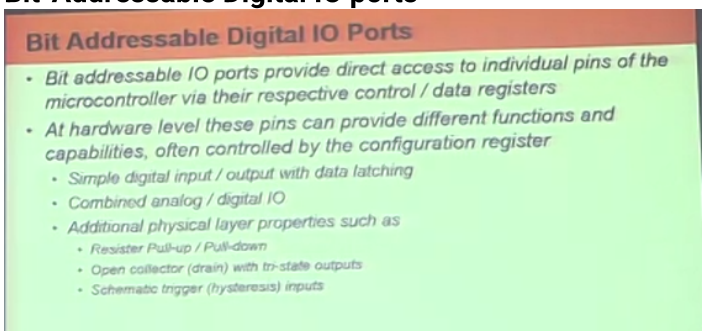
H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

https://pdf1.alldatasheet.com/datasheet-pdf/view/12624/ONSEMI/74138.html

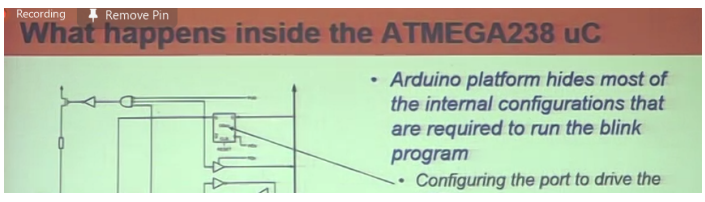
- If the Enablers are disabled where
  - the E active low inputs (E1 and E2) are HIGH
  - Or E active HIGH inputs (E3) are LOW
    - Then all outputs are HIGH irrespective of the other values in E's or A, B, C
- When Enablers are enabled then the Low signal is sent to output with the binary form of the 3 inputs ABC while other outputs are High
  - Refer demultiplexers and decoders in 2nd Sem

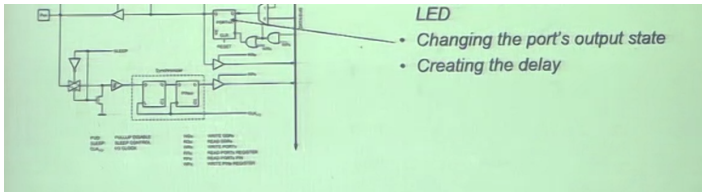
Assignment find the mistake in the above circuit diagram of the microcontroller

Bit-Addressable Digital IO ports



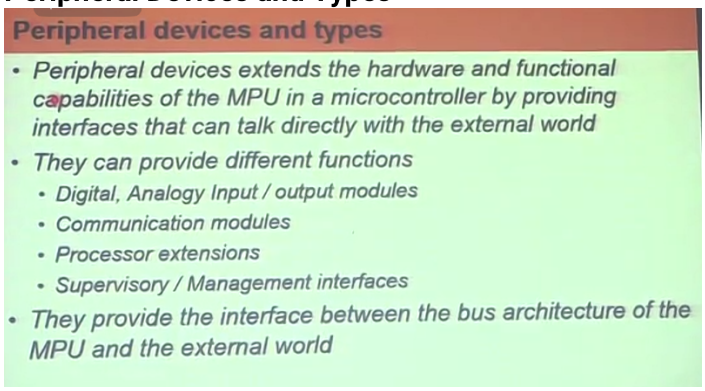
- Manufacturers have issues with handling too many ports with space
  - To handle it has multi-use ports
  - It may make the logic more complicated as well
  - But makes for a simpler connections
- Digital ports encapsulate many features within them
- But not all ports will not be needed in all cases
  - Hence these features have to be configurable and allow to change with some code depending on your uses





- When we give a command to make the port PIN as an output, there will be multiple changes and configuring to make it an output port
  - There is a output pin on the LHS itself
  - There is a pull up resistor between the pin and transistor
    - It ensures the logic is pulled up to 1 when there is nothing connected to the pin and to avoid any ambiguous values
  - The Latch will handle all the computation in here
    - And stores the output for the output device to read at a later stage when possible

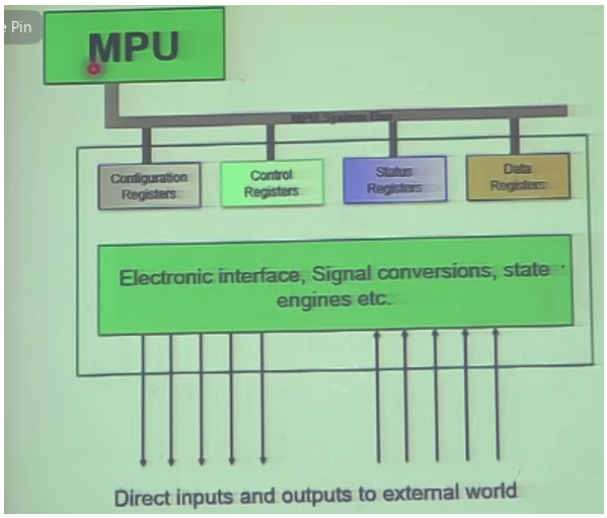
Peripheral Devices and Types



- Peripheral devices implement the functionalities of the microprocessor
- Some handle Analog to Digital convertors with built in components, and vice versa
  - Sometimes the Analog signals are represented a Pulse Modulated Circuits as well to stimulate the analog signal (even though its technically a digital)
- Inter-IC communication (I^2C)
- Processor extensions provide as additional features to the processor like memory
  - Then the ports will be needed to act as IO ports then
- Supervisory interfaces check for things like power surges
- These in-built interfaces allow easy implementation without the need to

build them separately

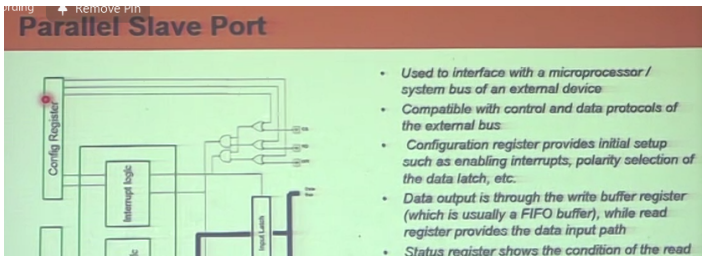
Peripheral Interfaces: General Structure



- Configuration registers
  - Can configure according to our use
  - Eg: the pull up resistor can be de/activate to change
  - The peripheral interface functionality can be changed and configured
- Control registers
  - To do specific tasks
  - Eg:
    - To start the transmission in the serial port to send data
    - Restart signal when required
- Status Registers
  - Read the operational status of the devices and interfaces
    - Like indicate if the data has been read or received or not
- Data Registers
  - Used to send transfer data out and in between the MPU and peripheral

- All type of registers may not be needed
  - Like Status Registers are not required if the module is only working with Outputs only with no data reading

Parallel Slave Port



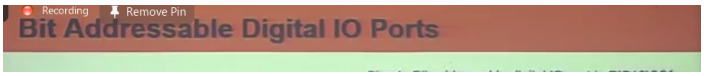
- An example of a Peripheral Device
- Able to send multiple data at the same time
  - The larger green box is the parallel port
- Configure register
  - Used to tell if Input/Output
  - Or configure whether some of them to be input and rest to be output

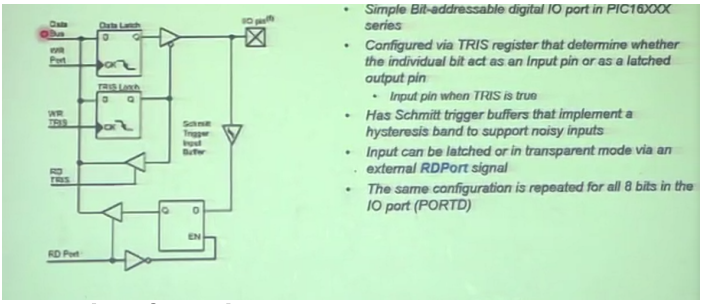
<<<<<9th of May >>>>>

- The Data Register will store the data that is read or to be written

- Depending on whether Input or Output the values read by the Latch will differ

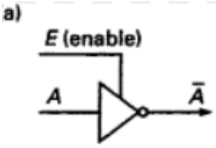
- At the input side there is
  - Input Latch
    - Store values coming in until to be read by the microprocessor
    - It's possible to have a buffer before the Input latch as well
    - In this diagram if there is only a input latch, it should be able to read them quickly and frequently
  - And an output latch
    - A FIFO buffer is placed as an intermediate to help the difference in the speeds of transmission and processing
      - The 1st value is stored into the output latch
      - The successive values will then put into the FIFO buffer
      - If no buffer, then it will rewrite the values
        - Then we have to make sure the values are read/delivered before being delivered
        - We check this from a status register that indicates if the value reached the output
    - The risk of FIFO buffer is when there is a buffer limit





Port taken from the PIC Microcontroller

- TRIS Latch – not required to remember
- If input mode
  - The Data Latch path cannot be used
  - It uses the Lower Latch path passing though the Schmitt Trigger Input Buffer
  - In that case the Data Latch needs to avoid pushing in a value
    - For that there is a tristate buffer blocking the output in the data bus by going to the open state
    - And hence no impact onto the input being read
- In the output mode
  - The tristate buffer goes to enabled and will drive the IO Pin from the value sent from the Data Latch
- Whether it acts as I or O port will be decided by the TRIS latch by sending TRIS output to the tristate buffer



A	E (enable)	Output
0	0	High Z
1	0	High Z
0	1	1
1	1	0

Tri state buffer truth table
The Enable pin should be 1 for normal functionality

Question

- If we setup the port to be output, and we write a value of High to the port
  - And then we read the value of this port
    - Note that in this diagram there isn't any issue when reading even though it is in output mode
    - In this config it will read from the value that was just sent ot the output port

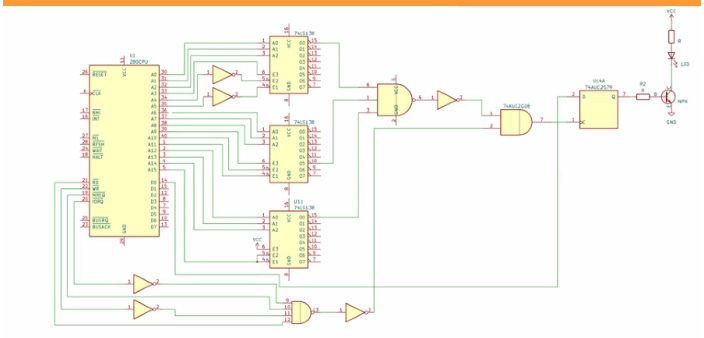
- **Note In different controllers the logic will work differently**

<<<<<</9th of May>>>>>>>

L4 – III – Microcontroller Internals

Tuesday, May 09, 2023 8:27 AM

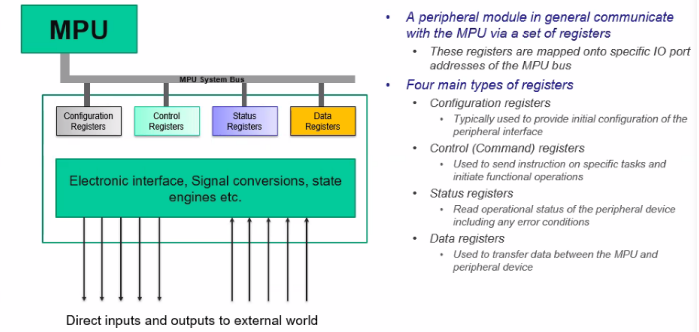
The microcontroller advantage…



- The objective is to design with minimum number of components
  - to reduce cost when building for production
  - To avoid a new point of failure in the device

- When adding a component, we can implement it using a software component or through hardware
  - Adding a new hardware component gives a higher cost
  - Adding a software will demand more computation power and make the program more complex
  - There is no hard-rule to indicate which route to go and one must analyze and justify why
    - Should know the limitations and benefits of the approaches considered

Pheriperal interfaces: The general structure



- *A peripheral module in general communicate with the MPU via a set of registers*
  - *These registers are mapped onto specific IO port addresses of the MPU bus*
- *Four main types of registers*
  - *Configuration registers*
    - *Typically used to provide initial configuration of the peripheral interface*
  - *Control (Command) registers*
    - *Used to send instruction on specific tasks and initiate functional operations*
  - *Status registers*
    - *Read operational status of the peripheral device including any error conditions*
  - *Data registers*
    - *Used to transfer data between the MPU and peripheral device*

- Designers build ports and interfaces to be more generic so it can be used in multiple ways
  - The ports will be designed to handle a voltage level

depending on the standard it works

- The output port in the microcontroller will have a limit on the current allowed through it
- Physical parameters such as
  - Current it can consume (sink)
  - Current it can provide
    - Are needed to be considered
- Pins will be needed to be configure as input or output
  - Buffering also may be needed to store the value using a latching port to allow enough time
- Status Registers check the status of ports and whether the flags of the port have changed over time

- So a Microcontroller should have those 4 components in the picture above with the electronic circuits
- The configuration of this done by writing values to those registers

- Understand the interpretation of the schematic diagram

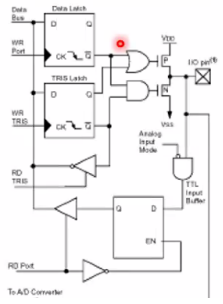
<<<Refer the Part 2 lecture note for info about Parallel Slave Port>>>

Bit Addressable Digital IO Ports

- *Bit addressable IO ports provide direct access to individual pins of the microcontroller via their respective control / data registers*
- *At hardware level these pins can provide different functions and capabilities, often controlled by the configuration register*
  - *Simple digital input / output with data latching*
  - *Combined analog / digital IO*
  - *Additional physical layer properties such as*
    - *Resister Pull-up / Pull-down*
    - *Open collector (drain) with tri-state outputs*
    - *Schematic trigger (hysteresis) inputs*

<<<Refer the Microcontroller part 2 lecture note for info about Bit Addressable Digital IO Ports>>>

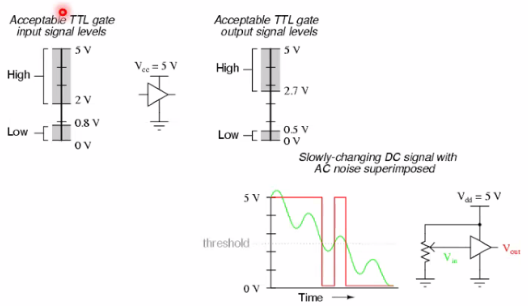
Bit Addressable Digital IO Ports



- *Digital bit addressable port with high-current capability and analog input support (PORTA in PIC16XXX)*
- *Configured via two registers*
  - *TRIS for direction control*
  - *Analog Input Mode – enable analog input*
- *Two MOSFET at the output stage allow high current driving and sinking*
- *Selecting Analog input mode disable digital reading*
  - *Pin must be in input mode to support analog input mode*
  - *Separate AtoD converter (not shown in the schematic) is required*

- We may have to take a decision when can't find ports to handle the logic voltage and currents we need

Digital IO – Practical voltage levels

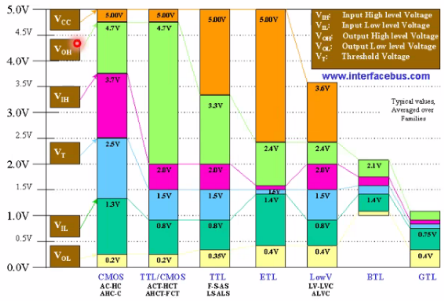


- Practically there will tolerances in the values of voltages sent due to resistances
- Hence it will give a range of values accepted
- For the Input gate
  - 2V to 5V – High
  - 0V to 0.8V - Low
    - 2V is the V_{IH} - minimum input to be high
    - 0.8V is the V_{IL} – maximum input to be low
- For the Output gate
  - 2.7 to 5V - High
  - 0V to 0.5V - Low
    - 2.7V is the V_{OH} - minimum output to be high
    - 0.5V is the V_{OL} – maximum output to be low
- From the range of values, we can see that the output ranges are within inside the larger input range values
  - Hence the output of gate can be connected to the input gate of another

- When the voltage value in the prohibited band is set to the input the output of the gate will be producing an output always but it will be unpredictable

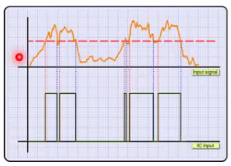
- The prohibition band is there to avoid noise changing the voltage values – and is a method to solve the problem of non-exact voltage values

Logic Levels – Different families

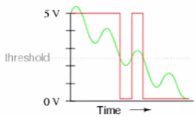


- Different technologies have different logic levels
- The reason the 5V in TTL logic gates are not popular is due to the high heat dissipation of the processor due to high current
- Hence we can use a lower voltage as the high logic level
  - Like 1 V
  - But then the guard bands in between will be less then
- Then we may have to convert/regulate the voltages between the different logic voltages
  - Or maybe a voltage buffer
  - Or a voltage matcher

Schmitt Trigger Inputs



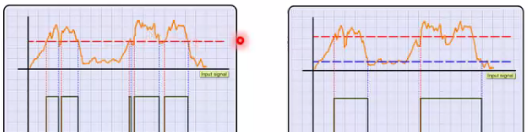
- *Use of single threshold is problematic*
  - *Threshold may not be consistent*
  - *Signal may cross threshold due to noise*
- ***Solution**: Use different thresholds for Low -> High and High - > Low transitions*



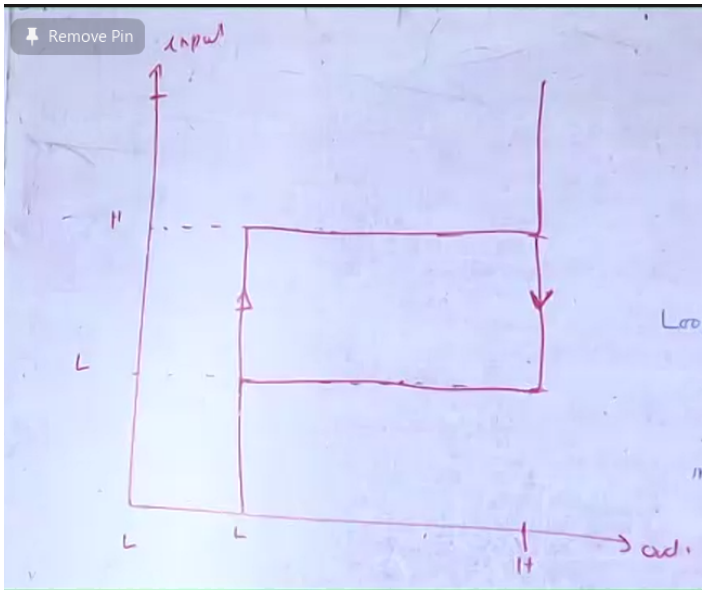
- Explains the problems from using a single threshold values to divide logic high and logic low

Schmitt Trigger Inputs

- *Use of different thresholds avoid the problem of prohibited range values*



- This issue is very apparent in cases where the voltage values are changing very slowly
- So the solution is having a double threshold
  - Then sort of prohibited band is created there in between
- When value > higher threshold
  - Logic 1
- When value < lower threshold
  - Logic 0
- When in between
  - it will give the previous logic level
  - In the sense it will wait until it hits the other threshold and then only change the logic level appropriately



- These arrows indicate how the input voltage changes (either from L to H or H to L)

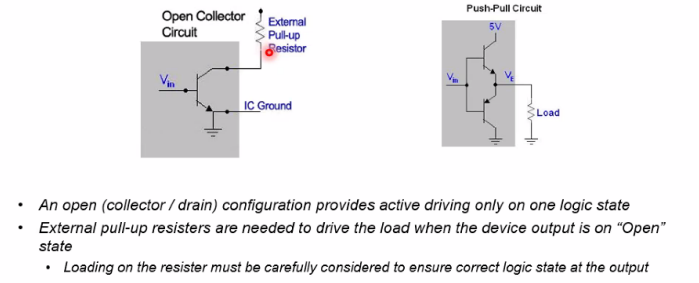
Gates with Schmitt Trigger Inputs

- Schmitt Inverting acts as NOT gate in a way

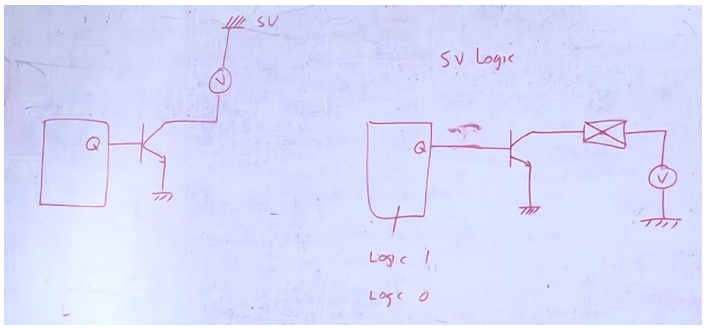
- The Schmitt Gate may impact on the value read from the I/O pin at

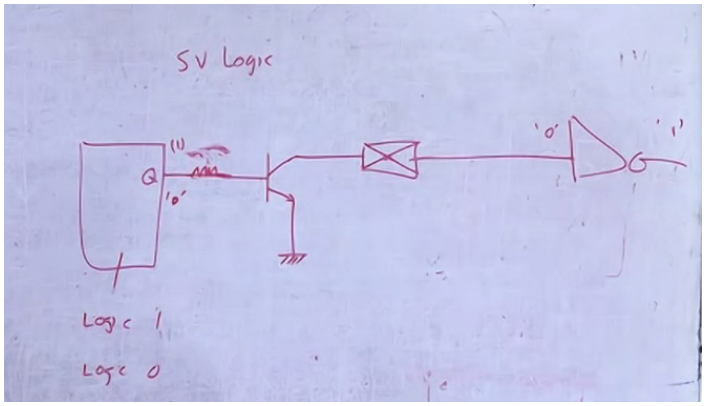
Output mode (refer the Parallel Slave Port and Bit Addressable Port)

Digital IO – Push-Pull Vs Open outputs



- *An open (collector / drain) configuration provides active driving only on one logic state*
- *External pull-up resisters are needed to drive the load when the device output is on "Open" state*
  - *Loading on the resister must be carefully considered to ensure correct logic state at the output*





- The values become undefined at logic level 0
- To prevent this we add a voltage source of Vcc of 5V with a resistance

---

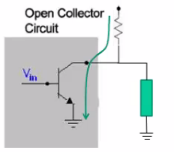
To visualize the circuits discussed in class

Circuits-Transisto...

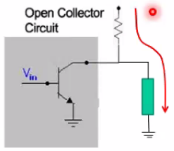
Import this file to https://www.falstad.com/circuit/

(Circuit diagrams maybe incorrect – refer the video recording too https://dms.uom.lk/s/NJnAxM9PTERxtZg)

Digital IO – Push-Pull Vs Open outputs



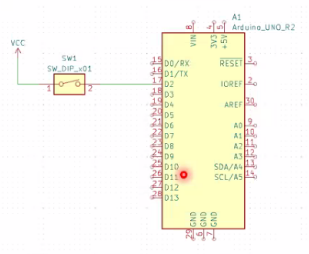
When the output is at Logic –Low state the load voltage is clamped at Vce of the output transistor. The external pull-up resister act as a current limiter

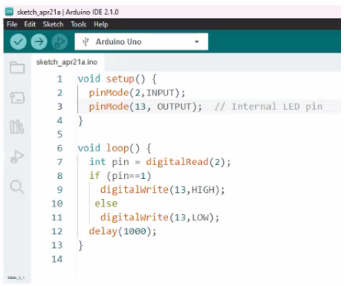


When the output is at Logic–High state current flows through the load. Output voltage is determined by potential division between the load and the external pull-up resister. Care must be taken to maintain output voltage at logic high level.

- To visual the operation above operation check from this link https://tinyurl.com/2e9zwkj2

Reading one bit input



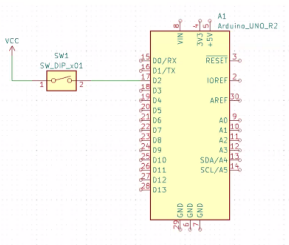


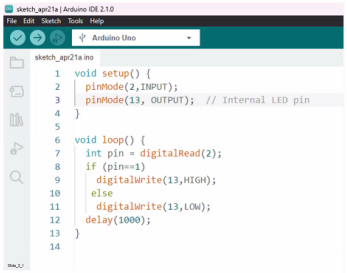
- *How would you expect the output to be?*
- *Does it behaves as expected?*
- It will be surprise
-

L4 – IV – Microcontroller Internals

Tuesday, May 16, 2023 8:26 AM

Reading one bit input





- *How would you expect the output to be?*
- *Does it behaves as expected?*

- In the setup function
  - Pin 2 is input
  - Pin 13 is the output

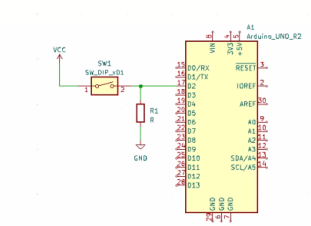
- In the loop
  - Reads pin 2 into "pin"
  - If the value is 1 then outputs High to pin 13
    - And vice versa
  - And waits 1000 seconds

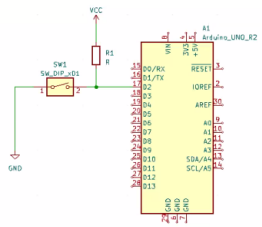
- The switch used here is push switch
  - Not a bi-stable where it stays in on or off state
  - The push switch is a mono-stable and stays ON as long as pressed

- The expectation is for when pressed the light is on

- But the light remains on
  - It means the pin 2 is reading HIGH
- There is the high input impedance between input and ground
  - Then if there is some current then high voltage may still go through
- Therefore we should ensure the input never floats
  - And there should be a path for the voltage to pass out

Reading one bit input: Using pull-up/Pull-Down



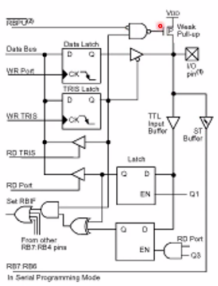


- There will be a resistor to pass the current down to the ground
  - We may think batter to have a lower resistor to make the induced current pass easily
  - The current passing through the R1 depends on the value
    - But note the voltage between R1 maybe fixed due to the high impedance of the microcontroller
  - So R1 should be large enough to ensure there is no large current drawn out from the power source

- 1K to 10k ohms is the better range to select

- Why it isn't R1 inbuilt to the microcontroller inputs?
  - Depending on the circuit it may not be needed
    - This R along with some Capacitance acts as a low pass filter in a way
  - So manufacturers make the R1 configurable to your use

Bit Addressable Digital IO Ports: Configuration

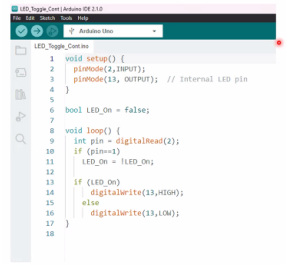


- *Digital IO port with open collector (drain) outputs, programmable pull up resisters and interrupt capability (PORTB in PIC16XXX)*
- *Configured via*
  - *TRIS register*
    - *Set pin direction (input , output)*
  - *PULLUP*
    - *Enable / Disable internal weak pull-up resisters*
  - *INTCONF*
    - *Enable interrupt on pin change*
- *Open collector (drain outputs) with low current sink capabilities*
- *Also support schimmitt trigger inputs to other peripheral devices when not used for digital IO*

  - The pull up resistor ensures that the resistance is active only for input mode and not output mode
  - But not all pins will have that capability
  - So we have to select a pin with that pull resistor for inputs
  - Note that
    - It takes up storage space
    - Makes another point of failure
      - And the mean time to failure reduces (the reciprocal addition)
      - Hence then more devices need to be made
    - Therefore important to reduce external device count

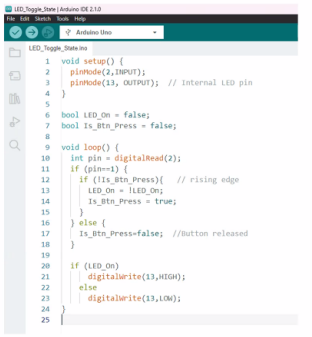
- There is also some impedance from the connections of the computer connected to the board (I'm not sure why and what sir meant there)

A Toggle button?



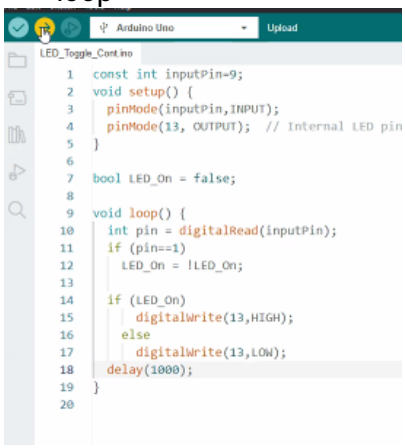
```
void setup() {
  pinMode(2,INPUT);
  pinMode(13, OUTPUT);  // Internal LED pin
}

bool LED_On = false;

void loop() {
  int pin = digitalRead(2);
  if (pin==1)
    LED_On = !LED_On;

  if (LED_On)
      digitalWrite(13,HIGH);
    else
      digitalWrite(13,LOW);
}
```

What's wrong here...?



```
void setup() {
  pinMode(2,INPUT);
  pinMode(13, OUTPUT);  // Internal LED pin
}

bool LED_On = false;
bool Is_Btn_Press = false;

void loop() {
  int pin = digitalRead(2);
  if (pin==1) {
    if (!Is_Btn_Press){   // rising edge
      LED_On = !LED_On;
      Is_Btn_Press = true;
    }
  } else {
    Is_Btn_Press=false;  //Button released
  }

  if (LED_On)
      digitalWrite(13,HIGH);
    else
      digitalWrite(13,LOW);
}
```

- Why doesn't the first code work?
    - The light sometimes respond to the push switch
    - And when press hold the light shines very low
  - One thing is there is no delay in the loop



```
const int inputPin=9;
void setup() {
  pinMode(inputPin,INPUT);
  pinMode(13, OUTPUT);  // Internal LED pin
}

bool LED_On = false;

void loop() {
  int pin = digitalRead(inputPin);
  if (pin==1)
    LED_On = !LED_On;

  if (LED_On)
      digitalWrite(13,HIGH);
    else
      digitalWrite(13,LOW);
  delay(1000);
}
```

- Try adding the delay
- Without the delay the loop runs very fast
  - Even a during small push of 50 ms
    - The light will be already switching between on or off continuously and very fast
  - And this makes the result of the circuit unpredictable
- So this is a physical issue (rather than a logical issue)

- But adding a delay is not practical as it slows the whole program
  - Nothing happens during the whole wait

- Behavioral modelling
  - Define the behavior using variables

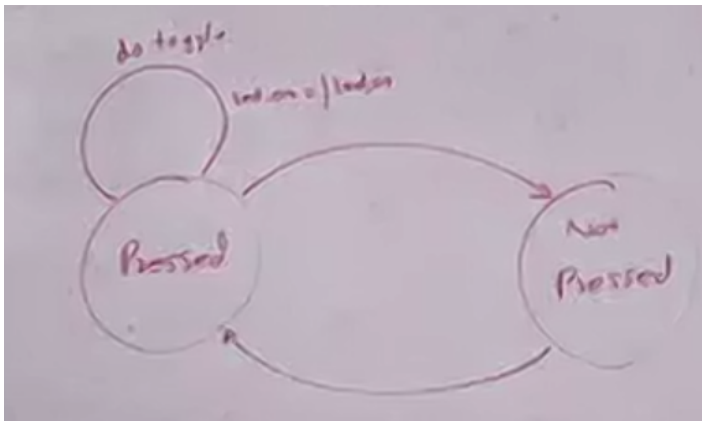
- And drive the behavior using those variables
- The logic may need to be handled from multiple locations
- So it's a good practice to have the logic variables to define the behavior

Trying out the circuit 2

LED_Toggle_State.ino

```
void setup() {
  pinMode(2,INPUT);
  pinMode(13, OUTPUT);  // Internal LED pin
}

bool LED_On = false;
bool Is_Btn_Press = false;

void loop() {
  int pin = digitalRead(2);
  if (pin==1) {
    if (!Is_Btn_Press){   // rising edge
      LED_On = !LED_On;
      Is_Btn_Press = true;
    }
  } else {
    Is_Btn_Press=false;  //Button released
  }

  if (LED_On)
     digitalWrite(13,HIGH);
   else
     digitalWrite(13,LOW);
}
```

- Moved from a dynamic logic to state-based logic programming in this case
- Earlier when pressed the toggling happens
- Now we wish to make the toggling takes place when moving between the two states
-



- This ensures the repeated changes for too long presses are handled
- In basic sense is to do the toggling in the rising edge of the button itself

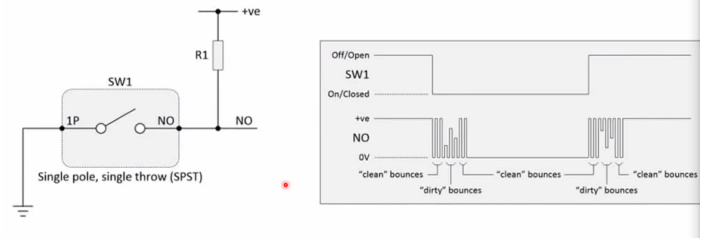
- The Is_Btn_Press checks if the variable is pressed or not
- If the input is 1 (pressed)
  - But the variable btn is false
    - Then do the toggling
    - And change the btn to true

    - Then the toggling code line will not be executed again
- When the button is released (physically) and input pin is false
    - Then we make the btn false

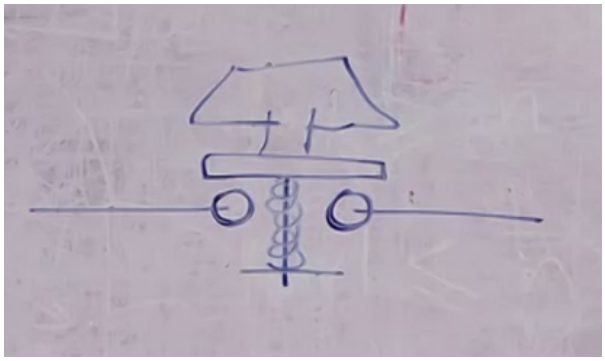
- BUT still there is some error at times and light works incorrectly

- This is a HARDWARE issue
    - Called **Bouncing**

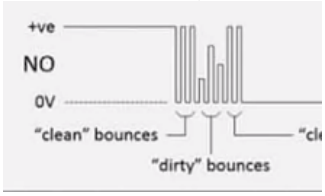
Switch bouncing and Debouncing….



- Single Pole, Single Throw (SPST)
    - Pole indicates no of the inputs
    - Throw indicates no of outputs
- This switch is a mechanical switch



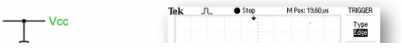
- Due to the spring there can be vibrations when pressing
- So there will be some bouncing between the high/low levels

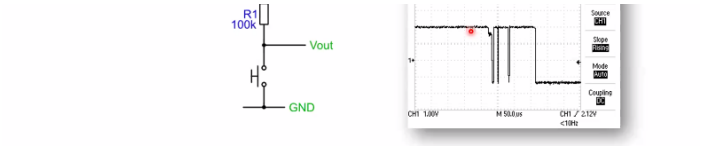


- It doesn't straight away go to the low from high

Bouncing and De-bouncing ccts

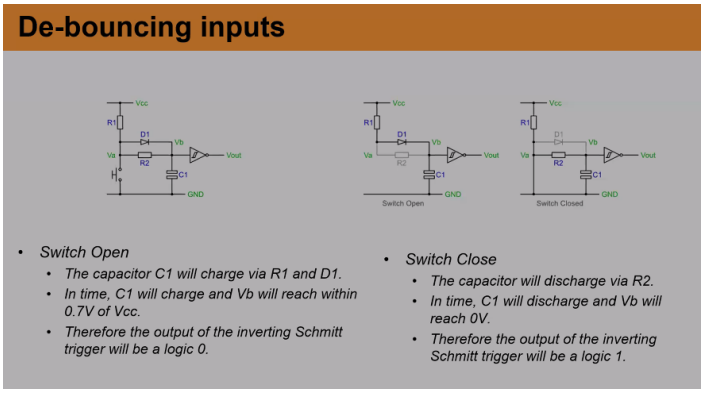
- *Pressing or releasing a mechanical switch in general does not produce a clean edge in the output waveform*
    - *Digital input may interpret this as multiple edges*
    - *Referred to a "Bouncing" on the input*





- Ways to handle bouncing
  - Slow the microcontroller by adding a delay
    - Not a good practice in all cases
    - Maybe 10ms a small delay be enough
      - For fairly good switches it may be enough to have 1ms
    - Depending the device it may be difficult

- Other ways to handle the bouncing is using debouncing circuits



- Schmitt trigger ensures predictable value outputs in the forbidden band of inputs

  When switch open
  - D1 is forward biased
  - C1 is charged
    - And the voltage across the capacitor doesn't increase suddenly
      - Because it depends on the rate of current and impossible to have infinite current
  - REMEMBER we require to have the voltage change as slow and smooth as possible
  - Then the Schmitt gives the correct output for the threshold

  When closed (pressed)
  - D1 is reversed
  - Capacitor will begin discharged

    - Current will discharged through R2 slowly
- The Schmitt trigger makes the output zero once it reaches the lower threshold

But need a large number (and fairly large in size) components to implement with hardware

But through software we can treat those multiple transitions as a single transition

- **Software debugging**

Software de-bouncing

COUNTER METHOD

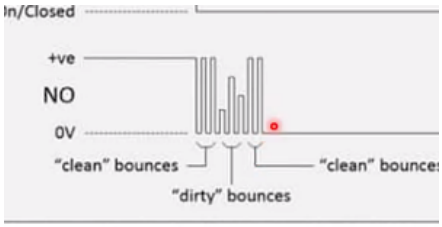
- *Setup a counter variable, initialize to zero.*
- *Setup a regular sampling event, perhaps using a timer. Use a period of about 1ms.*
- *On a sample event:*
- *if switch signal is high then*
- *Reset the counter variable to zero*
- *Set internal switch state to released*
- *else*
- *Increment the counter variable to a maximum of 10*
- *end if*
- *if counter=10 then*
- *Set internal switch state to pressed*
- *end if*

SHIFT REGISTER METHOD

- *Setup a variable to act as a shift register, initialise it to xFF.*
- *Setup a regular sampling event, perhaps using a timer. Use a period of about 1ms.*
- *On a sample event:*
- *Shift the variable towards the most significant bit*
- *Set the least significant bit to the current switch value*
- *if shift register val=0 then*
- *Set internal switch state to pressed*
- *else*
- *Set internal switch state to released*
- *end if*

Counter Method

- We make an internal counter
- Assume the circuit takes 10ms to run all instructions in the loop(). (input is read every 10ms)
- When input is read to be high
    - count =0 (resetting)
- When input is low
    - Count +=1

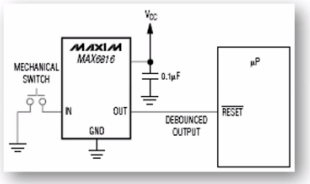


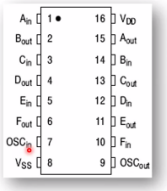
- So when the voltage is bouncing
    - If the counter is 0
        - Consider not pressed and stably at High
    - If between (0 to 10)
        - Consider as not stable
    - If above 10
        - then stably at LOW
    - Note: (10 is arbitrary and depends how long the bouncing happens)

- The advantage compared to the delay

- Is it doesn't block the whole circuit (delay is blocking function)
- And we can do some other task in the program still
- And allows to respond to critical inputs at the same time

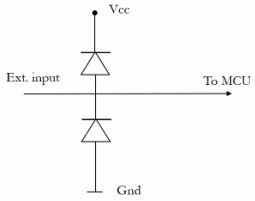
Specialized de-bouncer ICs





- Some IC's have debouncing handled within
  - Here we can add a single IC instead of many components
  - But costly

Protecting inputs



- *Clamping prevent the MCU input from over voltages / reverse polarities*

- It is possible to receive a voltage not in the range of the operating range of microcontroller
- So need to protect from high voltages and low voltages (beyond the ranges)
- Assume MCU handles in the range is 0 to 5V
  - If we receive 4.75V
    - The top diode is reverse (the Vcc would be 5 > 4.75)
    - The bottom diode is reversed
  - If we receive 10V
    - The top diode is forward biased (10 > 5)
      - Then the circuit is short circuit
    - Which prevents high voltage to the MCU
  - If we receive −5V
    - The bottom diode is forward biased

        - Then the circuit is protected from reverse or low voltages

- But note
  - The diodes have a max current
  - So if the diode works for a long time with extreme voltages then diodes will damaged
    - Then the extreme voltage will go to the MCU
- To fix this we add a fuse (fusable resistor) to the input
  - So the fuse will melt and disconnect the circuit
  - Likely to occur when lightning strikes

L5 – I – Peripheral Modules

Tuesday, May 30, 2023 8:25 AM

- Microprocessors need peripherals separately
  - While the Microcontroller devices have them integrated within them
    - This leads to have less cases for points of failure
- Microprocessors are designed to work faster with large data
  - Microcontroller is a scaled down version of above
- Hence even if it has the computational power it may not be able to respond on time
- When trying to do in all work in software – small routine work needs to be done and less is delegated to the important work

- Peripheral devices allow to do those processes without using the CPU's processing power
- Eg: GPU can do the image processing without the main processing unit doing it

- Peripherals Implement some of the peripheral tasks relieving the main unit
- Eg: GPU can do the image processing without the main processing unit doing it

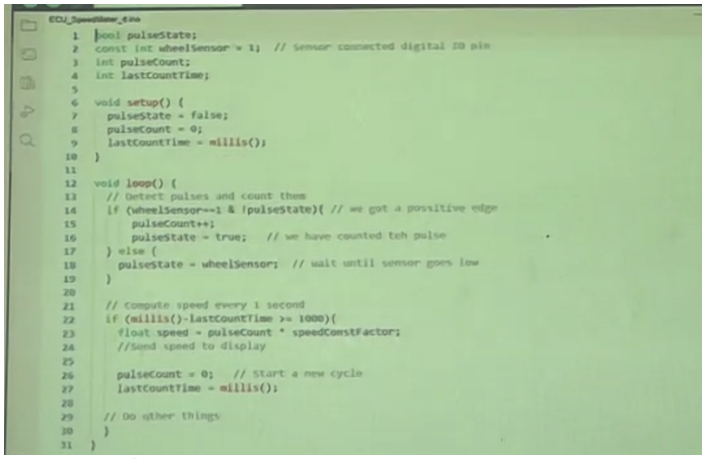
- Peripherals Implement some of the peripheral tasks relieving the main unit

Example

- ABS system looks to change the frictional force on the each wheel such that it doesn't end as an accident
- It looks to ensure the wheel doesn't get lock when it is skidding or in low frictional cases
- The vehicle needs to measure the rotational speed of each wheel
  - Measured using a ABS speed sensor
- Uses a Hall effect sensor

    - Then the gear teeth with a rotating sensor passes the ABS sensor will get a short pulse signal

- Modern vehicles uses the steering angle and the rotational speeds of the wheels to compute whether some wheels are rotating first
    - Helps also to determine if the tire pressure too low in some wheels
    - Also helps for the engine to determine the power needed
- Hence the car has a computer which may control some of the functions of the car ensuring the safety of the driver
- Hence measurements like these are important for different functions

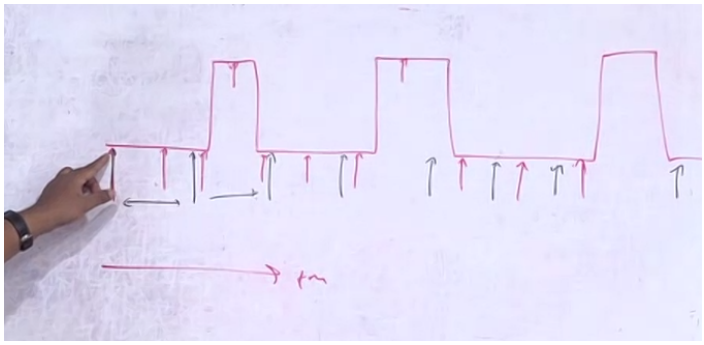


```
bool pulseState;
const int wheelSensor = 1;  // Sensor connected digital IO pin
int pulseCount;
int lastCountTime;

void setup() {
  pulseState = false;
  pulseCount = 0;
  lastCountTime = millis();
}

void loop() {
  // Detect pulses and count them
  if (wheelSensor==1 && !pulseState){ // we got a possitive edge
      pulseCount++;
      pulseState = true;   // we have counted teh pulse
  } else {
    pulseState = wheelSensor;  // wait until sensor goes low
  }

  // Compute speed every 1 second
  if (millis()-lastCountTime >= 1000){
    float speed = pulseCount * speedConstFactor;
    //Send speed to display

    pulseCount = 0;   // Start a new cycle
    lastCountTime = millis();

  // Do other things
  }
}
```

- Millis() is an Arduino library that returns the time since last power cycle
    - Like an internal clock for the Arduino
- Timing accuracy is not guaranteed though
- The IC also needs to
    - Manage the display
    - Handle debouncing
    - Do multiple other tasks

- In the loop()
    - It checks for the positive edge of the signal coming from the ABS sensor
    - Checks if the wheel sensor is received and if the previous state of the pulse was false
        - Then it detects a new positive edge

- In this example, it keeps sampling the ABS sensor signal
- In this code, there is no digitalRead() to check the input
    - Big mistake – can't believe I missed that

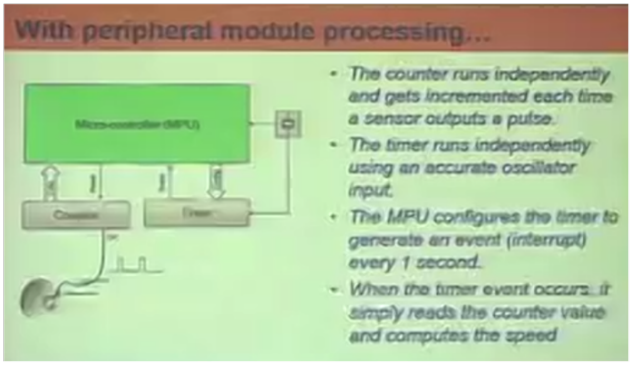


- But due to work overload of the processor
    - And the asynchronous rate between the reading and when the signal is happening
    - It is possible to miss out the pulse signal

By Nyquist theorem

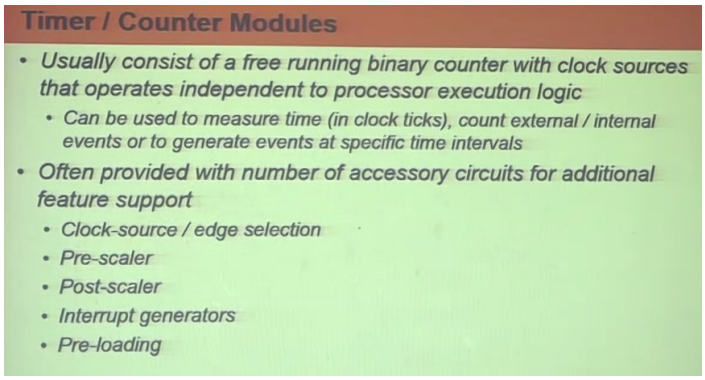
- We require to have at minimum twice the frequency for the sampling process

- But going at that speed will be difficult for the processor
    - Hence can delegate this to a peripheral device



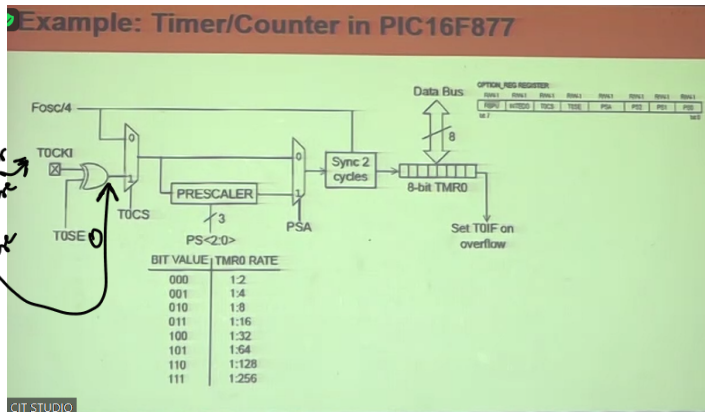
- It uses a separate counter which is connected to the MPU
    - The counter will do the counting
        - Which the MPU will communicate and obtain the information
- We could use millis() function – but since it is a software it can have issues

- Hence we have a timer() function which has a fixed frequency (eg: 1MHz)
    - And informs when it has passed 1ms of time and sends an "event" to the MPU informing that
- Hence no need to have the MPU synchronize with all the physical occurrences
- So these tasks are delegated to the peripherals



- Used for both timer/counter purposes
- Typically have 2/3 in built

- Works parallel as a separate unit independent of the MPU
    - Sends a timer event when required
        - It is configurable depending on timing required



- Used in the PIC controller

- The 8-bit TMR0 is the counter (aka Timer 0)
- Data bus can read from it and even over write it
- The T0IF is the overflow flag to indicate it is completes the cyle
    - Indicates didn't read the input within that cycle

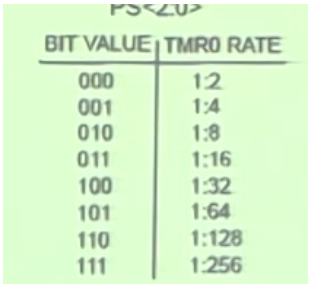
- F0sc/4 is the input pulse
  - /4 indicates for every 4 signals a signal is generated
- T0CKI is connected the clock on the processor board

There is a configuration registers that store the information needed to send to each device

The only way to interrupt the controller is through the flags like T0IF
- For a 8-bit the limit is 255
- If 100 ms is needed to put as the timing then we can use set the starting point as 156 (for each cycle we overwrite with 156)
  - Then exactly 100 pulses are done
  - When overflowed it will release an event

T0CS – Timer 0 Clock Select
- It decides which timer is used
  - The natural clock or the Arduino clock

The XOR in a way acts as an optional invertor
  (If one input is 1 then the output will be inverse of other input)
- T0SE – Timer 0 select edge
  - Decides if to read the positive edge or the negative edge

Prescaler – Frequency divider
- A simpler counter based
- Divides the signal by a value
  - The value depends on the value sent in the PS<2:0> bus
  - 
PS<2:0>

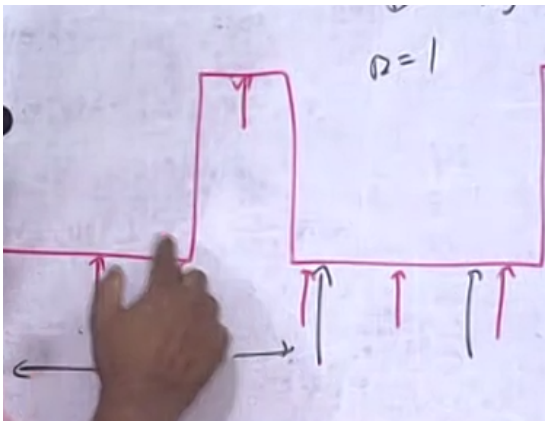
BIT VALUE	TMR0 RATE
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

- The scaling is done before the sent to the clock (pre)
  - TMR0 rate indicates how many input cycles should appear for each output cycle
  - Like if the there are 16 gear teeth in the wheel and needs to

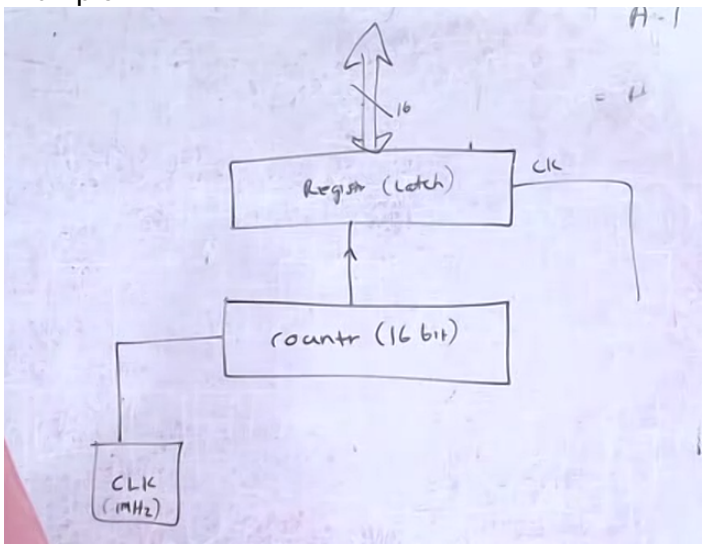
look for the each rotation we can select the 1:16 rate and put the 011 signal code to configure that

- Post scalers also exist

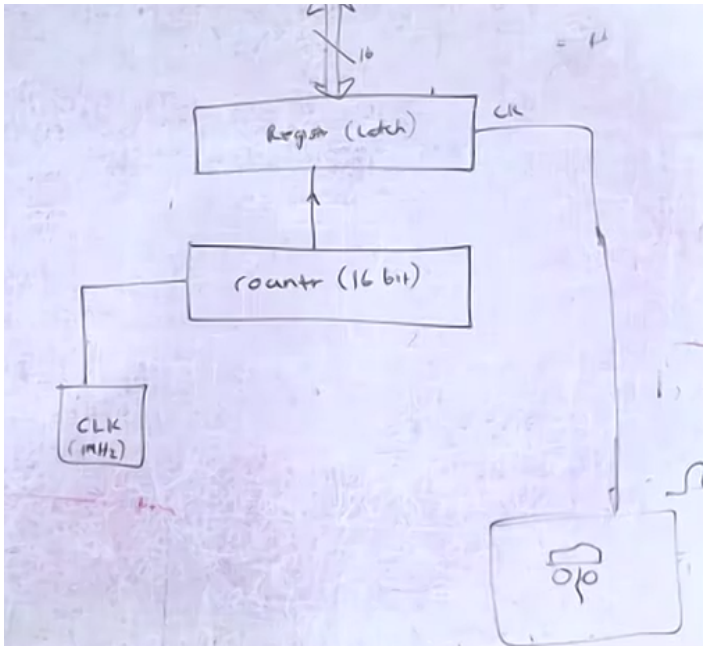
Capture/Compare Timer Modules

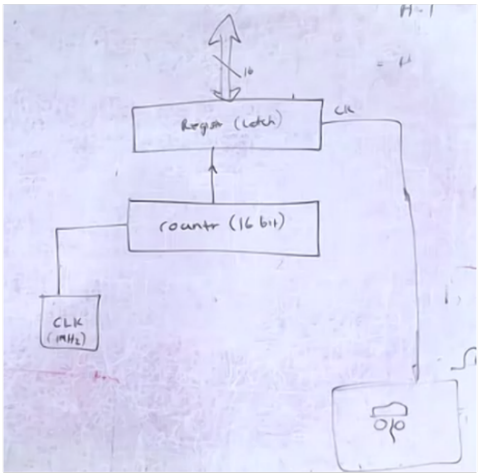
- Capture - for the ordometer.
  - Capturing the time when the event has happened accurately
  - Then the loop needs to be done very fast to catch the exact moment
  - 

Example



- 16 bit counter
  - Which Updates every microsecond
- Have an output register connected to the counter
  - Which has data bus read/writeable
- The requirement is to find when the user presses the button exactly





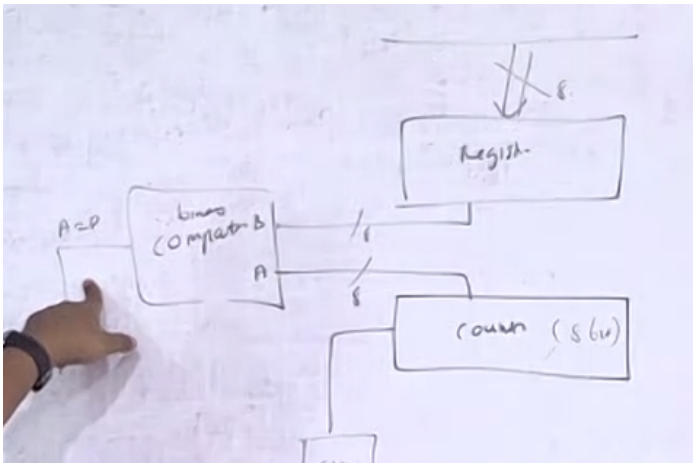
- When pressed makes a pulse sent to the register latch
  - And the latch takes a 'snapshot' of the counter when it happens and remains there
    - Until pressed again which overwrites it-

- When the MPU notices the register latch has changed its value then it can be notified that the event has happened

- But we have to ensure the register is read before it overflows at 64 seconds
  - And a cycle will be missed

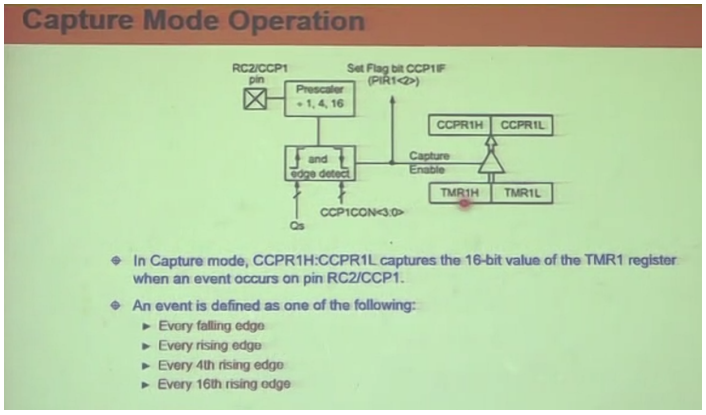
- But the advantage is the port is only needed to be read every 64s and but in 5ns while having the exact timing

- **Compare**
  -
  -

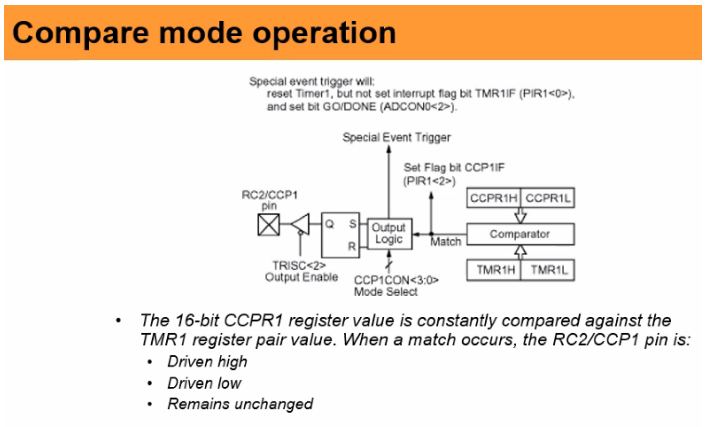


- Compares the two values in the registers and check if there are equal (or less than or greater than)
    - Sends an event when the timer reaches a certain tick count

<<<Get the Slide>>>



- When the capture signal is made it will be sent and stored at the corresponding to the latch register
- A flag is used to detect if a new capture is made
- The prescaler counts for how many edges should be made before a capture happens.
- And we can configure whether for rising/falling edges



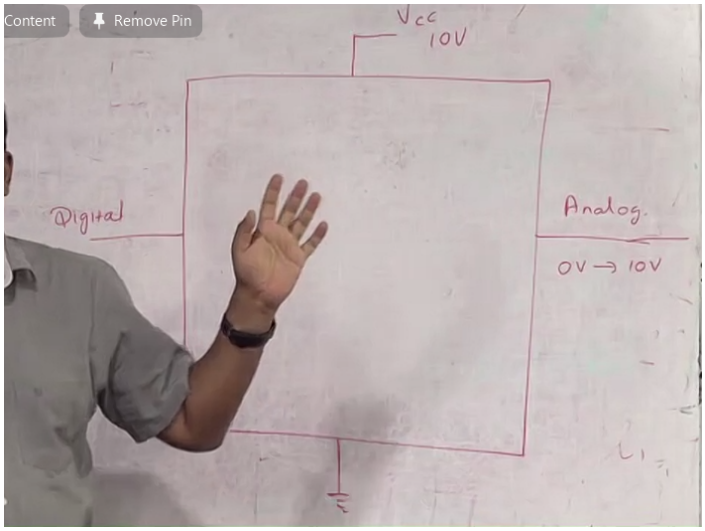
- Instead of the capturing mode
    - There will be a comparator
- another flag can have to indicate when the comparison was successfully made

- Some may have both the timer and capture/compare modules are built with the same hardware
- Or independent or the same at times

L5 – II – Peripheral Devices

Tuesday, June 06, 2023 8:25 AM

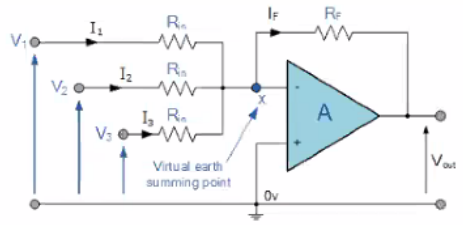
- IoT devices need to work in the real world
- Hence need to work with Analog signals
  - Hence need to do Analog-Digital conversions



- We need to map the Analog signal to the corressponding digital signal
  - Which depends on the supply voltage
  - The analog voltage values range from 0 to 10 V
  - The digital signal is depicted using n bits and has 2^n – 1 finite voltage values
    - Hence there is Discrete range of analog values
    - There are only 2^n –1 different values of analog values that can represented
  - In the case of n=4
    - With 16 possible values in range of 0 – 10V
    - The step size will be 10/(2^4) = 10/16 = 0.625 V for the different analog signals
- But when converting Analog to Digital

- The analog value may not be exactly fitting to those step sizes
- Hence may need to round off

Digital to Analog converters



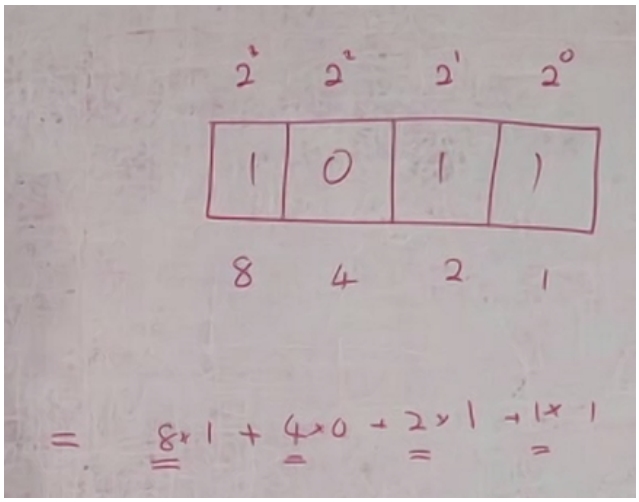
$$I_F = I_1 + I_2 + I_3 = -\left[\frac{V1}{Rin} + \frac{V2}{Rin} + \frac{V3}{Rin}\right]$$

Inverting Equation: $Vout = -\frac{Rf}{Rin} \times Vin$

then, $-Vout = \left[\frac{R_F}{Rin}V1 + \frac{R_F}{Rin}V2 + \frac{R_F}{Rin}V3\right]$

- OpAmps can be used as Summing Amp
  - Where input voltages are added to the output

- You know how OpAmps work right?

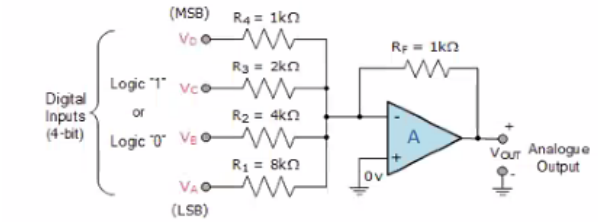


- When considering Binary numbers each bit has a place value
- Another way of saying is addition of the weight factors that have bit value of 1

then, $-Vout = \left[\frac{R_F}{Rin}V1 + \frac{R_F}{Rin}V2 + \frac{R_F}{Rin}V3\right]$

- Using OpAmps we can build that equation as well
  - Where V1, V2, V3,.. Represents the value of each bit
  - Then the R_{in} should be selected suitably to match those weights

An implementation of that



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

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

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

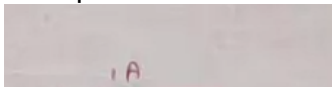
- A Digital to Analog Convertor can be built using a Summing Amplifier
- This is voltage based convertor

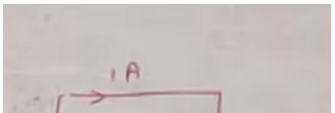
- Pros
  - Easy to add more components
    - Just another resistor for each bit to add
- Cons
  - But hard to control voltage
    - Controlling the power using voltage is not accurate/suitable
  - Hence need to think how to control the power instead
    - Since devices work under a specific range of voltage/current for optimum efficiency
    - To solve this issue uses Pulse Width Modulation

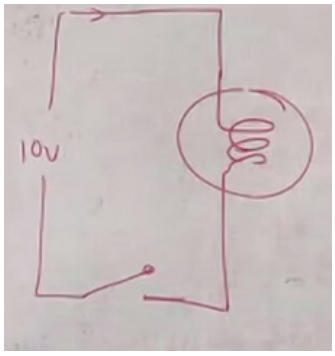
Pulse Width Modulation

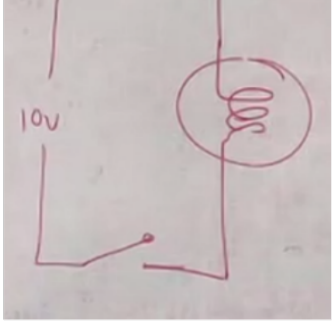
- *The amplitude is maintained constant but the width of each pulse is varied in accordance with instantaneous value of the analog signal.*

Example



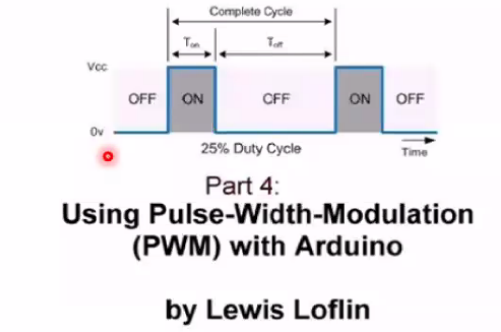






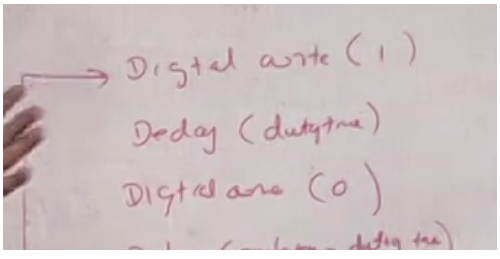
- Practically we can see that filament taking some time to go on and off
- The energy consumed in time t is VIt
- If it was on for 50% time – it will have VIt/2 energy consumed
- If we could turn on/off the light very fast in quick time, the voltage will never go down to 0 or up to the max and instead will get an average voltage
- Hence even when the switch in not on/off for a long time we can still get some energy

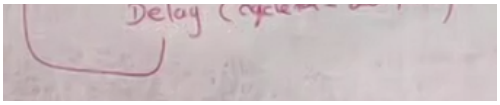
Pulse Width Modulation explained



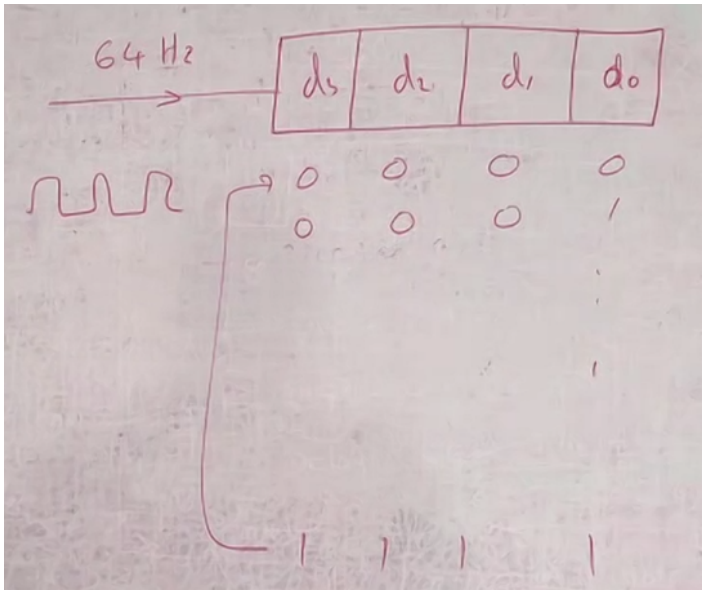
- Duty Cycle is the percentage of time the circuit is on compared to one cycle
- If need to get more power need to increase duty cycle
- And also it allows us to keep the pulse frequency signal fixed at its optimal

- If we wanted we can simulate it like this



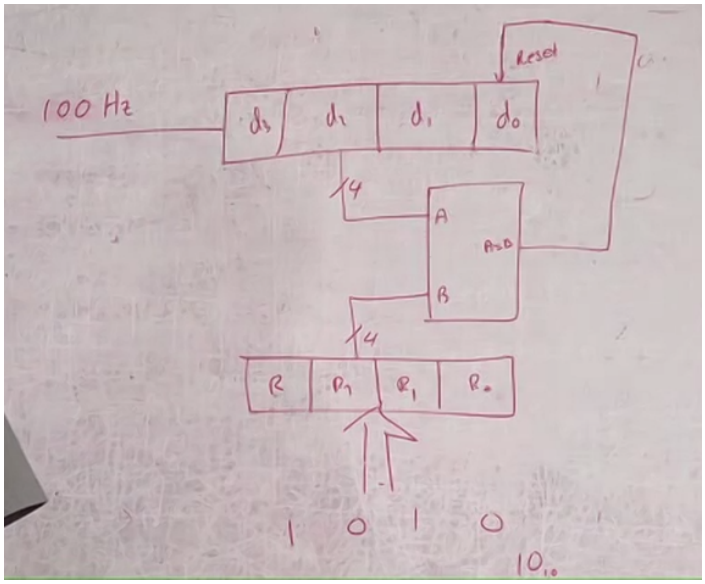


- It will be difficult to do any other work during this period



- A binary counter is also like a clock divider
  - The LSB changes at 32 Hz
  - The 2nd LSB at 16Hz
  - D2 at 8Hz
  - D3 at 4Hz

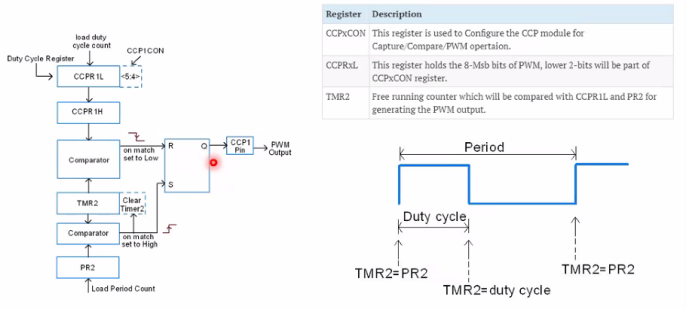
But this only works for 2^n freuency values



- The center componnet is an comparator to check equality
- This checks when to reset the counter when it receives a certain value
  - Like counts upto any value we need

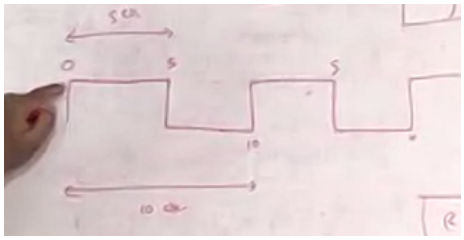
- If we take the signal from the reset line, we can get a clock signal of the time period we require
    - Which fixes the fixed time cycle in the microcontroller

PWM module in PIC microcontroller



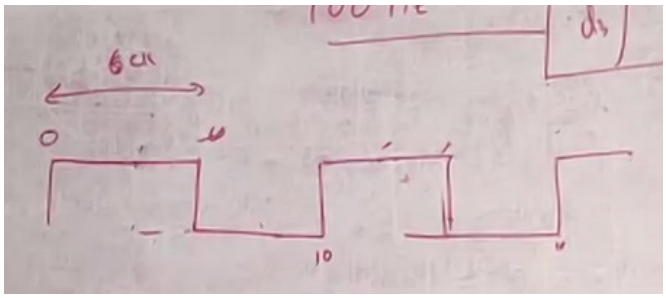
Register	Description
CCPxCON	This register is used to Configure the CCP module for Capture/Compare/PWM opertaion.
CCPRxL	This register holds the 8-Msb bits of PWM, lower 2-bits will be part of CCPxCON register.
TMR2	Free running counter which will be compared with CCPR1L and PR2 for generating the PWM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

- In this example the comparator sends its output to the clock clearer and the set of the flip flop



- Changing the CCPRIL we can control the duty cycle
- Changing the CCPRIH to change the cycle time

<<Check the recording at 9:14am>>

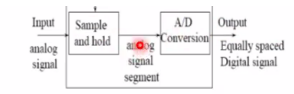


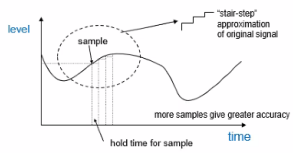
- When CCPRIH – 6

- When Changing from 6 to 3, for one cycle (the first) we might miss the chance to give the output
    - For critcal devices it is important not miss them
- Hence there is master slave tbetween the CCPRIL and CCPR1H to start the new cycle with different time sharp at the new cycle

Analog to Digital Conversions

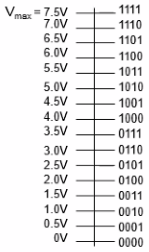
- *Converts analog signals into*



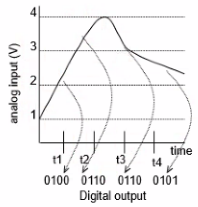


binary words

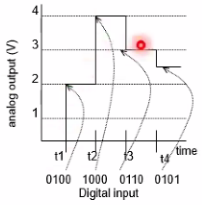
- *Involves a 3-step process*
  - *Sample and Hold*
    - *Freeze ever changing analog voltage until conversion is done*
  - *A/D conversion*
    - *Quantization*
    - *Encoding*



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:

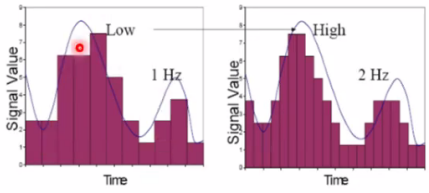
$Q=(V_{max}-V_{min})/N = (10V - 0V)/8 = 1.25V$

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

<<Sorry lost focus at this section>>

- The range gives an idea of the accuracy of the conversion
- Need to consider the sampling rate

Sampling Rate



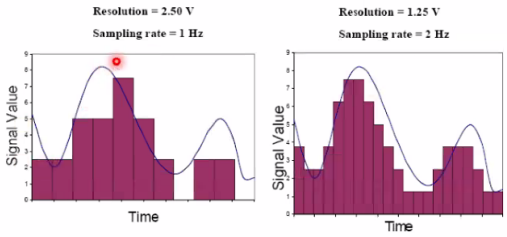
Frequency at which ADC evaluates analog signal. As we see in the second picture, evaluating the signal more often more accurately depicts the ADC signal.

- A higher sampling rate will get a more accurate representation of the analog

- But more samples means – the conversion should be done more faster

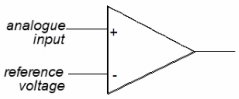
Overall Better Accuracy

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

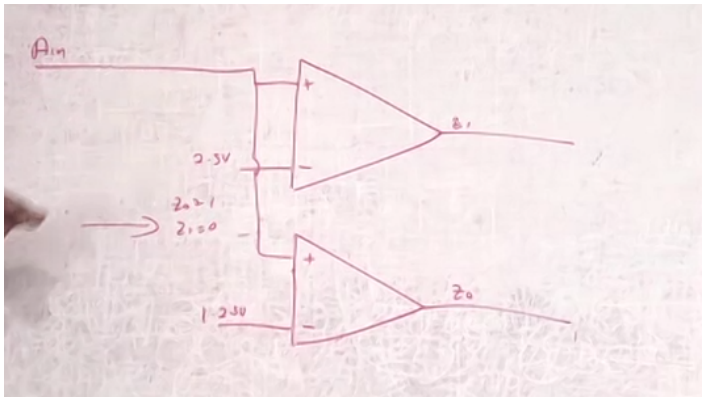


AtoD converter types

- *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*



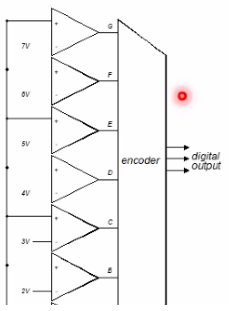
- *A comparator compares 2 signals A and B*
  - *if A > B the comparator output is in one logic state (0, say)*
  - *if B > A then it is in the opposite state (1, say)*
- *A comparator can be built using an op amp with no feedback*



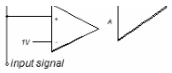
- Can have a series set of OpAmps (comparators) to check some decision points
  - If the analog signal is greater than the reference voltage than it will give 1

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 compactors*
  - *6-bits would require 63*



- *8-bits would need 256*



- The Encoder will then check the value and give the corresponding digital output

But the bad thing is

- We need 2^n comparators for n-bits
- And need some for logic to be implemented
- And not practical to fit

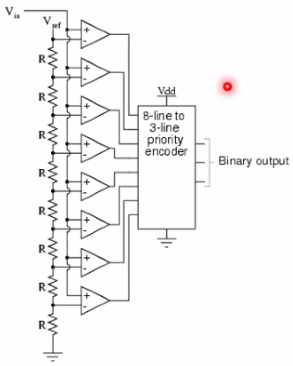
But the good thing is

- Only delay is the propagation delay
- Hence fast
- Flash conversion

- Hence this method is good for low accurate, fast conversions

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*



- The line from V_{af} acts as a potential divider

- Good and fast
- But has some practical difficulties
  - <<<Check the recording>>>

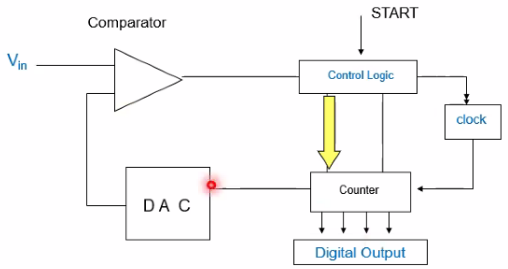
- Or change the reference voltage sequentially one at time without using a lot of fixed comparators

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)*

- We go through each of the limits until we find one where it is below that reference level
  - No fixed reference level at each checking
  - Hence need to generate them
- For that purpose we use a Digital to Analog Convertor to generate the analog reference signal
  - Just by changing the digital register value

Counter type converters



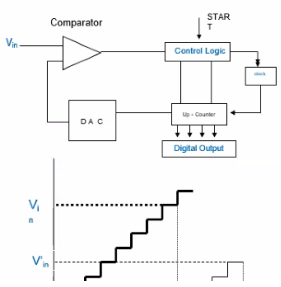
- *When START is received, control logic initializes the system, (sets counter) and modify counter value in steps so that DAC generates the corresponding analog voltage refererice for compactor.*
- *Process stops when a match is found*

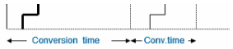
- If the analog signal is greater than the current reference level
  - Update the counter which writes to the reference level
  - Repeat until the analog signal is less than the current reference level

- Cons
  - Sometimes will need to take a lot of time
    - For small values it will come faster
  - But for large values, the time taken is variable
  - The value is not correctly represented until the whole conversion is completed by the way

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 Vin*
- *Each conversion cycle starts from the same initial, zero value*
- *Conversion time is variable and depends on absolute value of Vin*





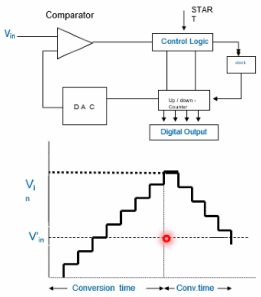
- Only few components are required for this
- It's like during a search

- In order to shorten the time taken to search
  - Since most analog signals are continuous like sine waves
  - We can continue the searching from the around the previously found value

Like 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 Vin*
- *Each conversion cycle starts from the last matching reference value*
- *Conversion time is variable and depends on the change in Vin*



- But need two comparators to check which way to look for

- This method is used especially in continuous signals
- The first point will take more time
  - But subsequent points can be searched quicker

- But can do the tracking as a Binary search instead
  - Then a sequential search is not needed
- But can just do a binary search
  - And reduce the searching space recursively

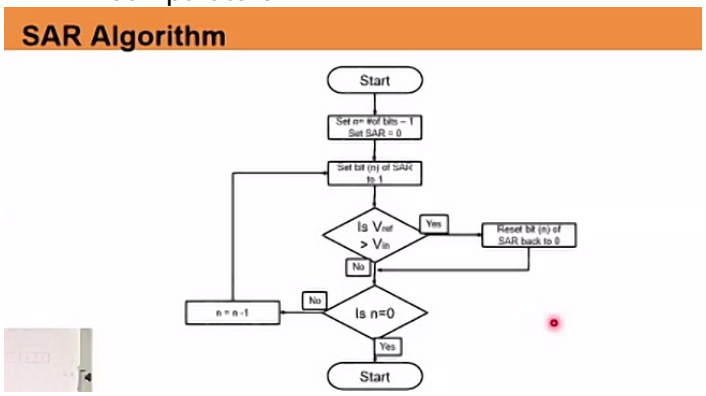
<<<Paste Slide Here>>>

Eg: For a 4 bit
Searching for 13

- First mark 1 in the MSB (then 1000)
  - Compare
  - Since the 1101 > 1000
  - Keep MSB =1
- Then mark 1 in the second MSB (then 1100)
  - Compare
  - Since 1101 > 1100
  - Keep 2nd MSB = 1

- Then mark 1 in 3rd MSb (then 1110)
  - Compare
  - Since not 1101 > 1110 XXXXXX
  - Put 0 for 3rd MSB
- And so on

- Good things
  - Fixed time to do all the searching
  - Number of steps is fixed
  - And relatively fast (n comparisions for n –bits)
  - For any # bits just need 2 comparators



- And most common algorithm – SAR

L5 – III – Peripheral Devices

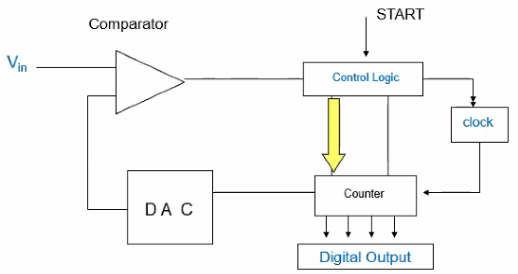
Tuesday, June 13, 2023 8:18 AM

- 50 MCQs online
  - Exclusding the last 2 weeks

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)*

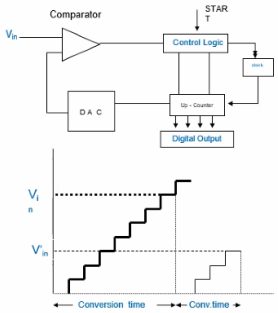
Counter type converters



- *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

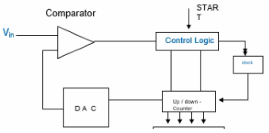
- *Conversion process*
  - *When START signal is received, conversion starts by initializing up-counter to zero*
  - *On each clock tick the counter is incremented by 1, gradually increasing the quantization reference level in a ramp pattern*
  - *Conversion completes at the point where DAC output meets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



- Control Logic does the estimation near the start
- The output of D to A conv will be sent to compare with the input to check if it reached the correct estimation

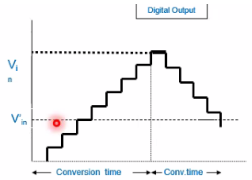
Tracking (Servo) Converter

- *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 Vin

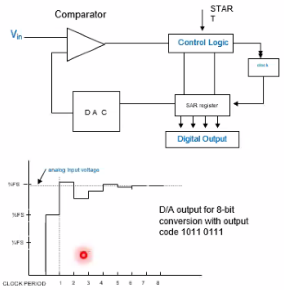
- *Each conversion cycle starts from the last matching reference value*
- *Conversion time is variable and depends on the change in Vin*



- This is a faster approximation which can do the process faster – if the wave is continuous (hence it starts reading from the last read location)
  - But needs to measure from both up and down until it reaches correct one
  - The control logic may be more complicated in this case

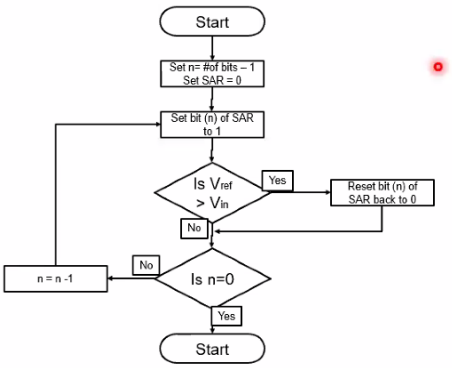
Successive Approximation Register Converter

- *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 Vin, approximately n-cycles for a n-bit conversion*



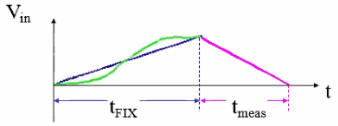
- The even faster one is SAR
  - Uses the binary search
  - The search space reduces by half at each round

SAR Algorithm



- Explained earlier ??

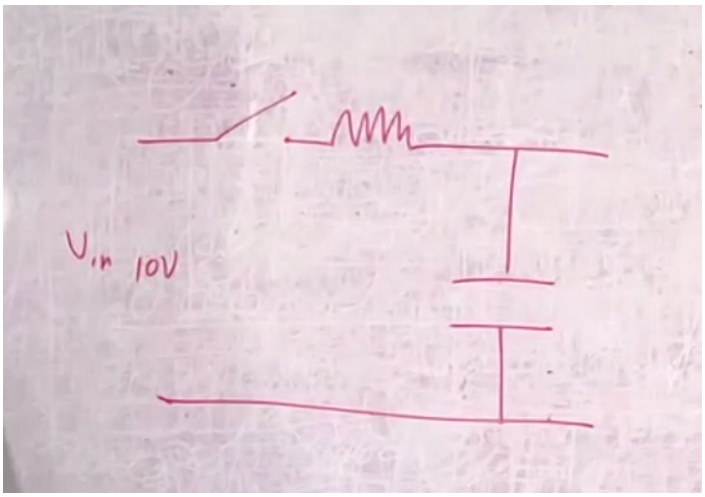
Dual Slope Converter

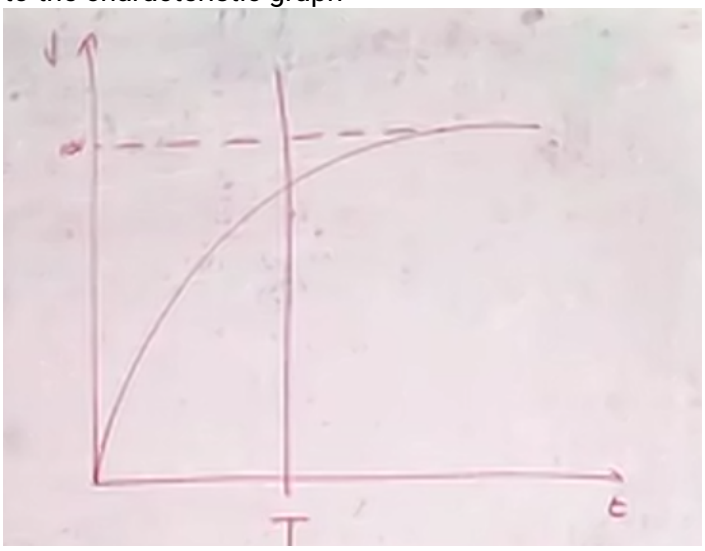


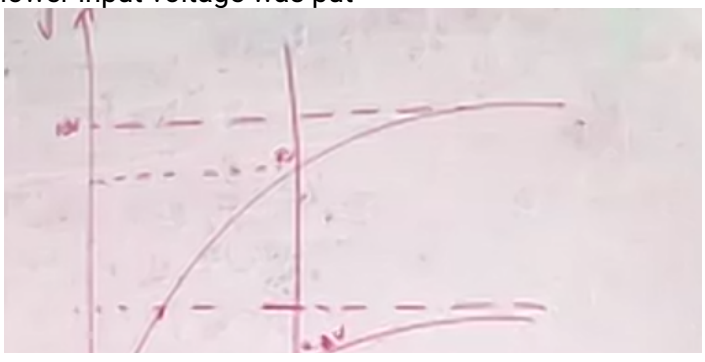
- *The sampled signal charges a capacitor for a fixed amount of time*
  - *By integrating over time, noise integrates out of the conversion*
- *Then the ADC discharges the capacitor at a fixed rate with the counter counts the ADC's output bits*

rate with the counter counts the ADC's output bits. A longer discharge time results in a higher count

- Earlier ones we make an approximation and do a comparision
  - But the accuracy of DtoA conv are not reliable
- The dual slope convertor uses the capacitor charging and discharging patterns

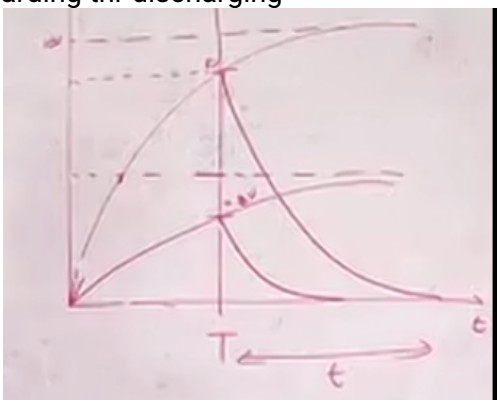
- The capacitor is charged according to the input Voltage for a fixed amount of time
  - 

  - When the switch is closed the capaccitor charges according to the characteristic graph
  - 

  - If a lower input voltage was put
  - 

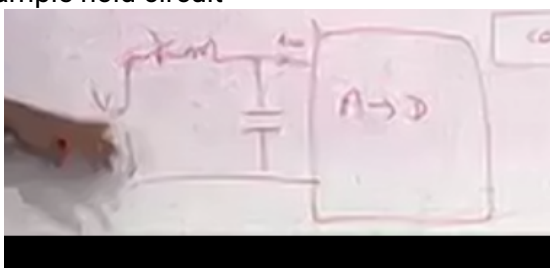
- As we can see for a fixed time, the charge accumulated will depend on the input voltage

- Regarding thr discharging



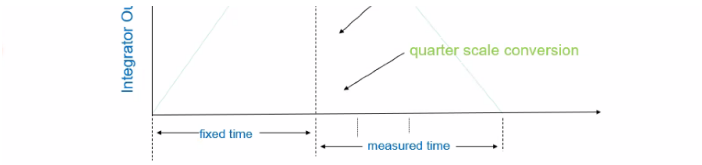
- Depedning on input voltage, the time taken to discharge differs
  - This can be measured by a counter

- An advantage of this process is not relying on A to D convertors
- Another advantage is no need a sample and hold circuit
  - To keep the voltage signal as it is for a period of time in order for the convertors to read them
    - But this will make it hard to track smoothly the signal as it takes time to be spent
  - A sample hold circuit
    -



- But a disadvantage of Dual Slpoe
  - The times to charge can also vary due to reasons like temperature

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



- Dual Slope is used when no need for high accuracy
- SAR is best for fast and better accuracy
- Flash is also fast but very expensive

- Depending on the application need to decide
  - Like audio waves are continuous
  - But video waves are random and hard to predict the next signal

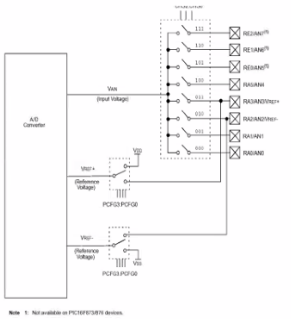
ADC Types Comparison

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

- Dual Slope is slow since it takes time to charge and discharge

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.*



- In IoT devices need to deal with multiple analog signals
- But A to D convertors need a lot of Silicon and take a lot space in the microcontroller
- And the comapred to most signals in the environment analog signals like sounds changes very fast

- Hence most microcontrollers uses one A-to-D convertors
    - But with a multiplexer connecting to the different analog channels
        - All those input channels share the same convertor
- But this configuration may not be suitable if need to sample very fast

L6 – Physical Interfacing

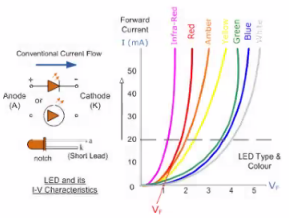
Tuesday, June 13, 2023 8:47 AM

- When connecting to a output device – we should note that devices work with a fixed vol/current rating
- Hence the driving voltage/current must also match to the device

- Looking at the basic interfacing needed

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.*



- LEDs light when forward biased
- After a certain voltage limit the current through the LED increases exponentially
- If too much current goes to the LED, it will get burned very quickly
- Hene challenging

- Most cases the voltage cutoff is like 2.2V for LED
- But microcontroller high voltage level is likely 5V or 3.3V
  - Hence can't really connect to microcontroller
  - Microcontroller pins also limit the current through as well (like 50mA)
    - Any further the input resistance will go high and no brightness will come
  - Or it will burn

- Also note even though a single output pin in the Microcontroller can give a high current rating like 75mA
  - But the totally for all pins it can only provide 500mA

Either we should

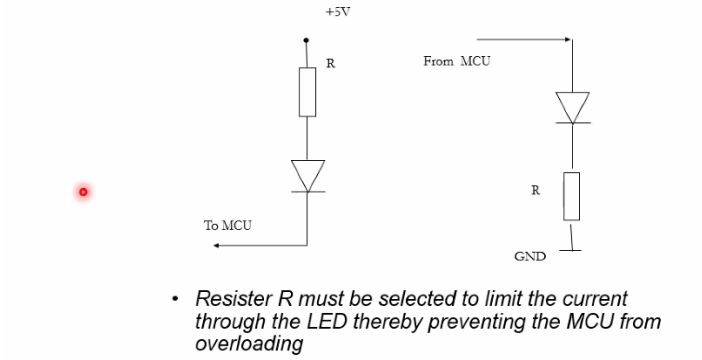
- Get constant voltage for the LED to keep it within the current limit

- Or fix the current using a current source

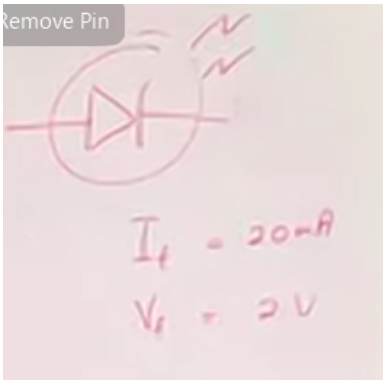
But Both these techniques uses a lot of circuits and transistors

- So use a resistor then

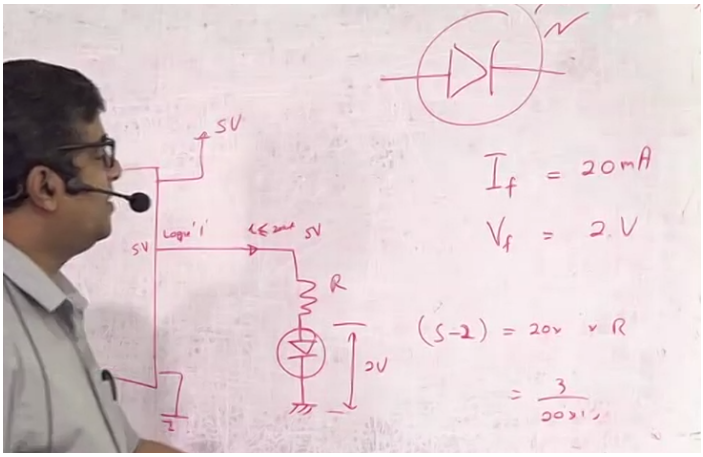
Driving LED's



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

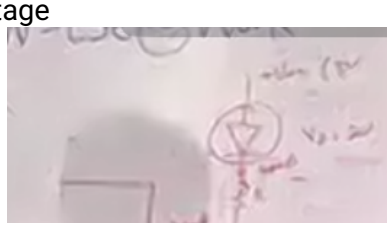


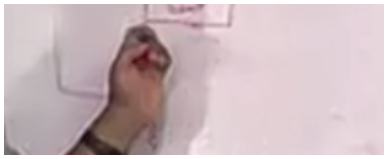
- Assume the forward bias is 2V



- Calculate the resistor to make the high 5V into 2V after passing through the resistor

What if need to make a LED light at low voltage

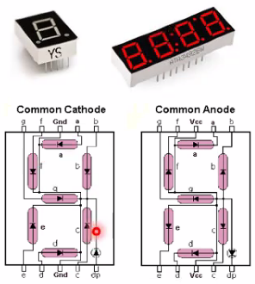




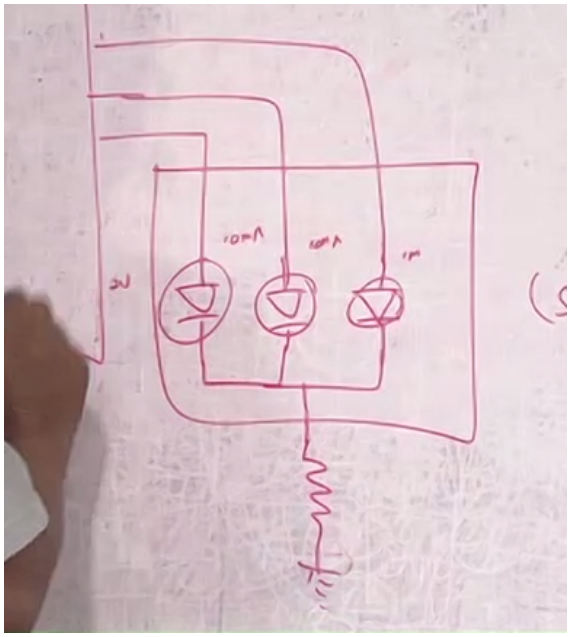
- If R is not selected correctly the LED will go too bright or too dimly

7-Segement Displays and Multiplexed driving

- *Consist of 7 rectangular LEDs arranged in the formation of a decimal digit (8th LED for decimal point)*
- *Available in two types*
  - *Common cathode*
  - *Common anaode*
- *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*

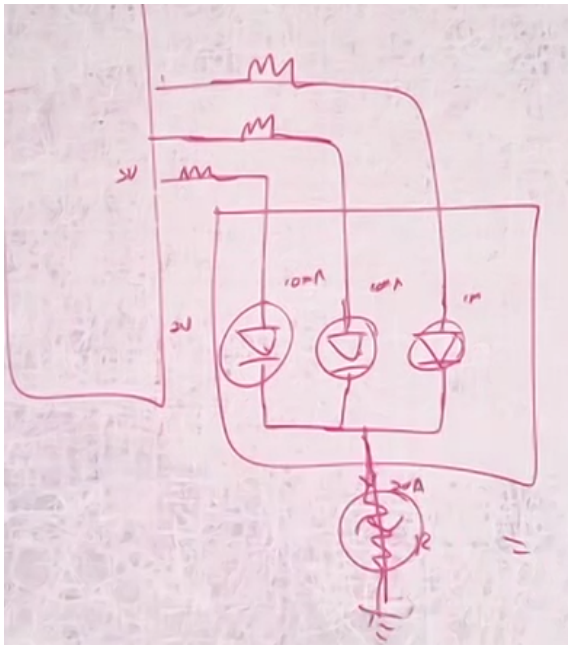


- 7 Segment Displays uses multiple LEDS needed to be driven
  - Two Types
    - Common Cathode
      - We give Voltage high to the required segment
      - And ground the common cathode
    - Common Anode
      - We ground the individual ones
      - And high Vcc to the common anode



- We cannot do like this since the current can varying

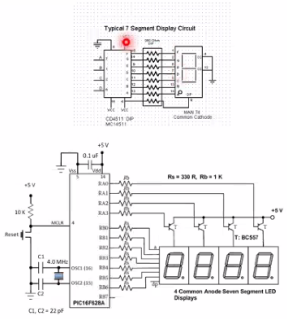
- Also it will work only if one of them were working
  - But if multiple were on the voltage drop will much larger then the LEDS will be dim



- Hence instead of a common resistor at the bottom
  - Need to have separate resistors for each LED
  - Or a common current driver at the bottom
- To ensure the same current through each LED

Direct and Multiplexed driving

- *A resister 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*



- The top circuit shows how separate ressitors are placed accordingly

- Multiplexing maybe needed to light multiple 7 Segments
  - In this each 7 Segment is light up one at a time at a very fast speed

Driving Inductive Loads

- *Special care must be taken when driving inductive loads*

directly as well as through buffer stages.

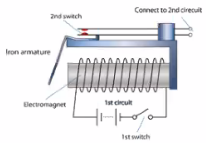
- *Large transient currents and voltages involved in inductive loads are likely to cause noise and other related fluctuations in the supply voltage.*
- *These must be filtered to prevent un-predictable behavior of the system.*

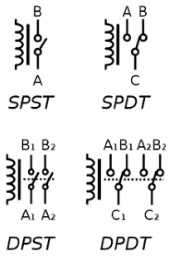
- Examples include that have electomagnetic loads in them
- When current passes through them, electromagnetic field is formed
- And vice versa when an electromagnetic field is arround it causes a voltage

- But

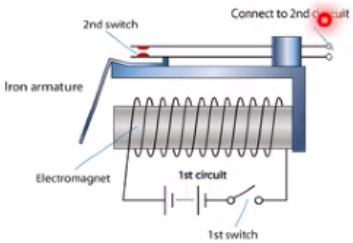
(Missed something here)

Electromagnetic relays

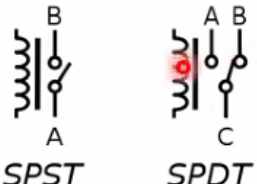


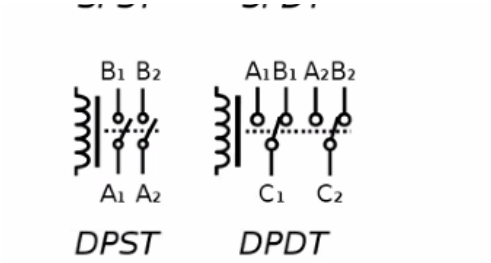


- Relay – unlike a mechanical switch – it will turn on/off by some electromagnetic power
- When a voltage is put to the primary circuit (electormagnet) the other 2nd s
-



-
-





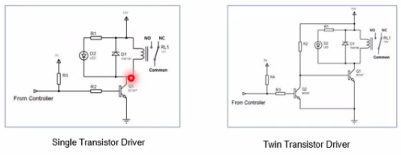
- DT –Dual Terminal – can switch to another circuit and not only one connection
- Dual Pole – There are two terminals that work simulatenously and parallely

- More coli current need a larger armature (did I hear that right?)

- Need a buffer (Heard just that)

- Next need to handle the back EM – which happens even turning off
  - It causes some unpredictable power passing through them

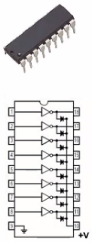
Driving Relays



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

- Uses a transitor that can handle higher current (compared to microcontroller)
- When transistor is off no current goes through the relay
- R3 resistor is there to ensure the relay is not turned on unintentionally
  - Especially important when it may end up at a floating stage

- There can be a large reverse voltage across the relay and could damage the microcontroller
  - Therefore we place a diode in reverse bias
    - At normal operation nothing happens to it
    - But if a backward voltage is formed, the back EMF can

be neutalized by going through that voltage
    - A protection/shunt diode to protect the circuit only

The 2nd circuit above uses 2 transistors

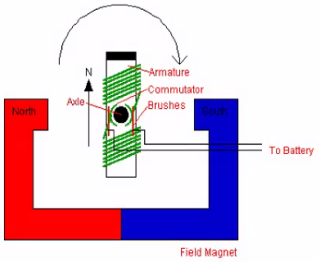
- When one is on the other will be turned off
- The benefit is
    - The 2nd transitor (near the realy) can handle higher current and therefore the relay can handle higher current

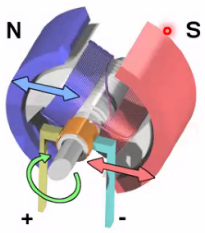
An IC can be used itself which have the driving delays including the protections (but without the relays)

- Hence no need to build something like that

Direct Current Motors

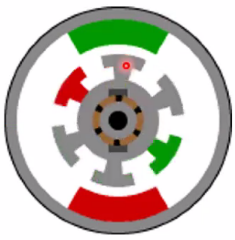
- *overall plan of a simple 2-pole DC electric motor*
- *A simple motor has 6 parts, as shown in the diagram*





- Now Motors
- Large motors can't be driven
- Small motors can be dirven directly
- Few considerations when driving
    - Speed of motor
    - Direction of motor

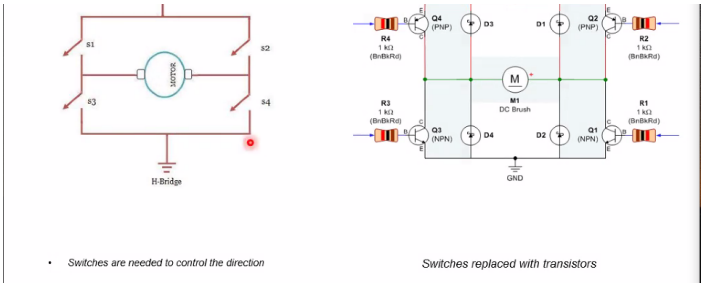
Brushed DC Motor



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

- If reverse of input voltage then it will turn the other way

DC Motors – Direction control



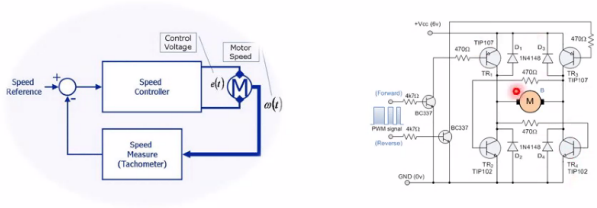
- Uses a H circuit
  - Like the letter H
- The two directions can be formed are
  - Only S1 and S4 closed
  - Only S2 and S3 closed
- Any other two switches being turned on will give power surge/short curcuit burned

- To ensure no damage we might have to disconnect the ciruit before changing to the other pair
  - To allow for mechanical delays in the realy

- In the 2nd circuit it will have shunt diodes to handle the back EMF as well
- And transistors for switches

- Speed is controlled by changing the electrical energy going to the motor
- PWM (pulse width modulation) can help handle that

DC Motor speed controller

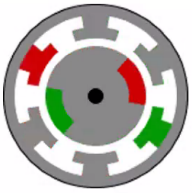


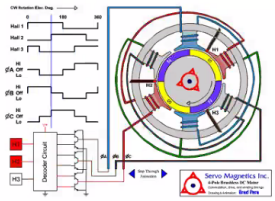
- A higher duty cycle will make the motor faster

- The mechanical load the motor will also impact the speed of the motor
  - Where even the same PWM signal can have different speeds due loads
- Hence need to have a feedback control loop to ensure the output is at the required speed we needed

- This is a server controller circuit

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*

- The motor turns in steps
  - Not continuos
  - Like clocks second hand moving only at the second

The armature only turns here when the current sent to the brushes are changed

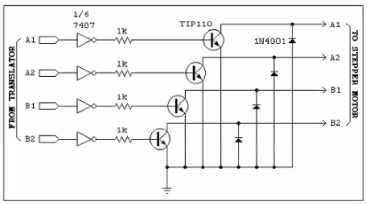
If curent fixed at one place the armature stops there
(Is that right ICE people??)

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*

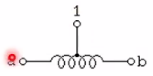
- Driving a Stepper motor may need to drive multiple parts

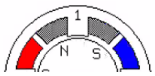
UNIPOLAR STEPPER MOTOR

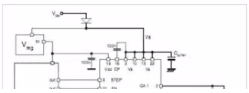


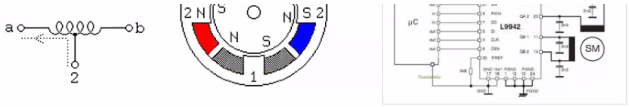
- *Relatively easy to control*
- *simple 1-of-'n' counter circuit can generate the proper stepping sequence*
- *1 transistor per winding*

UNIPOLAR STEPPER MOTOR









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

- A lot of things are there
- Hence use a Stepper controller IC can be used
- L9942 IC drives the motor based on signals by microcontrollers

THE THEORY IS SIMPLE

- Just 1's and 0's
- But what makes it complicated are the small things that need to consider and handling those small things that make the device fail even for a millis seconds
- And need to be careful since it gets complex when combining them togther

- ***Make a mistake, many people will die (no pressure)***

- Safety and planning is critical for IoT devices and relaiably
  - This is the challenge
  - And of course make it cheap and cost effective
    - Challenge for the engineer (not like people who learn from YouTube only 😅)