College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Course Information

Embedded Systems

Contents

Rules & Regulations

I will let it on your **Behavior !!**

Prerequisites:

C programming, Microprocessors, Control Systems, Signals & Systems

References

1.” Introduction to Embedded Systems” , David Russel, 2010
2. “Embedded Control System” , 2th edition
3. Embedded System Design: A Unified Hardware/Software Introduction
  " Online Book

Weekly hours and module units

- Each week, you have
  - Theoretical lectures: **3 hours**
  - Practical lectures: **2hours**
- **Module Units: Three**

Assessment

** Final Exam:

- Theoretical part: 35%
- Practical part: 15%

** Midterm Exam

- Theoretical part: 20%
- Practical part: 15%

** Quizzes, H.Ws and attendance: 15%

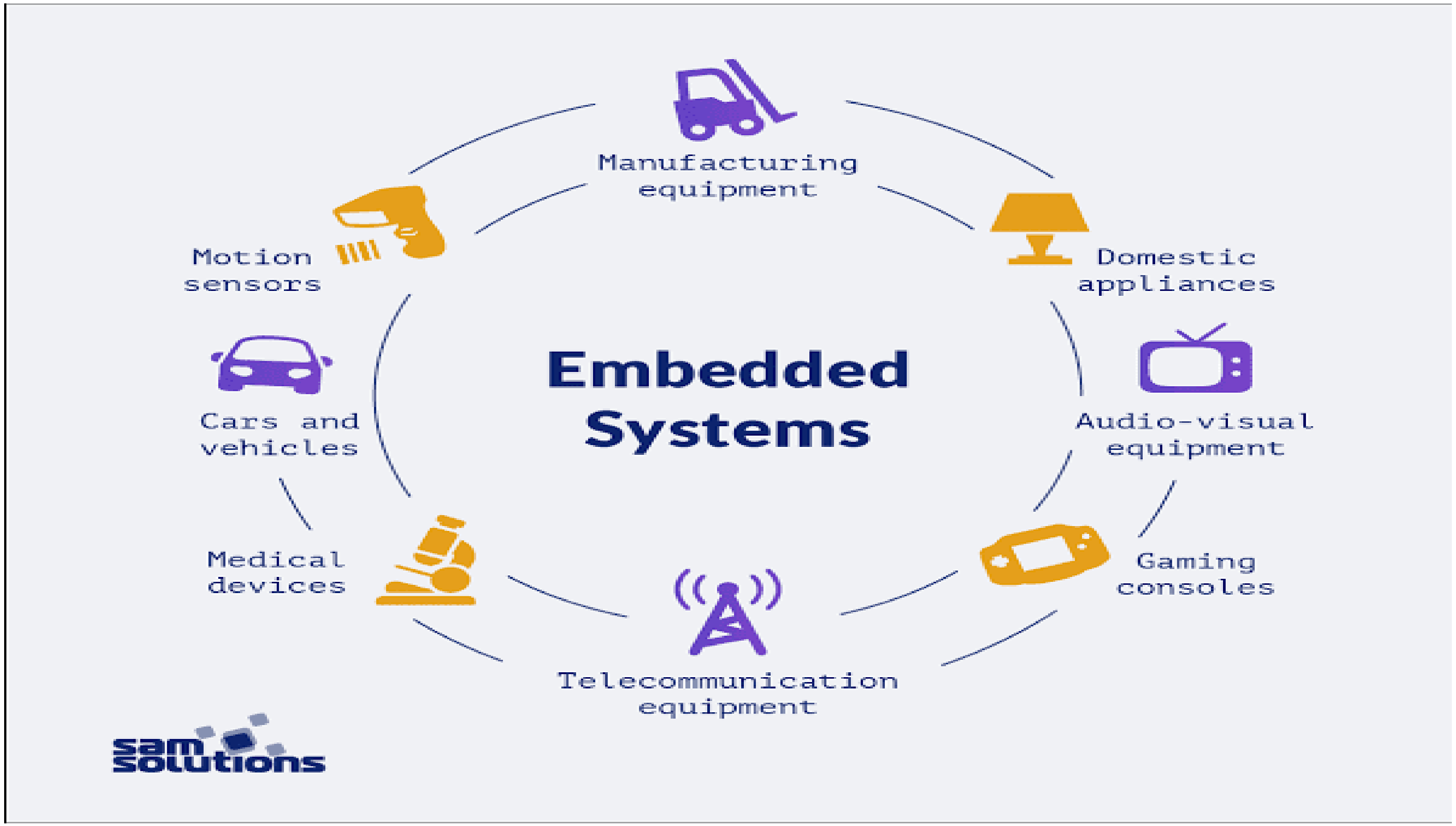
College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

What's an Embedded System (E.S)

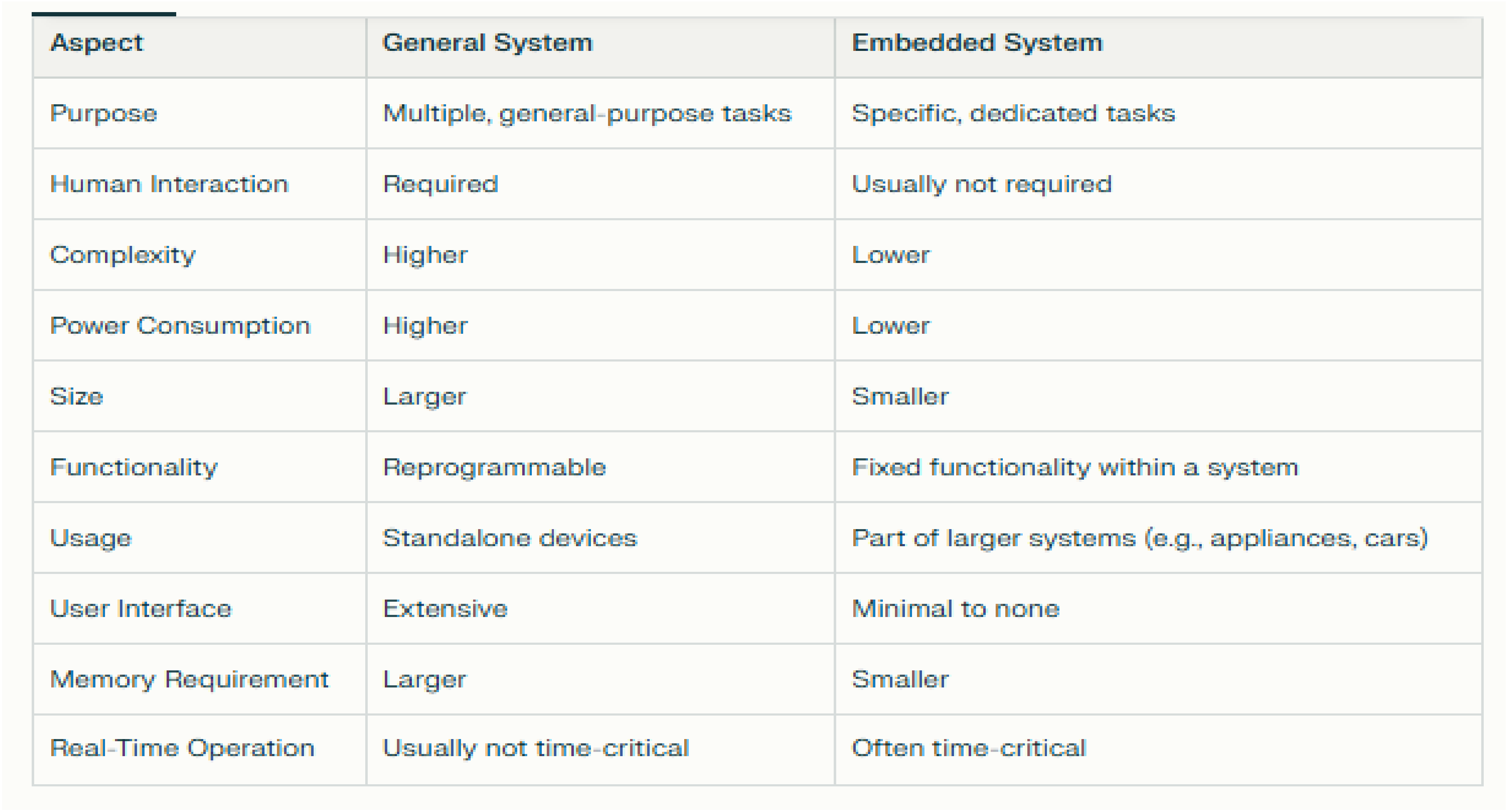
Examples of everyday devices that use embedded systems!!!!!!



What is an Embedded System?

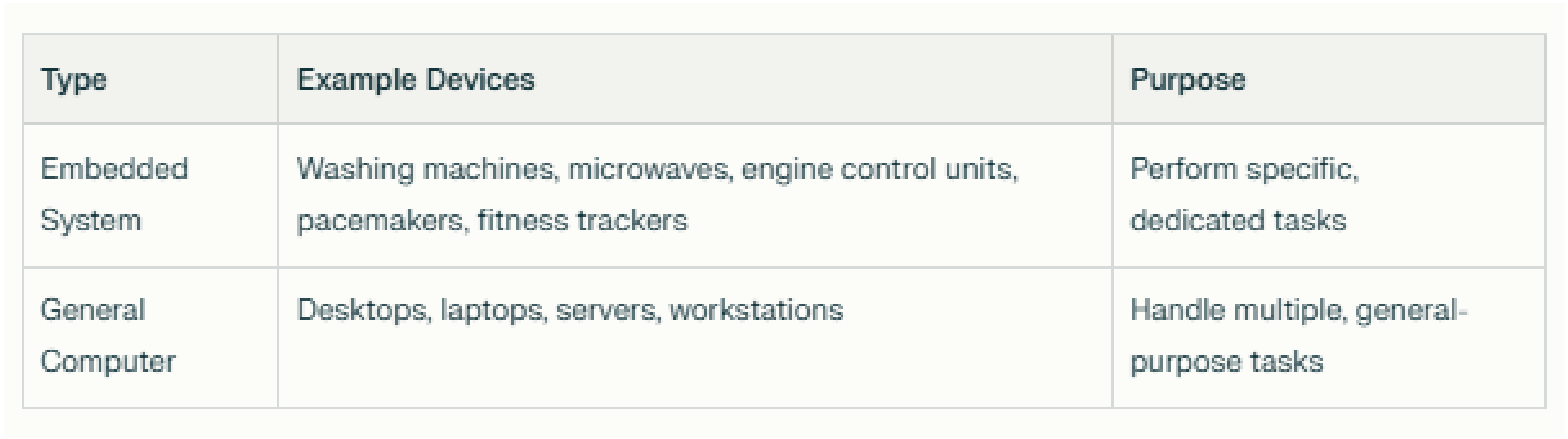
- **An embedded system is a combination of hardware and software designed to perform a dedicated function within a larger mechanical or electronic system. It typically consists of a microprocessor or microcontroller, memory, and input/output interfaces, and is optimized for specific tasks rather than general-purpose computing.**

The difference between a system (general computer system) and an embedded system

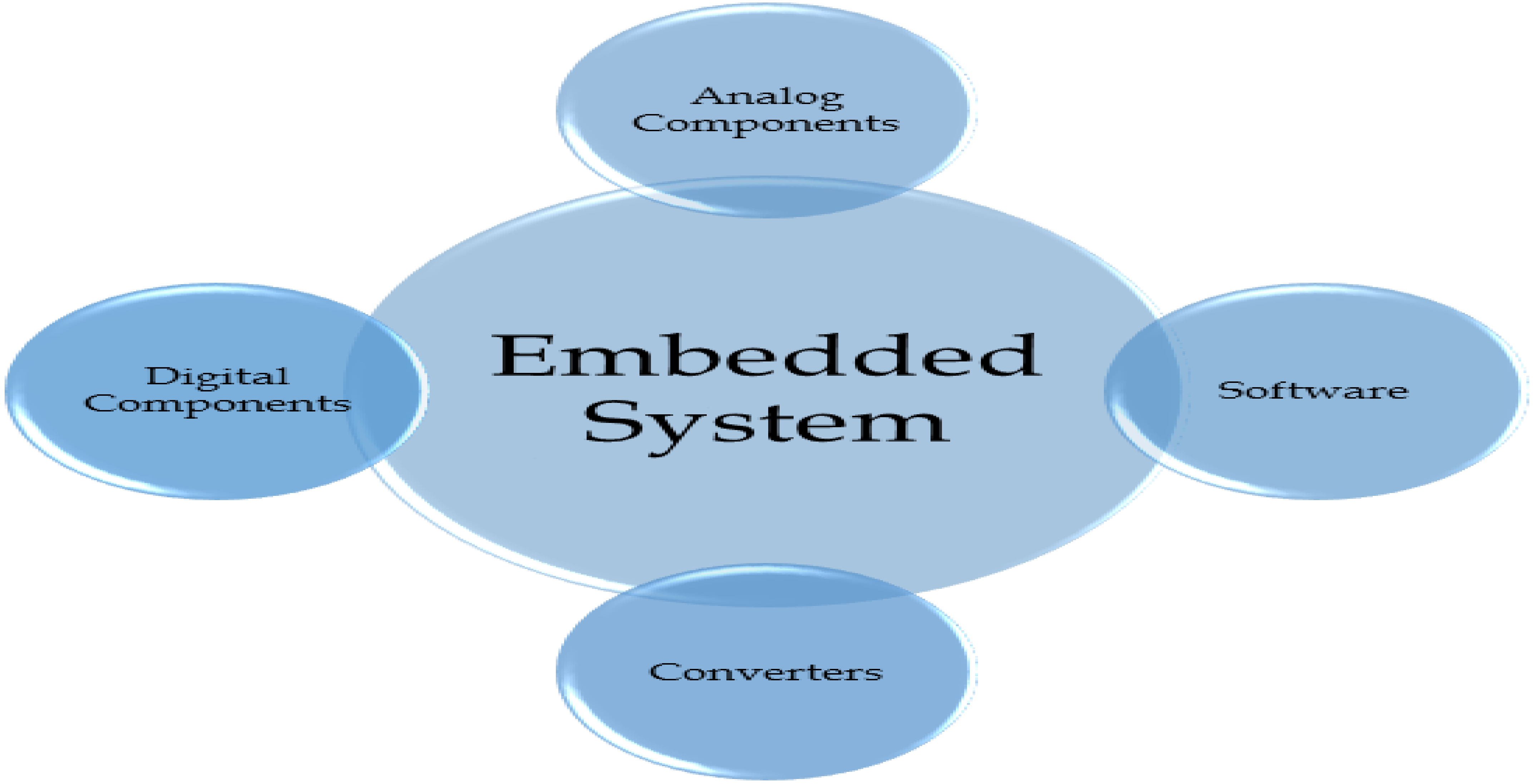


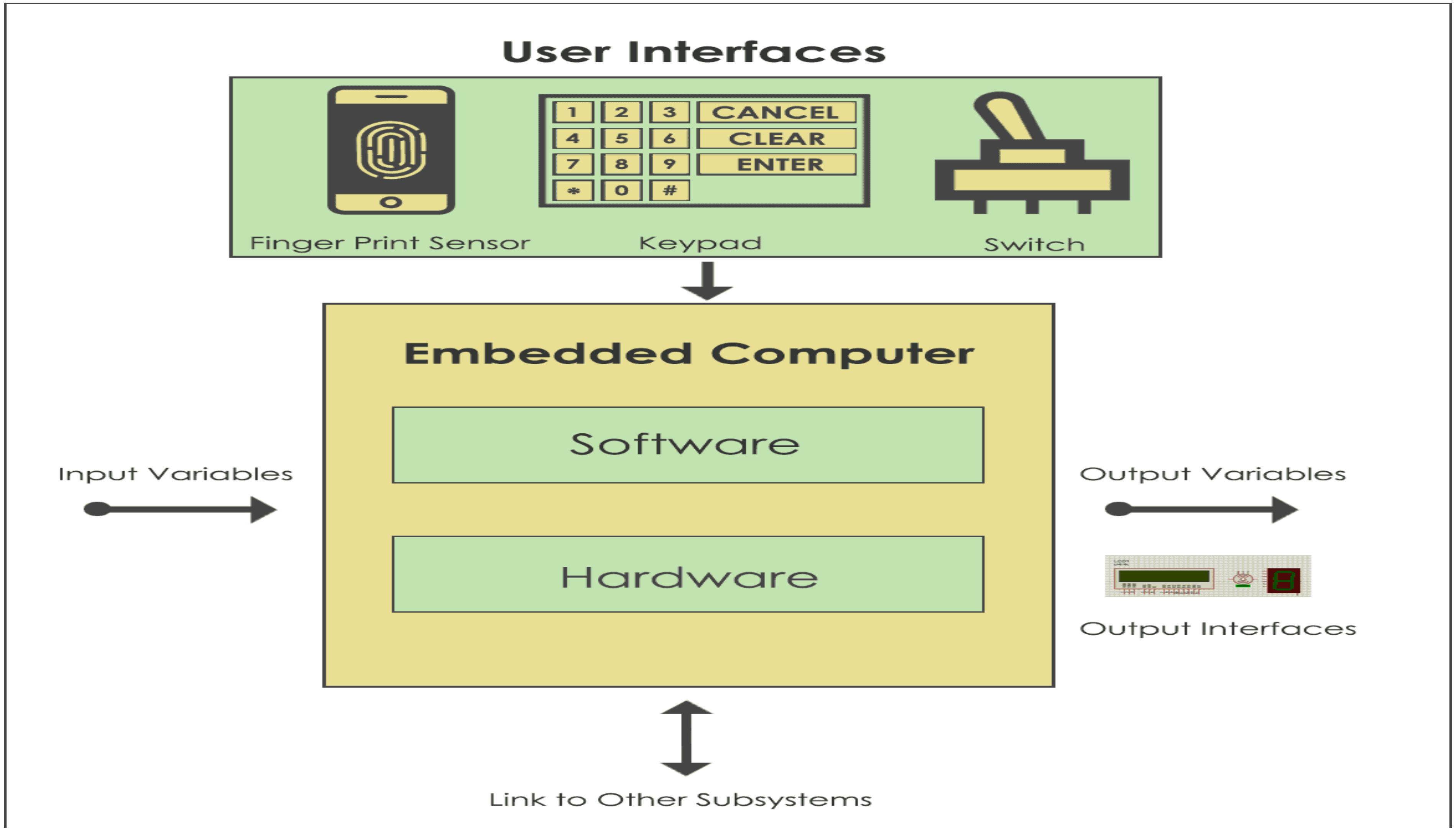
Aspect	General System	Embedded System
Purpose	Multiple, general-purpose tasks	Specific, dedicated tasks
Human Interaction	Required	Usually not required
Complexity	Higher	Lower
Power Consumption	Higher	Lower
Size	Larger	Smaller
Functionality	Reprogrammable	Fixed functionality within a system
Usage	Standalone devices	Part of larger systems (e.g., appliances, cars)
User Interface	Extensive	Minimal to none
Memory Requirement	Larger	Smaller
Real-Time Operation	Usually not time-critical	Often time-critical

Typical examples of embedded systems versus general computers illustrate their distinct purposes and functionalities.



Type	Example Devices	Purpose
Embedded System	Washing machines, microwaves, engine control units, pacemakers, fitness trackers	Perform specific, dedicated tasks
General Computer	Desktops, laptops, servers, workstations	Handle multiple, general-purpose tasks


Analog Components
Digital Components
Embedded System
Software
Converters


User Interfaces
1
2
3
CANCEL
4
5
6
CLEAR
7
8
9
ENTER
*
0
#
Finger Print Sensor
Keypad
Switch
Embedded Computer
Software
Hardware
Input Variables
Output Variables
Output Interfaces
Link to Other Subsystems

Characteristics of Embedded Systems

•Designed for specific tasks or functions within a larger system.

•May include integrated circuits with processors, memory, and peripherals.

•Can range from very simple (a single chip) to highly complex with multiple components.

•Often have real-time computing constraints due to controlling physical operations.

•Typically optimized for reliability, low power consumption, and minimal size.

•Interfaces can be non-existent or as complex as graphical user interfaces.

Classification Of E.S.s

E.S.s can be classified into 3 types:

1. **Small – scale E.S.s:**
    - these systems are designed with a single 8-bit, or 16-bit microcontroller.
    - They have little h/w & s/w complexities and involve board-level design.
    - They may even be battery operated.
    - An editor & assembler specific to the µC are used.
    - C – language (or similar languages) is used for developing these systems.
    - Commonly used microcontrollers
2. **Medium – scale E.S.s:**
    - These are designed with a single or few 16-bit or 32-bit microcontroller.
    - They have both h/w& s/w complexities
    - Programming tools: RTOS, source code eng. Tool, simulator, debugger & Integrated Development Environment (IDE).

3. **Sophisticated E.S.s:**
   - These systems have enormous h/w & s/w complexities and may need scalable processors or configurable processors & PLAs.
   - They are used for applications that need h/w & s/w co-design & integration in the final system.
   - Development tools may not be available at a reasonable cist or may not be available at all. In some cases, a compiler might have to be developed for these systems.
   - Commonly used microcontrollers: Intel80960, ARM7, MPC604.

Appropriate Microcontroller Use

A microcontroller is the correct tool to use when:

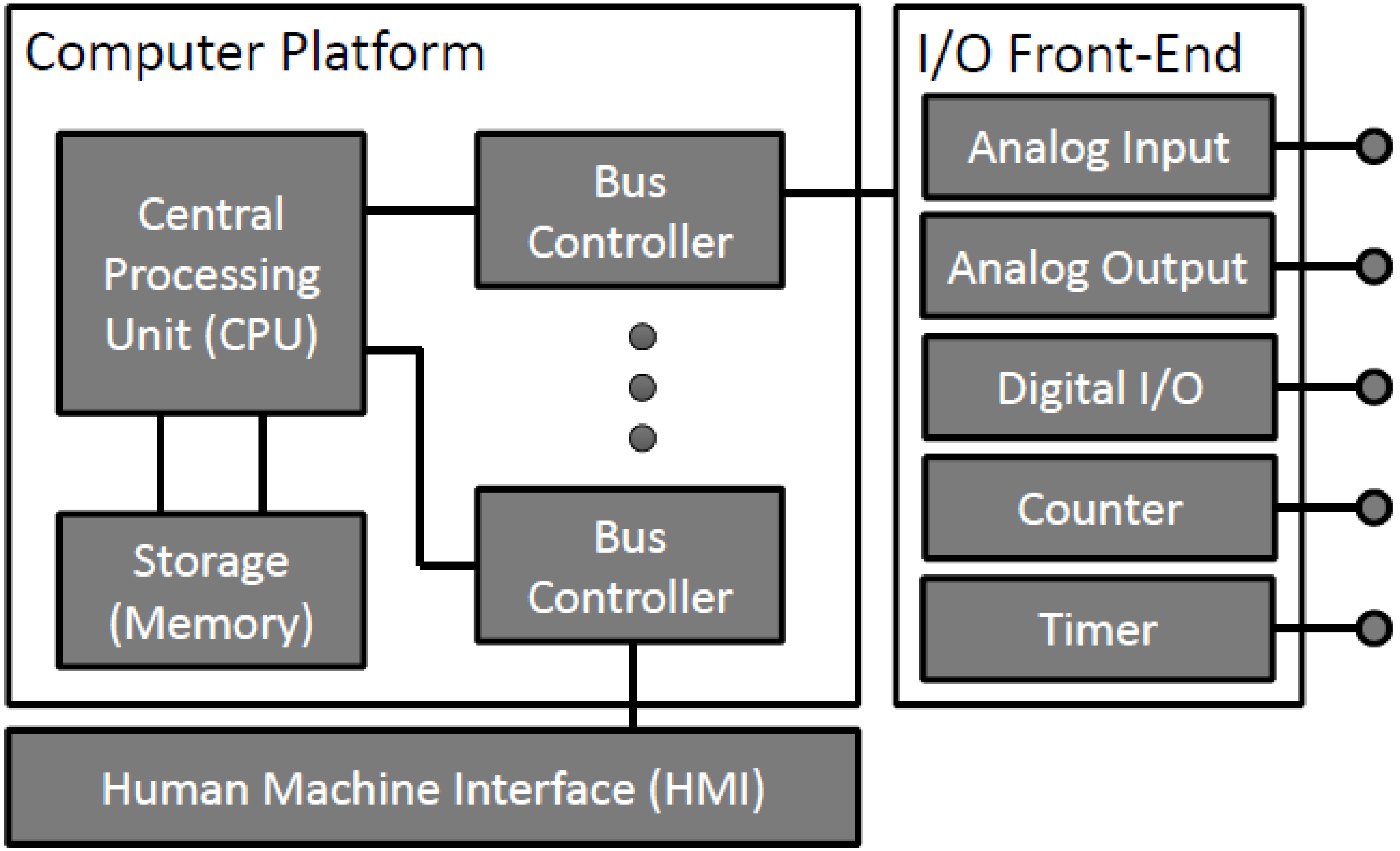
- Intelligence is required in the system.
- The complexity of a system is reduced when using one.
- The cost of the microcontroller is “less” than using discrete components to do the same job.
- A variety of sensors and actuators must be integrated in the system.
- Communication with other devices is necessary.

Appropriate Microcontroller Use

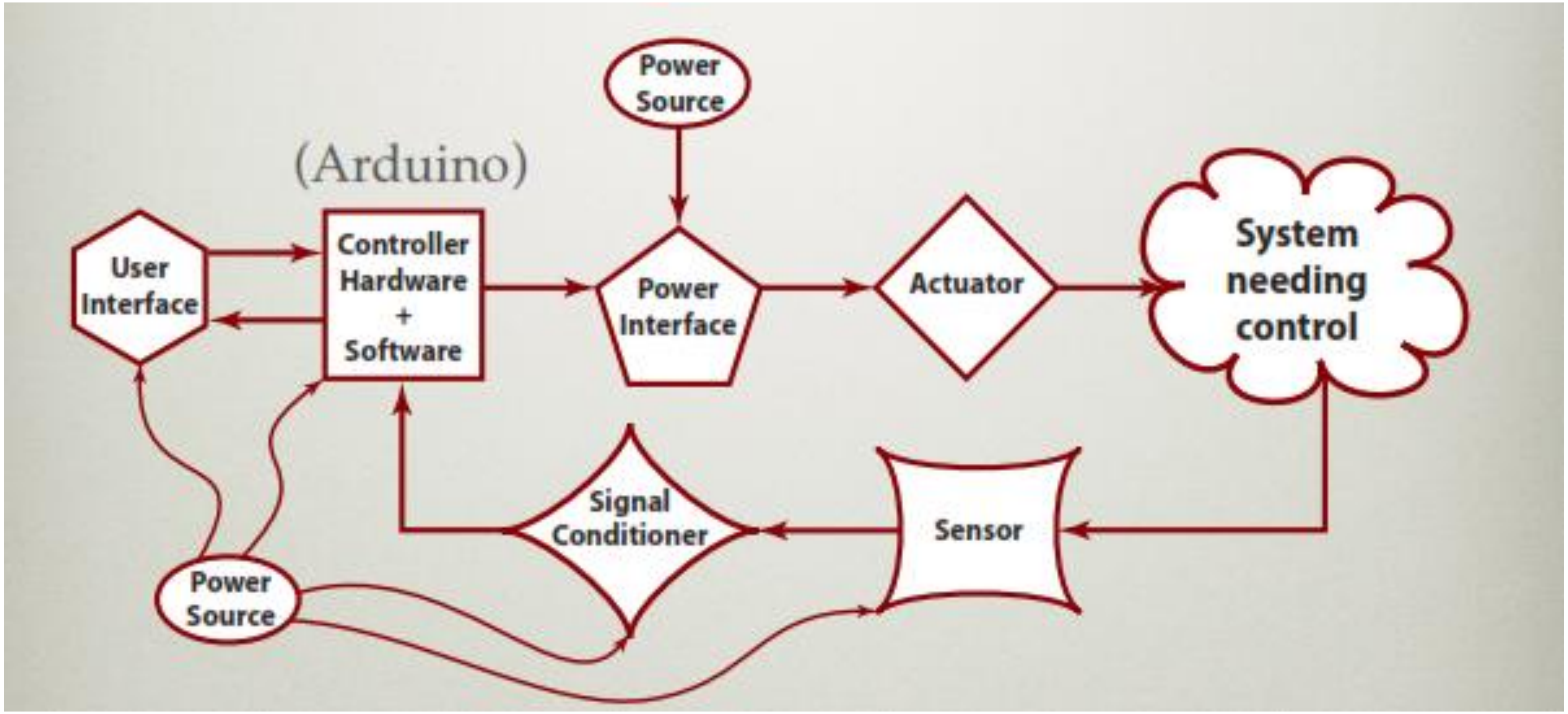
A microcontroller is **NOT** the correct tool to use when:

- The system requires little or no intelligence.
- The system can be made easier and/or cheaper using discrete-components.

Controller of Embedded System parts can be put in following diagram:



The Embedded System “Concept Map”



Systems Need Control

Systems need control could be:

- Mechanical (i.e. ……….)
- Electrical (i.e. ……….)
- Electro-Mechanical (Mechatronics)
- Biological (i.e. ……….)
- Thermodynamic (i.e. ……….)
- Chemical (i.e. ……….)
- Other Systems ?

The questions need to be asked are:

- What’s wrong with that system?
- What would we like to do with it?
- What can we do with it?

Examples Linked to Control Engineering

- **ON/OFF control:** Thermostat fan, water pump.
- **Proportional control:** Motor speed ∝ potentiometer error.
- **PID control :** Arduino PID library controlling motor or balancing robot .
- **Discrete logic/sequential control:** Traffic light, elevator simulation.
- **Data acquisition:** Serial logging of sensor values.

Systems Need Control

Robotics Applications:

- Controlling joint motors for motion.
- Reading sensor feedback (position, distance, vision).
- Executing tasks like object pickup, navigation.

- **Control Concept:** Closed-loop feedback control (PID for motor position).

Systems Need Control

Industrial Automation Applications:

- Managing conveyor belts (speed and direction).
- Controlling pumps and valves for fluid flow.
- Coordinating robotic arms for assembly lines.
- **Control Concept:** Sequential logic + real-time control**.**

Systems Need Control

Automotive Control Systems Applications:

- Engine Control Unit (ECU) → fuel injection, ignition timing.
- Anti-lock Braking System (ABS) → preventing wheel lock.
- Airbag deployment → triggered by crash sensors in milliseconds.

- **Control Concept:** Embedded real-time safety-critical control.

Systems Need Control

Home Automation Applications:

- Smart thermostats controlling HVAC systems.
- Automatic lighting systems (motion/light sensors).
- Security systems (cameras, alarms, smart locks).

- **Control Concept:** ON/OFF control + IoT connectivity.

Systems Need Control

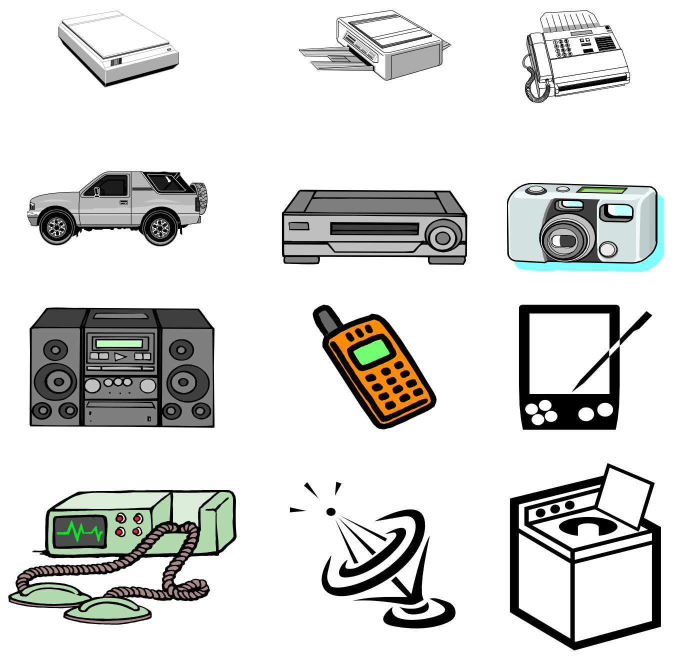
Medical Devices Applications:

- Insulin pumps regulating drug delivery.
- Pacemakers maintaining heart rhythm.
- Patient monitoring systems (vital signs, alarms).

- **Control Concept:** Precision closed-loop biomedical control.

A “short list” of embedded systems

- Anti-lock brakes
- Auto-focus cameras
- Automatic teller machines
- Automatic toll systems
- Automatic transmission
- Avionic systems
- Battery chargers
- Camcorders
- Cell phones
- Cell-phone base stations
- Cordless phones
- Cruise control
- Curbside check-in systems
- Digital cameras
- Disk drives
- Electronic card readers
- Electronic instruments
- Electronic toys/games
- Factory control
- Fax machines
- Fingerprint identifiers
- Home security systems
- Life-support systems
- Medical testing systems
- Modems
- MPEG decoders
- Network cards
- Network switches/routers
- On-board navigation
- Pagers
- Photocopiers
- Point-of-sale systems
- Portable video games
- Printers
- Satellite phones
- Scanners
- Smart ovens/dishwashers
- Speech recognizers
- Stereo systems
- Teleconferencing systems
- Televisions
- Temperature controllers
- Theft tracking systems
- TV set-top boxes
- VCR’s, DVD players
- Video game consoles
- Video phones
- Washers and dryers

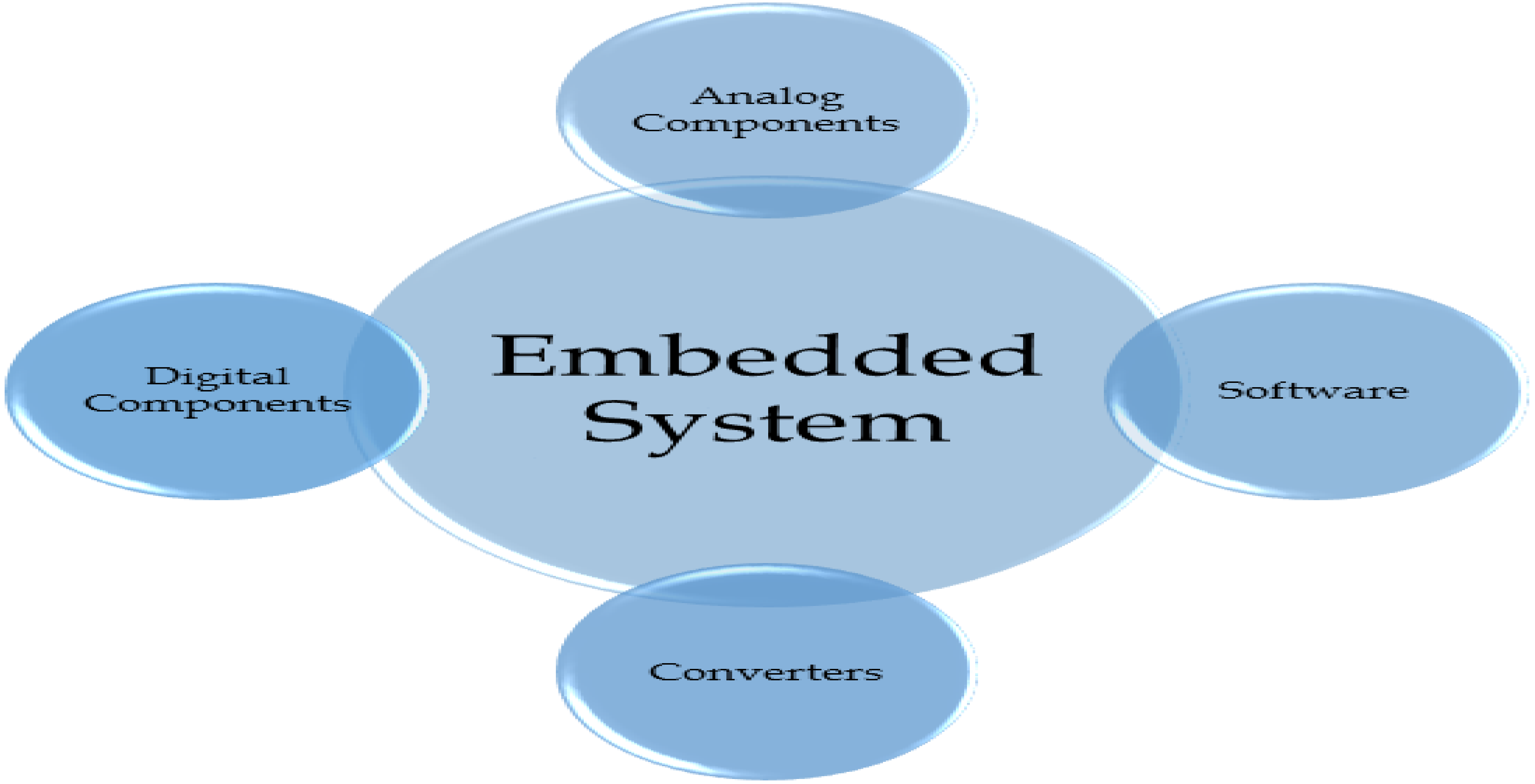


College of Electronics Engineering

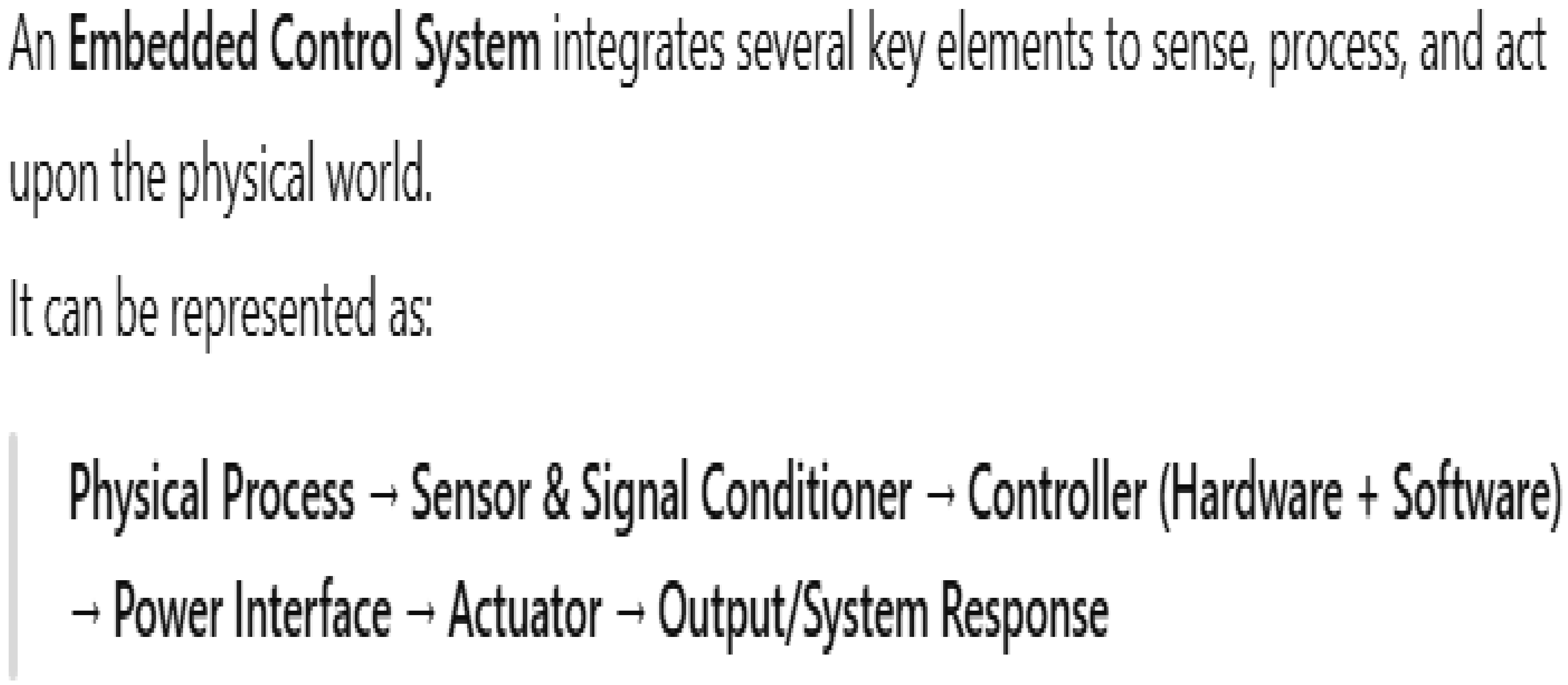
Systems and Control Engineering Department

Embedded Systems
4th Year

Major Components of an Embedded Control System 3


Analog Components
Digital Components
Embedded System
Software
Converters

1. Overview



An **Embedded Control System** integrates several key elements to sense, process, and act upon the physical world.

It can be represented as:

> Physical Process → Sensor & Signal Conditioner → Controller (Hardware + Software) → Power Interface → Actuator → Output/System Response

1. Sensor and Signal Conditioner

Type	Physical Quantity	Example Sensor
Temperature	Heat	LM35, TMP36
Position	Angle/displacement	**Potentiometer,** encoder
Pressure	Fluid/gas pressure	**Piezo sensor**
Light	Illumination	**LDR, photodiode**
Speed	Rotational speed	Hall-effect sensor

1. Sensor and Signal Conditioner

- A sensor converts a physical quantity (temperature, pressure, speed, position, light, etc.) into an electrical signal (voltage or current).
- Acts as the **input device** for the embedded system.

Signal Conditioner

- Sensors usually produce **weak or noisy analog signals**.
- A **signal conditioning circuit** modifies the signal to make it **usable** by the microcontroller.

Signal Conditioner

Main functions:

•**Amplification** → Op-amp increases signal amplitude.

•**Filtering** → Removes noise (low-pass, high-pass, band-pass).

•**Level shifting** → Matches voltage to ADC input range (e.g., 0–5 V).

•**Isolation** → Protects controller from high voltages (opto-isolators)

Power Sources

•Provide energy for **all electronic components** in the system.
•Must supply **stable and regulated voltage** to both the controller and actuators.

Common Power Sources

•**Battery** (Li-ion, NiMH, etc.) – used in portable/robotic systems.
•**DC Power Supply** (5 V, 12 V) – for laboratory/industrial systems.
•**Energy Harvesting** (solar, vibration) – low-power remote devices

Power Sources

Power Management:

- **Voltage Regulators:**
- **Converters:**
  - DC–DC converters (buck/boost) for efficiency.
  - **AC–DC rectifiers for mains input**.

Power Interface

Power Management:

•**Voltage Regulators:**

•**Converters:**

- DC–DC converters (buck/boost) for efficiency.
- AC–DC rectifiers for mains input.

Protection: fuses, transient suppressors, reverse polarity protection.

Power Interface

The bridge between the low-power controller and high-power actuators.Ensures the microcontroller can safely control larger loads.

Types of Power Interfaces

1. **Transistor or MOSFET Drivers** – switch higher current loads.
2. **Relay Circuits** – electromechanical switches for AC or DC loads.
3. **Opto-isolators** – provide electrical isolation between control and power sections.
4. **Motor Drivers / H-bridges** – enable direction and speed control of DC motors

Power Interface

The bridge between the low-power controller and high-power actuators.Ensures the microcontroller can safely control larger loads.

Types of Power Interfaces

1. **Transistor or MOSFET Drivers** – switch higher current loads.
2. **Relay Circuits** – electromechanical switches for AC or DC loads.
3. **Opto-isolators** – provide electrical isolation between control and power sections.
4. **Motor Drivers / H-bridges** – enable direction and speed control of DC motors

Actuators

Actuators convert electrical control signals into mechanical motion or physical action.

They represent the output stage of the embedded control loop.

Actuators

Type	Function	Example
Electromechanical	Rotational or linear motion	DC motor, stepper, servo
Electromagnetic	Switching or movement	Relay, solenoid
Thermal	Heat generation	Heater element
Hydraulic/Pneumatic	Force/pressure control	Valves, cylinders

User Interface (UI)

Allows the **operator or user** to interact with the embedded system.

Types

- **Input Devices:** push buttons, switches, keypads, rotary knobs.
- **Output Displays:** LEDs, LCDs, seven-segment displays, touch screens.
- **Communication Interfaces:** serial monitor (UART), USB, Bluetooth, Wi-Fi, CAN.

User Interface (UI)

Role in Control

•Provides manual overrides or adjustments (e.g., setpoint entry).

•Displays measured variables, system status, and faults.

Control link: Enables real-time monitoring and manual tuning of controller parameters

Controller Hardware

The **physical computing unit** executing control algorithms. Usually a **microcontroller**, **microprocessor**, or **digital signal processor (DSP)**.

Controller Hardware

Key Components

•**CPU:** performs calculations and logic operations.

•**Memory:**

- Flash/ROM – stores program (firmware).
- RAM – stores temporary data.

•**Peripherals:** timers, ADC, DAC, serial interfaces (UART(Universal Asynchronous Receiver/Transmitter), I^{2}C, SPI,Serial Peripheral Interface, CAN, Controller Area Network).

•**Clock:** determines speed of execution.

•**Reset & Power Circuits:** ensure stable startup

Controller Software

The **firmware or program code** running inside the controller.
Defines how the system responds to inputs and generates outputs.

Controller Software

Software Layers

1. **Application Code:** control algorithms (ON/OFF, PID, etc.).
2. **Drivers:** interface with peripherals (ADC, PWM, UART).
3. **Real-Time Scheduler or Loop:** manages timing and task execution.
4. **Interrupt Service Routines (ISRs):** handle immediate events (e.g., sensor triggers).

Controller Software

Software Development Flow

1. Write code in **C/C++** (or Arduino language).
2. Compile → Generate machine code (hex file).
3. Upload to microcontroller via programmer/USB.
4. Test, debug, and tune parameters.

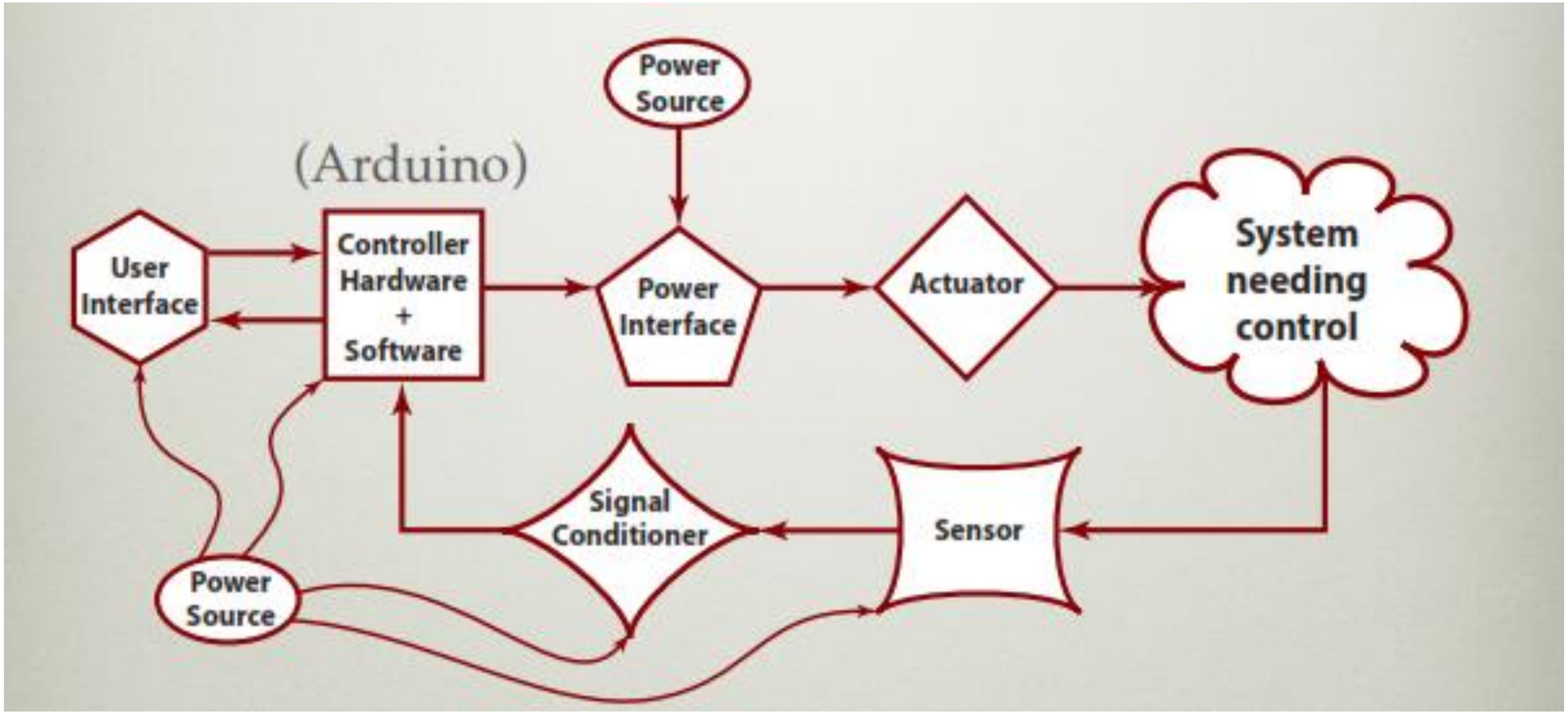
College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Embedded System (E.S)
4

The Embedded System “Concept Map”



Microprocessor

•A **microprocessor** is the **CPU (central processing unit) on a single chip**.
•It requires **external memory (RAM, ROM)** and **peripherals** to build a complete system.
•Used in **general-purpose computing systems** (like PCs, laptops).
Example: Intel i5, AMD Ryzen, ARM Cortex-A, Pentium, etc.

Microcontroller

- A **microcontroller** is a **complete computer system on a single chip**.
- It contains:
  - CPU
  - Memory (RAM, ROM/Flash)
  - I/O ports (digital & analog)
  - Timers/Counters
  - Communication interfaces (UART, I2C, SPI)
- Designed for **specific control applications** (automation, robots, instruments).

Example: ATmega328 (Arduino), PIC16F877A, STM32, 8051.

Comparison: Microprocessor vs. Microcontroller

Feature	Microprocessor (µP)	Microcontroller (µC)
Definition	CPU on a chip	CPU + memory + peripherals on a chip
Memory	External (RAM/ROM) required	Internal (RAM, ROM/Flash)
I/O Ports	Mostly external	Built-in GPIO, ADC, timers
Application Type	General-purpose computing	Dedicated / specific task
System Cost	Higher (needs more components)	Lower (integrated design)
Speed	Typically higher clock rates	Moderate, but optimized for control
Power Consumption	High	Low (battery-friendly)
Programming	Complex OS/software	Simple C or assembly
Examples	Intel i7, AMD Ryzen, ARM Cortex-A	ATmega328, PIC, STM32, 8051
Use in Control	Rare (for large systems)	Common (real-time embedded control)

Advantages of Microcontrollers

1. **Compact and integrated** → CPU, memory, and I/O on one chip.
2. **Low cost** → fewer external components.
3. **Low power consumption** → ideal for portable and embedded applications.
4. **Easy to program** → C/C++ using tools like Arduino IDE.
5. **Fast response** → suitable for real-time control tasks.
6. **Reliable** → fewer parts, lower failure rate.
7. **Flexible I/O** → supports analog/digital signals, PWM, interrupts, communication.

Disadvantages of Microcontrollers

1. **Limited memory and processing power** (compared to microprocessors).
2. **Application-specific** → cannot perform general computing tasks.
3. **Difficult to upgrade hardware** (chip-specific).
4. **Restricted multitasking** (unless RTOS or multi-core MCU used)

Key Characteristics of Microcontrollers

- **Dedicated task operation:** designed for one main function.
- **On-chip integration:** CPU + memory + peripherals.
- **Real-time operation:** must respond immediately to sensor input.
- **Embedded in larger systems:** not stand-alone like PCs.
- **Programmable control logic:** runs firmware written in C/C++.
- **Low power & cost-efficient:** used in millions of consumer and industrial devices.

Example architecture:

- CPU core (e.g., 8-bit, 16-bit, 32-bit)
- Flash memory for program
- RAM for data
- GPIO pins for sensors/actuators
- Timers, ADC, UART, I^{2}C, SPI modules

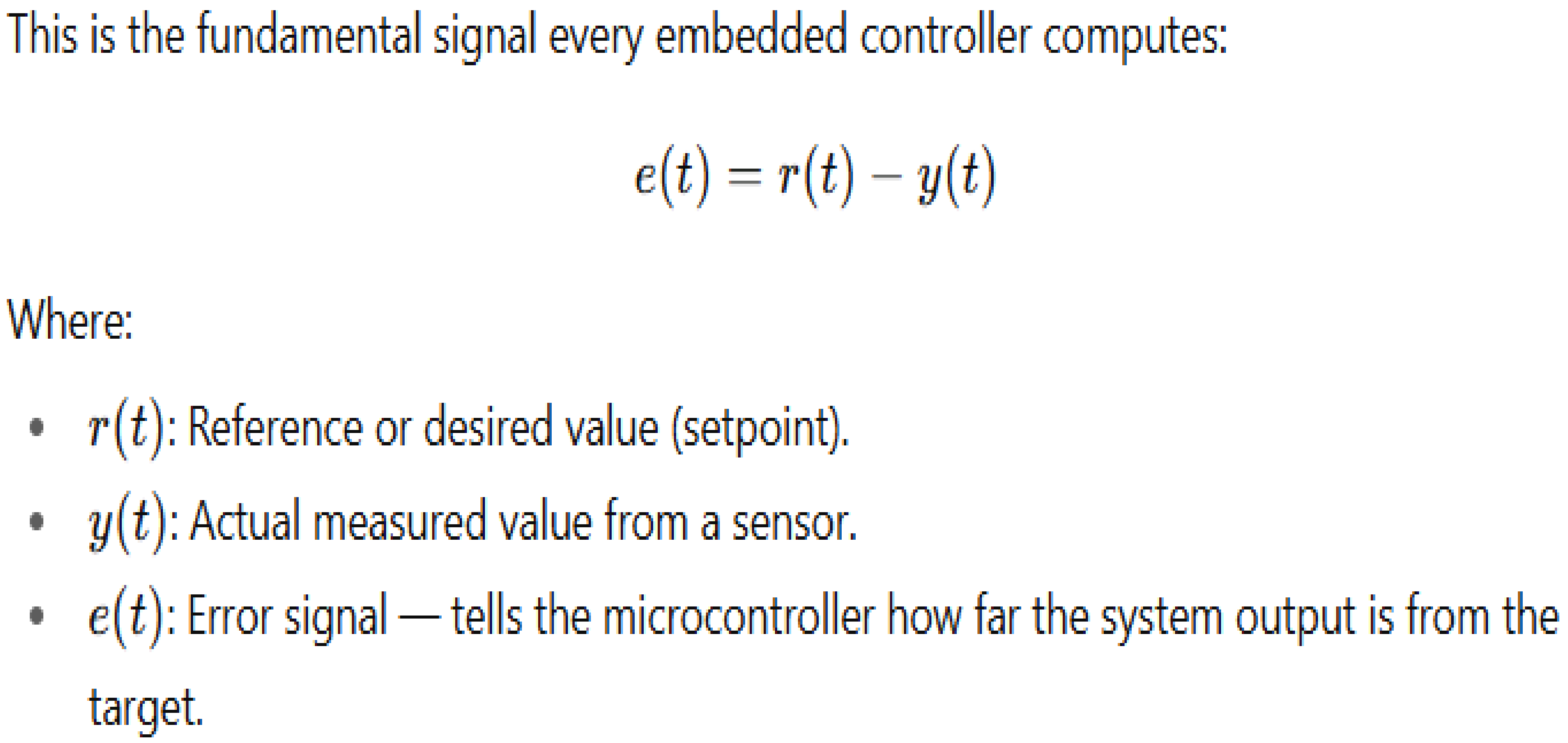
Control Engineering Context

- Microcontrollers are the heart of modern control systems:
- Measure signals from sensors (temperature, pressure, position).
- Compute control law (ON/OFF, PID, etc.).
- Drive actuators (motors, valves, relays).

Summary Slide:

- Microcontroller = compact embedded computer for control tasks.
- Microprocessor = powerful CPU for general computing.
- Control systems use microcontrollers for real-time embedded feedback control.

Error Equation



This is the fundamental signal every embedded controller computes:

$$e(t) = r(t) - y(t)$$

Where:

- $r(t)$: Reference or desired value (setpoint).
- $y(t)$: Actual measured value from a sensor.
- $e(t)$: Error signal — tells the microcontroller how far the system output is from the target.

Microcontroller role:

- Continuously reads sensor data.
- Calculates $e(t)$.
- Uses it to adjust actuators (motor, heater, etc.).

Discrete-Time Control Law (as implemented in software)

Since microcontrollers use **digital sampling**, the control law is expressed in discrete time:

$$u[k] = K_p\, e[k] + K_i \sum_{i=0}^{k} e[i] T_s + K_d \frac{e[k] - e[k-1]}{T_s}$$

Where:

- $u[k]$: Control signal at sample k (output to actuator).
- $e[k]$: Error at sample k.
- T_s: Sampling period (time between controller updates).
- K_p, K_i, K_d: Proportional, Integral, and Derivative gains.

Microcontroller role:

- Executes this PID equation in a timed loop (every T_s).
- Sends PWM or analog signal to the actuator.

Sampling Period Relation

For stable and responsive control:

$$T_s \leq \frac{1}{10 f_c}$$

Where:

- f_c: bandwidth or cutoff frequency of the control loop.
- T_s: sampling period (microcontroller loop delay).

- The controller (microcontroller) must sample at least **10× faster** than the system's dominant dynamic frequency for accurate control.

First-Order System Response (typical controlled plant)

Used to model many physical systems (temperature, DC motor, etc.):

$$\tau \frac{dy(t)}{dt} + y(t) = K\, u(t)$$

Where:

- τ: time constant (how quickly system reacts).
- K: system gain.
- $u(t)$: control input (from microcontroller).
- $y(t)$: system output (sensor reading).

Microcontroller role:

- Applies $u(t)$ (e.g., PWM duty cycle).
- Measures $y(t)$.
- Updates control action according to error.

Summary :

Equation	Purpose	Implemented by
$e(t) = r(t) - y(t)$	Compute system error	Sensor + MCU
$u[k] = K_p e[k] + K_i \sum e[i] T_s + K_d \frac{e[k]-e[k-1]}{T_s}$	Control algorithm	MCU firmware
$T_s \leq \frac{1}{10 f_c}$	Sampling condition	MCU timer interrupt
$\tau \frac{dy}{dt} + y = Ku$	System model	Physical process

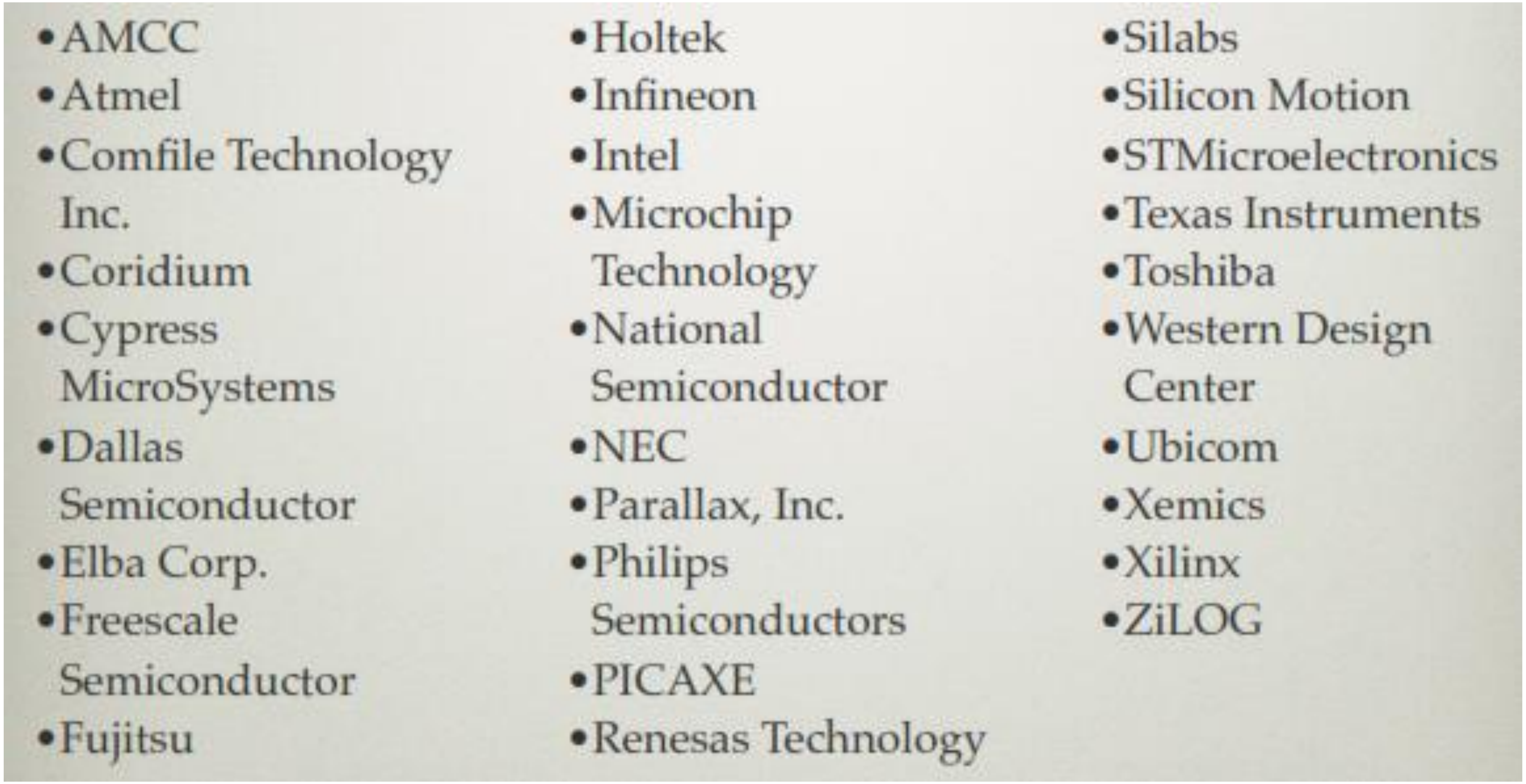
College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Microcontrollers MFGRs

Microcontroller MFGRs



- AMCC
- Atmel
- Comfile Technology Inc.
- Coridium
- Cypress MicroSystems
- Dallas Semiconductor
- Elba Corp.
- Freescale Semiconductor
- Fujitsu
- Holtek
- Infineon
- Intel
- Microchip Technology
- National Semiconductor
- NEC
- Parallax, Inc.
- Philips Semiconductors
- PICAXE
- Renesas Technology
- Silabs
- Silicon Motion
- STMicroelectronics
- Texas Instruments
- Toshiba
- Western Design Center
- Ubicom
- Xemics
- Xilinx
- ZiLOG

TERMINOLOGIES

A. Open-Source Software (OSS): Type of computer software in which source code is released under a license in which the copyright holder grants users the rights to use, study, change, and distribute the software to anyone and for any purpose. Examples: Linux, Android, Firefox etc.

B. Open-Source hardware (OSH): Physical artifacts of technology designed and offered by the open-design movement. Information about the hardware is easily discerned so that others can make it – coupling it closely to the maker movement. Hardware design (i.e., mechanical drawings, schematics, bills of material, PCB layout data, HDL source code and integrated circuit layout data), in addition to the software that drives the hardware, are all released under free/libre terms. Examples: RepRap (3D printing), Arduino etc.

TERMINOLOGIES

C. Microcontroller: An integrated circuit (IC) device used for controlling other portions of an electronic system, usually via a microprocessor unit (MPU), memory, and some peripherals. These devices are optimized for embedded applications that require both processing functionality and agile, responsive interaction with digital, analog, or electromechanical components. A typical microcontroller includes a processor (CPU), memory and input/output (I/O) peripherals on a single chip. MCUs are found in vehicles, robots, office machines, medical devices, mobile radio transceivers, vending machines and home appliances, among other devices. They are essentially simple miniature personal computers (PCs) designed to control small features of a larger component, without a complex front-end operating system (OS).

What is the Arduino?

- Arduino is an open-source electronics platform based on easy-to-use hardware and software.

- **Arduino boards** are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online

- You can tell **your board** what to do by **sending a set of instructions** to the microcontroller on the board.
- To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

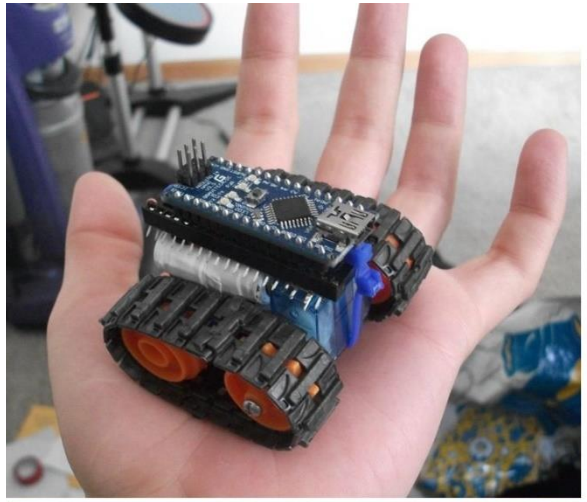
Why the Arduino?

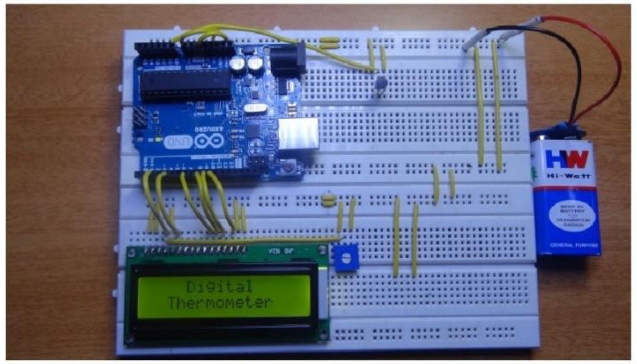
- Improvement over other micro-controllers for hobby usage.
- Inexpensive
- IDE on multiple platforms (Mac, PC, Linux)
- Simple, clear programming environment: “C”
- **Open source** and extensible software/hardware.
- Arduino Shields (elements that can be plugged onto a board to give it extra features).
- Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments.
- **Arduino board** started changing to **adapt** to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and **embedded environments.**

Arduino Shields

Shield Name	Function	Example Application
Motor Shield	Controls DC motors and servos	Robotics, automation
Ethernet Shield	Adds wired network (LAN) capability	IoT data logging
Wi-Fi Shield	Connects to Wi-Fi networks	Smart home, remote control
Bluetooth Shield	Enables Bluetooth communication	Wireless sensors
GPS Shield	Receives geographic coordinates	Vehicle tracking
LCD/Touch Shield	Provides display and user input	Interface panels
Relay Shield	Controls high-power devices	Home automation
Sensor Shield	Simplifies connection of multiple sensors	Environmental monitoring
GSM/GPRS Shield	Mobile communication (SIM-based)	Remote alert systems

Arduino Applications





https://playground.arduino.cc/Projects/Ideas

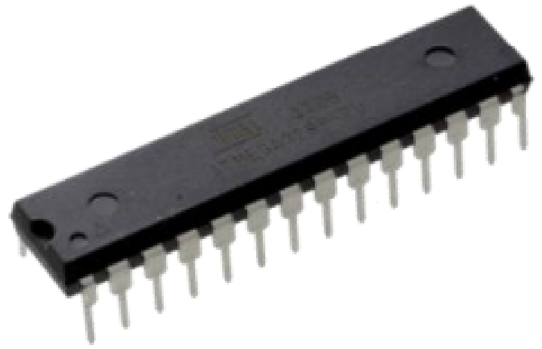
Arduino UNO

- The UNO is the best board to get started with electronics and coding.
- Arduino Uno is a microcontroller board based on the **ATmega328P.**
- It has 14 digital input/output pins (of which 6 can be used as PWM outputs).
- 6 analog inputs.
- 6 MHZ quartz crystal
- USB connection.
- Power jack
- ICSP header
- Reset button.
- It contains everything needed to support the microcontroller.
- Simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
- You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

ATmega328P

ATmega328P is 8-bit microcontroller from Atmel (now owned by Microchip Technology). It's one of the most popular microcontrollers, especially known for powering the Arduino Uno and other Arduino-compatible boards.

- 28 pins DIP (dual in-line package)
- **CPU:** 8-bit AVR RISC-based CPU
- **Clock Speed:** Up to 20 MHz
- **Flash Memory:** 32 KB (used to store program code)
- **SRAM:** 2 KB (for variables and temporary data during execution)
- **EEPROM:** 1 KB (for persistent data storage, even after power-off)
- **Operating Voltage:** 1.8V - 5.5V
- **I/O Pins:** 23 I/O pins, with 14 digital and 6 analog input pins (ADC-enabled)
- **Communication Interfaces:** UART, SPI, and I2C for connecting to sensors and other peripherals.



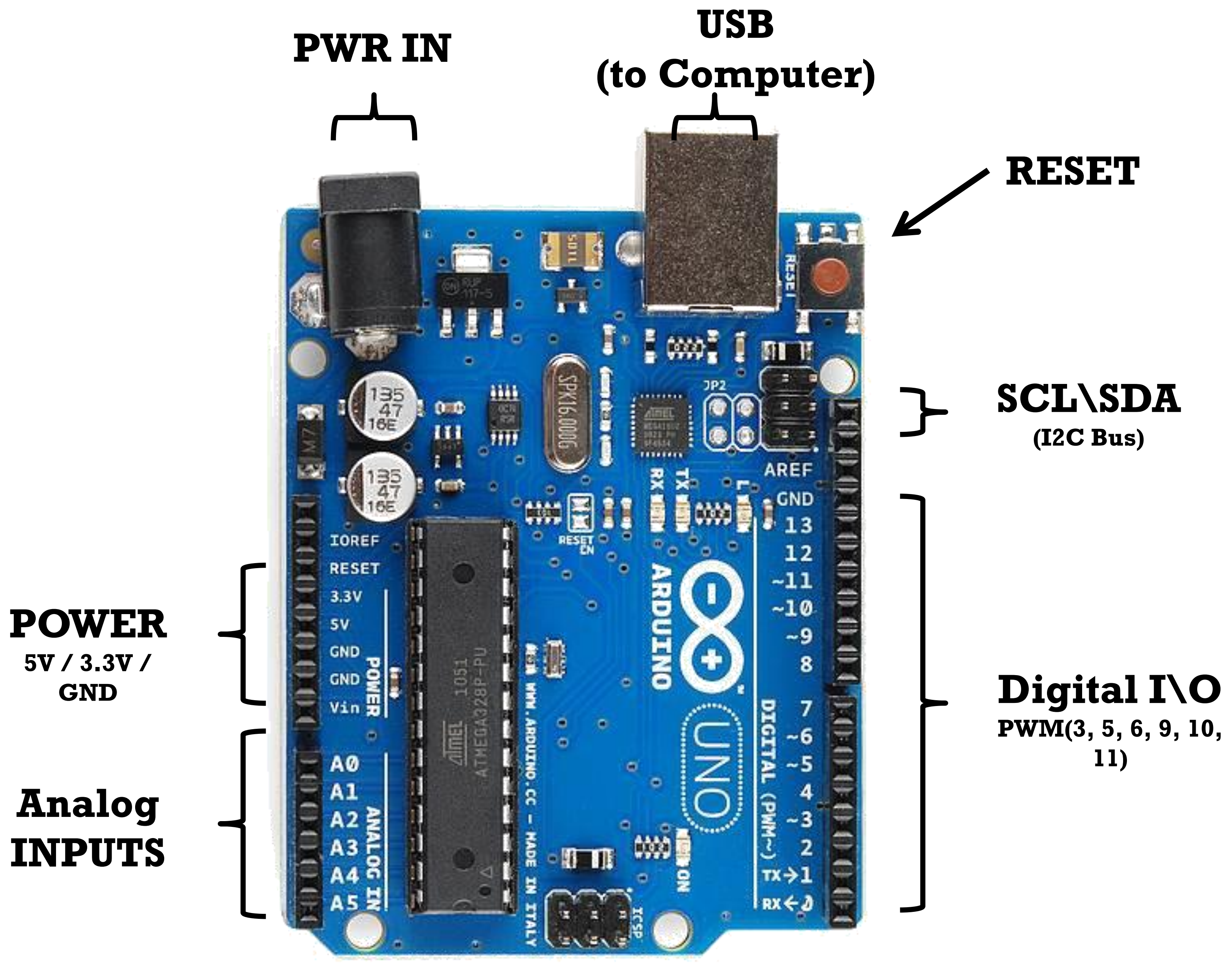
ATmega328P

ATmega328P is 8-bit microcontroller from Atmel (now owned by Microchip Technology). It's one of the most popular microcontrollers, especially known for powering the Arduino Uno and other Arduino-compatible boards.

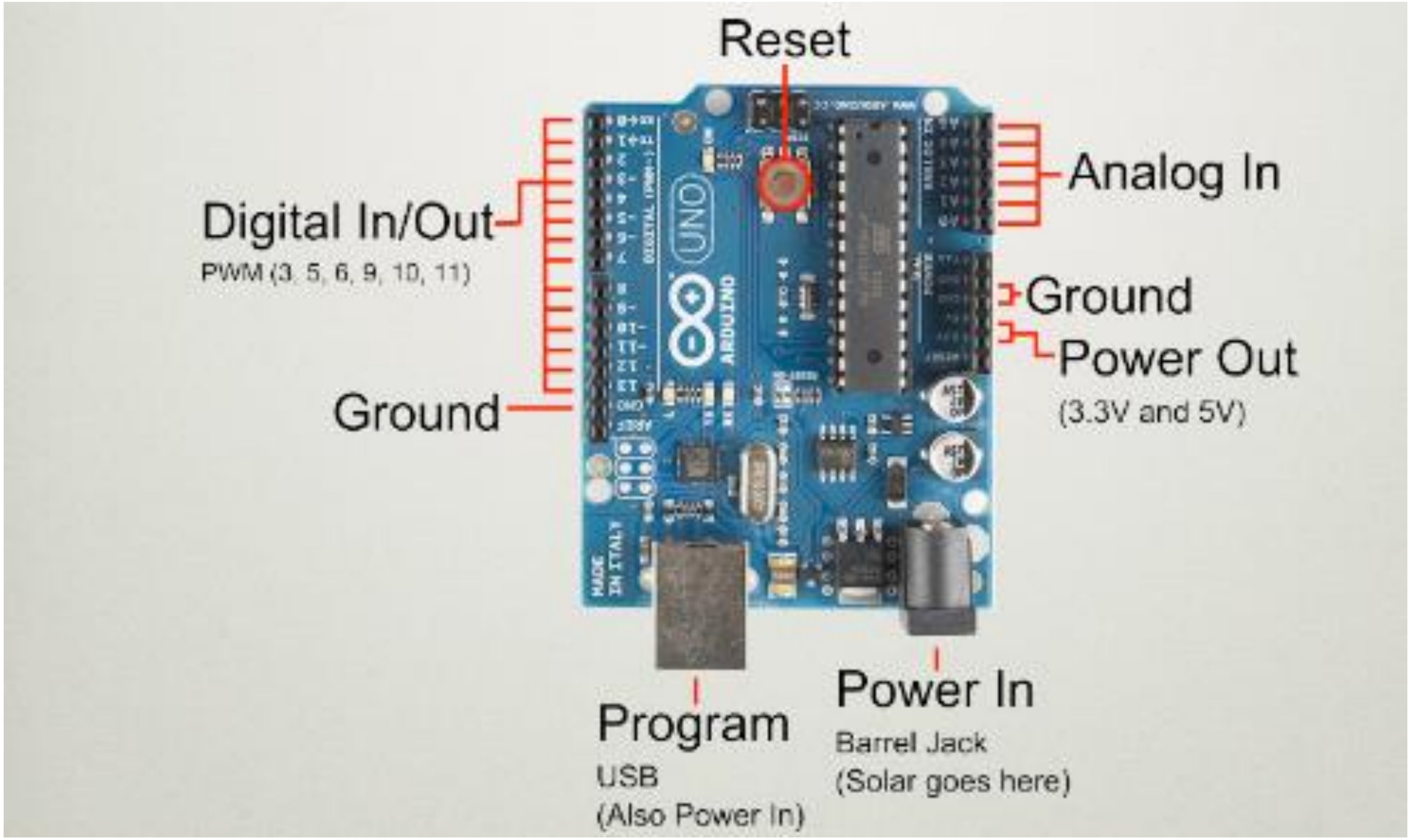
- 28 pins DIP (dual in-line package)
- **CPU:** 8-bit AVR RISC-based CPU
- **Clock Speed:** Up to 20 MHz
- **Flash Memory:** 32 KB (used to store program code)
- **SRAM:** 2 KB (for variables and temporary data during execution)
- **EEPROM:** 1 KB (for persistent data storage, even after power-off)
- **Operating Voltage:** 1.8V - 5.5V
- **I/O Pins:** 23 I/O pins, with 14 digital and 6 analog input pins (ADC-enabled).

ATmega328P

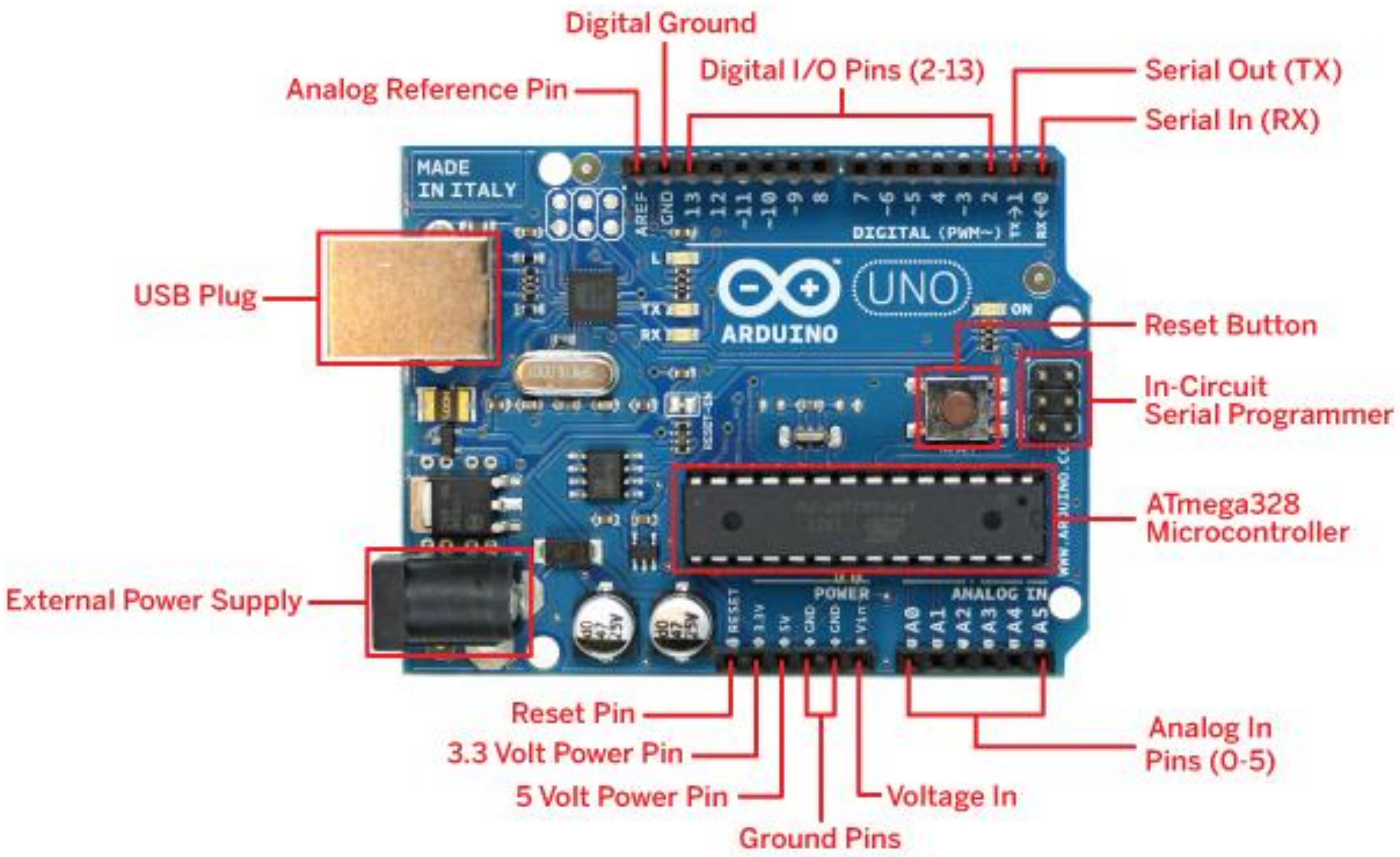
- **Timers:** Three timers (two 8-bit and one 16-bit) for timing and event management.
- **PWM:** Six PWM channels for tasks like dimming LEDs and controlling motor speeds.
- **Communication Interfaces:** UART, SPI, and I2C for connecting to sensors and other peripherals.
- **ADC:** 10-bit ADC with 6 channels for reading analog inputs.
- **Power Efficiency:** The "P" in ATmega328P stands for "PicoPower“, which means it has ultra-low power consumption modes, ideal for battery-powered applications.
- **RISC Architecture:** It uses a Reduced Instruction Set Computing (RISC) architecture, which means that the instructions are simple and fast.
- **Limitations:** Limited processing power and memory and hence not suitable for high-performance applications, low clock speed.


PWR IN
USB
(to Computer)
RESET
SCL\SDA
(I2C Bus)
POWER
5V / 3.3V /
GND
Digital I\O
PWM(3, 5, 6, 9, 10,
11)
Analog
INPUTS

Arduino UNO


Reset
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
Ground
Analog In
Ground
Power Out
(3.3V and 5V)
Program
USB
(Also Power In)
Power In
Barrel Jack
(Solar goes here)

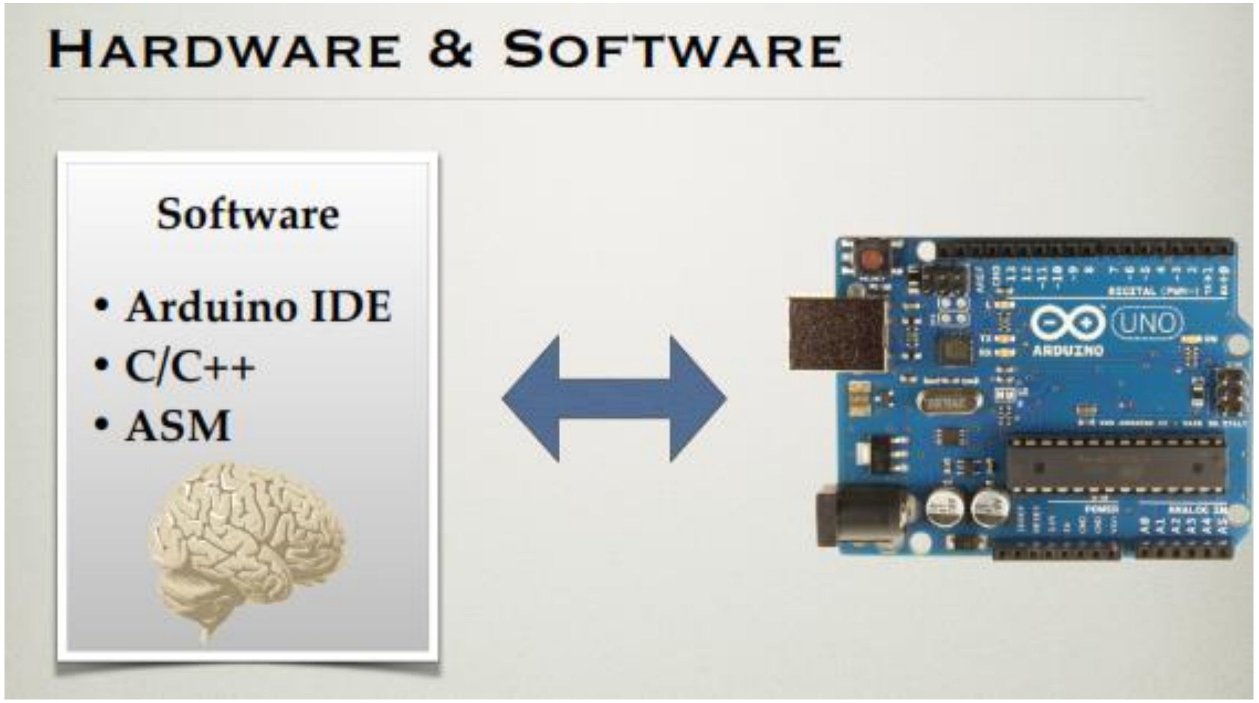
Arduino UNO


Digital Ground
Analog Reference Pin
Digital I/O Pins (2-13)
Serial Out (TX)
Serial In (RX)
USB Plug
Reset Button
In-Circuit Serial Programmer
ATmega328 Microcontroller
External Power Supply
Reset Pin
3.3 Volt Power Pin
5 Volt Power Pin
Ground Pins
Voltage In
Analog In Pins (0-5)
MADE IN ITALY
ARDUINO
UNO
DIGITAL (PWM~)
POWER
ANALOG IN

GPIO

- General Purpose Input/ Output
- Can be used as digital input or output.
- Each pin uniquely assignable.
- "Analog" (PWM) output on digital pins 3,5,6,9,10,11
- Analog pins 0-5 can also be used as GPIO.

Hardware & Software

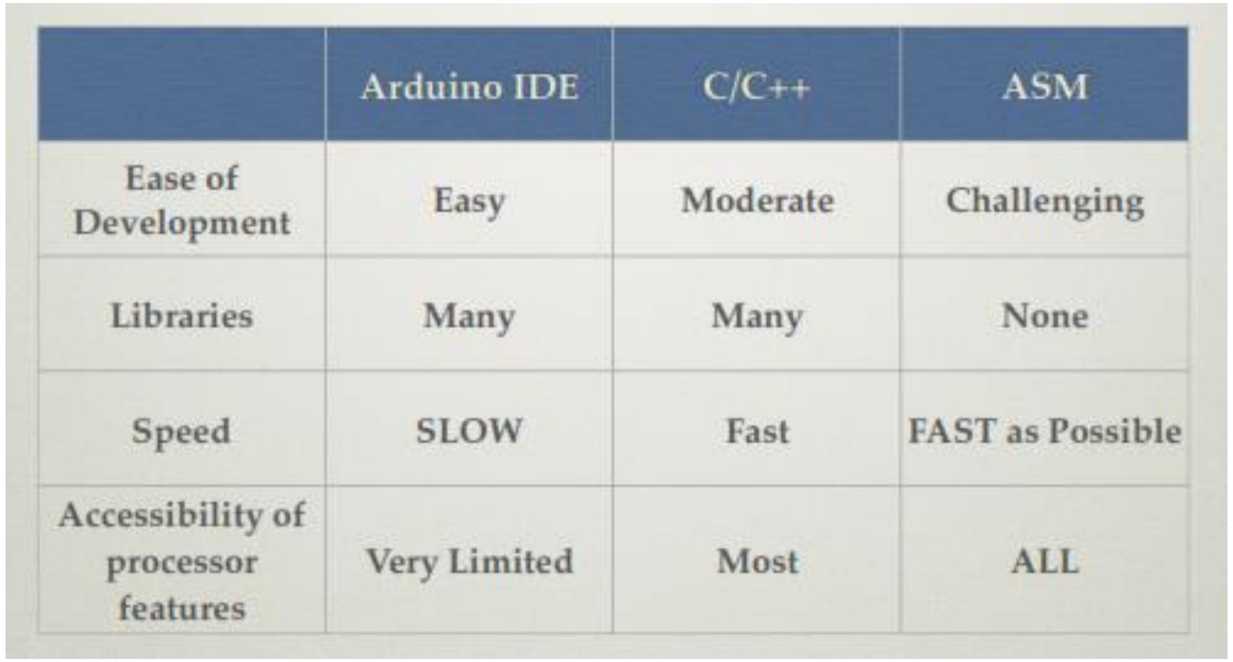

HARDWARE & SOFTWARE

Software

- Arduino IDE
- C/C++
- ASM

UNO
ARDUINO

Software Comparison



	Arduino IDE	C/C++	ASM
Ease of Development	Easy	Moderate	Challenging
Libraries	Many	Many	None
Speed	SLOW	Fast	FAST as Possible
Accessibility of processor features	Very Limited	Most	ALL

Software: Arduino IDE

- Free software
- Download from
  - http://arduino.cc/en/Main/Software
- Available for Mac, Windows, Linux

References:

1. https://www.arduino.cc/en/Tutorial/HomePage
2. Arduino Cookbook, Michael Margolis, O'Reilly Media (2011)
3. Getting Started with Arduino, Massimo Banzi, O'Reilly Media (2009)
4. Programming Arduino: Getting Started with Sketches by Simon Monk

Software:

Arduino IDE (Integrated Development Environment)

Hardware:

1. Arduino UNO
2. Basic electronic components, sensors

College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Embedded Software

Embedded Software

- Embedded Software is the software that controls an embedded system.
- All embedded systems need some software for their functioning.
- Embedded software or program is loaded in the microcontroller which then takes care of all the operations that are running.
- For developing this software, a number of different tools are needed.
- These **tools** include **editor**, **compiler**, **assembler**, **debugger, etc.**

1. Editor

- The first tool for Embedded Systems Software Development Tools is **text editor**.
- This is where **the code should be written** for the embedded system.
- The code is written in some programming language. Most commonly used language is C, C++ or IDE.
- The code written in editor is also referred to source code.

2. Compiler

- A compiler is used after finishing the editing part and made a source code.
- The function is to convert the source code in to object code.
- Object code is understandable by computer as it in low level programming language.
- It used to convert a high level language code in to low level programming language.

3. Assembler

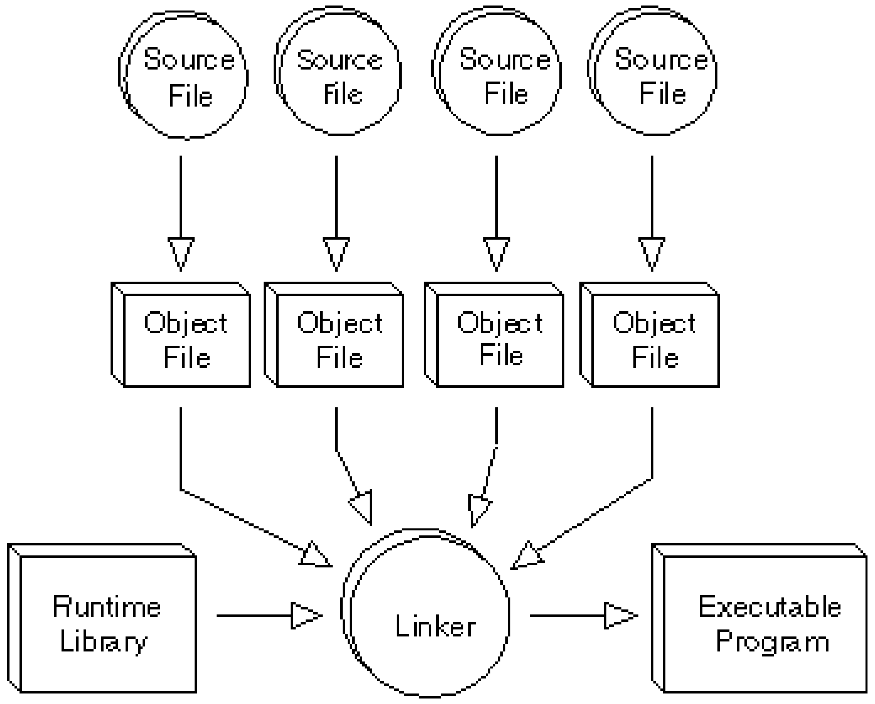
- The function of an assembler is to convert a code written in assembly language into machine language.
- All the mnemonics and data is converted in to opcodes and bits by an assembler.
- Computer understands binary and it works on 0 or 1, so it is important to convert the code into machine language.

4. Debugger

- It is important to test whether the code you have written is free from errors or not. So, a debugger is used for this testing.
- Debugger goes through the whole code and tests it for errors and bugs.
- It tests the code for different types of errors. For example a run time error or a syntax error and notifies you wherever it occurs
- The line number or location of error is shown by debugger so it is easy to go ahead and modify it.
- So from the function, it can show how important tool a debugger is in the list of Embedded Systems Software Development Tools.

5. Linker

- A linker is a computer program that combines one or more object code files and library files together in to executable program.
- It is very common practice to write larger programs in to small parts and modules to make job easy and to use libraries in your program.
- All these parts must be combined into a single file for execution, so this function requires a linker.



6. Libraries

- A library is a pre written program that is ready to use and provides specific functionality.
- For Embedded Systems Software Development Tools, libraries are very important and convenient.
- Library is a file written in C or C++ and can be used by different programs and users.
- For example, Arduino microcontroller comes with a number of different libraries that you can download and use while developing your software.
- For instance, controlling LED or reading sensor like an encoder can be done with a library.

7. Simulator

- A simulator helps to show how the code will work in real time.
- It can show how sensors are interacting, the input from sensors can be shown, and it can show how the components are working and how changing certain values can change parameters.

Integrated Development Environment (IDE)

- An Integrated Development Environment is software that contains all the necessary tools required for embedded software development.
- For creating software for your embedded system, you need all the mentioned ***tools***.
- An **IDE** normally consists of a code editor, compiler and a debugger. Also it provide user developer.
- Depending on what kind of microcontroller you are using, you can choose from many different software applications.

Example of Embedded Systems Software Development Tools

- MPLAB
- Arduino Software
- Keil
- MATLAB
- LabVIEW
- Pspice
- Proteus
- Visual Studio
- EasyEDA
- Altium

College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Analog Inputs

- An important aspect to most embedded systems involves one or more **input signals**.
- Most naturally occurring signals are analog, meaning the voltage level can be any value within some interval $V_{min} \leq V_A \leq V_{max}$

- In many systems, different actions and decisions are made based on the value of V_A.

- For example, consider the following list of **analog sensor** devices.
  - **Potentiometer** - a mechanical knob is used to adjust the resistance between terminals;
  - **Thermistor**- a device that changes resistance based on temperature;

- **Accelerometer-** a device that measures the acceleration of gravity in three dimensions and out- puts an associated analog value for each dimension

- **Ambient light sensor**-a device that measure the environmental ambient light intensity and outputs an associated analog value (e.g., many laptop computers will adjust the backlight level based on the surrounding environment light level);

- **Microphone**- a device that converts vibrations (i.e., acoustic-waves) into analog levels.

- In order for a microcontroller to operate on the sensor outputs, an **Analog-to-Digital Conversion (ADC)** circuit must be used,

Parallel ADC

- Uses typical values for V_{min} and V_{max} where the minimum voltage is simply set to ground and V_{max} is controlled externally via the user-defined reference voltage V_{ref}.

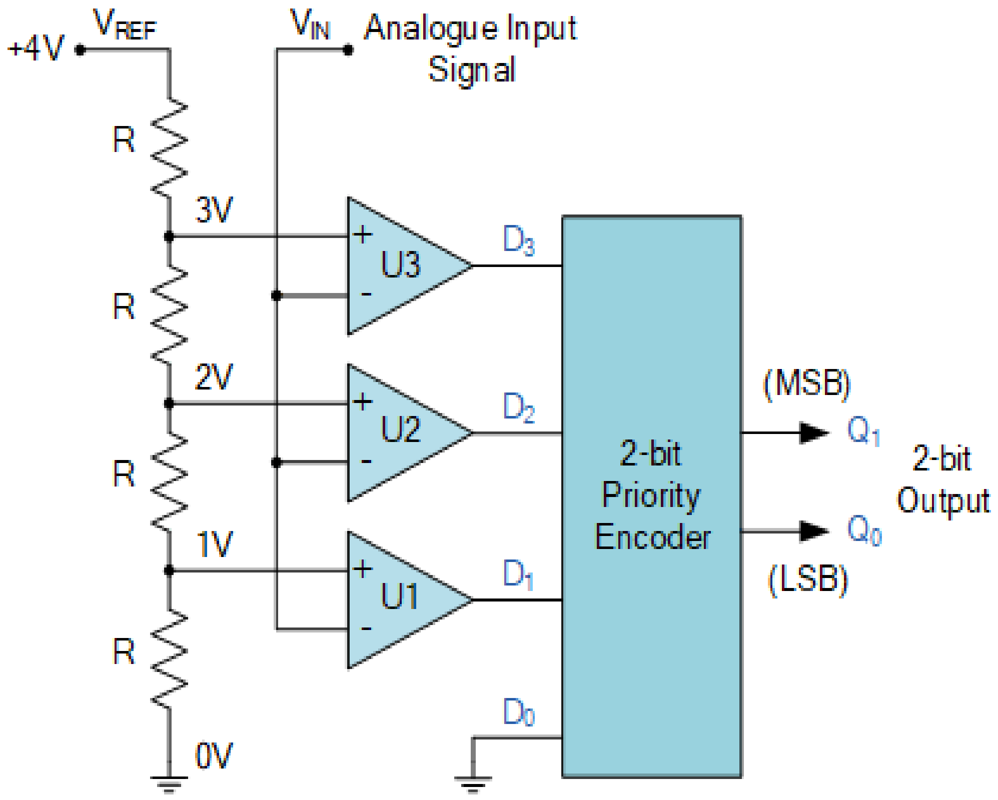
- The resistor network on the left-hand side is designed such that :
  - $V_{ref} - V_2 = V_2 - V_1 = V_1 - V_0 = V_0$

- which implies there is uniform spacing between the inverting inputs on all of the comparators.

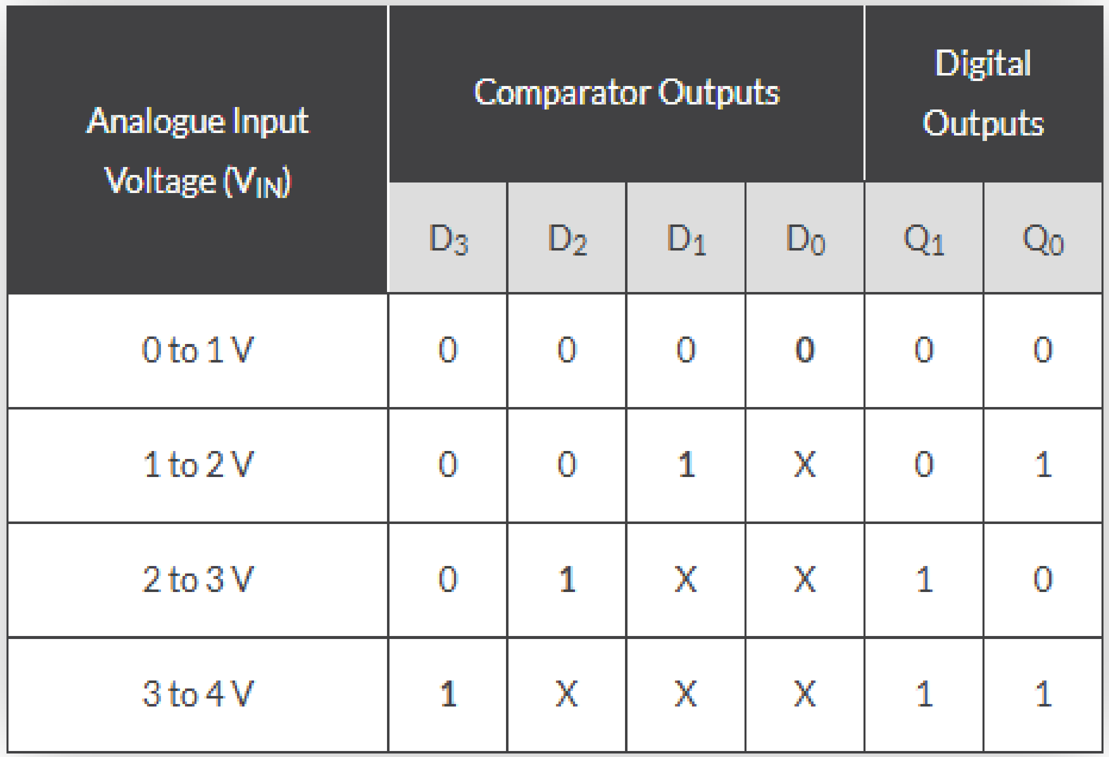
- The analog voltage V_A is first converted into a **three-bit** code word defined by (c_2, c_1, c_0), where each bit is the output of a comparator

A two-bit parallel ADC.

2-bit Analogue to Digital Converter Circuit



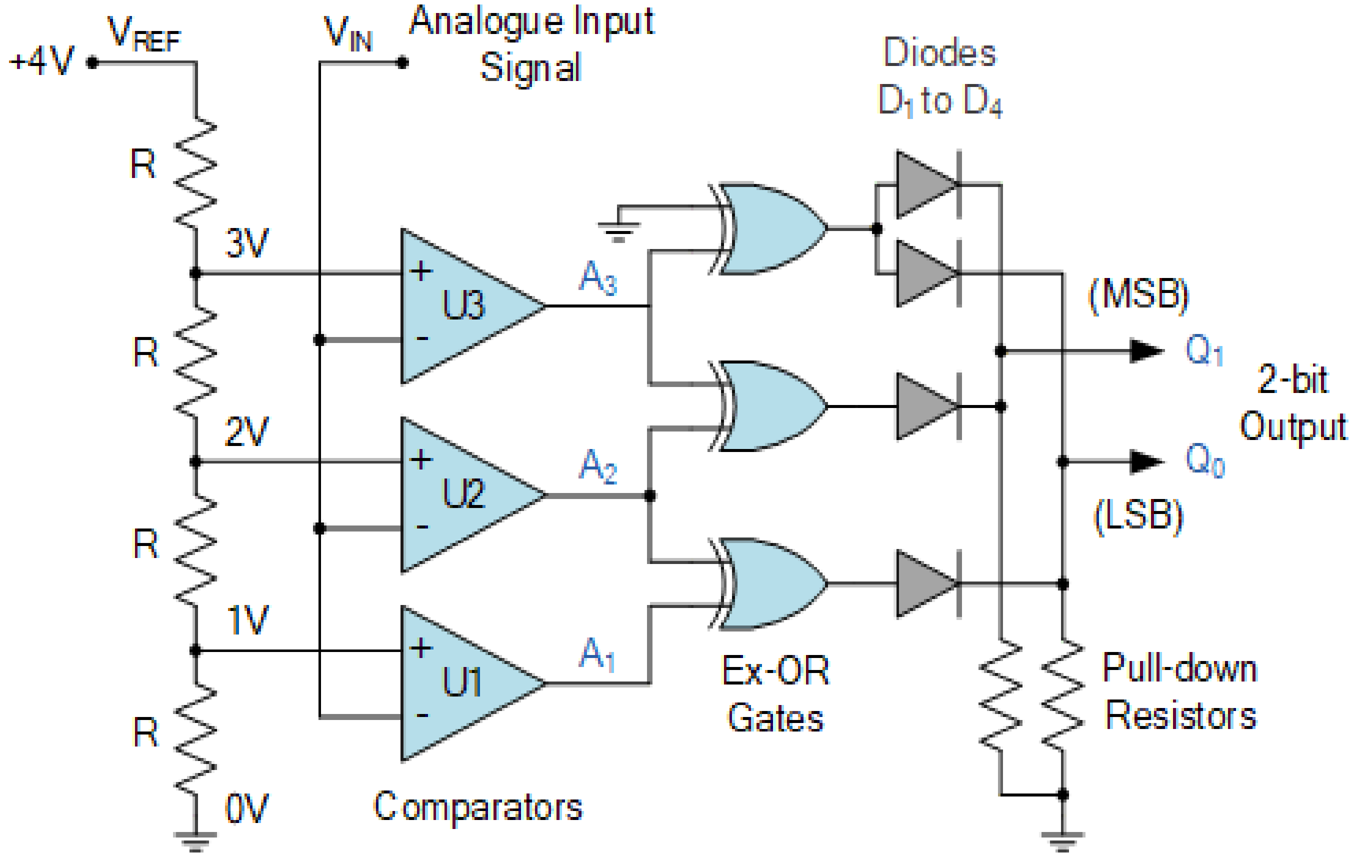
A two-bit parallel ADC.



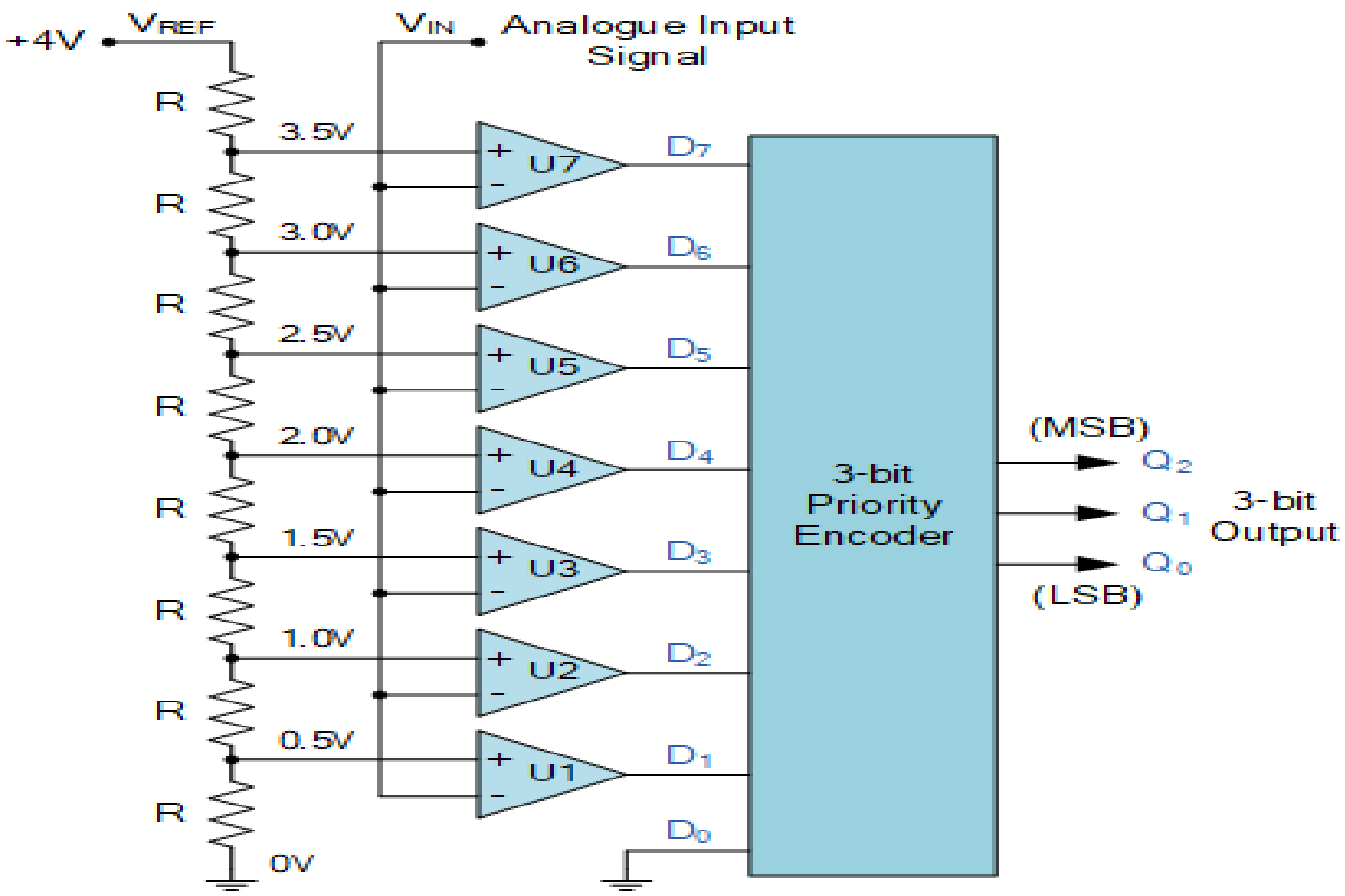
Analogue Input Voltage (V_{IN})	Comparator Outputs				Digital Outputs	
	D_3	D_2	D_1	D_0	Q_1	Q_0
0 to 1 V	0	0	0	0	0	0
1 to 2 V	0	0	1	X	0	1
2 to 3 V	0	1	X	X	1	0
3 to 4 V	1	X	X	X	1	1

Where: "X" is a "don't care", that is either a logic "0" or a logic "1" condition.

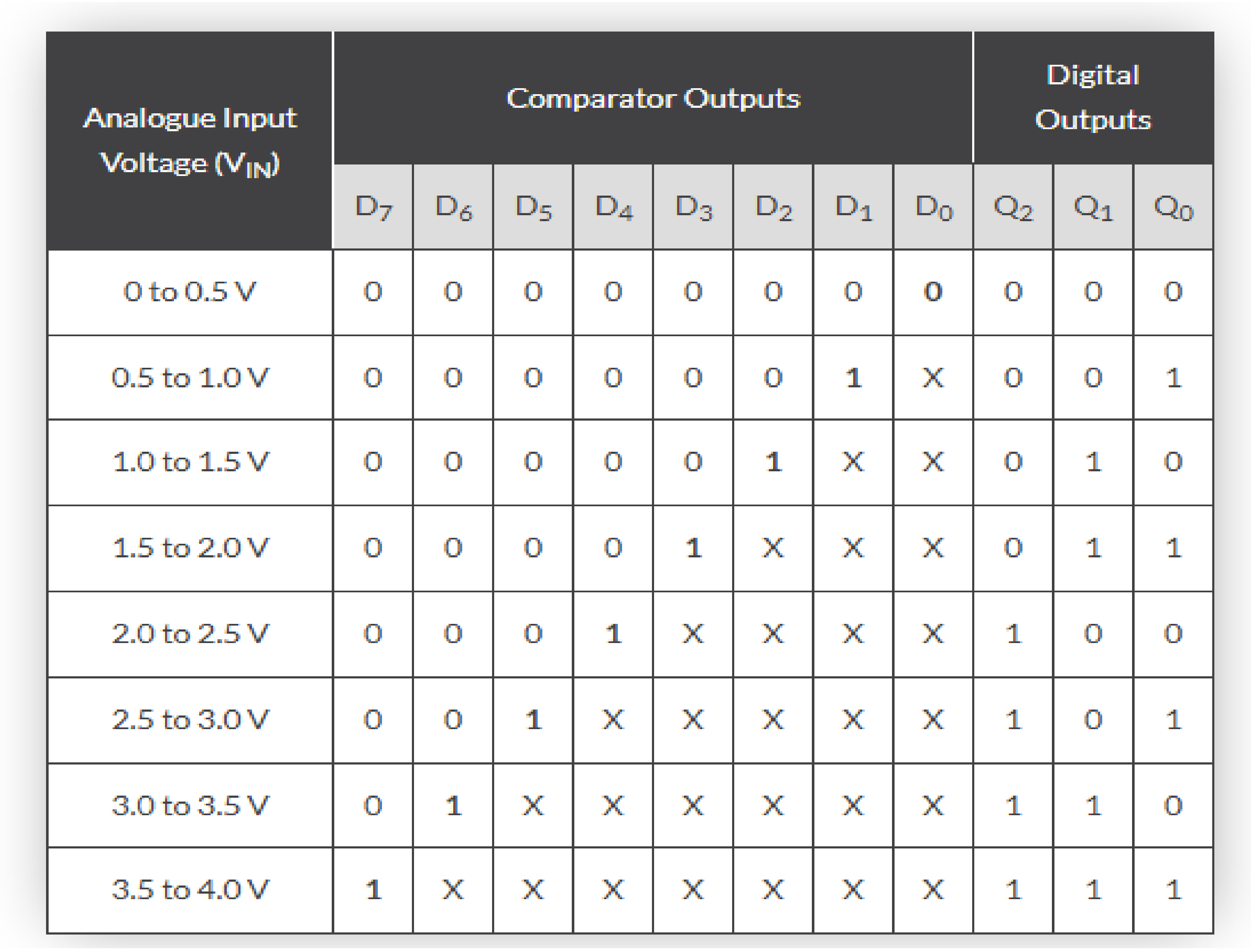
2-bit ADC Using Diodes



3-bit Analogue to Digital Converter Circuit



3-bit A/D converter Output



Analogue Input Voltage (V_{IN})	Comparator Outputs								Digital Outputs		
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0 to 0.5 V	0	0	0	0	0	0	0	0	0	0	0
0.5 to 1.0 V	0	0	0	0	0	0	1	X	0	0	1
1.0 to 1.5 V	0	0	0	0	0	1	X	X	0	1	0
1.5 to 2.0 V	0	0	0	0	1	X	X	X	0	1	1
2.0 to 2.5 V	0	0	0	1	X	X	X	X	1	0	0
2.5 to 3.0 V	0	0	1	X	X	X	X	X	1	0	1
3.0 to 3.5 V	0	1	X	X	X	X	X	X	1	1	0
3.5 to 4.0 V	1	X	X	X	X	X	X	X	1	1	1

- The four possible code-words are then passed through an encoder that outputs the final two-bit value (b_1, b_0).

- Most microcontrollers provide at least one ADC, which will often output 10-bit resolutions for a very reasonable digital representation of the analog voltage.

- For example, with a reference voltage of 5 V, a 10-bit ADC provides an accuracy of $\frac{5}{2^{10}} = 4.9mV$

- While the parallel ADC is very fast, it suffers from the problem of requiring :
  - $2^N - 1$ comparators
  - 2^N resistors

  to convert an analog signal into an N-bit digital representation.

- This circuit will require **a lot of space** and **power** in order to function, which are both **drawbacks in an embedded** processing environment

The counting ADC

- It functions by **comparing** the analog voltage V_A to a voltage level that is output from a Digital-to-Analog Converter (*DAC*) circuit.
- The level output from the DAC is controlled by the N -bit digital input vector ($b_{N-1} \dots \dots b_0$).
- When the ADC process begins, the initial N-bit vector is cleared to (0…0) which will generate the lowest analog level from the DAC

- As long as V_A is above the DAC output voltage, the **comparator output (c)** will be 5 V.
- This signal is used to enable the counter to **increase the N -bit vector by one** when the next clock edge occurs.

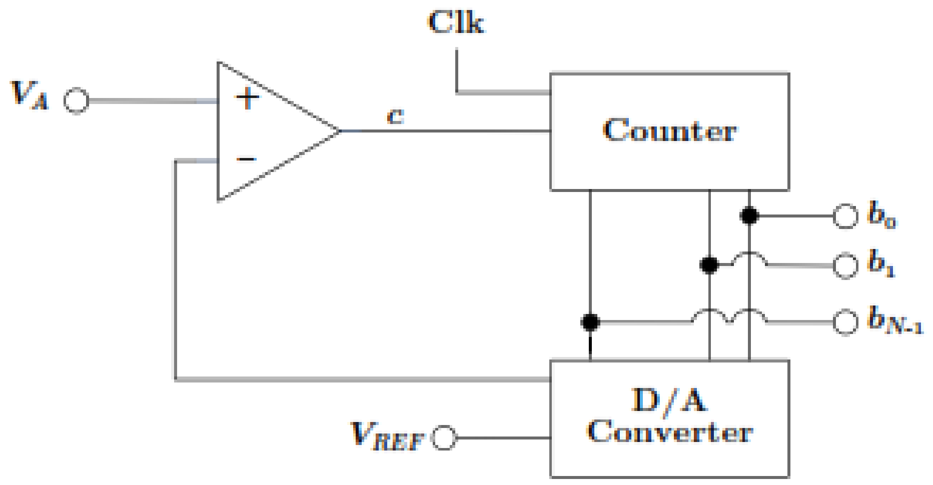
- Eventually, the N -bit vector will increase to the point that the DAC output voltage is greater than V_A, at which time, c will go to 0, which will stop the counter.

- In this way, the N -bit value used to control the DAC output voltage is also the same value that represents V_A.

- The **drawback** of the counting ADC is the time necessary for the counter to lock onto the proper binary output vector.

- It is often the case that ADCs found in **embedded processors** do require some **significant delay** before the digital value can be trusted

An N -bit counting ADC.



ADC PERIPHERAL

- In the case of the ATmega328P, there are six 10-bit ADCs (note: there are eight 10-bit ADCs on some packages all of which are found on Port C.

- For generic usage of the ADC port functionality, the following steps should be addressed.
  1. Register ADCSRA needs to be enabled, started, auto trigger enabled, and an appropriate prescalar selected;
  2. register ADCSRB needs to have the auto trigger source set to free running, so the program can read the converted value of a low-frequency sensor (which is often our application);
  3. source selected, right or left justification selected, and the desired ADC channel selected; register ADMUX needs to have a VREF
  4. register DIDR0 should have the input pin associated with the ADC channel selected in ADMUX disabled.

- Once the ADC is initialized, the program can poll the ADC output by reading the 16-bit ADC register which contains the 10-bit result.

- As the math implies, eight bits sit in one 8-bit register and the remaining two bits are provided in the other register, with six bits being unused

- It turns out that the ADC hardware will not update the ADC register until the ADCH portion is read out by itself, or the entire 16-bit register is read out as a single unit.

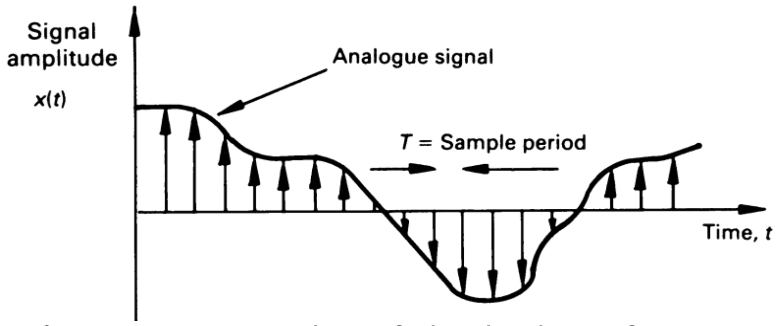
- So, if only eight bits of resolution are needed, the ADC value can be left-justified in ADMUX and the high-order byte may be read, as in the following code snip.

Advantages and Disadvantage of storing, processing and transmitting signals digitally:

- Digital technology is more ***advanced*** and more ***powerful*** than analogue technology.
- Digital signals are ***less sensitive to transmission noise*** than analogue signals. It is ***easy*** to ***error-protect*** and ***encrypt*** digital signals so that digital transmission can be made very ***secure***.
- Digital signals of different types can be treated in a unified way and, provided adequate ***decoding*** arrangements exist, can be ***mixed*** on the same channel.
- One ***disadvantage*** of digital communication is that it ***requires greater channel bandwidth.*** This can be several times the bandwidth of an equivalent analogue channel.

Sampling

- Sampling is carried out at discrete intervals of time Ts, where ***Ts*** is known as the ***sample period***. The number of samples per second or the sampling frequency fs in Hz is equal to the reciprocal of the sample period, that is ***fs = 1/Ts***.



- ***Nyquist's sampling theorem*** states that if the highest frequency component present in the signal is $\boldsymbol{f_m}$ ***Hz***, then the ***sampling frequency must be at least twice this value***, that is $\boldsymbol{f_s \geq 2f_m}$ in order that the signal may be properly re-constructed from the digital samples.
- The sampling rate $\boldsymbol{f_s = 2f_m}$ is called the ***Nyquist rate***.

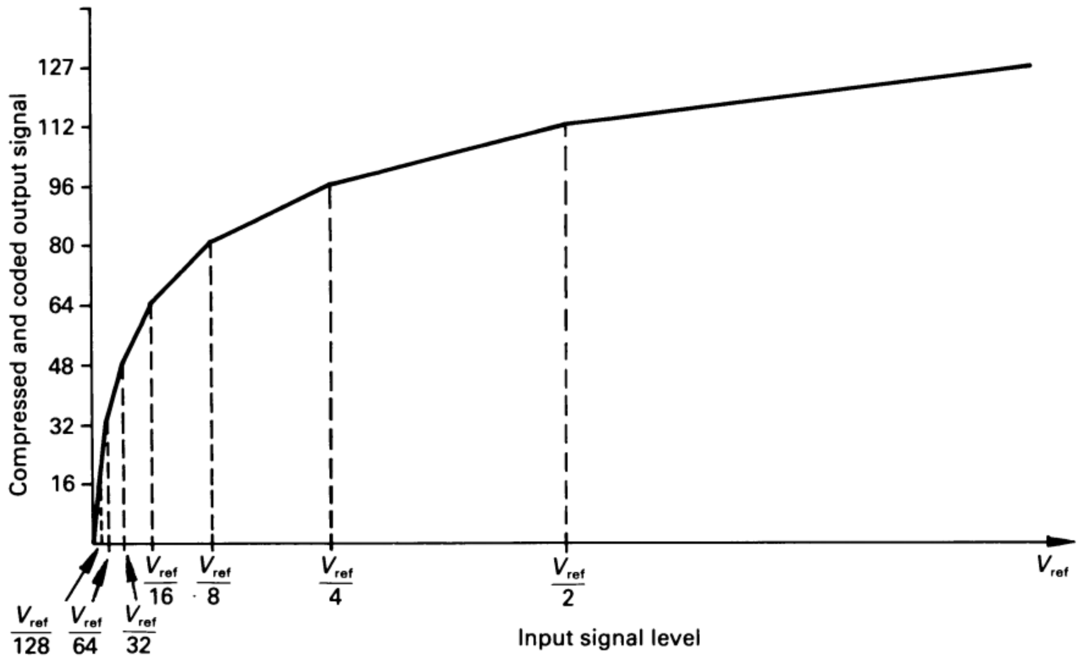
μ-Law Companding

- The μ-law companding equations are different from those of the A -law, though the resulting characteristic is very similar.

$$x'[n] = x_{max} \frac{log(1 + \mu \, |x[n]|/x_{max})}{\log(1+\mu)} \; \mathrm{sgn}\{x[n]\},$$

- where μ is a parameter which determines the level of compression and all of the other symbols have the same meaning as before. A value of μ = 255 is often used.
- In a practical system, the companding equations are not used directly on each input sample.
- Instead, the quantisation intervals are defined using a piecewise linear approximation to the companding equations.

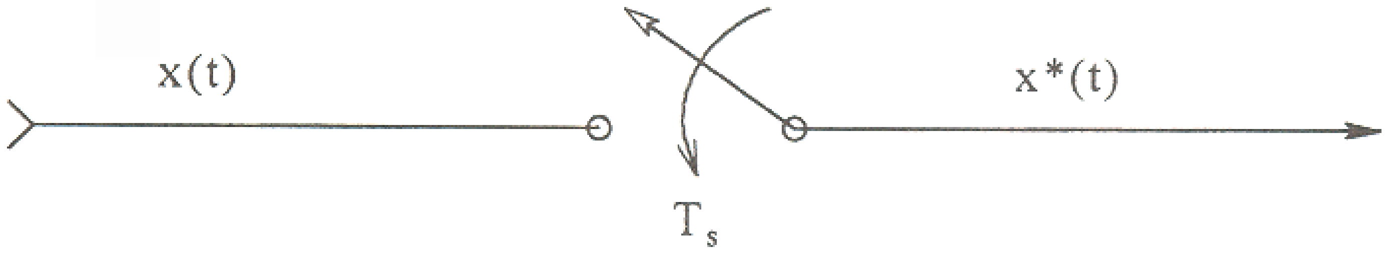
- As illustrated in figure for the A-law for the case of positive sample values, The complete characteristic exhibits odd symmetry about the vertical axis. The positive portion of the characteristic consists of 8 linear segments.



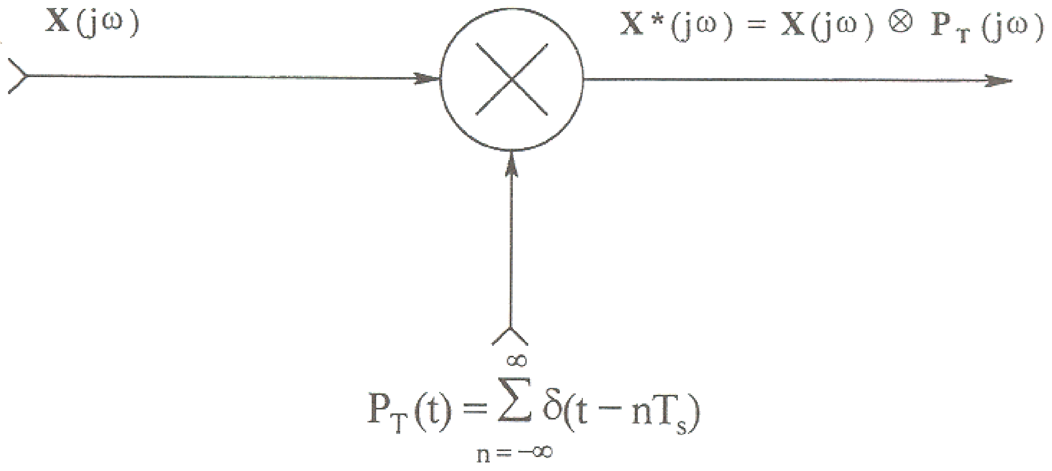
Sampling

- Data conversion from analog to digital form or digital to analog form is generally done continuously and periodically.
- **Sampling**—process of picking one value of a signal to represent the signal for some interval of time.
- Because ADC data conversion is generally a periodic process, we will first analyze what happens when an analog signal, x(t), is periodically and ideally sampled.
- The ideal sampling process generates a data sequence from x(t) defined only at the sampling instants, when $t = nT_s$, where n is an integer and $-\infty < n < +\infty$.

Sampling



•Ideal sampling process ADC. T_s—period between successive samples.

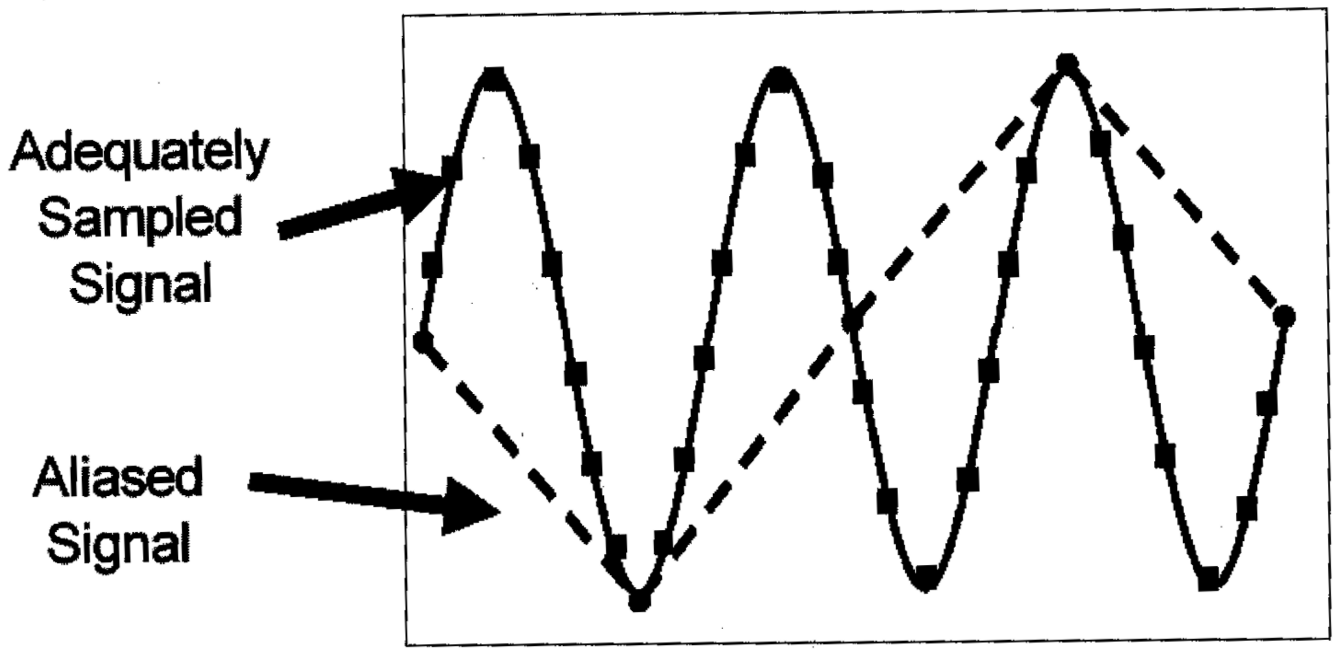


$X(j\omega)$
$X^*(j\omega) = X(j\omega) \otimes P_T(j\omega)$
$P_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s)$

- Impulse modulation equivalent to ideal sampling

Aliased Signal

- If the sampling rate is not high enough, aliased signal comes out.



Nyquist’s Theorem

- To avoid aliasing, the sampling rate must be greater than twice the maximum frequency component in the signal to be acquired.

Quantization

- Once the analog signal sample has been converted to digital form, it is represented by a digital (binary) number of a finite number of bits (e.g. 12), which limits the **resolution** of the sample (one part in 4096 for 12 bits). This rounding-off of the digital sample is called **quantization**.

Quantization

- E.g. an analog signal whose values range from 0 to +10V. We wish to convert this signal to digital form and the required output is a 4-bit signal.
- We know that a 4-bit binary number can represent 16 different values, 0 to 15. Then, the resolution of this conversion = 10 V/15 = 2/3 V. So, an analog signal of 0 V will be represented by 0000, 2/3 V will be represented by 0001.

Quantization

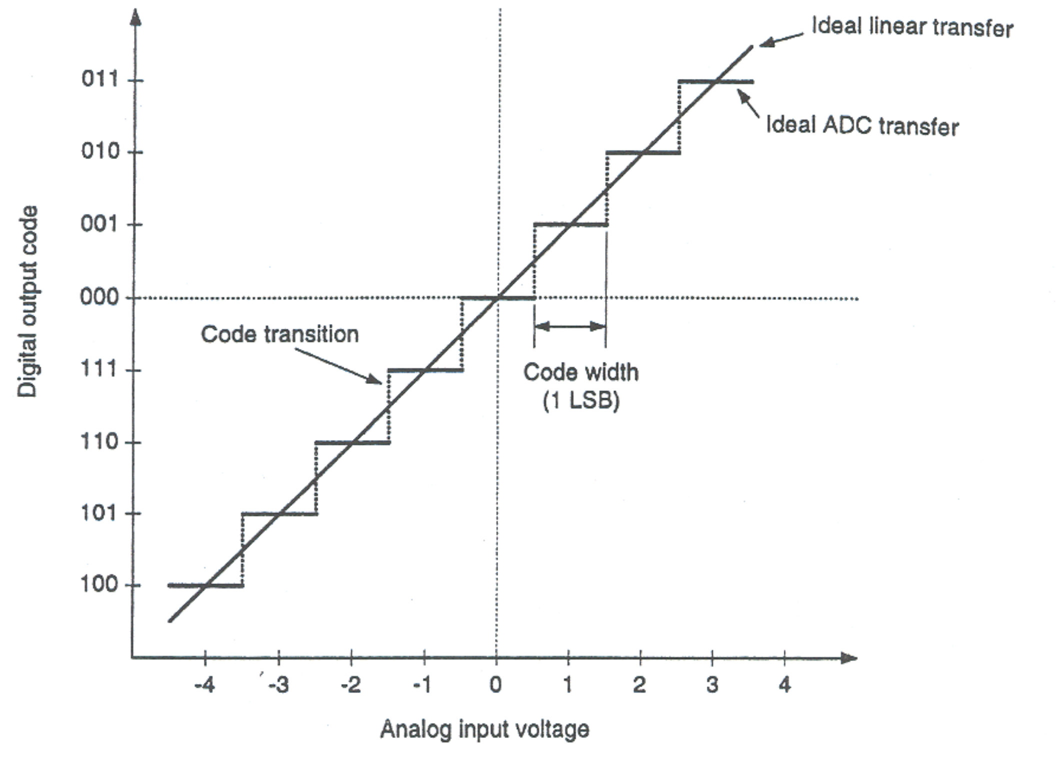
- From the above assumption, all the sample numbers were multiples of the basic increment—2/3 V.
- There is a question raised—what is the treatment when the conversion of numbers that fall between these successive incremental levels.

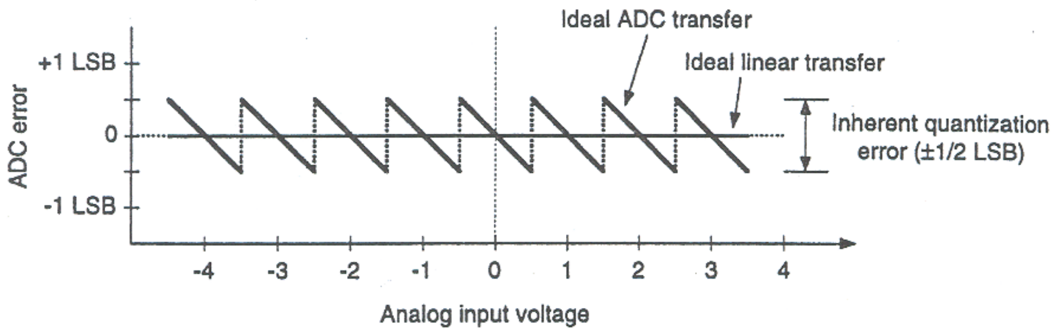
Quantization

- While sampling is done in the time domain, quantization is performed in the amplitude domain.
- The process of digitization is not complete until the sampled signal is reduced to digital information.
- The sampled signal is still in analog form. Therefore, an ADC quantizes it by picking one integer value from a predetermined, finite list of integer values to represent each analog sample.

Quantization

- Normally, an ADC chooses the value closest to the actual sample from a list of uniformly spaced values. This rule gives the **transfer function** of analog input-to-digital output a uniform “staircase” characteristic.





- The ideal 3-bit quantizer has 8 possible digital outputs. The bottom graph shows the ideal transfer function (a st. line) subtracted from the staircase transfer function.

Analog-to-digital converter (ADC)

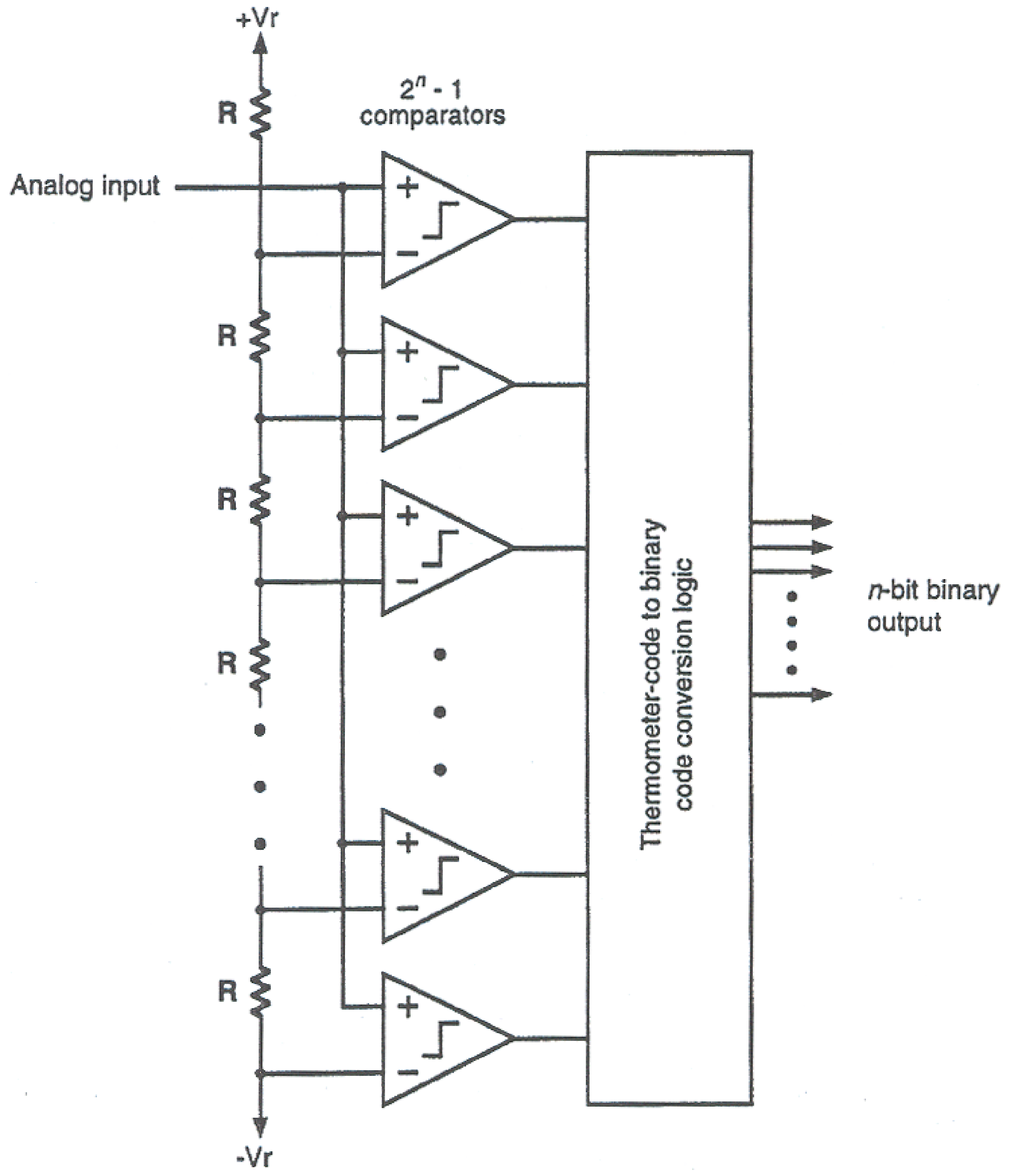
- An **analog-to-digital converter (ADC)** converts real-world signals (usually voltages) into digital numbers so that a computer can:
  - acquire signals automatically,
  - store and retrieve information about the signals,
  - process and analyze the information,
  - display measurement results.

Types of ADCs

- There are 5 major types of ADCs:
  - Flash (parallel) converters,
  - Dual-slope, integrating converters,
  - Successive-approximation converters,
  - Tracking (servo) types,
  - Dynamic range, floating point converters.
- The fastest ADCs are the flash converters. They can convert 8 bits with a sampling period of less than 1 ns. Such fast speed is useful for measuring transient phenomena. E.g. Transient events in particle physics and lasers.

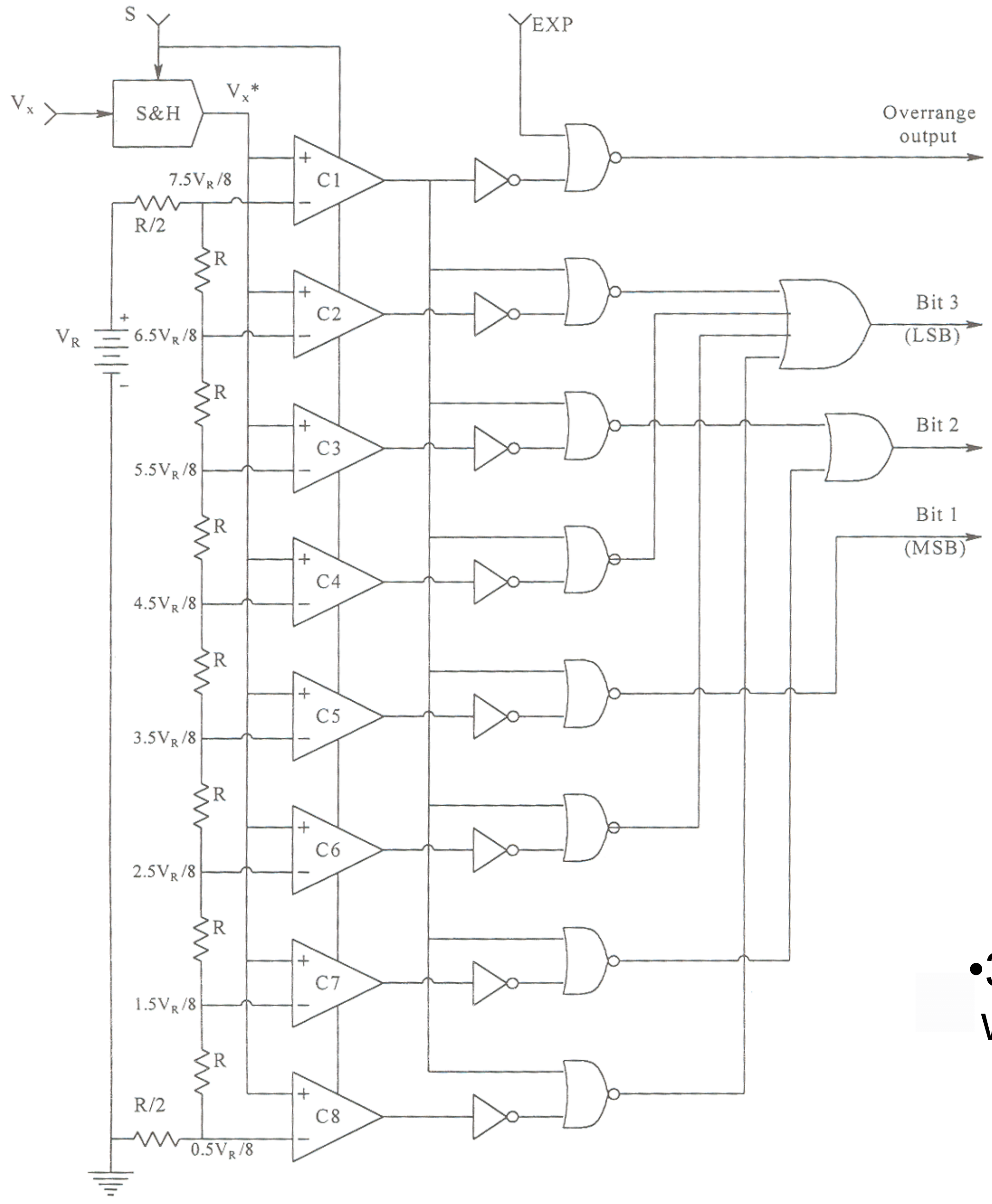
Flash (parallel) converter

- Since **Flash ADCs (FADCs)** is simple-structured, they are fast.
- A string of resistors between two voltage references supplies a set of uniformly spaced voltages that span the input range, one for each comparator. The input voltage is compared with all of these voltages simultaneously.
- Comparator outputs = 1 for all voltages below the input voltage
- Comparator outputs = 0 for all the voltages above the input voltage.
- The resulting collection of digital outputs is called a “thermometer code”.



- A flash converter has 2^n-1 comparators operating in parallel.

Flash converter (More Complicated example)



- 3-bit flash ADC with binary output.

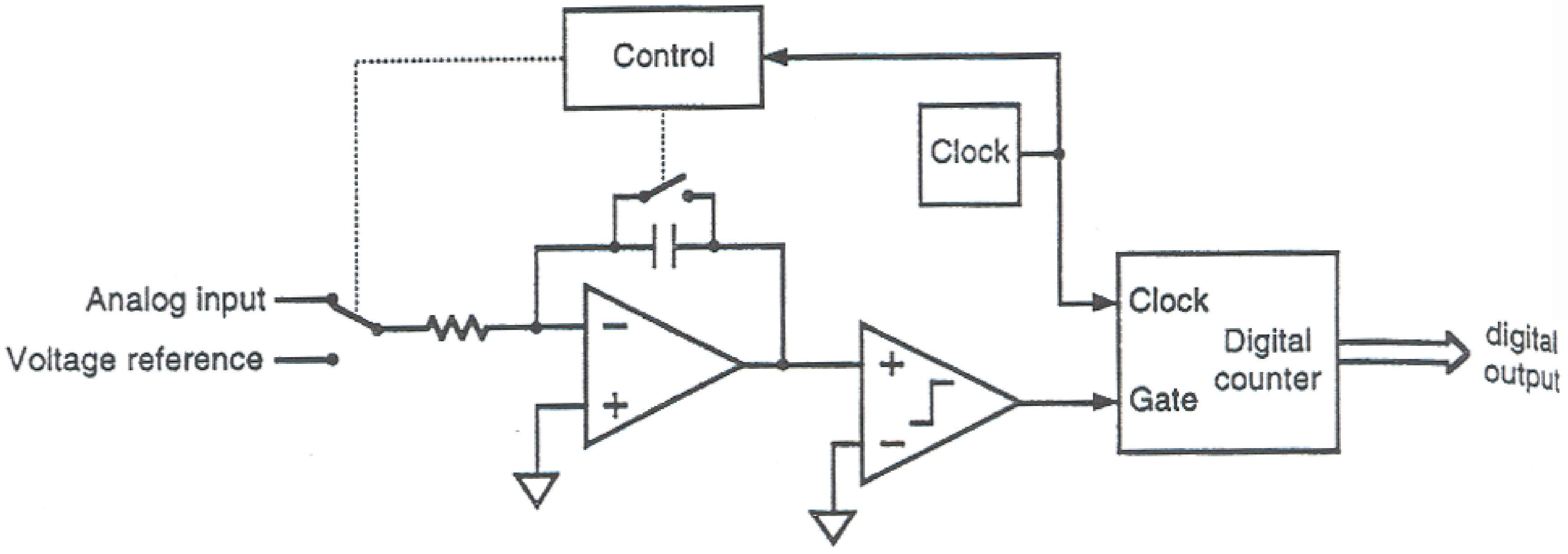
Integrating converter

- Integrating converters are used for low-speed, high-resolution applications such as voltmeters. They are conceptually simple, consisting of an integrating amplifier, a comparator, a digital counter, and a very stable capacitor for accumulating charge.

Integrating converter

- The most common integrating ADC in use is the dual-slope ADC. Its action is illustrated in the next slide.

Integrating converter



- A dual-slope integrating converter uses a comparator to determine when the capacitor has fully discharged.

Digital-to-analog converter (DAC)

- A **digital-to-analog converter (DAC)** is an integrated circuit (IC) device which converts an N bit digital word to an equivalent analog voltage or current. It allows digital information which has been processed and/or stored by a digital computer to be realized in analog form.
- After digitalization, a staircase waveform can be smoothed by a low-pass filter. In this way, an analog output signal is reconstructed.

Digital-to-analog converter (DAC)

- At sampling instants, the difference between DAC output and the analog input signal is called a **quantization error**.
- The quantization error of an ADC is equivalent to $\pm$½ least significant bit (LSB).

Digital to Analog Conversion

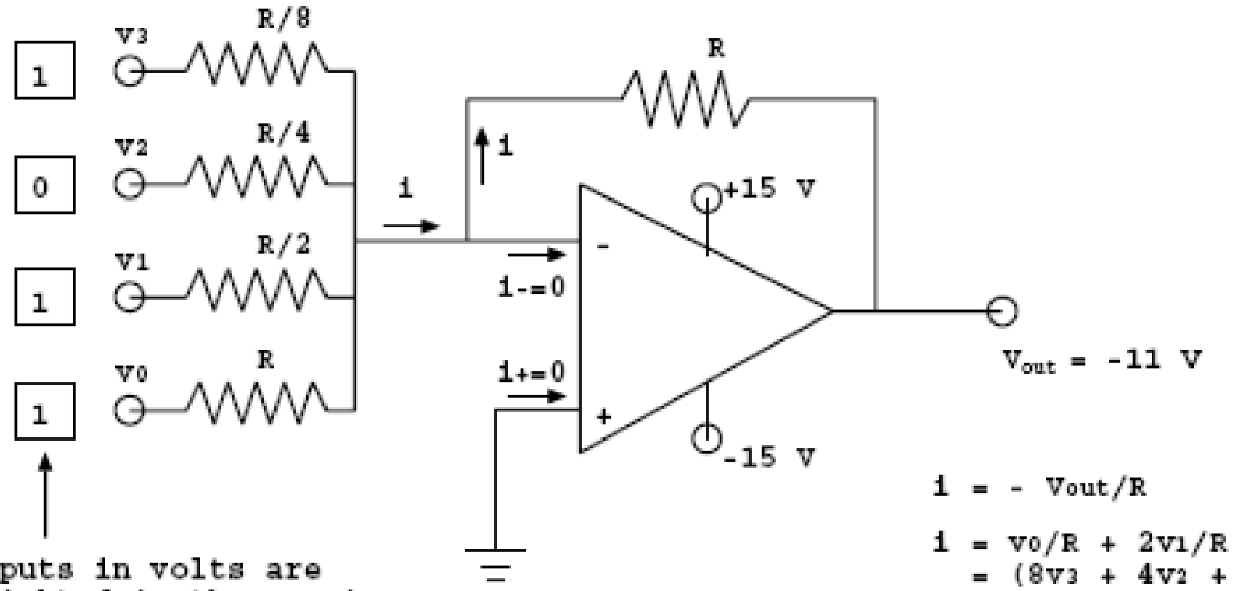
A D/A Converter produces an analog voltage proportional to the digital input.

Example:

D/A designed to output 0 to 10 volts.

- Input to D/A is an 8 bit digital value.
- Digital value 0 produces a 0 volt output.
- Digital value 1 produces a 1 x 10/256 volt ouput.
- Digital value 2 produces a 2 x 10/256 volt output.
- Digital value 255 produces a 255 x 10/256 volt output.

Typical 4-bit D/A Converter



Inputs in volts are weighted in the summing amplifier to produce the corresponding analog voltage.

1x8 + 0x4 + 1x2 + 1x1 = 11

$$i = -V_{out}/R$$

$$i = v_0/R + 2v_1/R + 4v_2/R + 8v_3/R$$
$$= (8v_3 + 4v_2 + 2v_1 + v_0)/R$$

$$-V_{out}/R = (8v_3 + 4v_2 + 2v_1 + v_0)/R$$

$$V_{out} = -(8v_3 + 4v_2 + 2v_1 + v_0)$$

Parameters Affecting D/A Converter Performance

Output Range

- The voltage difference between the max and min output voltages of the D/A.

Accuracy

- Expressed as a percentage of the maximum output voltage
- Or as an error of the least significant bit (e.g.,+1/2 LSB).
- Expected amount of accuracy in actual output based on digital input.

College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Servo Motors

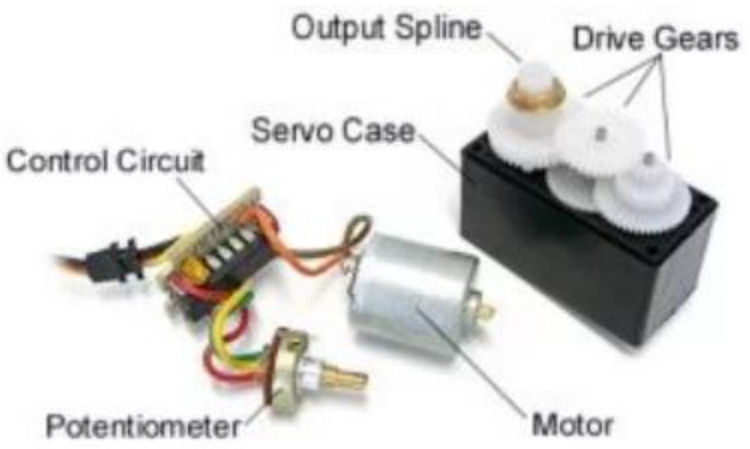
How Do Servo Motors Work?

- **Servo motors** have been around for a long time and are utilized in many applications.
- They are small in size but pack a big punch and are very energy-efficient.
- These features allow them to be used to operate remote-controlled or radio-controlled toy cars, robots and airplanes.
- Servo motors are also used in industrial applications, robotics, in-line manufacturing, pharmaceutics and food services.

- Now, let see how they work

- The **servo circuitry** is built right inside the motor unit and has a positionable shaft, which usually is fitted with a **gear**
- The motor is controlled with **an electric signal** which determines the amount of movement of the shaft.

- To fully understand how the servo works, we need to take a look inside the servo motor.
- Inside there is a pretty simple set-up:
  - Small DC motor
  - Potentiometer
  - control circuit.



- The motor is attached by gears to the control wheel.

- As the motor rotates, the potentiometer's resistance changes, so the control circuit can precisely regulate how much movement there is and in which direction.

- When the shaft of the motor is at the desired position, **power** supplied to the motor is stopped. If not, the motor is turned in the appropriate direction.

- The desired position is sent via electrical pulses through the **signal wire.**

- The motor's speed is proportional to the difference between its actual position and desired position.
  - So if the motor is near the desired position, it will turn slowly, otherwise it will turn fast.
  - This is called ……………. **control**.

How is the servo controlled?

- Servos are controlled by sending an electrical pulse of variable width, or **pulse width modulation** (PWM), through the control wire.

- There is a minimum pulse, a maximum pulse, and a repetition rate.

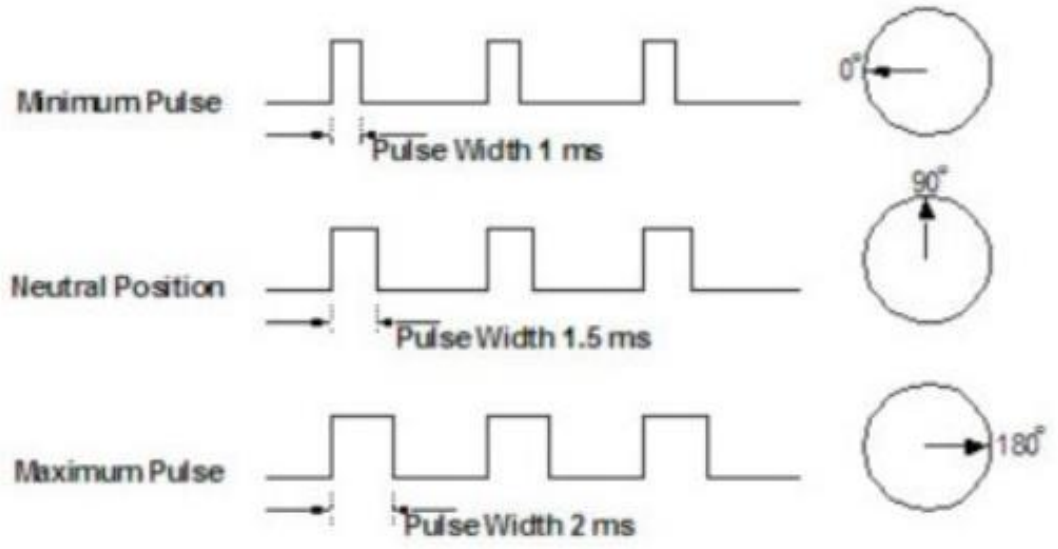
- **A servo motor can usually only turn 90° in either direction for a total of 180° movement.**

- The motor's **neutral position** is defined as the position where the servo has the same amount of **potential rotation** in the both the clockwise or counter-clockwise direction.

- The **PWM** sent to the motor **determines position of the shaft**, and based on the **duration of the pulse sent via the control wire**; the **rotor will turn to the desired position.**

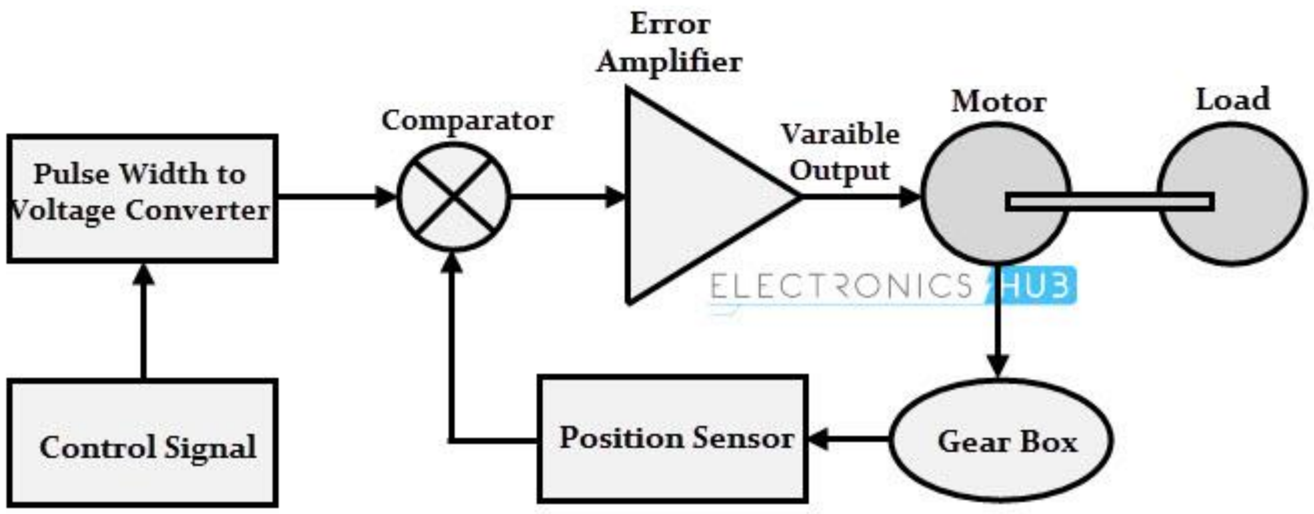
- The servo motor expects to see a pulse every **20 milliseconds (ms)** and the length of the pulse will determine how far the motor turns.

- For example:
  - 1.5ms pulse will make the motor turn to the 90° position.
  - Shorter than 1.5ms moves it in the counter clockwise direction toward the 0° position
  - Any longer than 1.5ms will turn the servo in a clockwise direction toward the 180° position.



Variable Pulse width control servo position

- When these servos are commanded to move, they will move to the position and **hold that position**.
- If an external force pushes against the servo while the servo is holding a position, the servo will resist from moving out of that position. **[This consider as one of the most important features of the servo motor]**
- The maximum amount of force the servo can exert is called the **torque rating** of the servo.
- Servos will not hold their position forever though; the position pulse must be repeated to instruct the servo to stay in position.



Types of Servo Motors

- There are two types of servo motors - AC and DC.

  - AC servo can handle higher current surges and tend to be used in industrial machinery.
  - DC servos are not designed for high current surges and are usually better suited for smaller applications.

  (Generally speaking, DC motors are less expensive than their AC counterparts.)

- These are also servo motors that have been built specifically for **continuous rotation**, making it an easy way to get your robot moving
  - They feature two ball bearings on the output shaft for reduced friction and easy access to the rest-point adjustment potentiometer.

Servo Motor Applications

- Servos are used in radio-controlled airplanes to position control surfaces like elevators, rudders, walking a robot, or operating **grippers.**
- Servo motors are small, have built-in control circuitry and have good power for their size.

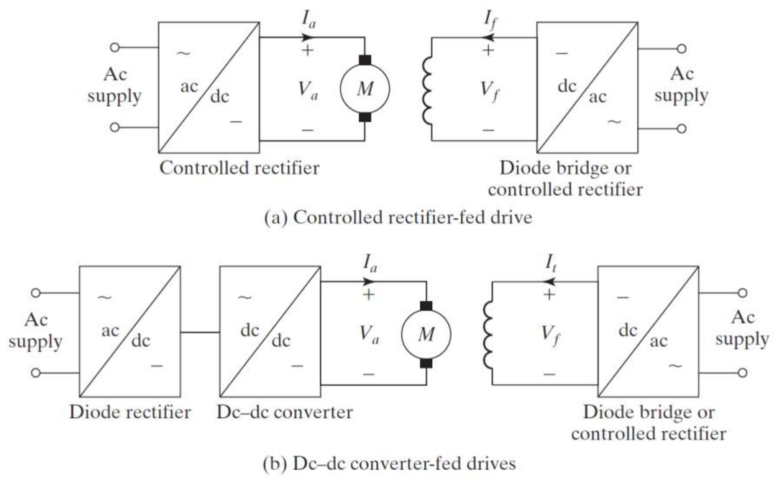
The learning objectives of this lecture are as follows:

- To learn the basic characteristics of dc motors and their control parameters
- To understand the types and operating modes of dc drives
- To learn about the control requirements of four-quadrant drives
- To understand single phase DC drives
- Solved example on Single Phase Drives

1- Introduction

Direct current (DC) motors have variable characteristics and are used extensively in variable-speed drives. Dc motors can provide a high starting torque and it is also possible to obtain speed control over a wide range. The methods of speed control are normally simpler and less expensive than those of ac drives. Dc motors play a significant role in modern industrial drives. Both series and separately excited dc motors are normally used in variable-speed drives, but series motors are traditionally employed for traction applications. Due to commutators, dc motors are not suitable for very high speed applications and require more maintenance than do ac motors. With the recent advancements in power conversions, control techniques, and microcomputers, the ac motor drives are becoming increasingly competitive with dc motor drives. Although the future trend is toward ac drives, dc drives are currently used in many industries. It might be a few decades before the dc drives are completely replaced by ac drives. Controlled rectifiers provide a variable dc output voltage from a fixed ac voltage, whereas a DC-DC converter can provide a variable dc voltage from a fixed dc voltage. Due to their ability to supply a continuously variable de voltage, controlled rectifiers and dc-dc converters made a revolution in modern industrial control equipment and variable-speed drives, with power levels ranging from fractional horsepower to several megawatts. Controlled rectifiers are generally used for the speed control of dc motors, as shown in Figure 1a. The alternative form would be a diode rectifier followed by DC-DC converter, as shown in Figure 1b. Dc drives can be classified, in general, into three types:

1. Single-phase drives 2. Three-phase drives 3. Dc-dc converter



Ac supply
ac
dc
I_a
V_a
M
Controlled rectifier
I_f
V_f
Diode bridge or controlled rectifier
(a) Controlled rectifier-fed drive
Diode rectifier
Dc–dc converter
(b) Dc–dc converter-fed drives

- General arrangement for variable speed drive.

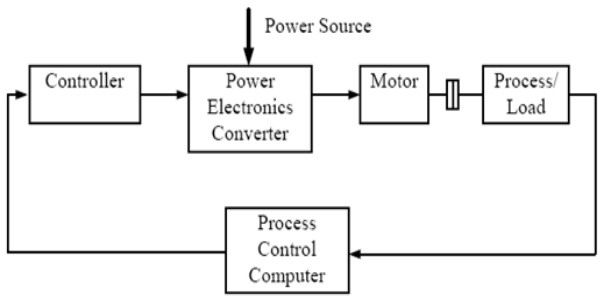


FIGURE.1 Controller rectifier- and dc-dc converter-fed drives.

Single-phase drives are used in low-power applications in the range up to 100 kW.Three-phase drives are used for applications in the range 100 kW to 500 kW. The converters are also connected in series and parallel to produce 12-pulses output. The power range can go as high as 1 MW for high-power drives. These drives generally require harmonic filters and their size could be quite bulky.

2- BASIC CHARACTERISTICS OF DC MOTORS

1- Separately Excited Dc Motor

The equivalent circuit for a separately excited dc motor is shown in Figure .2

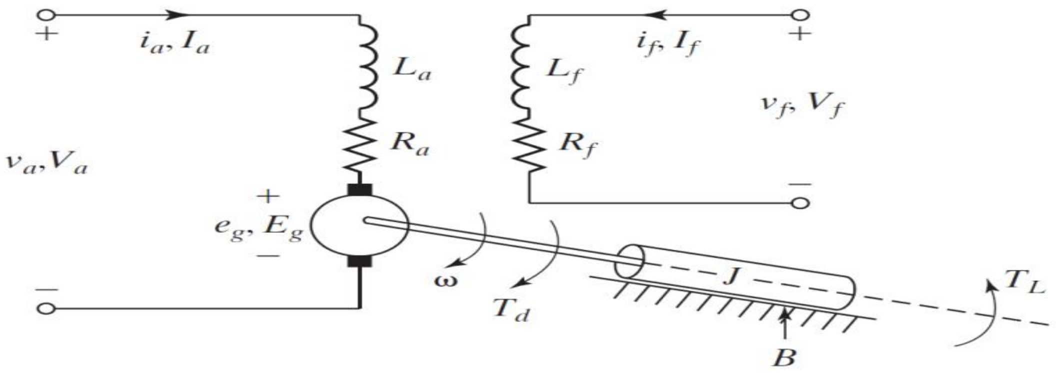


Figure .2 Equivalent circuit of separately excited dc motors.

The equations describing the characteristics of a separately excited motor can be determined from Figure 2. The instantaneous field current *if* is described as

$$v_f = R_f i_f + L_f \frac{di_f}{dt}$$

The instantaneous armature current can be found from

$$v_a = R_a i_a + L_a \frac{di_a}{dt} + e_g$$

The motor back emf, which is also known as speed voltage, is expressed as

$$e_g = k_v w i_f$$

The torque developed by the motor is

$$T_d = k_t i_f i_a$$

The developed torque must be equal to the load torque:

$$T_d = Bw + j\frac{dw}{dt} + T_L$$

Where
w = motor angular speed, or rotor angular frequency, rad/s.
B = viscous friction constant, N .m/rad/s.
Kv = voltage constant, V/A-rad/s.
Kt = torque constant, which equals voltage constant, Kv.
La = armature circuit inductance, H.
Lf = field circuit inductance, H.
Ra = armature circuit resistance Ω.
Rf = field circuit resistance, Ω.

TL = load torque, N . m.

Under steady-state conditions, the time derivatives in these equations are zero and the steady-state average quantities are

$$V_f = R_f I_f \qquad 1$$

$$E_g = K_v I_f\ w \qquad 2$$

$$V_a = R_a I_a + E_g$$

$$V_a = R_a I_a + K_v I_f\ w \qquad 3$$

$$T_d = k_t i_f i_a \qquad 4$$

$$T_d = Bw + T_L \qquad 5$$

The developed power is :

$$P_d = T_d w \qquad 6$$

From Eq. (3), the speed of a separately excited motor can be found from:

$$w = \frac{V_a - R_a I_a}{K_v +} \qquad 7$$

We can notice from Eq. (7) that the motor speed can be varied by (1) controlling the armature voltage *Va*, known as *voltage control*; (2) controlling the field current *If*, known as *field control*; or (3) torque demand, which corresponds to an armature current *Ia*, for a fixed field current *If*. The speed, which corresponds to the rated armature voltage, rated field current, and rated armature current, is known as the *rated* (or *base*) *speed*.
In practice, for a speed less than the base speed, the armature current and field currents are maintained constant to meet the torque demand, and the armature voltage *Va* is varied

to control the speed. For speed higher than the base speed, the armature voltage is maintained at the rated value and the field current is varied to control the speed. However, the power developed by the motor 1 = torque * speed2 remains constant. Figure.3 shows the characteristics of torque, power, armature current, and field current against the speed.

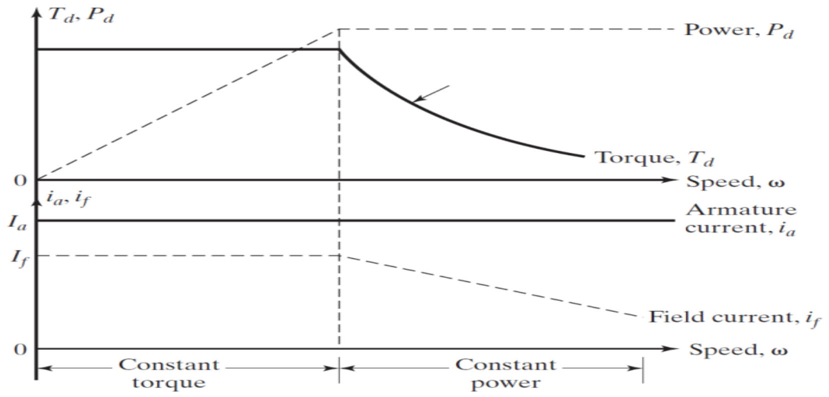


Figure .3 Characteristics of separately excited motors.

2- Series-Excited Dc Motor

The field of a dc motor may be connected in series with the armature circuit, as shown in Figure 4, and this type of motor is called a *series motor*. The field circuit is designed to carry the armature current. The steady-state average quantities are

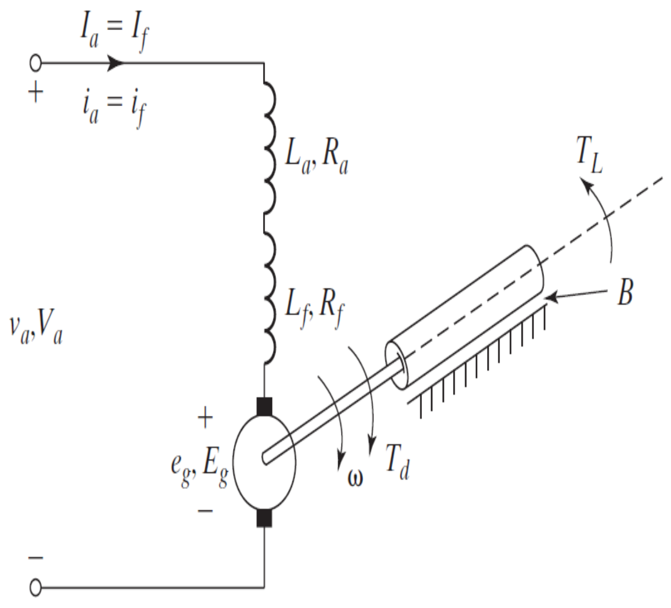


FIGURE .4.

Equivalent circuit of dc series motors.

$$E_g = K_v I_a\, w \qquad 8$$

$$V_a = (R_{a+} R_f) I_a + E_g \qquad 9$$

$$V_a = (R_{a+} R_f) I_a + K_v I_f\, w \qquad 10$$

$$T_d = k_t i_f i_a$$

$$T_d = Bw + T_L \qquad 11$$

The speed of a series motor can be determined from Eq.10

$$w = \frac{V_a - (R_a + R_f) I_a}{K_v I_f} \qquad 12$$

The speed can be varied by controlling the (1) armature voltage *Va*; or (2) armature current, which is a measure of the torque demand. Equation (11) indicates that a series motor can provide a high torque, especially at starting; for this reason, series motors are commonly used in traction applications.

3- Operating Modes

In variable-speed applications, a dc motor may be operating in one or more modes: motoring, regenerative braking, dynamic braking, plugging, and four quadrants. The operation of the motor in any one of these modes requires connecting the field and armature circuits in different arrangements, as shown in Figure 5. This is done by switching power semiconductor devices and contactors.

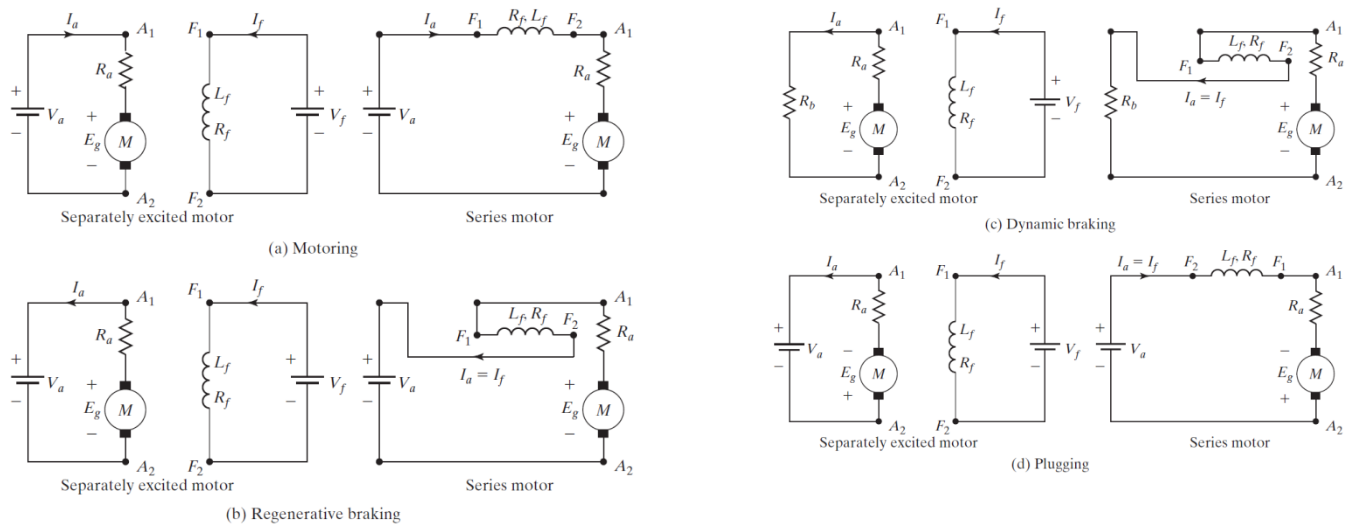


Figure 5 Operating modes.

Motoring. The arrangements for motoring are shown in Figure 5 a. Back emf *Eg* is less than supply voltage *Va*. Both armature and field currents are positive. The motor develops torque to meet the load demand.

Regenerative braking. The arrangements for regenerative braking are shown in Figure 5 b. The motor acts as a generator and develops an induced voltage *Eg*. *Eg* must be greater than supply voltage *Va*. The armature current is negative, but the field current is positive. The kinetic energy of the motor is returned to the supply. A series motor is usually connected as a self-excited generator. For self-excitation, it is necessary that the field current aids the residual flux. This is normally accomplished by reversing the armature terminals or the field terminals.

Dynamic braking. The arrangements shown in Figure 5 c are similar to those of regenerative braking, except the supply voltage *Va* is replaced by a braking resistance *Rb*. The kinetic energy of the motor is dissipated in *Rb*.

Plugging. Plugging is a type of braking. The connections for plugging are shown in Figure 5 d . The armature terminals are reversed while running. The supply voltage *Va* and the induced voltage *Eg* act in the same direction. The armature current is reversed, thereby producing braking torque. The field current is positive. For a series motor, either the armature terminals or field terminals should be reversed, but not both.

Four quadrants. Figure 6 shows the polarities of the supply voltage *Va*, back emf *Eg*, and armature current *Ia* for a separately excited motor.

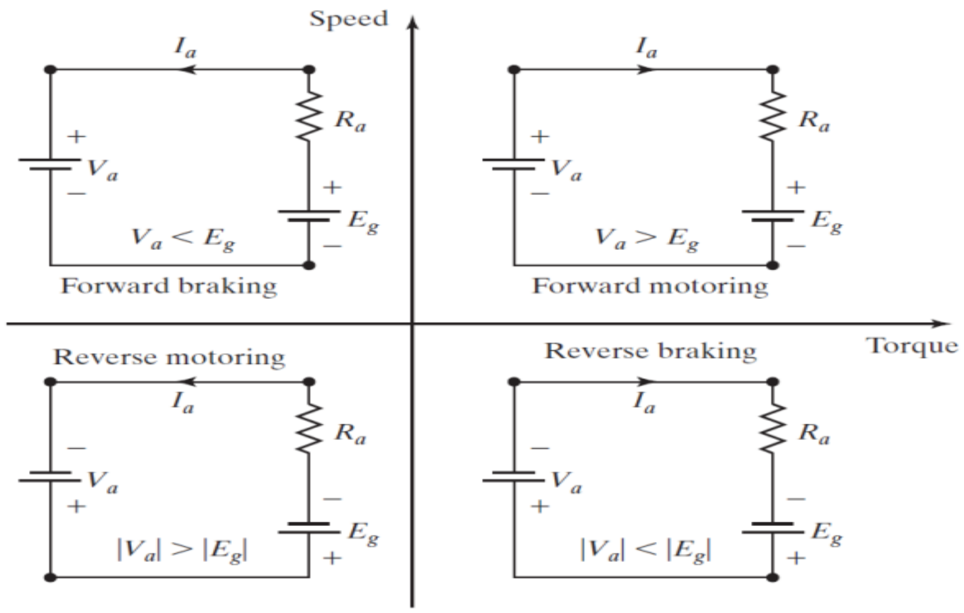


Figure 6 Conditions for four quadrants.

Single-Phase Drives

If the armature circuit of a dc motor is connected to the output of a single-phase Controlled rectifier, the armature voltage can be varied by varying the delay angle of the converter ∝*a*. The forced-commutated ac–dc converters can also be used to improve the power factor (PF) and to reduce the harmonics. The basic circuit agreement for a single-phase converter-fed separately excited motor is shown in Figure 7.

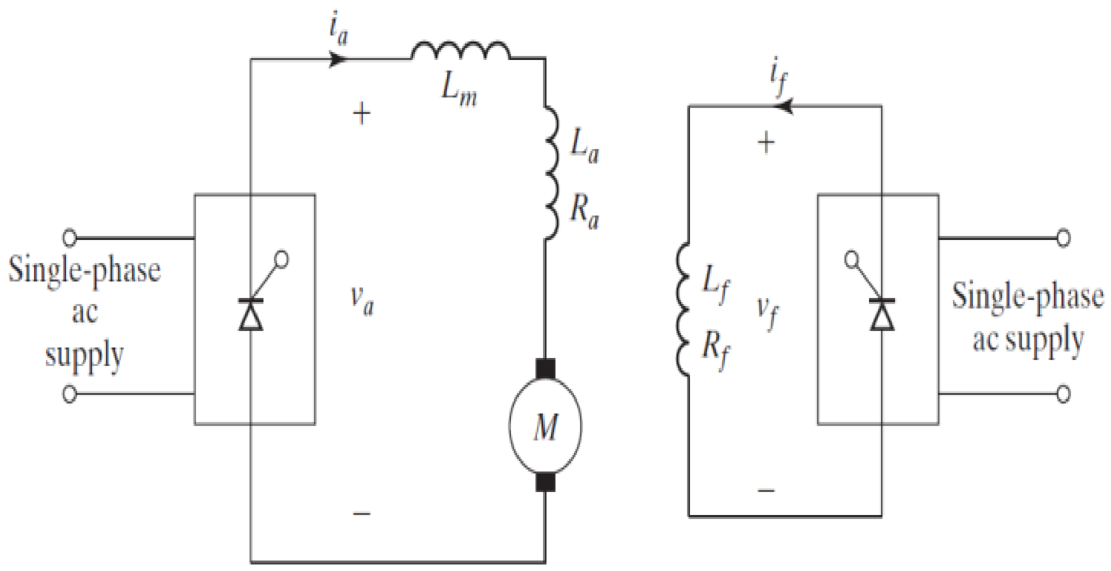


Figure 7.Basic circuit arrangement of a single-phase dc drive.

Depending on the type of single-phase converters, single-phase drives may be subdivided into:

1. Single-phase half-wave converter drives

2. Single-phase semiconverter drives

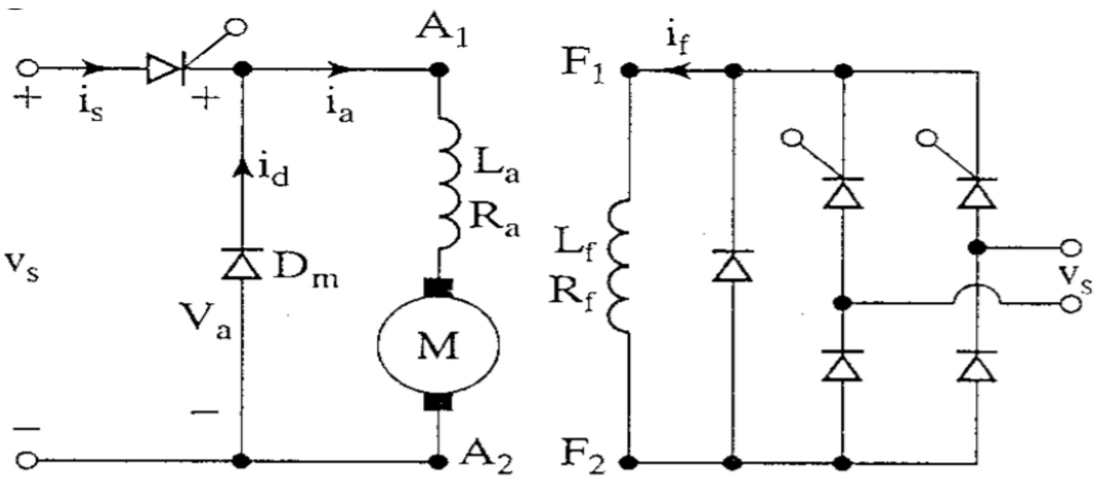
3. Single-phase full-converter drives

4. Single-phase dual-converter drives

The armature current of half-wave converter drives is normally discontinuous. This type of drive is not commonly used . A semiconverter drive operates in one quadrant in applications up to 1.5 kW. The full converter and dual drives are more commonly used.

1- Single-Phase Half-Wave Converter Drives

Figure 8 shows a single-phase half-wave converter drive used to control the speed of separately-excited motor. This d.c. drive is very simple, needs only one power switch and one freewheeling diode connected across the motor terminals for the purpose of dissipation of energy stored in the inductance of the motor and to provide an alternative path for the motor current to allow the power switch to commutate easily.



$$V_a = \frac{V_m}{2\pi}(1+\cos\alpha_a)$$

$$0 \le \alpha_a \le \pi$$

$$V_f = \frac{V_m}{\pi}(1+\cos\alpha_f)$$

$$0 \le \alpha_f \le \pi$$

Figure 8 a single-phase half-wave converter drive

2- Single-Phase Semiconverter Drives

A single-phase semiconverter feeds the armature circuit, as shown in Figure 9a. It is a one-quadrant drive, as shown in Figure 9b, and is limited to applications up to 15 kW. The converter in the field circuit can be a semiconverter . The current waveforms for a highly inductive load are shown in Figure 9c.

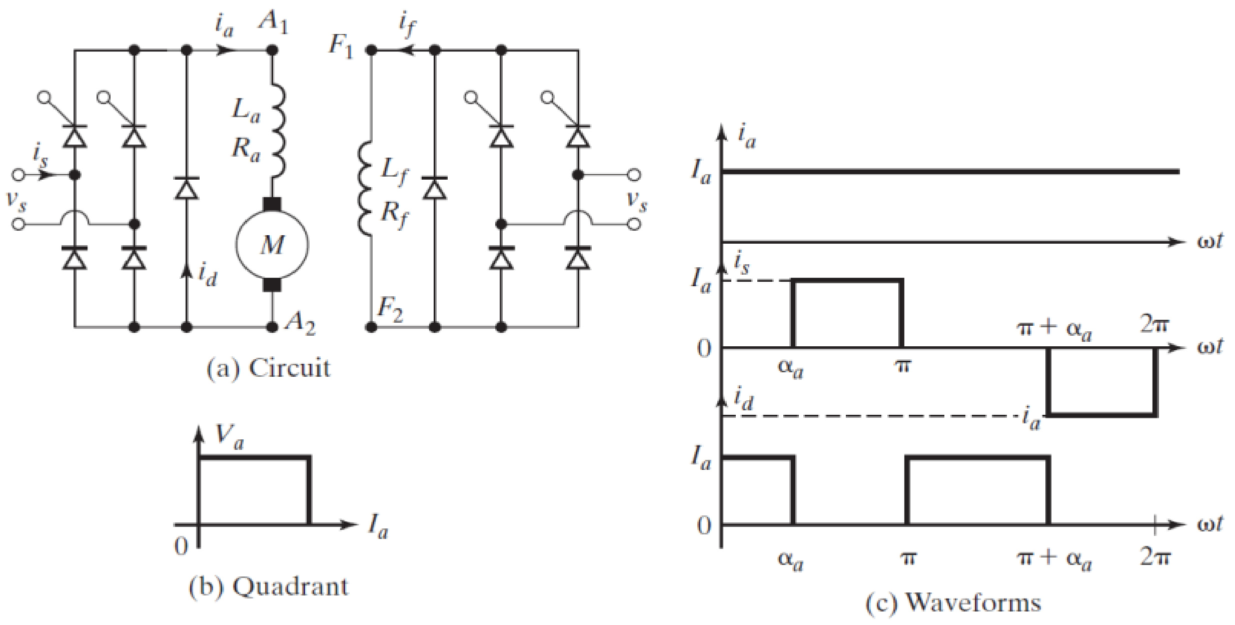


Figure 9. Single-phase semiconverter drive.

With a single-phase semiconverter in the armature circuit, the average armature voltage is can given by

$$V_a = \frac{V_m}{\pi}(1 + \cos \alpha_a) \qquad \text{for } 0 \le \alpha_a \le \pi \qquad 13$$

With a semiconverter in the field circuit, gives the average field voltage as

$$V_f = \frac{V_m}{\pi}(1 + \cos \alpha_f) \qquad \text{for } 0 \le \alpha_f \le \pi \qquad 14$$

3- Single-Phase Full-Converter Drives.

The armature voltage is varied by a single-phase full-wave converter, as shown in Figure 10a. It is a two-quadrant drive, as shown in Figure 10b, and is limited to applications up to 15 kW. The armature converter gives +Va or -Va, and allows operation in the first and fourth quadrants. During regeneration for reversing the direction of power flow, the back emf of the motor can be reversed by reversing the field excitation. The converter in the field circuit could be a semi-, a full, or even a dual converter. The reversal of the armature or field allows operation in the second and third quadrants. The current waveforms for a highly inductive load are shown in Figure 10c for powering action.

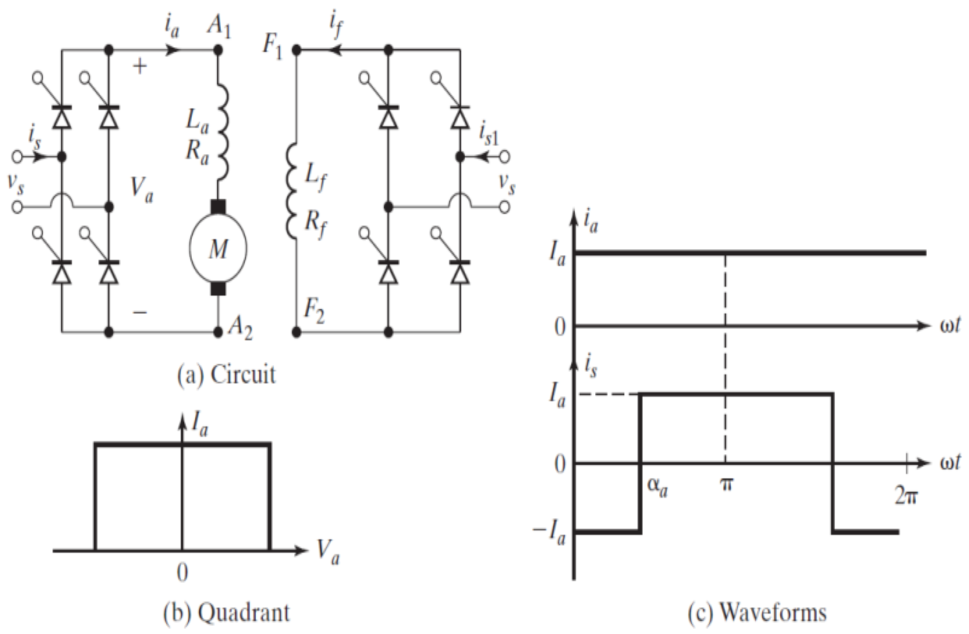


Figure 10 Single-phase full-converter drive.

With a single-phase full-wave converter in the armature circuit the average armature voltage is given by:

$$V_a = \frac{2V_m}{\pi} \cos \alpha_a \qquad \text{for } 0 \le \alpha_a \le \pi \tag{15}$$

With a single-phase full-converter in the field circuit,

$$V_f = \frac{2V_m}{\pi}\cos\alpha_f \qquad \text{for } 0 \le \alpha_f \le \pi \qquad 16$$

4- Single-Phase Dual-Converter Drives

Two single-phase full-wave converters are connected, as shown in Figure 11. Either converter 1 operates to supply a positive armature voltage, *Va*, or converter 2 operates to supply a negative armature voltage, -*Va*. Converter 1 provides operation in the first and fourth quadrants, and converter 2, in the second and third quadrants. It is a four-quadrant drive and permits four modes of operation: forward powering, forward braking (regeneration), reverse powering, and reverse braking (regeneration). It is limited to applications up to 15 kW. The field converter could be a full-wave, a semi-, or a dual converter.

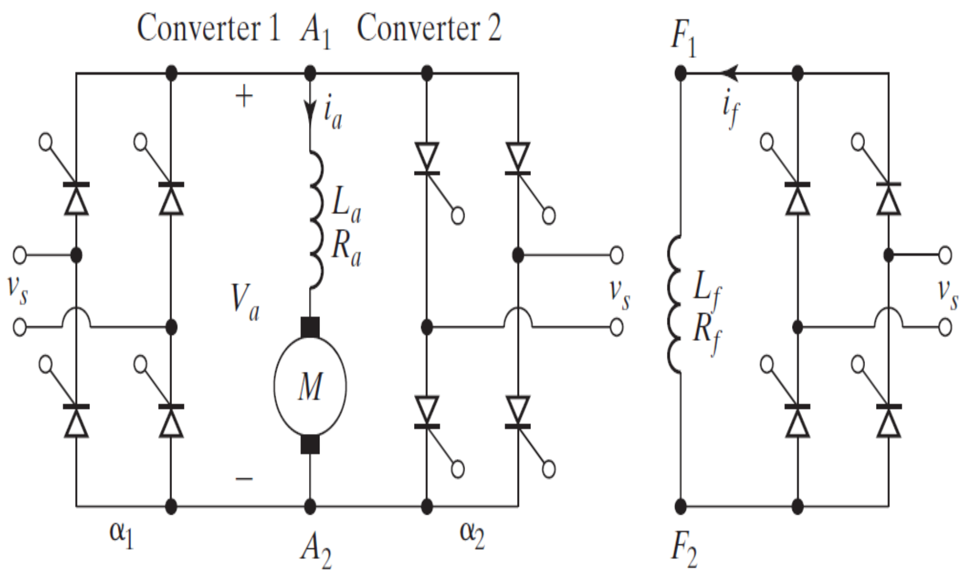


Figure11 Single-phase dual-converter drive.

If converter 1 operates with a delay angle of α*a*1:

$$V_a = \frac{2V_m}{\pi}\cos\alpha_{a1} \qquad \text{for } 0 \leq \alpha_{a1} \leq \pi \tag{17}$$

If converter 2 operates with a delay angle of α*a*2,

$$V_a = \frac{2V_m}{\pi}\cos\alpha_{a2} \qquad \text{for } 0 \leq \alpha_{a2} \leq \pi \tag{18}$$

With a full converter in the field circuit

$$V_f = \frac{2V_m}{\pi}\cos\alpha_f \qquad \text{for } 0 \leq \alpha_f \leq \pi \tag{19}$$

1- Example 14.3 Finding the Performance Parameters of a Single-Phase Semiconverter Drive[2] p 714.

The speed of a separately excited motor is controlled by a single-phase semiconverter in Figure 14.12a. The field current, which is also controlled by a semiconverter, is set to the maximum possible value. The ac supply voltage to the armature and field converters is one phase, 208 V, 60 Hz. The armature resistance is $R_a = 0.25\ \Omega$, the field resistance is $R_f = 147\ \Omega$, and the motor voltage constant is $K_v = 0.7032$ V/A rad/s. The load torque is $T_L = 45$ N · m at 1000 rpm. The viscous friction and no-load losses are negligible. The inductances of the armature and field circuits are sufficient enough to make the armature and field currents continuous and ripple free. Determine (a) the field current I_f; (b) the delay angle of the converter in the armature circuit α_a, and (c) the input power factor of the armature circuit converter.

Solution

$V_s = 208$ V, $V_m = \sqrt{2} \times 208 = 294.16$ V, $R_a = 0.25\ \Omega$, $R_f = 147\ \Omega$, $T_d = T_L = 45$ N · m, $K_v =$ 0.7032 V/A rad/s, and $\omega = 1000\ \pi/30 = 104.72$ rad/s.

a. From Eq. (14.19), the maximum field voltage (and current) is obtained for a delay angle of $\alpha_f = 0$ and

$$V_f = \frac{V_m}{\pi}(1 + \cos\alpha_f) \qquad V_f = \frac{2V_m}{\pi} = \frac{2 \times 294.16}{\pi} = 187.27\text{ V}$$

The field current is

$$I_f = \frac{V_f}{R_f} = \frac{187.27}{147} = 1.274\text{ A}$$

b. From Eq. (14.4), $T_d = K_t I_f I_a$

$$I_a = \frac{T_d}{K_v I_f} = \frac{45}{0.7032 \times 1.274} = 50.23\text{ A}$$

From Eq. (14.2),

$$E_g = K_v \omega I_f = 0.7032 \times 104.72 \times 1.274 = 93.82 \text{ V}$$

From Eq. (14.3), the armature voltage is $V_a = R_a I_a + E_g$

$$V_a = 93.82 + I_a R_a = 93.82 + 50.23 \times 0.25 = 93.82 + 12.56 = 106.38 \text{ V}$$

From Eq. (14.18), $V_a = 106.38 = (294.16/\pi) \times (1 + \cos \alpha_a)$ and this gives the delay angle as $\alpha_a = 82.2°$. $V_a = \frac{V_m}{\pi}(1 + \cos \alpha_a)$

c. If the armature current is constant and ripple free, the output power is $P_o = V_a I_a = 106.38 \times 50.23 = 5343.5$ W. If the losses in the armature converter are neglected, the power from the supply is $P_a = P_o = 5343.5$ W. The rms input current of the armature converter, as shown in Figure 14.12, is

$$I_{sa} = \left(\frac{2}{2\pi}\int_{\alpha_a}^{\pi} I_a^2 \, d\theta\right)^{1/2} = I_a\left(\frac{\pi - \alpha_a}{\pi}\right)^{1/2}$$

$$= 50.23\left(\frac{180 - 82.2}{180}\right)^{1/2} = 37.03 \text{ A}$$

and the input volt–ampere (VA) rating is $VI = V_s I_{sa} = 208 \times 37.03 = 7702.24$. Assuming negligible harmonics, the input PF is approximately

$$\text{PF} = \frac{P_o}{\text{VI}} = \frac{5343.5}{7702.24} = 0.694 \text{ (lagging)}$$

$$\text{PF} = \frac{\sqrt{2}\,(1 + \cos 82.2°)}{[\pi(\pi - 82.2°)]^{1/2}} = 0.694 \text{ (lagging)}$$

The input power factor is given by [12]

$$\text{PF} = \frac{\sqrt{2}(1 + \cos \alpha)}{\sqrt{\pi(\pi + \cos \alpha)}}$$

The speed of a separately excited dc motor is controlled by a single-phase full-wave converter in Figure 14.13a. The field circuit is also controlled by a full converter and the field current is set to the maximum possible value. The ac supply voltage to the armature and field converters is one phase, 440 V, 60 Hz. The armature resistance is $R_a = 0.25\ \Omega$, the field circuit resistance is $R_f = 175\ \Omega$, and the motor voltage constant is $K_v = 1.4$ V/A rad/s. The armature current corresponding to the load demand is $I_a = 45$ A. The viscous friction and no-load losses are negligible. The inductances of the armature and field circuits are sufficient to make the armature and field currents continuous and ripple free. If the delay angle of the armature converter is $\alpha_a = 60°$ and the armature current is $I_a = 45$ A, determine (a) the torque developed by the motor T_d, (b) the speed ω, and (c) the input PF of the drive.

Solution

$V_s = 440$ V, $V_m = \sqrt{2} \times 440 = 622.25$ V, $R_a = 0.25\ \Omega$, $R_f = 175\ \Omega$, $\alpha_a = 60°$, and $K_v = 1.4$ V/A rad/s.

a. From Eq. (14.21), the maximum field voltage (and current) would be obtained for a delay angle of $\alpha_f = 0$ and

$$V_f = \frac{2V_m}{\pi} \cos \alpha_f \qquad V_f = \frac{2V_m}{\pi} = \frac{2 \times 622.25}{\pi} = 396.14 \text{ V}$$

The field current is

$$I_f = \frac{V_f}{R_f} = \frac{396.14}{175} = 2.26 \text{ A}$$

$$T_d = K_t I_f I_a$$

From Eq. (14.4), the developed torque is

$$T_d = T_L = K_v I_f I_a = 1.4 \times 2.26 \times 45 = 142.4 \text{ N} \cdot \text{m}$$

From Eq. (14.20), the armature voltage is

$$V_a = \frac{2V_m}{\pi} \cos 60° = \frac{2 \times 622.25}{\pi} \cos 60° = 198.07 \text{ V}$$

The back emf is

$$E_g = V_a - I_a R_a = 198.07 - 45 \times 0.25 = 186.82 \text{ V}$$

b. From Eq. (14.2), the speed is

$$\omega = \frac{E_g}{K_v I_f} = \frac{186.82}{1.4 \times 2.26} = 59.05 \text{ rad/s or } 564 \text{ rpm}$$

c. Assuming lossless converters, the total input power from the supply is

$$P_i = V_a I_a + V_f I_f = 198.07 \times 45 + 396.14 \times 2.26 = 9808.4 \text{ W}$$

The input current of the armature converter for a highly inductive load is shown in Figure 14.13b and its rms value is $I_{sa} = I_a = 45$ A. The rms value of the input current of field converter is $I_{sf} = I_f = 2.26$ A. The effective rms supply current can be found from

$$I_s = (I_{sa}^2 + I_{sf}^2)^{1/2}$$

$$= (45^2 + 2.26^2)^{1/2} = 45.06 \text{ A}$$

and the input VA rating, $\text{VI} = V_s I_s = 440 \times 45.06 = 19{,}826.4$. Neglecting the ripples, the input power factor is approximately

$$\text{PF} = \frac{P_i}{\text{VI}} = \frac{9808.4}{19{,}826.4} = 0.495 \text{ (lagging)}$$

From Eq. (10.7),

$$\text{PF} = \left(\frac{2\sqrt{2}}{\pi}\right) \cos \alpha_a = \left(\frac{2\sqrt{2}}{\pi}\right) \cos 60° = 0.45 \text{ (lagging)}$$

If the polarity of the motor back emf in Example 14.4 is reversed by reversing the polarity of the field current, determine (a) the delay angle of the armature circuit converter, α_a, to maintain the armature current constant at the same value of $I_a = 45$ A; and (b) the power fed back to the supply due to regenerative braking of the motor.

Solution

a. From part (a) of Example 14.4, the back emf at the time of polarity reversal is $E_g = 186.82$ V and after polarity reversal $E_g = -186.82$ V. From Eq. (14.3),

$$V_a = E_g + I_a R_a = -186.82 + 45 \times 0.25 = -175.57 \text{ V}$$

From Eq. (14.20),

$$V_a = \frac{2V_m}{\pi} \cos \alpha_a = \frac{2 \times 622.25}{\pi} \cos \alpha_a = -175.57 \text{ V}$$

and this yields the delay angle of the armature converter as $\alpha_a = 116.31°$.

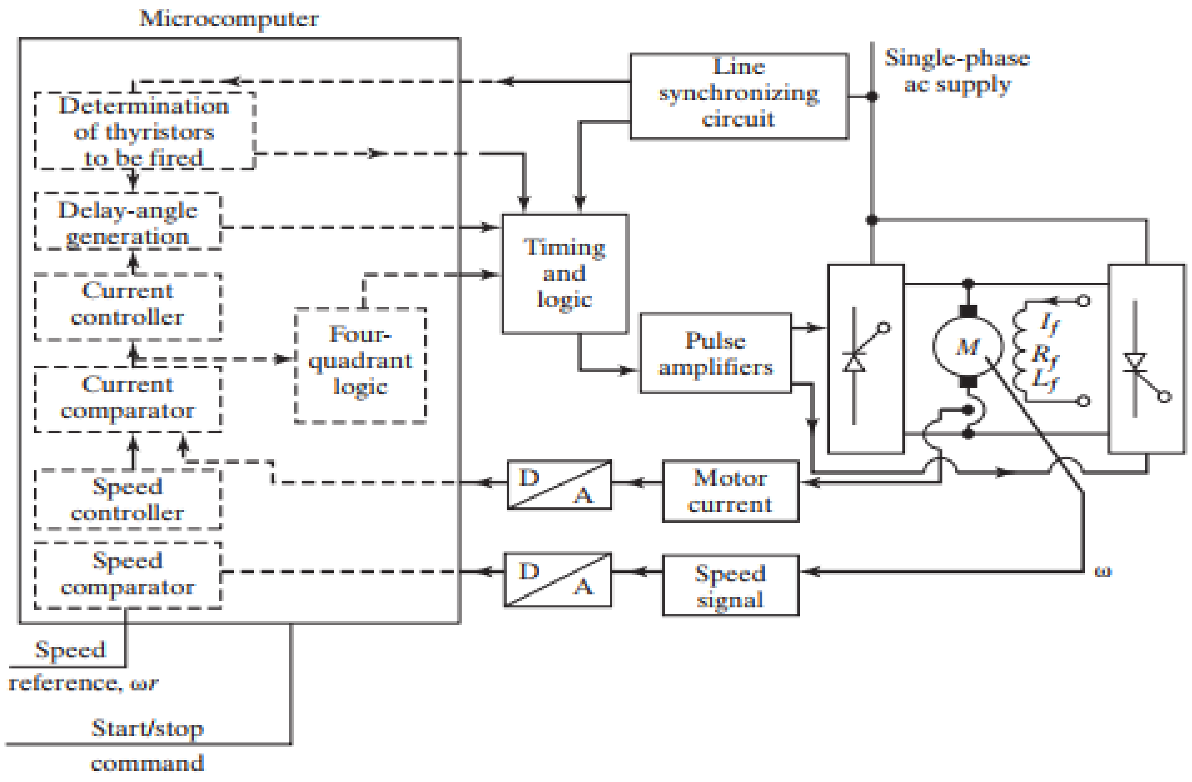
b. The power fed back to the supply is $P_a = V_a I_a = 175.57 \times 45 = 7900.7$ W.

Microcomputer Control of Dc Drives

The analog control scheme for a converter-fed dc motor drive can be implemented by hardwired electronics. An analog control scheme has several disadvantages: nonlinearity of speed sensor, temperature dependency, drift, and offset. Once a control circuit is built to meet certain performance criteria, it may require major changes in the hardwired logic circuits to meet other performance requirements.

A microcomputer control reduces the size and costs of hardwired electronics, improving reliability and control performance. This control scheme is implemented in the software and is flexible to change the control strategy to meet different performance characteristics or to add extra control features. A micro-computer control system can also perform various desirable functions: on and off of the main power supply, start and stop of the drive, speed control, current control, monitoring the control variables, initiating protection and trip circuit, diagnostics for built-in fault finding, and communication with a supervisory central computer. Figure 14.38 shows a schematic diagram for a mThe speed signal is fed into the microcomputer using an analog-to-digital (A/D) converter. To limit the armature current of the motor, an inner current-control loop is used. The armature current signal can be fed into the microcomputer through an A/D converter or by sampling the armature current. The line synchronizing circuit is required to synchronize the generation of the firing pulses with the supply line frequency. Although the microcomputer can perform the functions of gate pulse generator and logic circuit, these are shown outside the microcomputer. The pulse amplifier provides the necessary isolation and produces gate pulses of required magnitude and duration. A microprocessor-controlled drive has become a norm. Analog control has become almost obsolete. icrocomputer control of a converter-fed four-quadrant dc drive.

The speed signal is fed into the microcomputer using an analog-to-digital (A/D) converter. To limit the armature current of the motor, an inner current-control loop is used. The armature current signal can be fed into the microcomputer through an A/D converter or by sampling the armature current. The line synchronizing circuit is required to synchronize the generation of the firing pulses with the supply line frequency. Although the microcomputer can perform the functions of gate pulse generator and logic circuit, these are shown outside the microcomputer. The pulse amplifier provides the necessary isolation and produces gate pulses of required magnitude and duration. A microprocessor-controlled drive has become a norm. Analog control has become
almost obsolete.



Schematic diagram of computer-controlled four-quadrant dc drive.

College of Electronics Engineering

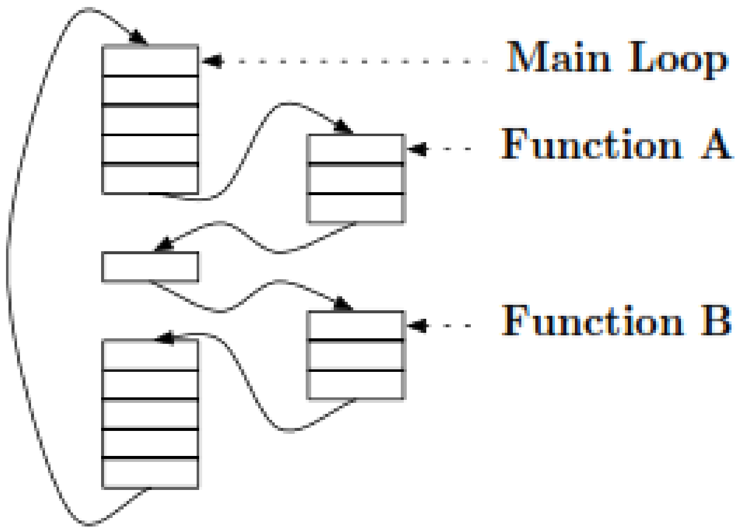
Systems and Control Engineering Department

Embedded Systems
4th Year

Interrupt Processing

Introduction

- Any time a program needed to retrieve(call back) an input signal, it did so by repeatedly checking the desired information source, a process called ***polling.*** For Example:



- Each block in the figure is meant to indicate a **single instruction**, and the solid directional lines show the path that the **Program Counter (PC)** follows.

- As can be seen in the example, the program executes Main Loop which makes two different function calls, one to **Function A** and one to **Function B.**

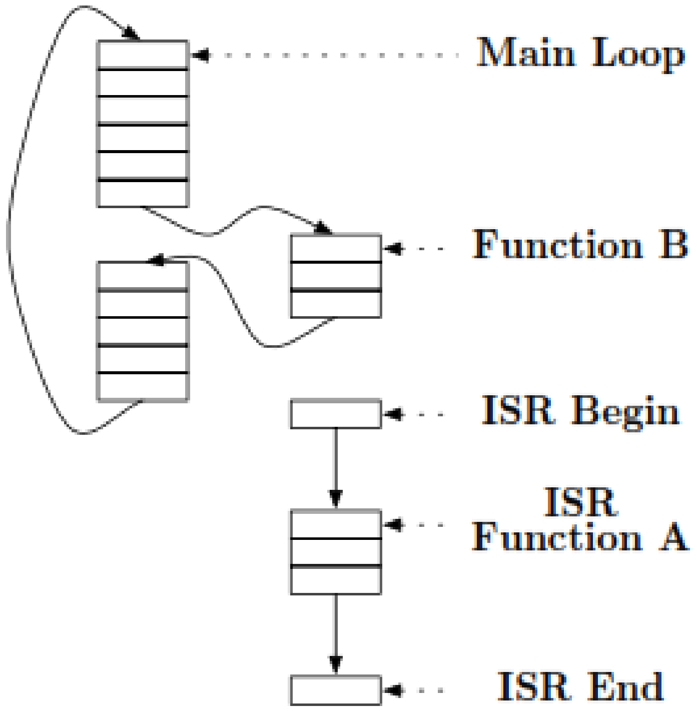
- While **polling** works fine, there are a couple of reasons to avoid them.
  - **First**, performance can suffer in more complicated embedded systems which require input from **many sources.**

    - (For example, a high-tier cell-phone has to monitor an entire key-pad, menu buttons, ambient light sensor,etc.)

    - When software is written such that the processor is constantly polling all inputs for activity, it will never have a chance to perform other operations that generally control the behavior of the device.

- **Second**, an important feature for many embedded systems is to periodically power down hardware, a process called **sleeping**
  - When a processor sleeps, it is able to conserve power, which leads to longer portable battery lives.
  - Many environments that involve polling don't allow the microprocessor to sleep at all, resulting in batteries draining much faster.

- An alternative version to the polling example is that **Function A** has been **moved** from the deterministic pathway of the polling software to an **isolated function** that gets called in response to an **interrupt condition**.



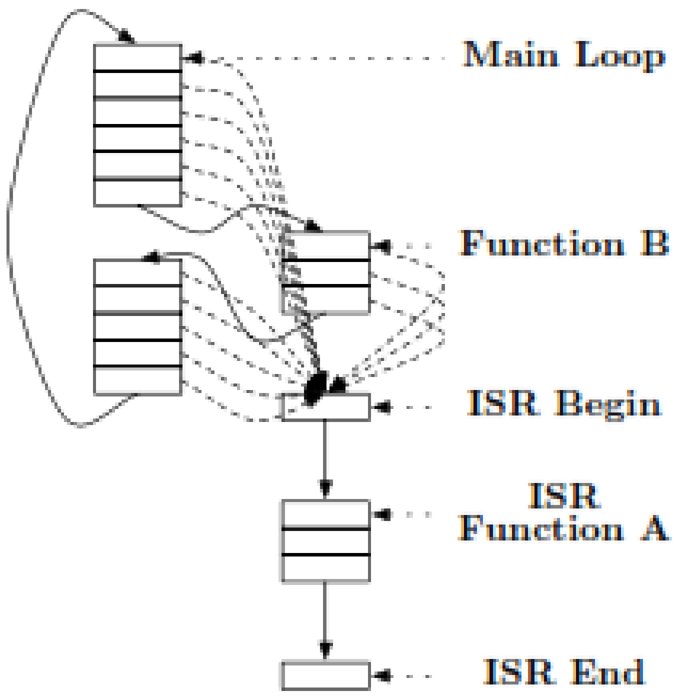
(a) Program with ISR

- **Interrupts** are signals used to notify the CPU that some new event has just occurred.
- Because most interrupt sources occur outside the CPU boundary, interrupts may be thought of as random signals that can occur at any point during the otherwise predictable flow of the primary algorithm.

- When an interrupt occurs, the CPU stops whatever it is doing and jumps to an associated **Interrupt Service Routine (ISR).**

Interrupt Service Routine (ISR)

- Is a special function written to handle the fact that the interrupt signal occurred.



(b) ISR called from anywhere

Interrupt Service Routine (ISR)

- In the polling example, **Function A** always occurred at the same time within the loop of processing**.**

- By contrast, the new example algorithm has moved **Function A,** processing outside the main algorithm's path, reducing the work constantly performed by the main program.

- The drawback is the added **complexity** of the fact that now **Function A** can be called at any time, and so software needs to be written with this behavior in mind.

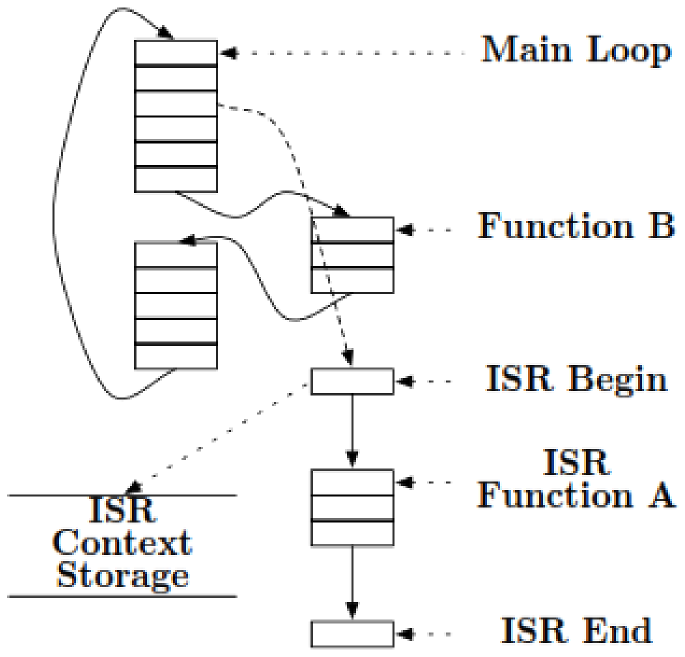
CONTEXT

- Typically, the software context may be specified by the set of **CPU registers**, including the PC (Program Counter) which points to the current assembly instruction.

- For the example interrupt-based algorithm to work properly, it is certainly necessary that **Main Loop** function as it did in the polling case.
  - This requires the current context to appear as though each assembly instruction is executed in the intended sequence.

- Interrupt processing is made possible by saving the context on ISR entry and restoring the context when the ISR completes.

- For example, suppose an interrupt occurs at the **third** (machine) instruction of **Main Loop**, as shown in figure

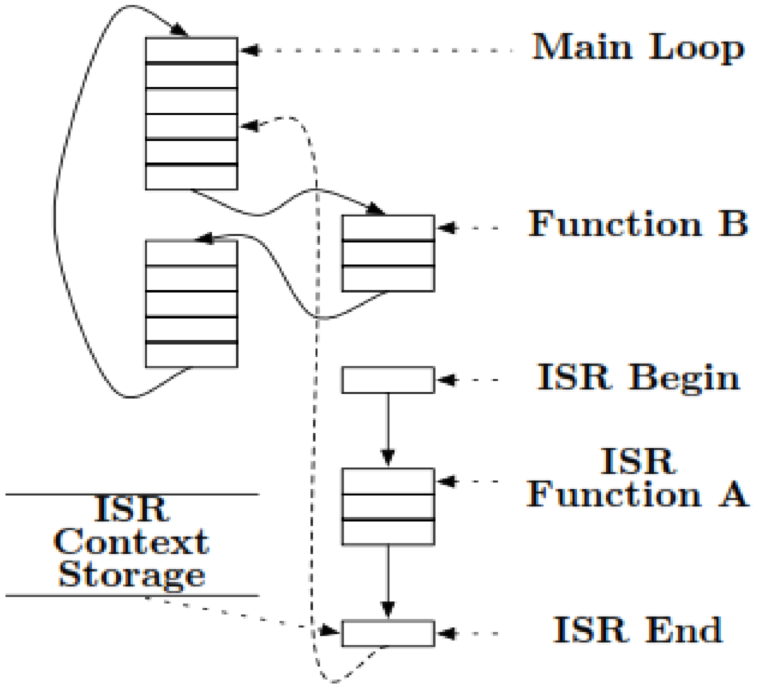


(a) ISR is called

- As a result, the address of the **fourth** (machine) instruction is stored as part of the current context.

- Then, any CPU registers used within **Function A** are also saved in memory before they are overwritten by **Function A's instructions.**

- After **Function A** finishes, the entire context is restored and the PC is reset to the fourth instruction of the **Main program**, as
- shown in Fig:



(b) ISR finishes

- The CPU begins executing the fourth instruction of **Main Loop with CPU** registers appearing as though the third instruction just finished because the context was saved prior to entering the ISR and then restored after the ISR completed.

ISR and main task communication

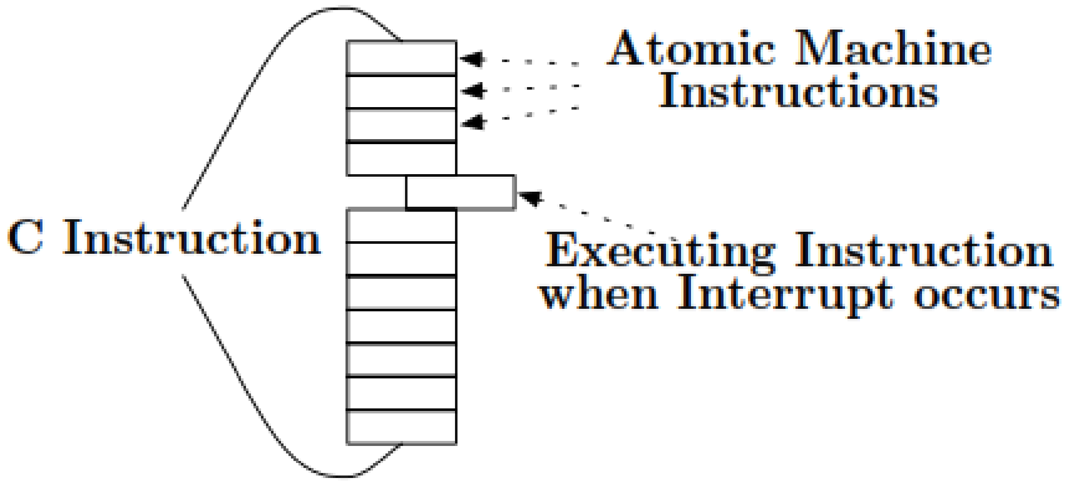
- The final general topic that needs attention is the method for an ISR to communicate with the main program.

- That is, the ISR is not a direct function call, so there are no parameters that can be passed in, nor is there a return statement that is able to pass back any values.

- Therefore, the only way for an ISR to communicate with the rest of the program is to use **shared memory**, such as global variables.

- However, because an interrupt can occur at any time, care must be taken when reading or writing variables that are accessed within ISRs.

- As seen in the figure, the reason is because individual C instructions are often composed of several machine instructions, especially when dealing with 16- and 32-bit variables.


Atomic Machine Instructions
C Instruction
Executing Instruction when Interrupt occurs

  - So, it is likely the main program will be interrupted part-way through variable access.
  - **The associated ISR could change the contents of the variable before it returns to the previous location.**
  - At that time, the main program would continue accessing the variable, which is now different.

- The result is that the main program operates on a variable value that is half correct and half incorrect.

- In order to mitigate these problems, any memory that is **shared between ISRs and the main program need to be protected**.

- **The protection necessary is usually in the form of turning off global interrupts before accessing the shared variable.**

- When finished, the global interrupts need to be restored to their previous state.

College of Electronics Engineering

Systems and Control Engineering Department

Embedded Systems
4th Year

Interrupt Processing

Example of interrupts

```
const byte LED = 13;
const byte BUTTON = 2;

// Interrupt Service Routine (ISR)
void switchPressed ()
{
  if (digitalRead (BUTTON) == HIGH)
    digitalWrite (LED, HIGH);
  else
    digitalWrite (LED, LOW);
}  // end of switchPressed

void setup ()
{
  pinMode (LED, OUTPUT);  // so we can update the LED
  digitalWrite (BUTTON, HIGH);  // internal pull-up resistor
  attachInterrupt (digitalPinToInterrupt (BUTTON), switchPressed, CHANGE);
}  // end of setup

void loop ()
{
  // loop doing nothing
}
```

- This example shows how, even though the main loop is doing nothing, you can turn the LED on pin 13 on or off, if the switch on pin D2 is pressed.
- To test this, just connect a wire (or switch) between D2 and Ground. The internal pull-up (enabled in setup) forces the pin HIGH normally.
- When grounded, it becomes LOW. The change in the pin is detected by a **CHANGE** interrupt, which causes the Interrupt Service Routine (ISR) to be called.
- In a more complicated example, the main loop might be doing something useful, like taking temperature readings, and allow the interrupt handler to detect a button being pushed.

DigitalPinToInterrupt function

- To simplify converting interrupt vector numbers to pin numbers you can call the function **digitalPinToInterrupt**, passing a pin number.
- It returns the appropriate interrupt number.
- For example, on the Uno, pin D2 on the board is interrupt 0 (INT0_vect from the table below).
- Thus these two lines have the same effect:

```
attachInterrupt (0, switchPressed, CHANGE);    // that is, for pin D2
attachInterrupt (digitalPinToInterrupt (2), switchPressed, CHANGE);
```

- However the second one is easier to read and more portable to different Arduino types.

Available interrupts

Below is a list of interrupts, in priority order, for the Atmega328:

1	Reset	
2	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
5	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
6	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)
7	Watchdog Time-out Interrupt	(WDT_vect)
8	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	Timer/Counter0 Compare Match A	(TIMER0_COMPA_vect)
16	Timer/Counter0 Compare Match B	(TIMER0_COMPB_vect)
17	Timer/Counter0 Overflow	(TIMER0_OVF_vect)
18	SPI Serial Transfer Complete	(SPI_STC_vect)
19	USART Rx Complete	(USART_RX_vect)
20	USART, Data Register Empty	(USART_UDRE_vect)
21	USART, Tx Complete	(USART_TX_vect)
22	ADC Conversion Complete	(ADC_vect)
23	EEPROM Ready	(EE_READY_vect)
24	Analog Comparator	(ANALOG_COMP_vect)
25	2-wire Serial Interface (I2C)	(TWI_vect)
26	Store Program Memory Ready	(SPM_READY_vect)

Summary of interrupts

The main reasons you might use interrupts are:

- To detect pin changes (eg. rotary encoders, button presses)
- Watchdog timer (eg. if nothing happens after 8 seconds, interrupt me)
- Timer interrupts - used for comparing/overflowing timers
- SPI data transfers
- I2C data transfers
- USART data transfers
- ADC conversions (analog to digital)
- EEPROM ready for use
- Flash memory ready.

The "data transfers" can be used to let a program do something else while data is being sent or received on the serial port, SPI port, or I2C port.

Wake the processor

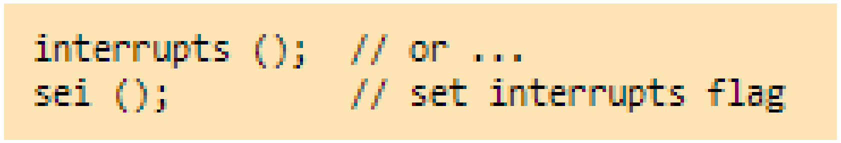
- **External interrupts**, **pin-change interrupts**, and the **watchdog timer interrupt**, can also be used to wake the processor up.
- This can be very handy, as in sleep mode the processor can be configured to use a lot less power (eg. around 10 micro-amps).
- A **rising**, **falling**, or low-level interrupt can be used to wake up a gadget (eg. if you press a button on it), or a "watchdog timer" interrupt might wake it up periodically (eg. to check the time or temperature).
- Pin-change interrupts could be used to wake the processor if a key is pressed on a keypad, or similar.
- The processor can also be awoken by a timer interrupt (eg. a timer reaching a certain value, or overflowing) and certain other events, such as an incoming I2C message.

Enabling / disabling interrupts

- The "reset" interrupt cannot be disabled. However the other interrupts can be temporarily disabled **by clearing the interrupt flag.**

Enable interrupts

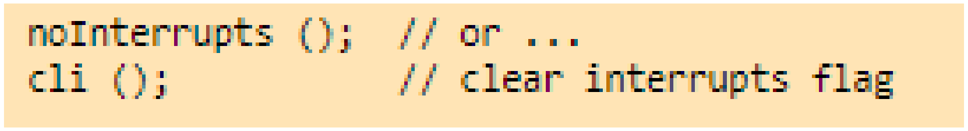
- You can enable interrupts with the function call "interrupts" or "sei" like this:



```
interrupts ();  // or ...
sei ();         // set interrupts flag
```

Disable interrupts

- If you need to disable interrupts you can "clear" the interrupt flag like this:



```
noInterrupts ();  // or ...
cli ();           // clear interrupts flag
```

College of Electronics Engineering

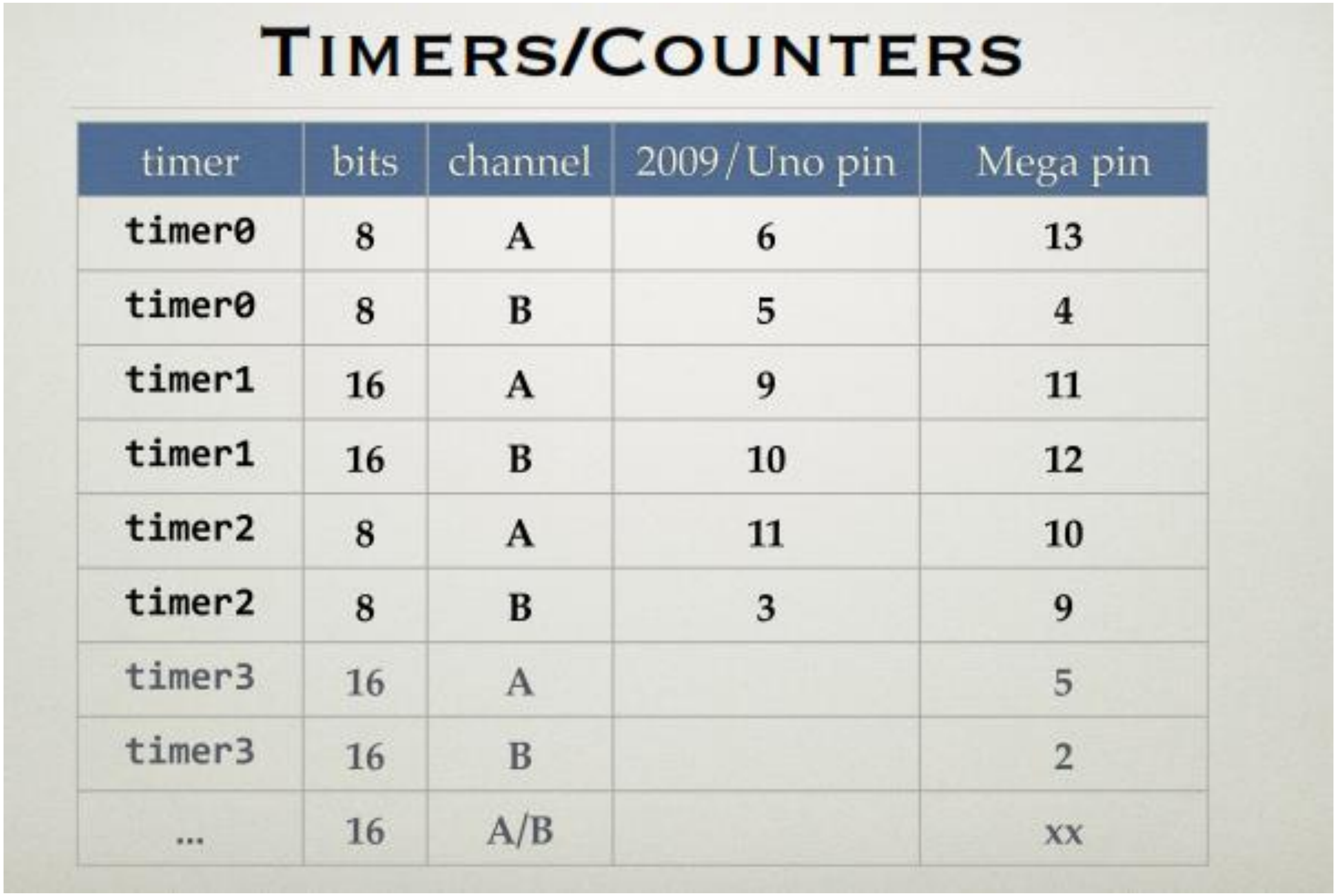
Systems and Control Engineering Department

Embedded Systems
4th Year

Counters & Timers

Existing Counter Use

- The millis() and micros() functions. [functions use **timer 0**)
- The analogWrite() function [Used with PWM].
- The Tone library.
- The Servo library [timer1].



Timers/Counters

timer	bits	channel	2009/Uno pin	Mega pin
timer0	8	A	6	13
timer0	8	B	5	4
timer1	16	A	9	11
timer1	16	B	10	12
timer2	8	A	11	10
timer2	8	B	3	9
timer3	16	A		5
timer3	16	B		2
...	16	A/B		xx

Initial Timer State

- Timers 1,2 set ready for analogWrite() PWM use.
- Set to 8-bit PWM **phase-correct***, with the clock pre-scaled to the system clock divided by 64.
- Timer0 set to **fast PWM** for use with the millis()/ micros()

Control Registers

- **T**imer/**C**ounter **C**ontrol **R**egister **A**: TCCRnA
- **T**imer/**C**ounter **C**ontrol **R**egister **B**: TCCRnB
- **T**imer **C**ou**NT**: TCNTn
- **O**utput **C**ompare **R**egister **A**: OCRnA
- **O**utput **C**ompare **R**egister **B**: OCRnB

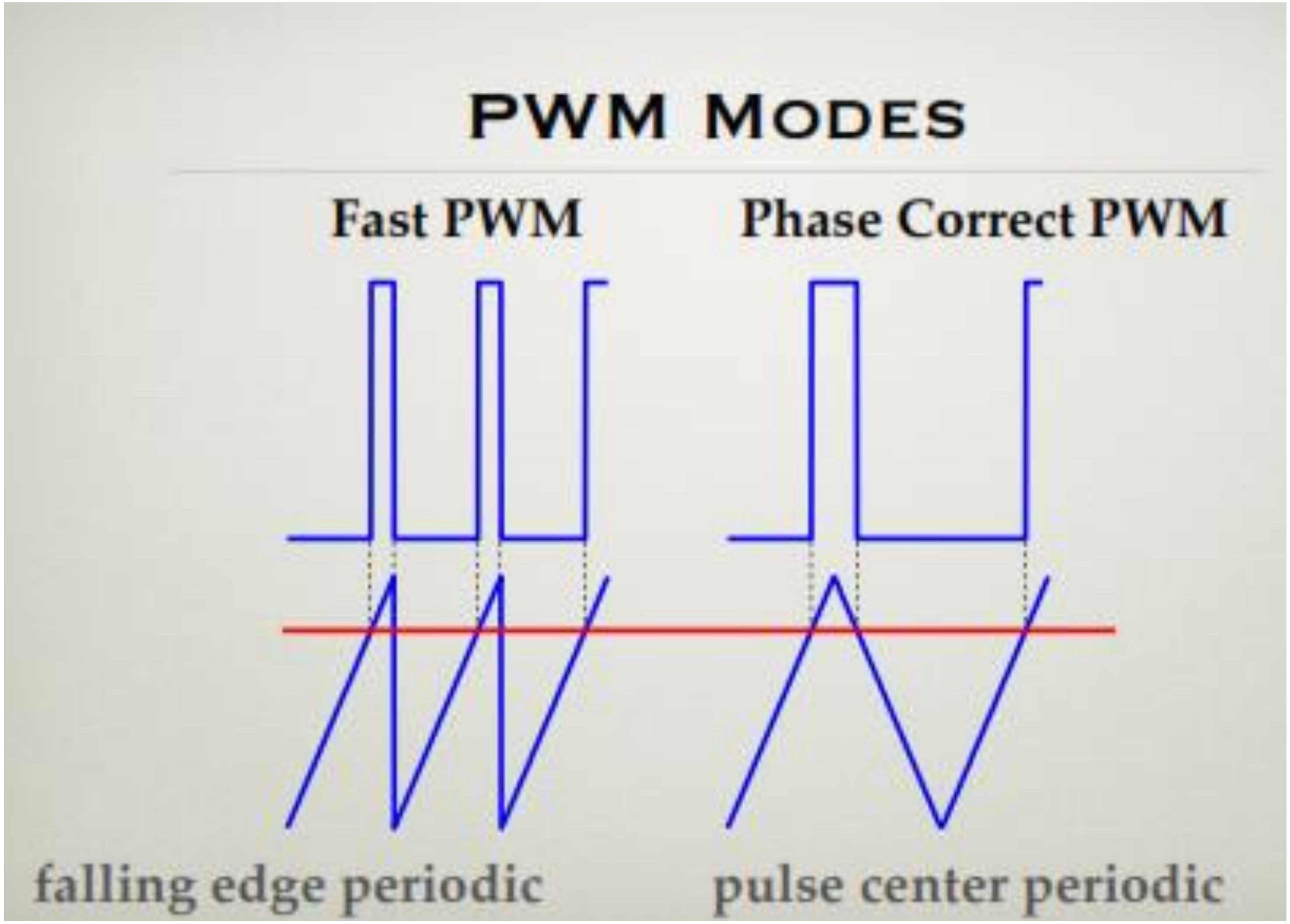
Timer 0

- The timer most used in applications.
- Can be used as:
  - simple counter [Used internally]
  - frequency generator (including PWM)
  - External source clock counter
  - Able to generate “tree” interrupts. *

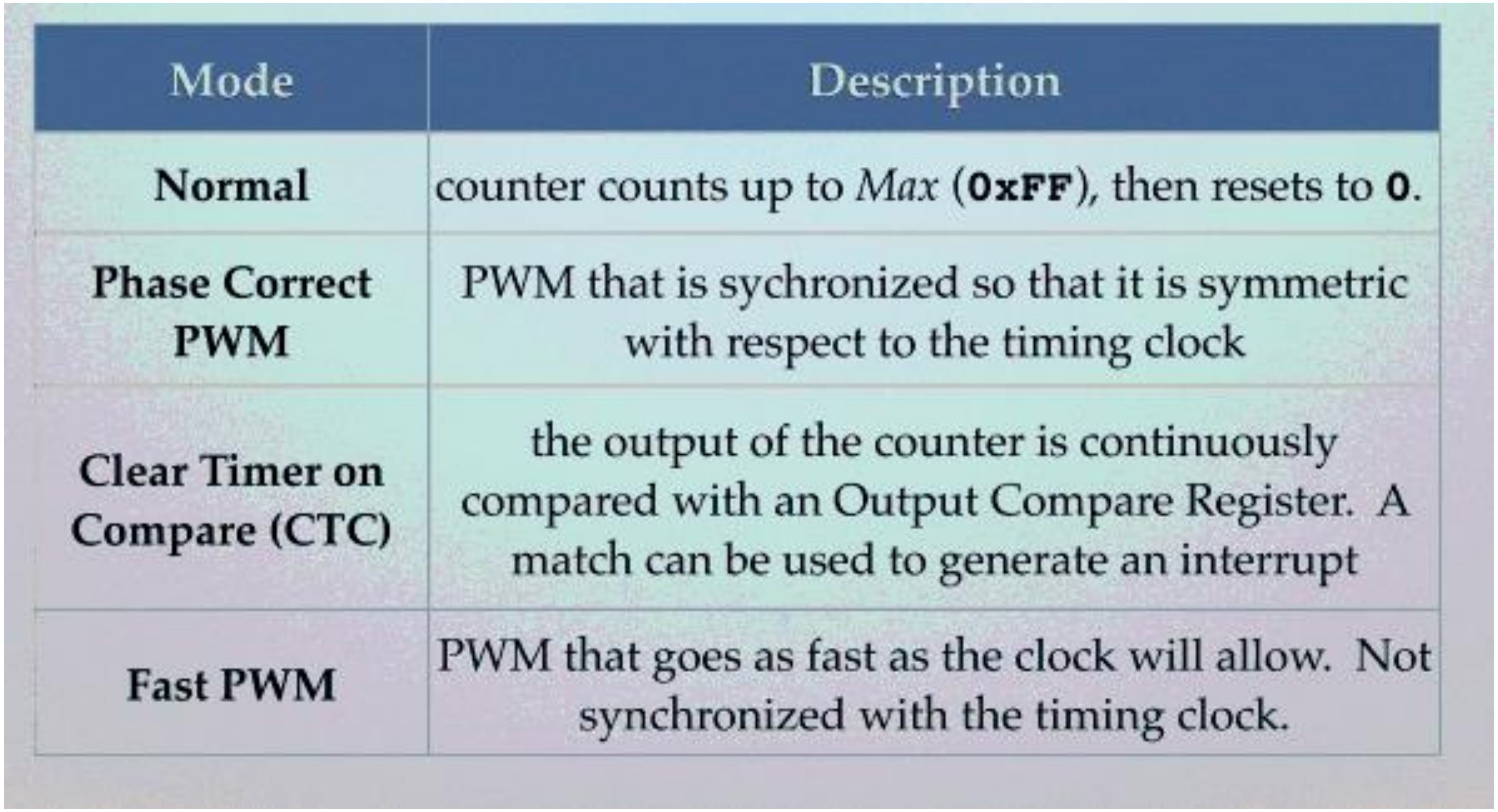
What do the Registers do?

- Set up the Mode of the timer/counter.
- Set up the various parameters of the timers/counters.
- Provide output waveforms.
- Provide options for comparing the waveform values with other values.
- **Note**: a timer is a counter with a clock input.

PWM Modes *

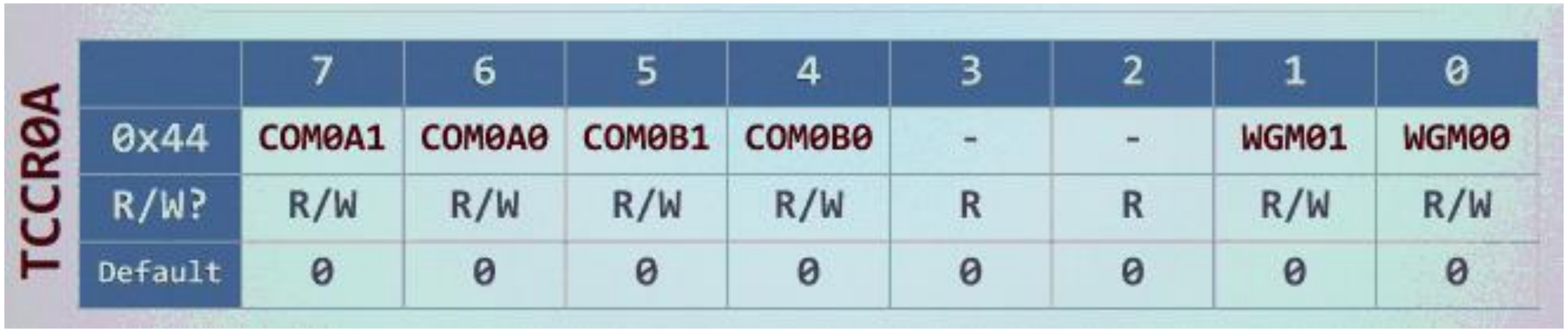

PWM Modes
Fast PWM
Phase Correct PWM
falling edge periodic
pulse center periodic

Timer/Counter Modes *



Mode	Description
Normal	counter counts up to *Max* (**`0xFF`**), then resets to **`0`**.
Phase Correct PWM	PWM that is sychronized so that it is symmetric with respect to the timing clock
Clear Timer on Compare (CTC)	the output of the counter is continuously compared with an Output Compare Register. A match can be used to generate an interrupt
Fast PWM	PWM that goes as fast as the clock will allow. Not synchronized with the timing clock.

Timer/Counter 0 Control Register A*



TCCR0A

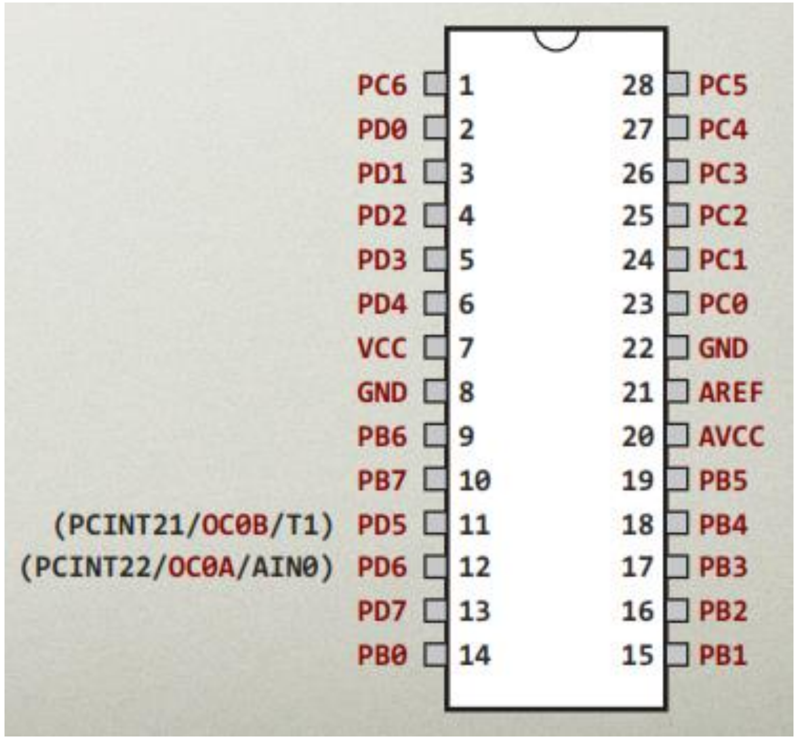
	7	6	5	4	3	2	1	0
0x44	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
R/W?	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

- The COM0x bits:

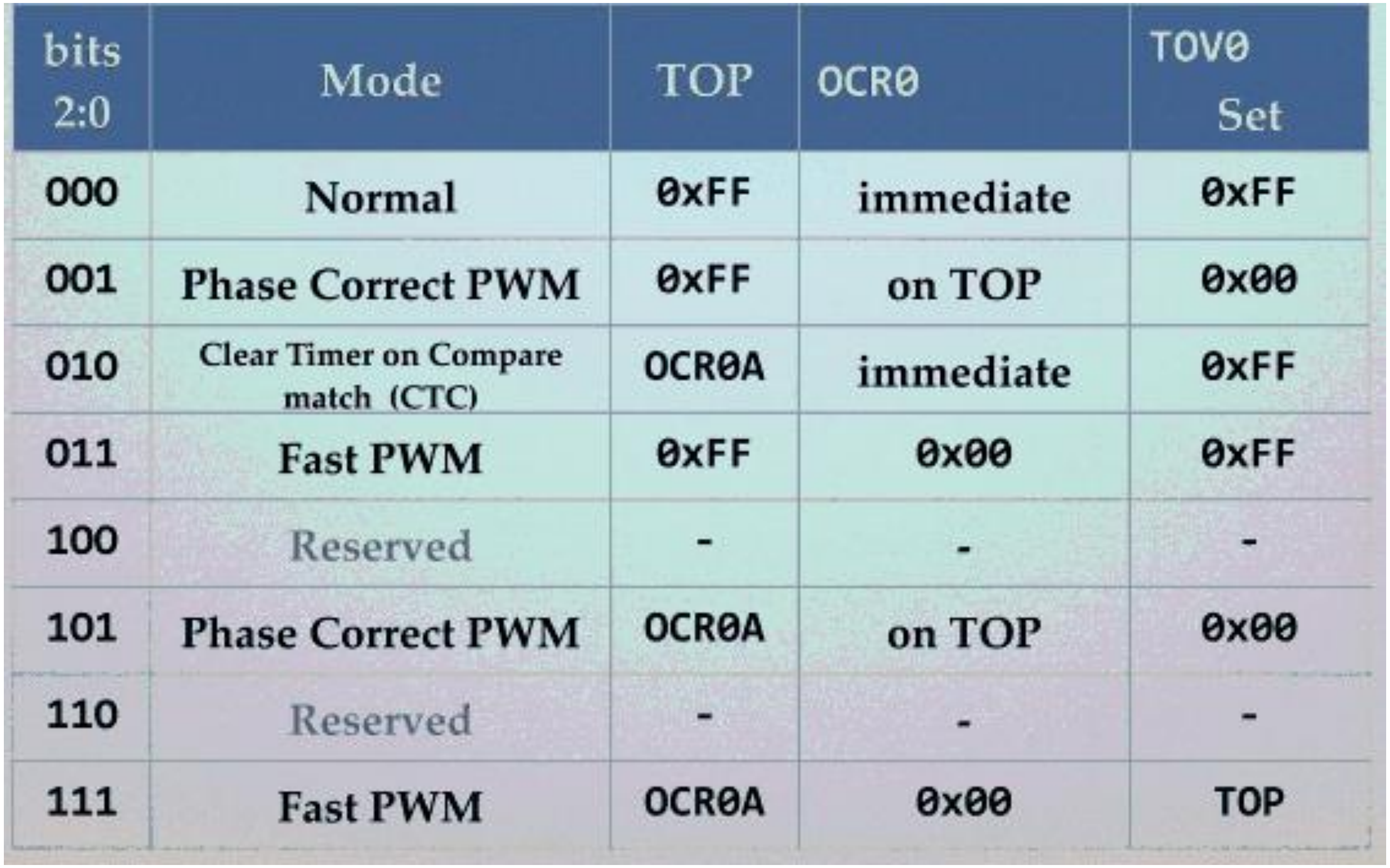
control the behavior of the output compare pins OC0A,B.

- The WGM0n bits.

control the function of the output compare pins OC0A,B

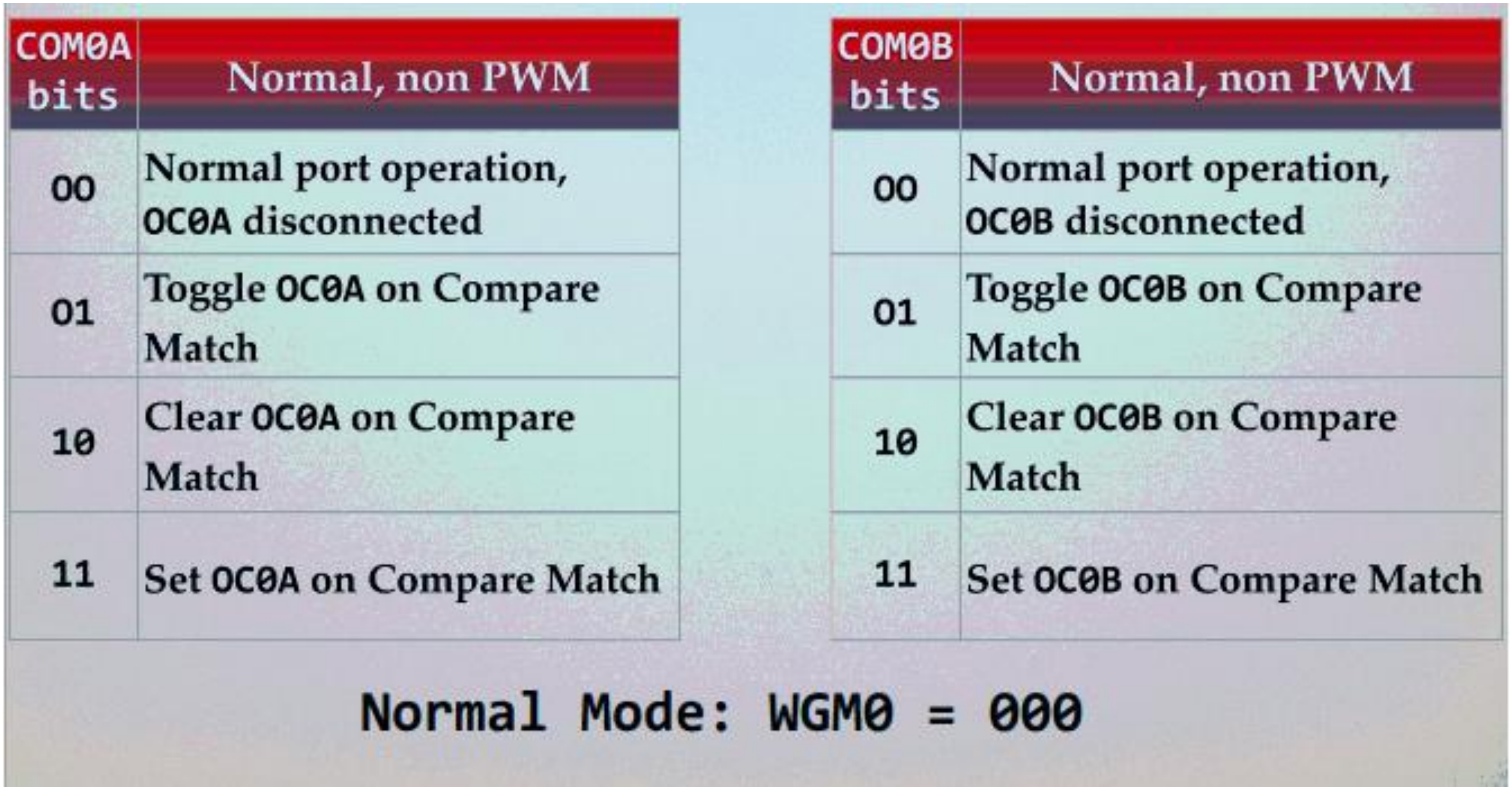


T/C0 Waveform Generation Mode Bits (WGM02:0)*



bits 2:0	Mode	TOP	OCR0	TOV0 Set
000	Normal	0xFF	immediate	0xFF
001	Phase Correct PWM	0xFF	on TOP	0x00
010	Clear Timer on Compare match (CTC)	OCR0A	immediate	0xFF
011	Fast PWM	0xFF	0x00	0xFF
100	Reserved	-	-	-
101	Phase Correct PWM	OCR0A	on TOP	0x00
110	Reserved	-	-	-
111	Fast PWM	OCR0A	0x00	TOP

Compare Output Mode T/C 0 Bits (COM0x) Normal Mode

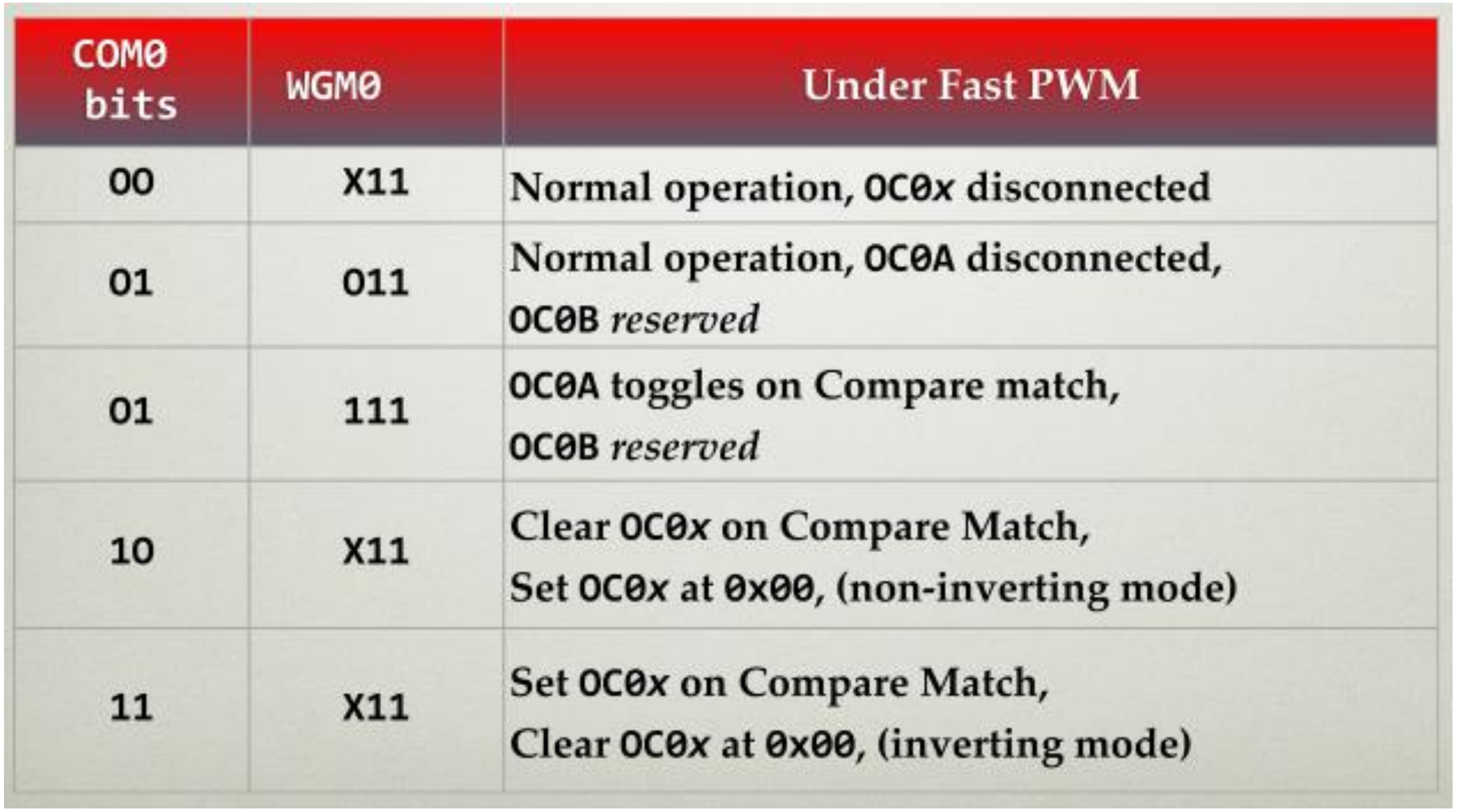


COM0A bits	Normal, non PWM
00	Normal port operation, OC0A disconnected
01	Toggle OC0A on Compare Match
10	Clear OC0A on Compare Match
11	Set OC0A on Compare Match

COM0B bits	Normal, non PWM
00	Normal port operation, OC0B disconnected
01	Toggle OC0B on Compare Match
10	Clear OC0B on Compare Match
11	Set OC0B on Compare Match

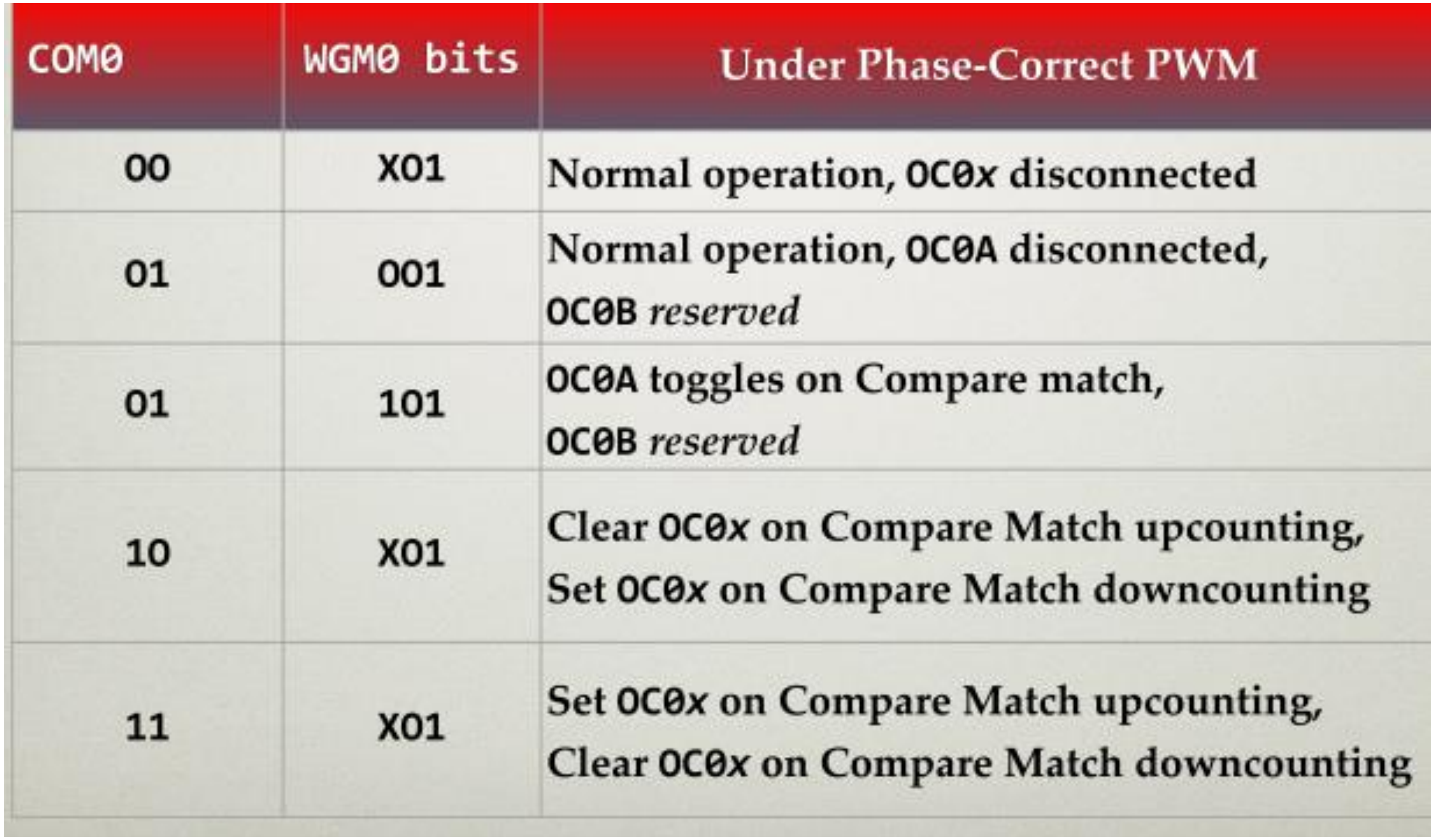
Normal Mode: WGM0 = 000

Compare Output Mode T/C 0 Bits (COM0x) Fast PWM



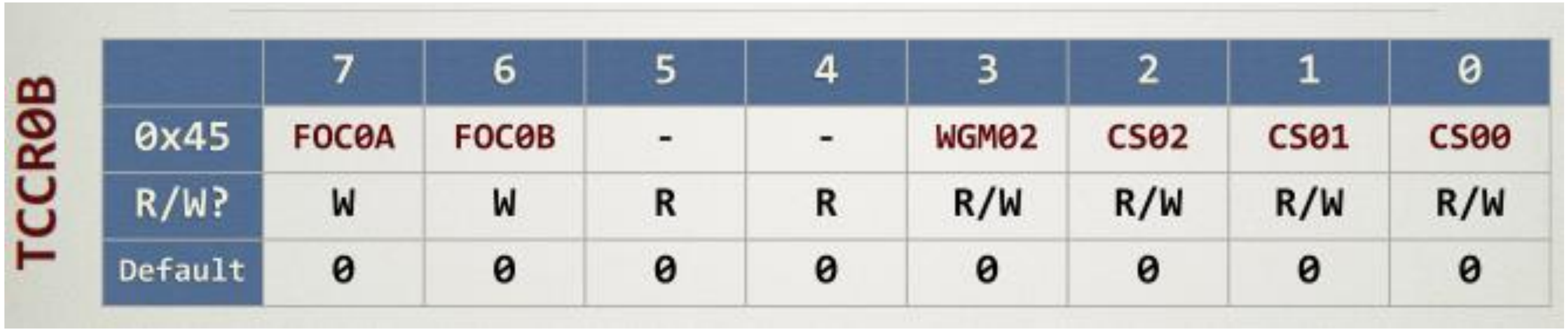
COM0 bits	WGM0	Under Fast PWM
00	X11	Normal operation, OC0x disconnected
01	011	Normal operation, OC0A disconnected, OC0B *reserved*
01	111	OC0A toggles on Compare match, OC0B *reserved*
10	X11	Clear OC0x on Compare Match, Set OC0x at 0x00, (non-inverting mode)
11	X11	Set OC0x on Compare Match, Clear OC0x at 0x00, (inverting mode)

Compare Output Mode T/C 0 Bits Phase-Correct PWM



COM0	WGM0 bits	Under Phase-Correct PWM
00	X01	Normal operation, OC0x disconnected
01	001	Normal operation, OC0A disconnected, OC0B *reserved*
01	101	OC0A toggles on Compare match, OC0B *reserved*
10	X01	Clear OC0x on Compare Match upcounting, Set OC0x on Compare Match downcounting
11	X01	Set OC0x on Compare Match upcounting, Clear OC0x on Compare Match downcounting

Timer/Counter 0 Control Register B



TCCR0B

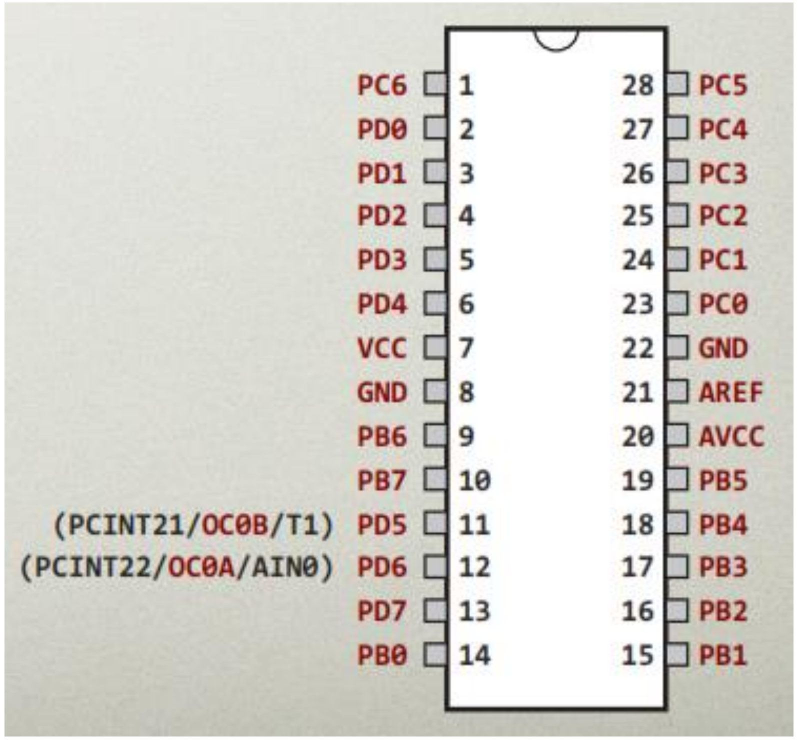
	7	6	5	4	3	2	1	0
0x45	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
R/W?	W	W	R	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- The FOC0x bits:

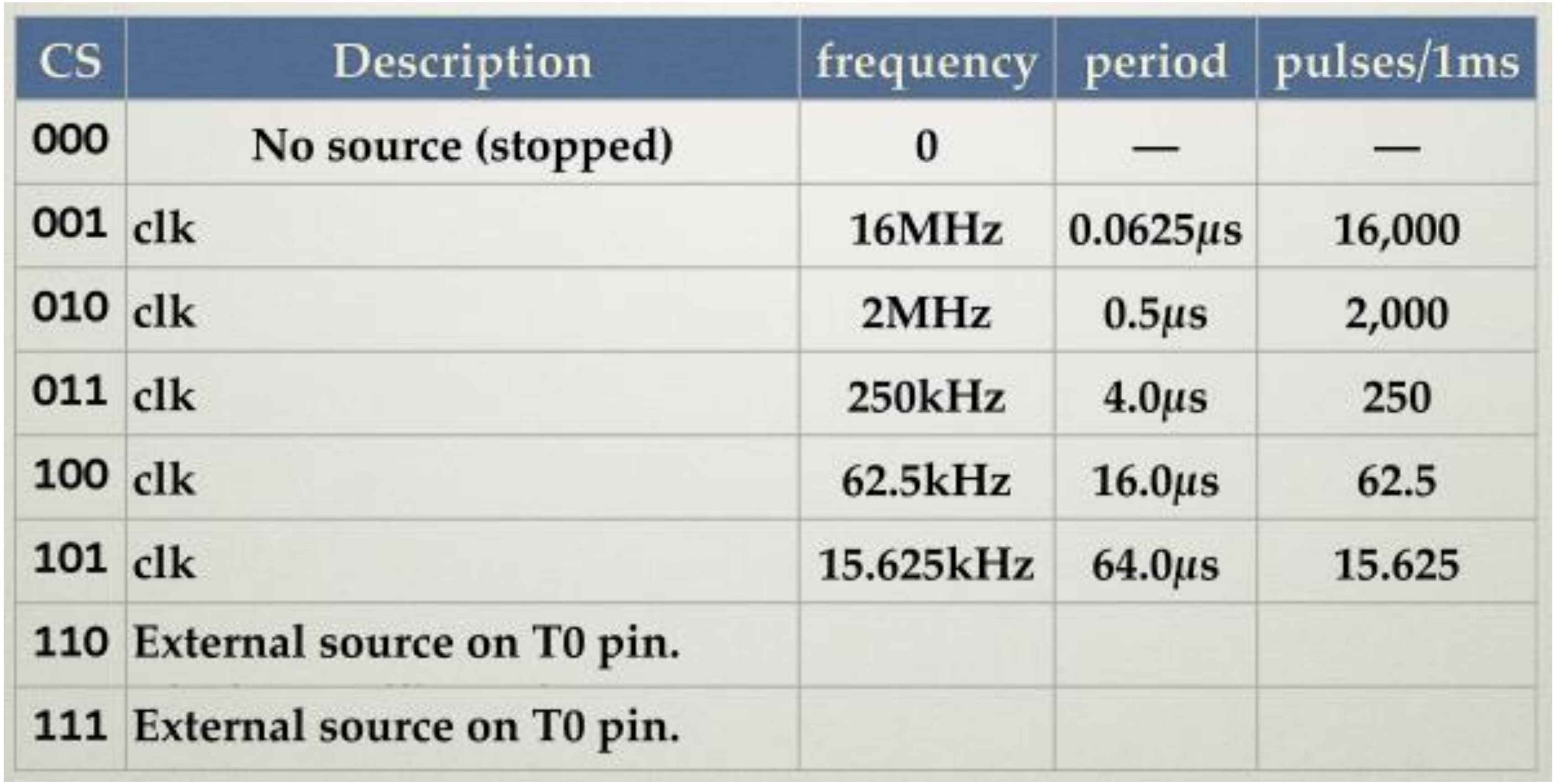
Force **o**utput **c**ompare for A or B for non-PWM modes.

- The CS0n bits.

Clock **S**elect bits for the clock source.



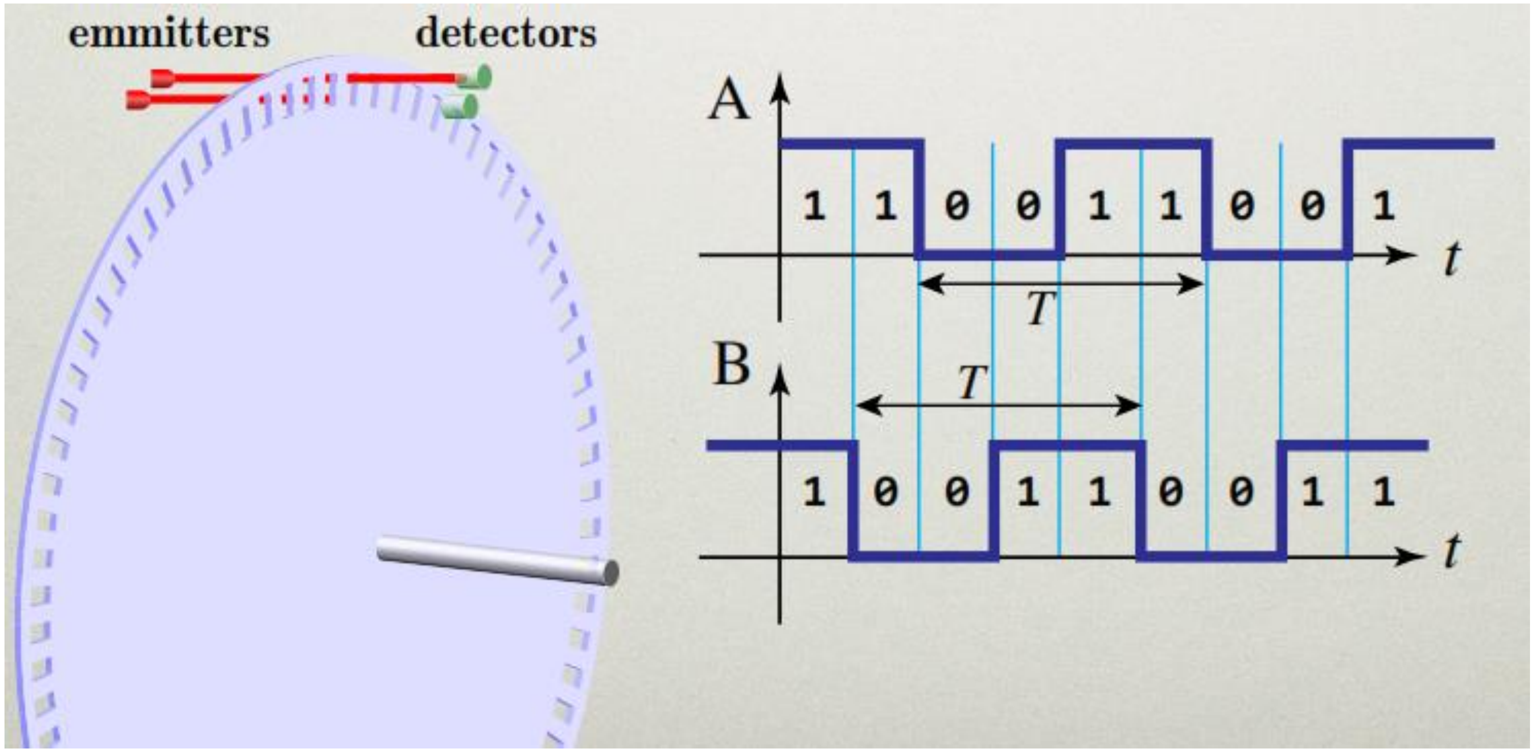
Timer 0 Clock Select



CS	Description	frequency	period	pulses/1ms
000	No source (stopped)	0	—	—
001	clk	16MHz	0.0625μs	16,000
010	clk	2MHz	0.5μs	2,000
011	clk	250kHz	4.0μs	250
100	clk	62.5kHz	16.0μs	62.5
101	clk	15.625kHz	64.0μs	15.625
110	External source on T0 pin.			
111	External source on T0 pin.			

Program Example

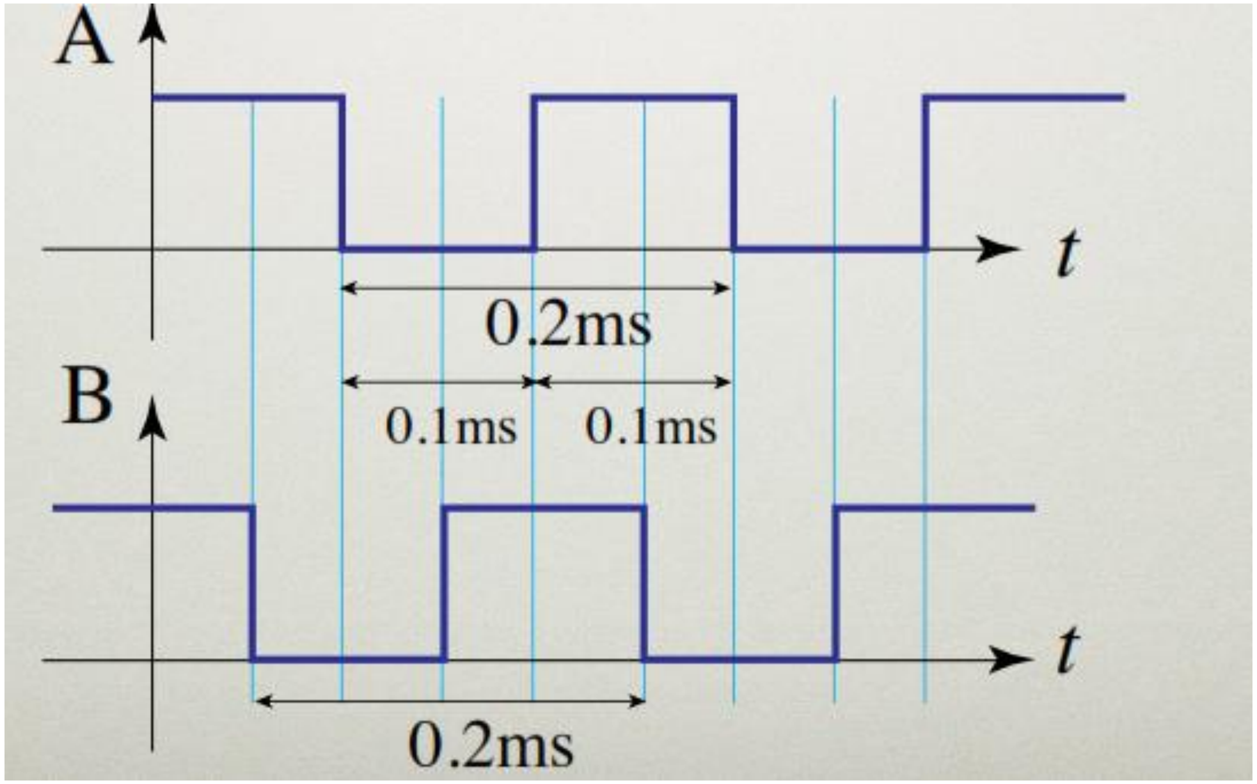
- Create a 500ppr quadrature encoder simulator that outputs quadrature signals out on pins from timer 0.
- Assume the rotation is a constant speed of 10rev/sec.



Program Example

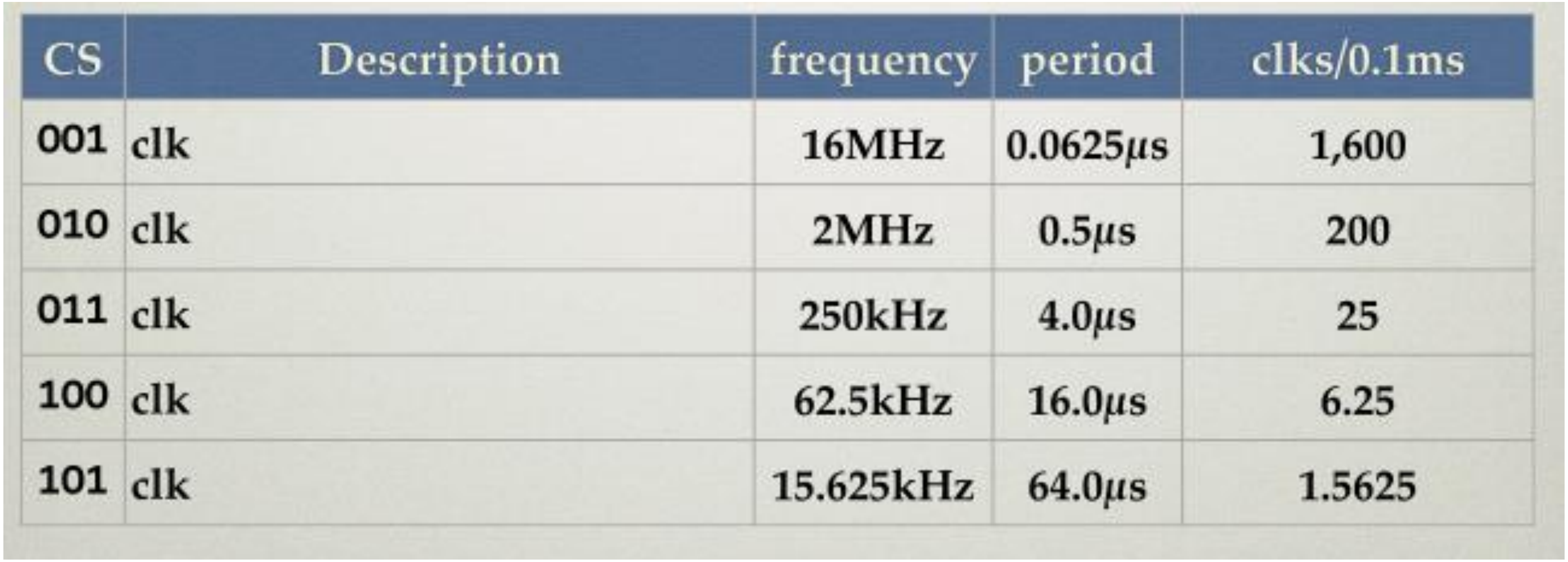
- A 500ppr quadrature encoder at a constant speed of 10rev/sec gives (500ppr)(10rev/sec) = 5000 pps (5.0 kHz).

- This corresponds to a period of 0.2ms, that toggles every 0.1ms.



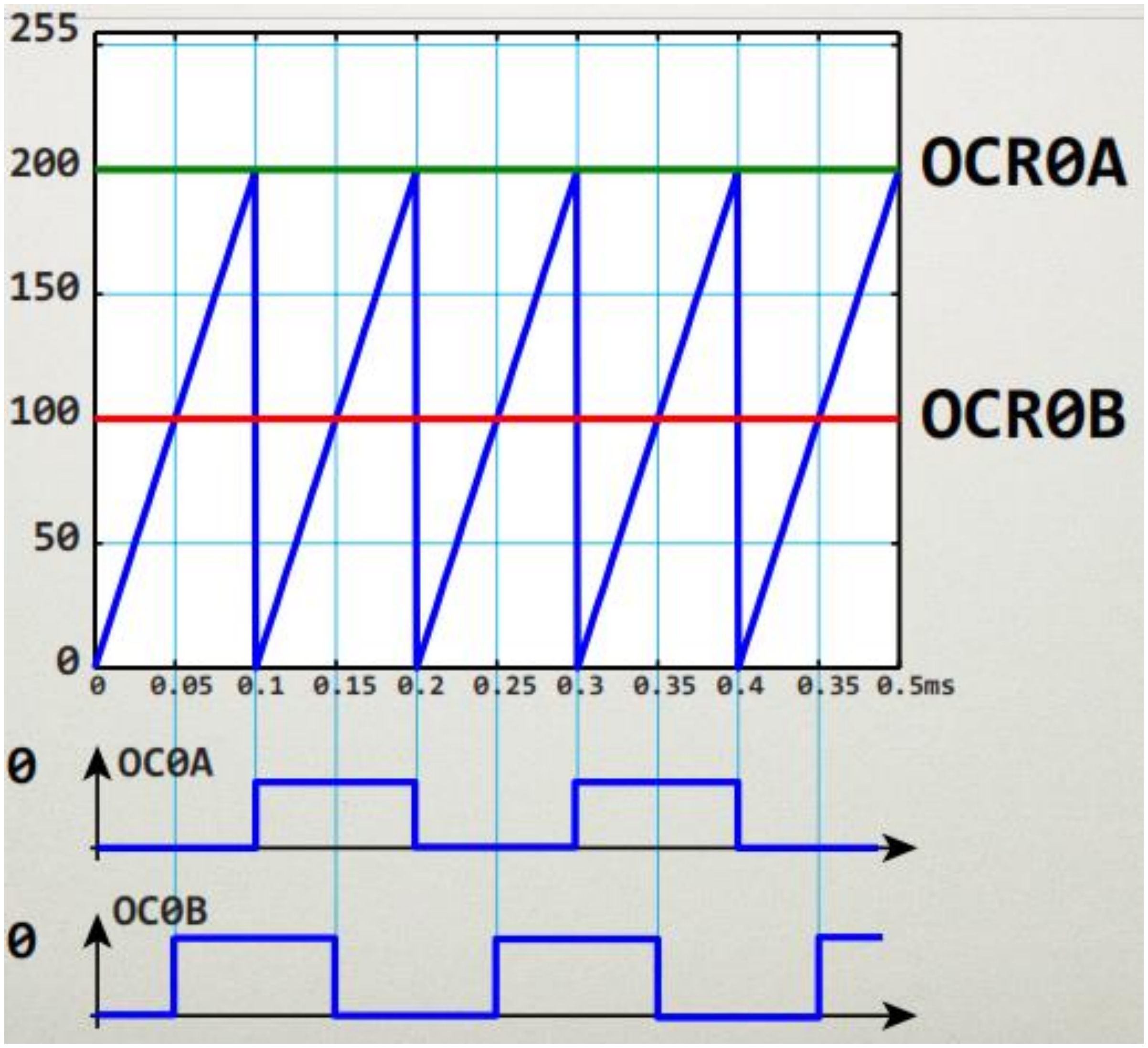
Program Example

- The clock select determines how many clock pulses every 0.1ms.
- Clock 0 has outputs on Arduino pins 5 & 6. (PORTD pins 5 & 6)



CS	Description	frequency	period	clks/0.1ms
001	clk	16MHz	0.0625μs	1,600
010	clk	2MHz	0.5μs	200
011	clk	250kHz	4.0μs	25
100	clk	62.5kHz	16.0μs	6.25
101	clk	15.625kHz	64.0μs	1.5625

Timing Diagram



(timer 0 resets when count hits OCR0A)

(toggles when timer 0 hits OCR0A)

(toggles when timer 0 hits OCR0B)

College of Electronics Engineering

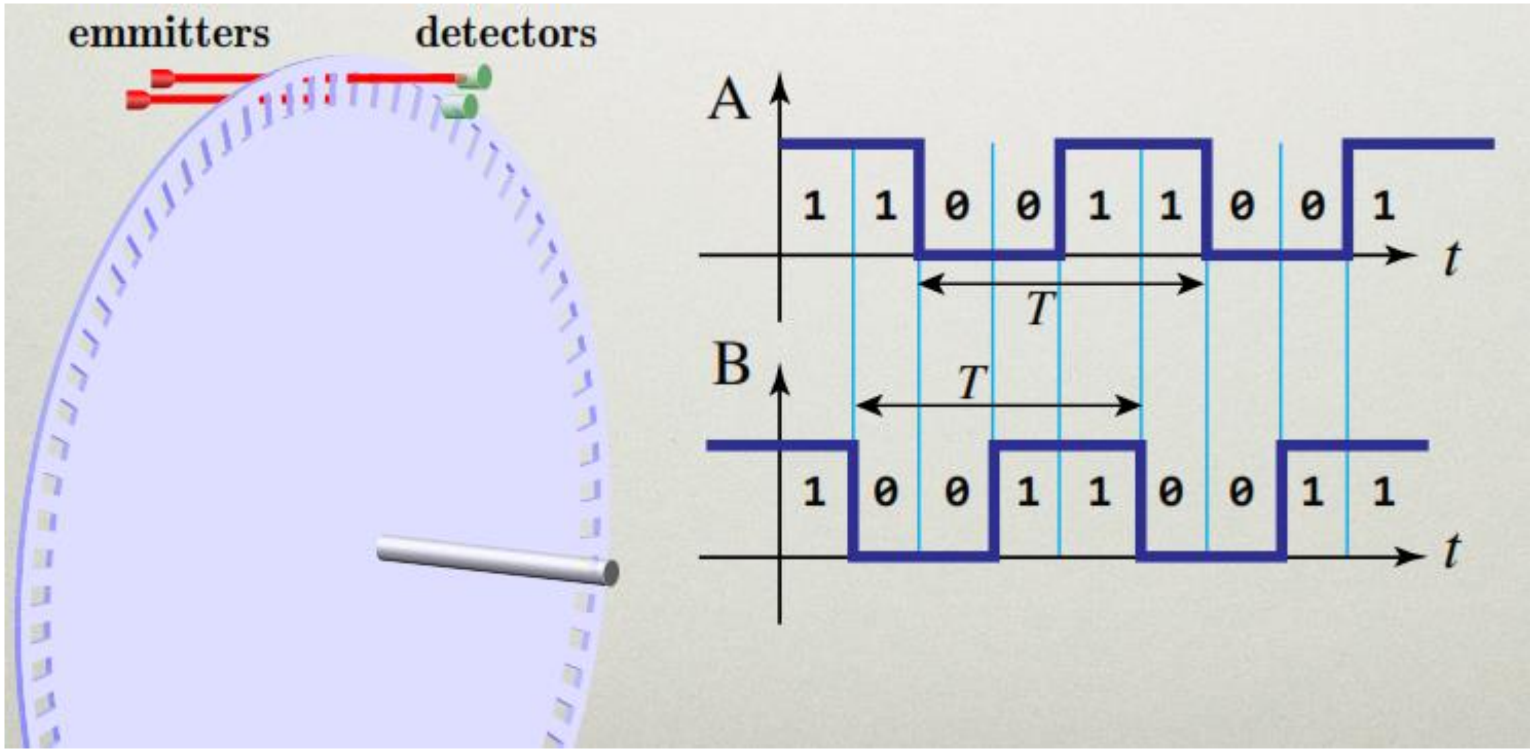
Systems and Control Engineering Department

Embedded Systems
4th Year

Counters & Timers

Program Example

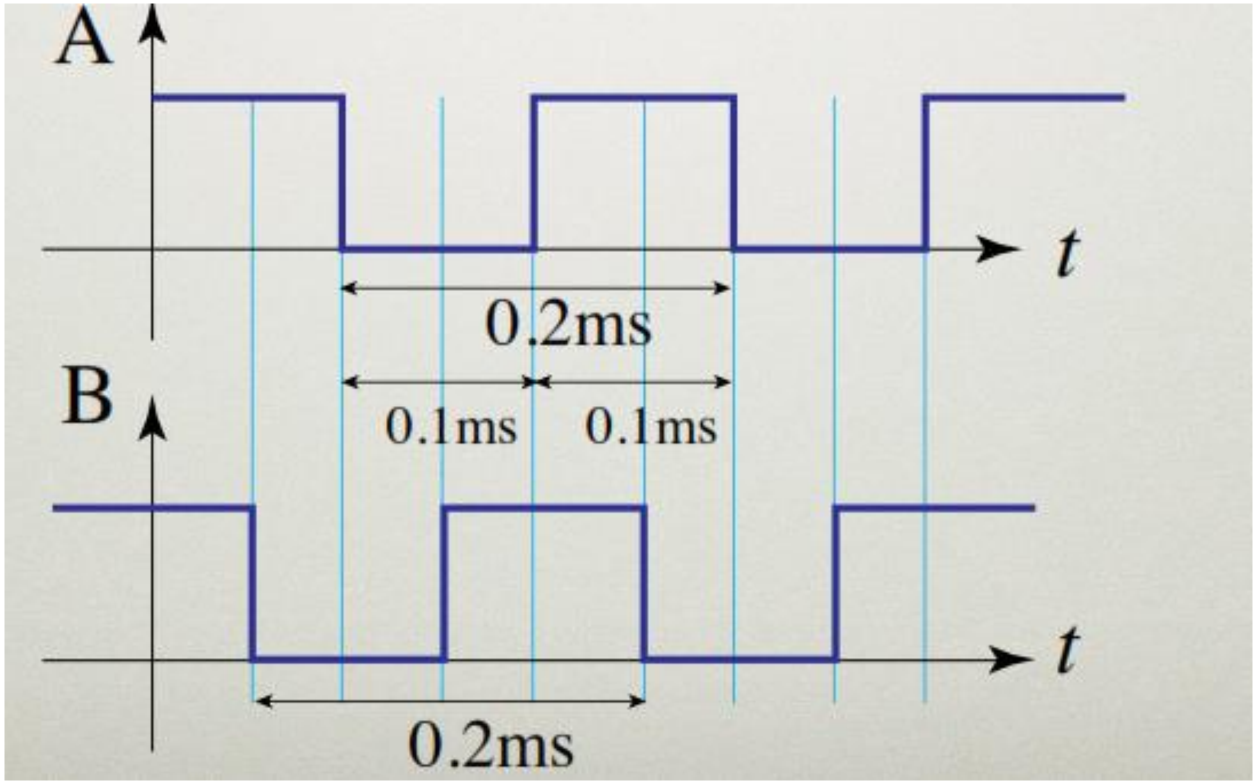
- Create a 500ppr quadrature encoder simulator that outputs quadrature signals out on pins from timer 0.
- Assume the rotation is a constant speed of 10rev/sec.



Program Example

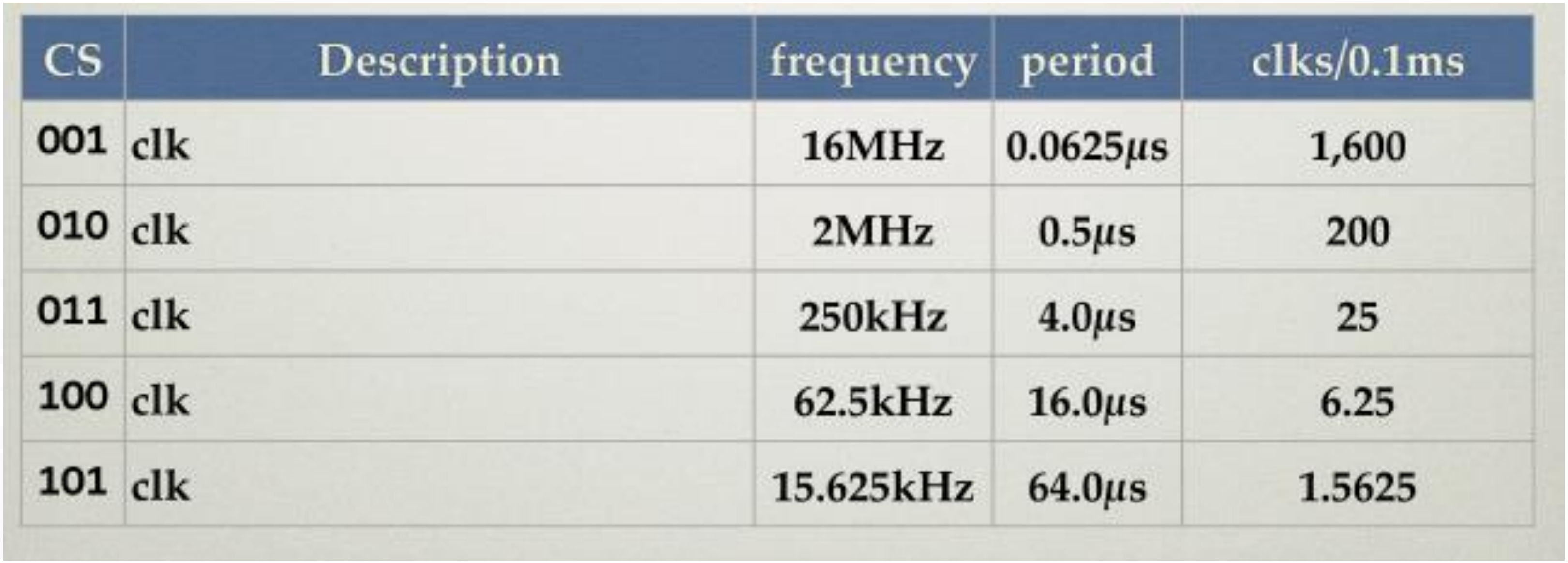
- A 500ppr quadrature encoder at a constant speed of 10rev/sec gives (500ppr)(10rev/sec) = 5000 pps (5.0 kHz).

- This corresponds to a period of 0.2ms, that toggles every 0.1ms.



Program Example

- The clock select determines how many clock pulses every 0.1ms.
- Clock 0 has outputs on Arduino pins 5 & 6. (PORTD pins 5 & 6)



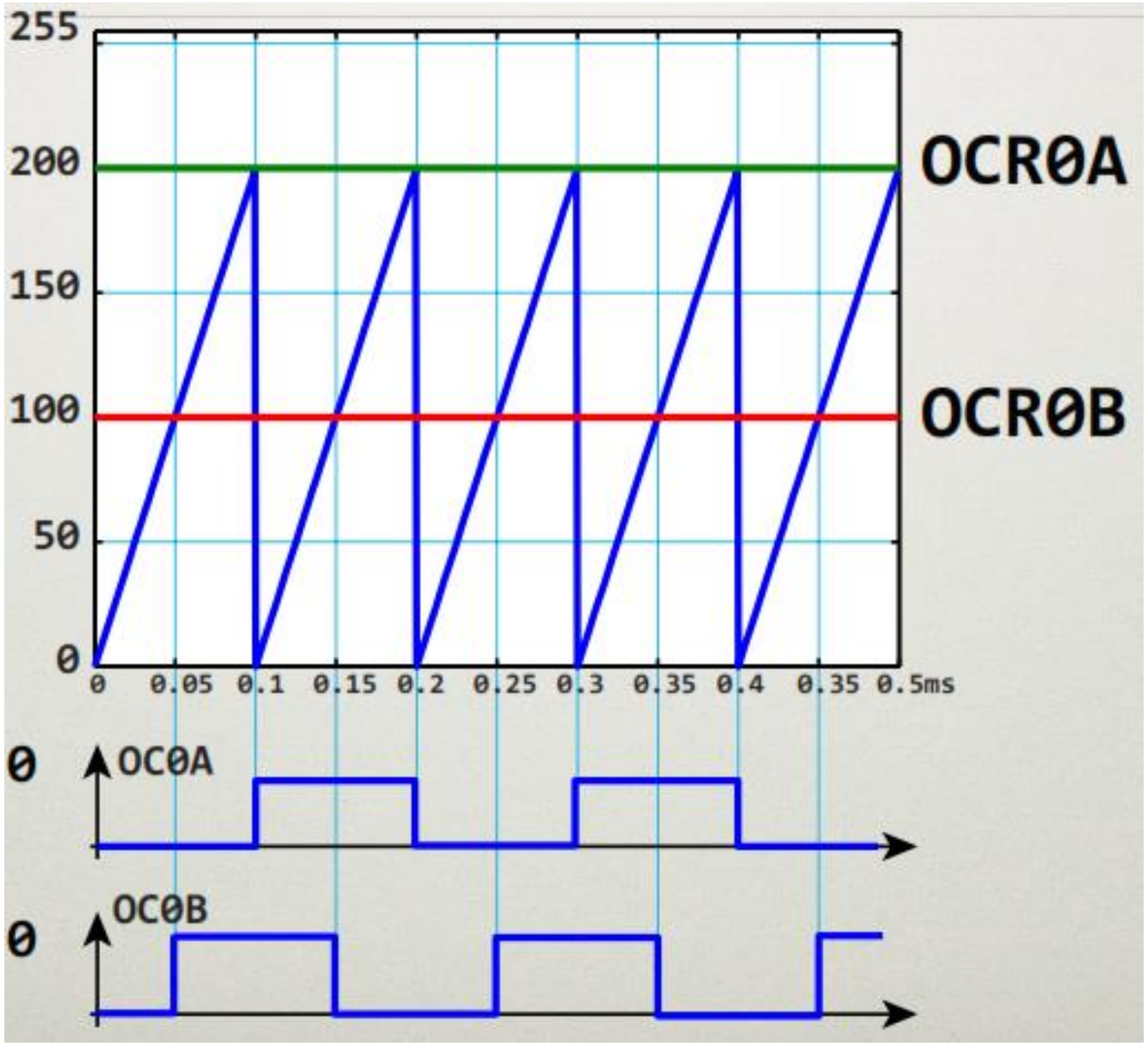
CS	Description	frequency	period	clks/0.1ms
001	clk	16MHz	0.0625μs	1,600
010	clk	2MHz	0.5μs	200
011	clk	250kHz	4.0μs	25
100	clk	62.5kHz	16.0μs	6.25
101	clk	15.625kHz	64.0μs	1.5625

Timing Diagram

(timer 0 resets when count hits OCR0A)

(toggles when timer 0 hits OCR0A)

(toggles when timer 0 hits OCR0B)



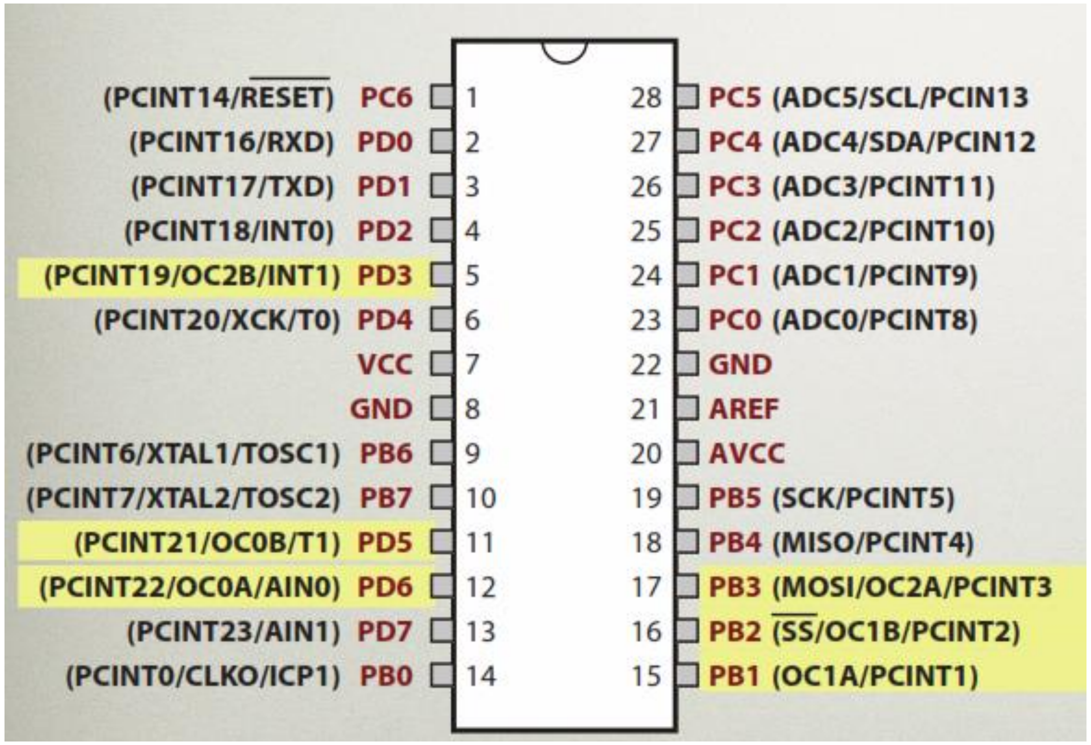
Registers & Parameters

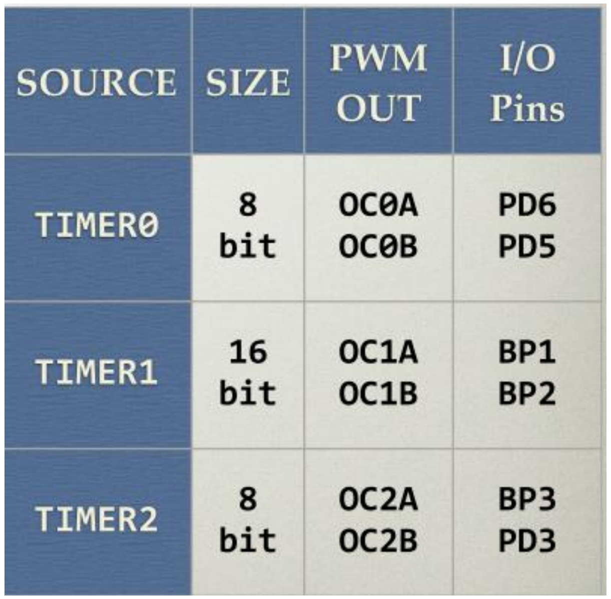
- OCR0A: 200
- OCR0B: 100
- COM0A: 01 (TCCR0A)
- COM0B: 01 (TCCR0A)
- WGM0: 010 (TCCR0A and TCCR0B)
- CS: 010 (TCCR0B)

Program Example

```
//Quadrature encoder simulator
void setup()
{
  PORTD &= 0b10011111;          //set pins 5,6 to zero
  DDRD |=(1<<6)|(1<<5);         //set pins 5,6 as outputs.

  //Initialize and setup the timer in CTC mode
  //and set pins PD5 and PD6 (OC0A, OC0B) to toggle

  TCNT0=0;                //initialize the timer 0 value
  OCR0A = 200;            //Set TOP value to OCR0A.
  OCR0B = 100;            //This TOP value determines phase of B

  TCCR0A |= (1<<COM0A0)|(1<<COM0B0)|(1<<WGM01);//WGM0: 010
  TCCR0B|=(1<<CS01);//CS: 010
}
void loop()
{
}
```

PWM Pins



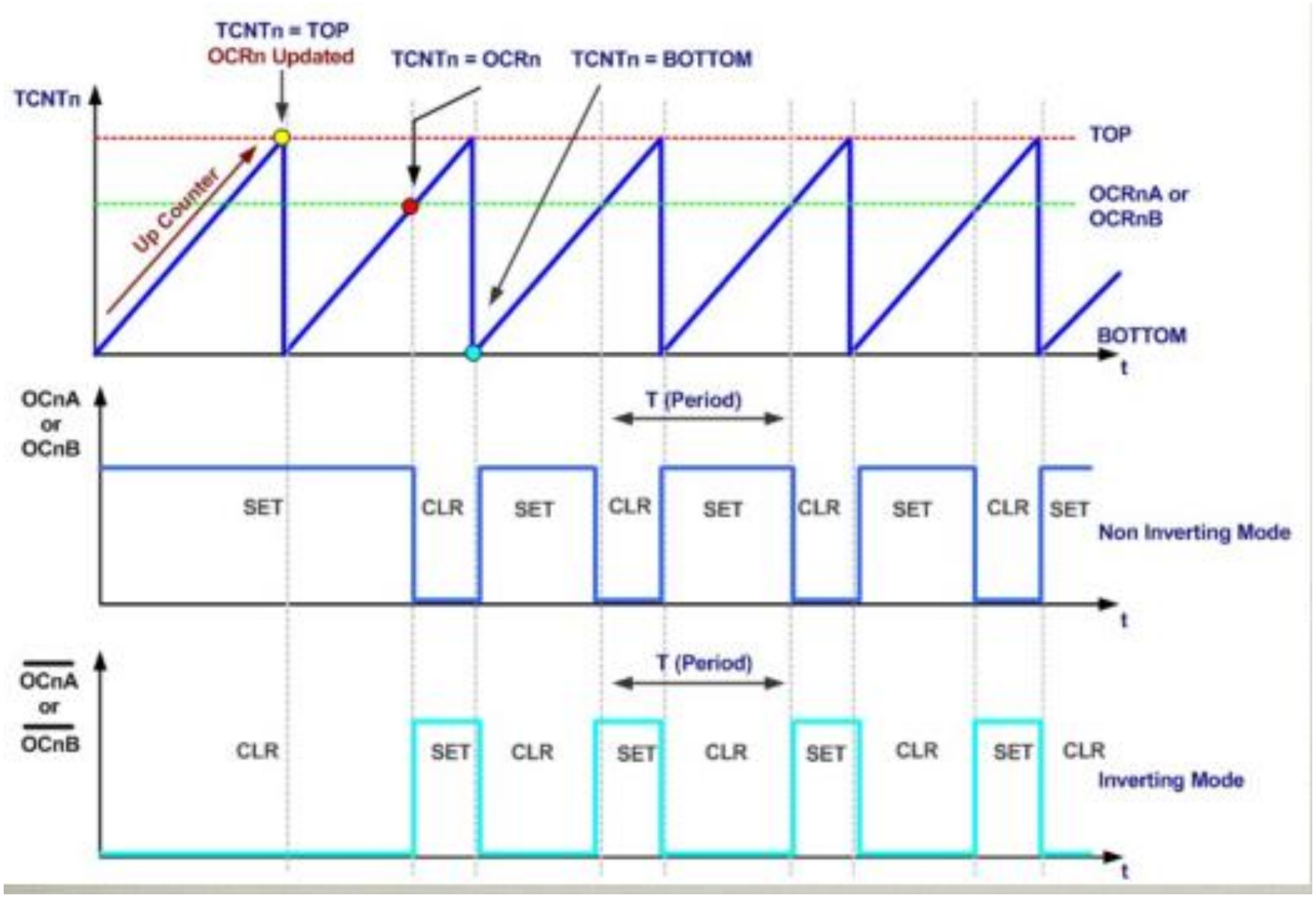


SOURCE	SIZE	PWM OUT	I/O Pins
TIMER0	8 bit	OC0A OC0B	PD6 PD5
TIMER1	16 bit	OC1A OC1B	BP1 BP2
TIMER2	8 bit	OC2A OC2B	BP3 PD3

Fast PWM Mode

- Produces the **fastest** PWM waveforms (compared to phase or phase/frequency correct)
- The timer count TCNTn simply goes from the BOTTOM (0x00) to the TOP (0xFF or 0xFFFF) then resets.
- When TCNTn reaches the output compare (OCRnA or OCRnB), the output compare bit (OCnA or OCnB) is cleared.   These are then set when TCNTn rolls over.

Fast PWM Illustrated



- Frequency

$$f_{pwm} = \frac{f_{clk}}{256\,N}$$

- Where N = pre-scalar

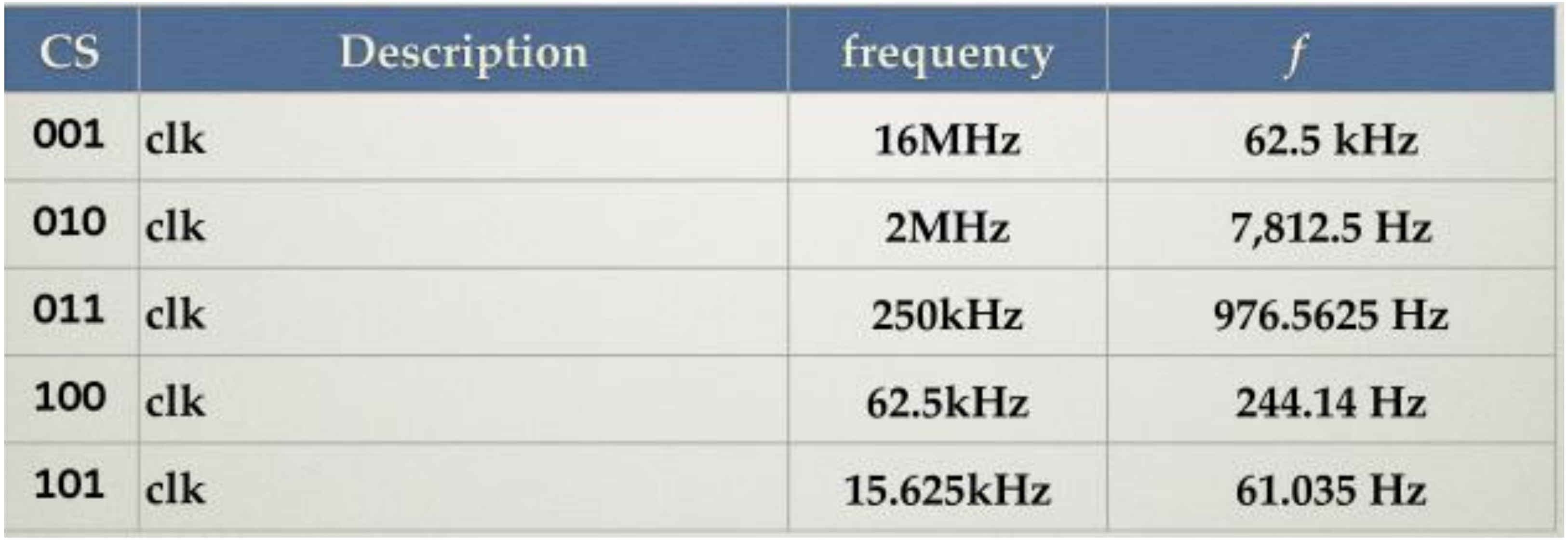
- Duty Cycle:

$$= \frac{OCR0A+1}{256} \times 100\%$$

$$= \frac{255-OCR0A}{256} \times 100\%$$

Fast PWM Example

- Use Fast 8 bit PWM.
- Use both A & B on Timer 0
- Clear the output compare bits on Compare Match, set on TOP
- Choose a PWM frequency approximately twice the default PWM frequency (approx. 1kHz)



CS	Description	frequency	*f*
001	clk	16MHz	62.5 kHz
010	clk	2MHz	7,812.5 Hz
011	clk	250kHz	976.5625 Hz
100	clk	62.5kHz	244.14 Hz
101	clk	15.625kHz	61.035 Hz

Fast PWM Example

- Registers & Parameters
- TCNT0: 0
- COM0A: 10 (TCCR0A)
- COM0B: 10 (TCCR0A)
- WGM0: X11 (TCCR0A and TCCR0B)
- CS: 011 (TCCR0B)

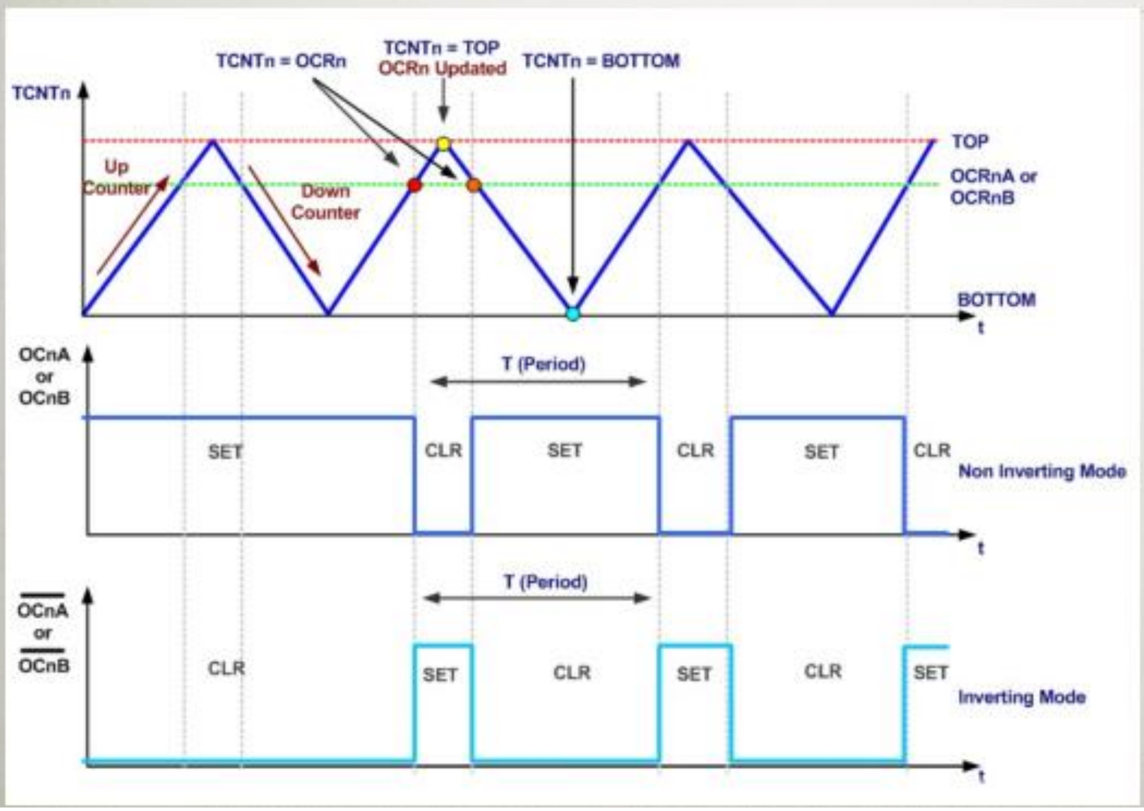
Fast PWM Example

```
void setup()
{
  PORTD = 0x00;
  DDRD = (1<<5)|(1<<6);
  TCCR0A = 0b10100011;
  TCCR0B = 0b00000011;
  TCNT0 = 0;
}
void loop()
{
  OCR0A = 128;//duty cycle for A
  OCR0B = 64;//duty cycle for B
}
```

Phase Correct PWM

- Produces “symmetric” waveforms, which provide smoother motions when used with motors.
- The center of each pulse occurs at the period, as opposed to the falling edge on the period.
- Up/Down counter required.
- Frequencies are half of that used for Fast PWM.

Phase Correct PWM



Frequency

$$f_{\text{PWM}} = \frac{f_{\text{clk}}}{510N}$$

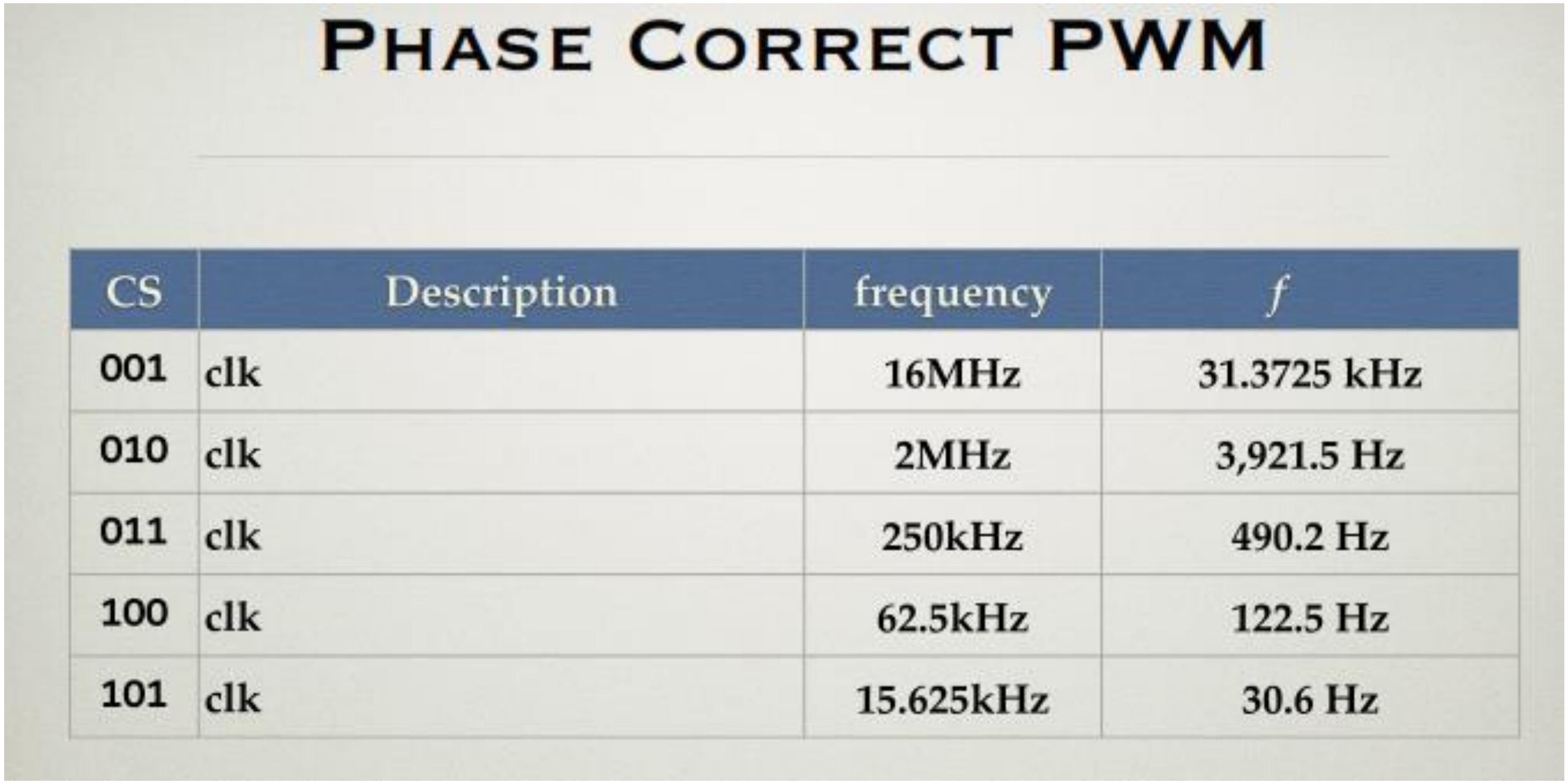
N = prescalar

Duty Cycle:

$$= \frac{\texttt{OCR0A}}{255} \times 100$$

$$= \frac{255 - \texttt{OCR0A}}{255} \times 100$$

(inverted)



Phase Correct PWM

CS	Description	frequency	f
001	clk	16MHz	31.3725 kHz
010	clk	2MHz	3,921.5 Hz
011	clk	250kHz	490.2 Hz
100	clk	62.5kHz	122.5 Hz
101	clk	15.625kHz	30.6 Hz