ERTOS: Embedded & Real Time Operating Systems (3-0-0)
Embedded Systems Architecture: Part - 2

Dr. Teja Mannepalli
Assistant Professor
Department of Electronics and Communications Engineering
NIT Warangal

January 5, 2026

1 Domain-Specific and General Purpose Processors

Embedded systems consist of a central core which may include a processor or specialized logic elements. These components are selected based on application needs, performance, cost, and flexibility requirements.

Processors that execute instructions and perform computations. These can be general-purpose or specialized for particular domains.

Types:

1. *General Purpose Processors (GPPs):* Designed to handle a wide variety of tasks and applications. They are flexible but may not be optimal for domain-specific tasks. Examples include Intel Core and AMD Ryzen chips.

2. *Domain-Specific Processors (DSPs):* Optimized for a particular application domain such as digital signal processing, multimedia, or communications. By tailoring instruction sets and hardware, they achieve higher efficiency for their target domain.

1.1 Application-Specific Integrated Circuits (ASICs)

ASICs are integrated circuits custom-designed for a specific application or product. They are not general-purpose but are optimised for dedicated tasks. *Characteristics:*

1. *High performance for target task:* ASICs execute specific functions very efficiently.

2. *Fixed functionality:* Once manufactured, ASIC logic cannot be reconfigured.

3. *Low power and area cost at scale:* When produced in large volumes, ASICs are power-efficient and cost-effective.

Examples: Custom mining chips for cryptocurrencies, dedicated network packet processors, and automotive control ICs.

1.2 Programmable Logic Devices (PLDs)

PLDs are reconfigurable logic devices that allow hardware functionality to be programmed or updated even after manufacturing.

1. *CPLDs (Complex Programmable Logic Devices):* Smaller in logic capacity and suitable for moderate complexity designs.
2. *FPGAs (Field-Programmable Gate Arrays):* Provide large logic capacity and high performance, enabling implementation of complex digital systems.

Key Features:

1. Reconfigurable logic allows hardware updates after deployment.
2. No need for long lead times for prototyping or manufacturing.
3. Lower non-recurring engineering costs compared to full ASIC designs.

1.3 Commercial Off-The-Shelf (COTS) Components

COTS components are ready-made products that can be purchased and used "as-is" without custom design. They may incorporate general-purpose or domain-specific processors, ASICs, or PLDs internally.

Advantages

1. *Readily available:* Components can be sourced easily from the market.
2. *Easy to integrate:* COTS modules typically follow industry standards enabling plug-and-play usage.
3. *Reduces development time and cost:* No need to design from scratch, leading to faster time-to-market.

Disadvantages:

1. *Vendor lock-in:* Lack of manufacturing or operational standardization.
2. *Product discontinuation risk:* Manufacturers may withdraw products due to technological shifts, impacting long-term support.

2 Sensors and Actuators

A *sensor* is a device that produces an output signal for the purpose of detecting a physical phenomenon. It senses a *physical phenomenon or stimulus* and converts it into a *measurable and scalable electrical parameter* that can be quantified and processed by an embedded system. The *Physical phenomena* like light, motion, sound, pressure, temperature, magnetic

field, and chemical composition and the *Electrical outputs* are voltage, current, resistance, capacitance, or digital values.

An *actuator* is a component of a machine that produces force, torque, or displacement, when an electrical, pneumatic or hydraulic input is supplied to it in a system (called an actuating system). It amplifies the control signal generated by the embedded system. Refer to Fig. 2.

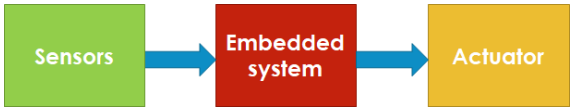

Figure 1: Sensors and Actuators in Embedded Systems

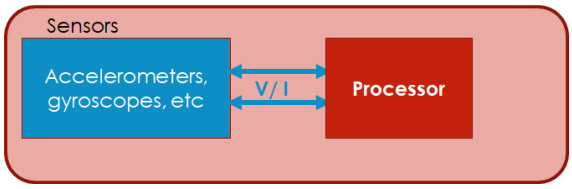

Figure 2: Interface to Processors

2.1 Example of Sensor Operation

Consider the measurement of light intensity from a tube light. Light emitted by the tube light is a physical phenomenon. A light sensing instrument (such as a photodiode or light sensor) captures the intensity.The sensor converts light energy into an electrical signal. The signal is quantified into numerical values for processing and analysis.

Modern embedded systems integrate multiple sensors for real-time monitoring and decision-making.

1. *Cameras*: Capture visual information for image and video processing.
2. *Accelerometers*: Measure linear acceleration and vibration.
3. *Gyroscopes*: Measure angular velocity and orientation.
4. *Strain Gauges*: Measure deformation or mechanical strain.
5. *Microphones*: Convert sound waves into electrical signals.
6. *Magnetometers*: Measure magnetic fields.
7. *Radar and Lidar*: Detect distance, speed, and spatial mapping of objects.
8. *Chemical Sensors*: Detect gas concentration or chemical composition.
9. *Pressure Sensors*: Measure force per unit area.

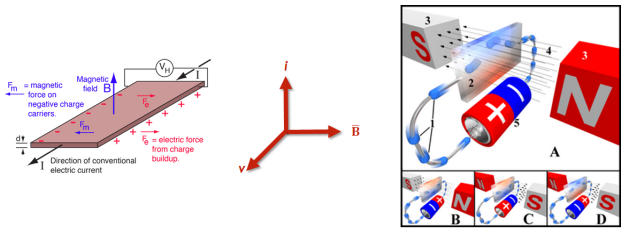

Figure 3: Magnetometer working principle (source: Wikipedia)

2.2 Sensors in Smartphones

Smartphones embed multiple sensors that operate continuously to detect real-time user activities.

1. *Accelerometers*: Detect motion, step count, and device orientation.
2. *Gyroscopes*: Enable precise orientation and rotation tracking.
3. *Magnetometers*: Detect Earth's magnetic field for compass and navigation.

The smartphone communicates with sensors using *voltage or current signals*, which are later digitized using Analog to Digital Converters or ADCs.

2.3 Magnetometer Principle of Operation

Magnetometers commonly operate based on the *Hall Effect* principle. A current-carrying conductor is placed in a magnetic field. The magnetic field exerts a force on moving charge carriers. A voltage is generated perpendicular to both the current direction and the magnetic field. The output voltage or current is proportional to the magnetic field strength. Refer to Fig. 3.

2.4 Sensors Used in Cars

Motion Sensors

1. *Accelerometers*: Measure vehicle acceleration and deceleration.
2. *Gyroscopes*: Detect angular velocity and orientation, used in electronic stability control.
3. *Wheel Speed Sensors*: Monitor individual wheel speeds, used in ABS and traction control systems.

Environmental Sensors

1. *Radar Sensors*: Measure distance and relative speed of objects, used in adaptive cruise control.
2. *Ultrasonic Sensors*: Detect nearby objects, used in parking assistance.
3. *Lidar Sensors*: Provide high-resolution 3D mapping for autonomous driving.

Position and Navigation Sensors

1. *GPS Sensors*: Determine vehicle position and navigation routes.
2. *Hall Effect Sensors*: Measure rotational speed and position in transmission systems.

Vision Sensors

1. *Cameras*: Used for lane departure warning, blind spot detection, and autonomous driving.
2. *Infrared Cameras*: Detect heat signatures for night vision systems.

Pressure Sensors

1. *Tire Pressure Sensors*: Monitor air pressure inside tires.
2. *Oil Pressure Sensors*: Track engine lubrication conditions.
3. *Fuel Pressure Sensors*: Ensure proper fuel delivery.

Temperature Sensors

1. *Engine Temperature Sensors*: Monitor cooling system performance.
2. *Exhaust Gas Temperature Sensors*: Control emissions and exhaust efficiency.
3. *Cabin Temperature Sensors*: Regulate interior climate control.

Proximity Sensors

1. *Blind Spot Sensors*: Detect vehicles in adjacent lanes.
2. *Obstacle Detection Sensors*: Assist in collision avoidance and parking.

Light Sensors

1. *Ambient Light Sensors*: Adjust dashboard and headlight intensity.
2. *Rain Sensors*: Automatically activate windshield wipers.

Gas and Airflow Sensors

1. *Oxygen Sensors*: Monitor exhaust oxygen levels for emission control.
2. *Mass Air Flow Sensors*: Measure incoming air for optimal combustion.

Physiological Sensors

1. *Heart Rate Sensors*: Monitor driver health and alertness.
2. *Eye-Tracking Sensors*: Detect driver fatigue and drowsiness.

2.5 Actuators in Embedded Systems

Actuators convert *electrical control signals* into *physical actions*.

1. *Motor Controllers*: Control speed, torque, and direction of motors.
2. *Solenoids*: Produce linear mechanical motion.
3. *LEDs, Lasers, LCD and Plasma Displays*: Convert electrical signals into visual output.
4. *Loudspeakers*: Convert electrical signals into sound.

Sensors collect real-time data from the environment. Embedded processors analyze sensor data. Actuators execute physical actions based on control decisions.

3 Communication Interfaces

3.1 Overview

Communication interfaces enable interaction *within an embedded system* and with the *external world*. These interfaces are broadly classified based on their scope of communication.

1. *Onboard communication interfaces*: Interconnect internal components of an embedded system. Examples include I^2C, SPI, and UART.
2. *External communication interfaces*: Transfer data between the embedded system and external devices. These interfaces may be wired or wireless, and serial or parallel. Examples include USB and Ethernet.

Not all embedded systems require external communication interfaces.

3.2 Communication Modes

1. *Synchronous communication*: Sender and receiver exchange data using a common clock and require immediate response.
2. *Asynchronous communication*: Data transfer does not require a shared clock and immediate response is not expected.
3. *Duplex communication*: Data transmission occurs in both directions between two devices.
4. *Full-duplex communication*: Data flows simultaneously in both directions over separate channels.

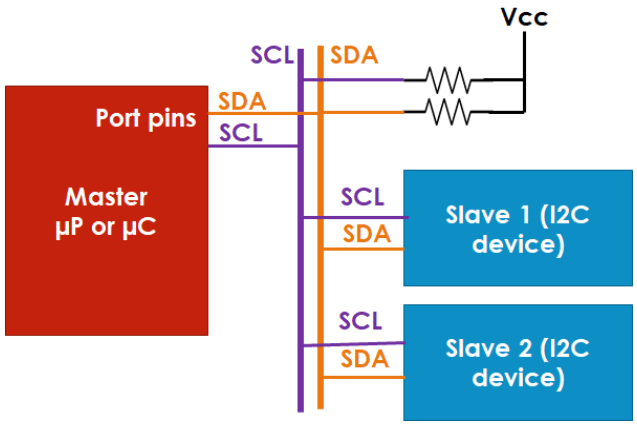

Figure 4: I^2C bus

3.3 I²C Bus Interface

I^2C (Inter-Integrated Circuit) is a *synchronous, bidirectional, multi-master, two-wire serial bus* protocol used for short-distance communication between integrated circuits. It was originally developed by Philips (now NXP) for communication between processors and peripheral devices.

I^2C Uses two bus lines:

1. *Serial Clock Line (SCL)*: Provides synchronization clock.
2. *Serial Data Line (SDA)*: Transfers serial data.

$I^2$2C bus supports multiple devices connected on the same bus. Devices can function as *master* or *slave*. Refer to Fig. 4.

Both SCL and SDA are *open-drain (or open-collector)* lines and require external pull-up resistors. In the idle state, both lines remain HIGH.

I^2C supports multiple devices connected to the same bus. Devices can operate as *masters* or *slaves*, as shown in Fig. 4.

Master–Slave Operation

1. The master initiates communication and controls data transfer and termination.
2. Slave devices wait for commands from the master and respond when addressed.
3. A device can act as a transmitter or receiver depending on the direction of communication.
4. The synchronization clock is generated by the active master.
5. The bus supports multiple masters with arbitration to prevent data corruption.

Bus Characteristics

1. The I^2C bus uses open-drain/open-collector outputs.
2. Both SDA and SCL are pulled HIGH by external pull-up resistors when the bus is idle.
3. Data on SDA must be stable while SCL is HIGH; changes in SDA are allowed only when SCL is LOW, except during START and STOP conditions.

Addressing

1. I^2C supports *7-bit* and *10-bit* addressing modes.
2. In 7-bit addressing, the address is followed by a *Read/Write (R/W)* bit.
3. Some address values are reserved for special purposes.

I^2C Communication Sequence

1. Bus remains idle with SDA and SCL HIGH.
2. Master generates a *START condition* by pulling SDA LOW while SCL is HIGH.
3. Master sends the slave address followed by the R/W bit (MSB first).
4. The addressed slave responds with an *ACK* by pulling SDA LOW during the ninth clock pulse.
5. Data bytes are transferred between master and slave, each followed by an ACK.
6. Master may issue a *Repeated START* to initiate another transfer without releasing the bus.
7. Master generates a *STOP condition* by releasing SDA HIGH while SCL is HIGH.

3.4 Serial Peripheral Interface (SPI)

1. SPI is a *bi-directional, synchronous, full-duplex, four-wire serial interface.*
2. Multiple masters may exist, but only one is active at a time.
3. Signal lines used:
   (a) *MOSI*: Data from master to slave.
   (b) *MISO*: Data from slave to master.
   (c) *SCLK*: Clock signal from master.
   (d) *Slave Select (SS)*: Active LOW slave selection.
4. Data transfer uses shift registers.

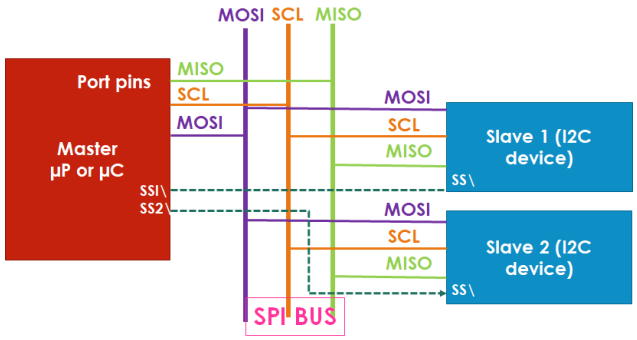

Figure 5: SPI BUS INTERFACING

5. Simultaneous data exchange occurs through MOSI and MISO.
6. SPI does not support acknowledgement.
7. Data is transferred in continuous streams.

Refer to Fig. 5.

3.5 Universal Asynchronous Receiver Transmitter (UART)

1. UART is an *asynchronous serial communication protocol.*
2. No clock signal is shared between transmitter and receiver.
3. Configuration parameters must match at both ends:
   (a) Baud rate
   (b) Data bits
   (c) Parity
   (d) Start and stop bits
4. Data frame consists of start bit, data bits (LSB first), parity bit, and stop bit.
5. Receiver samples data at the midpoint of each bit interval.
6. Parity bit provides basic error detection.
7. Start, stop, and parity bits are discarded after reception.

3.6 1-Wire Interface

Refer to Fig. 6.

1. 1-Wire is an *asynchronous half-duplex communication protocol.*
2. Uses a single data line for communication and power.

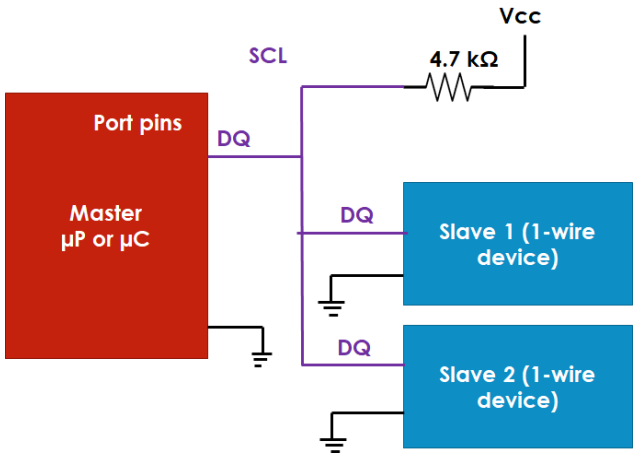

Figure 6: 1-wire communication system

3. Slave devices contain an internal capacitor for parasitic power.
4. Supports one master and multiple slave devices.
5. Each device has a unique *64-bit identification address*.
6. Address consists of family code, serial number, and CRC.

1-Wire Communication Sequence

1. The master issues a reset pulse on the bus.
2. Slave devices respond with a presence pulse.
3. The master sends a ROM command to address one or more slave devices.
4. The master sends function commands to perform read or write operations.
5. Data transfer between master and slave begins.
6. A typical time slot duration is approximately *60 μs*, corresponding to a data rate of about *16.3 kbps*.

4 System Support Components in Embedded Systems

4.1 Reset Circuit

A **Power-On Reset (POR)** circuit is responsible for generating a reset signal whenever power is applied to an electrical or embedded device. This ensures that the system always starts from a **known and deterministic state** before normal operation begins.

In microcontroller-based systems, the POR circuit initializes internal registers, clears previous states, and prevents undefined behavior during power-up.

MCLR Pin Operation

The **Master Clear (MCLR)** pin controls the reset functionality of the microcontroller:

- **MCLR = 0**: The microcontroller is held in reset mode and all internal activities are terminated.
- **MCLR = 1**: The reset condition is released and normal program execution begins.

Recommended Component Values

Typical resistor values used in a power-on reset circuit are:

- Pull-up resistor: $R < 40$ kΩ
- Series resistor (R_1): 100 Ω to 1 kΩ

Role of Circuit Components

The power-on reset circuit typically consists of the following components:

- **Diode (D)**: Allows electric current to flow in only one direction, enabling fast discharge of the capacitor during power-off.
- **Capacitor (C)**: Maintains a constant voltage for a short duration, providing the required delay for a stable reset signal.
- **Resistor (R)**: Limits current flow and forms an RC time constant with the capacitor to control the reset delay.

A *reset circuit* ensures that a processor or controller starts operation from a *known and defined state*, especially during power-on or fault conditions. When a system powers up, transitional events such as slow voltage rise, clock instability, and noise may cause unpredictable behavior. The reset circuit forces the processor into a safe initial condition and directs execution to the reset vector (usually at address 0x0000 on simple controllers) before normal firmware begins. Reset pulses can be applied through external RC circuits (using a resistor and capacitor) or specialized reset/supervisor ICs chosen based on the required logic levels (e.g., CMOS/TTL). Some microcontrollers incorporate internal reset circuitry, eliminating the need for discrete reset components. The width of the reset pulse must be sufficient to allow clock oscillator stability before the system exits reset state. Refer to Fig. 7.

4.2 Brown-Out Protection Circuit

Once during a lightning in my home,

A "brown out" of a microcontroller is a partial and temporary reduction in the power supply voltage below the level required for reliable operation. Many microcontrollers have a protection circuit that detects when the supply voltage drops below this level and puts the device into a reset state to ensure proper startup when power is restored. This action is called a "Brown Out Reset" or BOR.

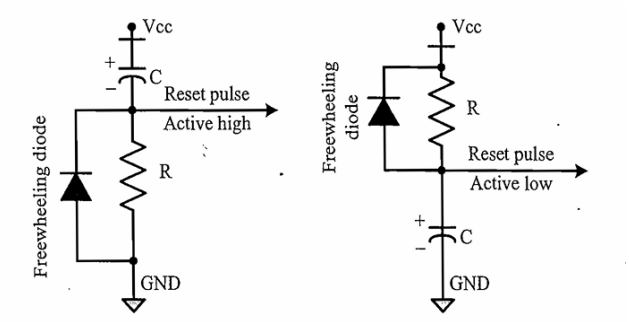

Figure 7: RC-based reset circuit. Credits: Shibu KV

A similar feature is called Low Voltage Detect (LVD) which is more complex and adds detection of multiple voltage levels and can produce an interrupt before a reset is triggered.

Brown-out protection prevents unpredictable behavior when the supply voltage falls below the minimum level required for reliable operation. In a brown-out event, the microcontroller may behave erratically, leading to data corruption or unintended instruction execution. A brown-out protection circuit monitors the supply voltage and holds the processor in reset until the voltage returns above a predefined threshold, ensuring that the system only runs within safe power levels. Many modern microcontrollers include built-in brown-out detection; if not, external supervisor circuits or passive voltage detectors can be used. This protection is particularly crucial for battery-powered and unstable power applications. Refer to Fig. 8.

Brownout is an important safety feature in electronics and microcontrollers. The hardware brown-out feature resets the microcontroller and keeps it until the power supply is returned to the operating range. This ensures that all parts of the circuit work correctly.

Software brown-out part - an interrupt based functionality that detects falling voltage, which allows the software to take care of critical components like saving vital information to non-volatile memory before resetting.

4.3 Oscillator Unit

An *oscillator unit* provides the clock signal required for synchronous operation of a microcontroller or processor. Internal logic and instruction execution depend on these regular clock pulses to sequence operations correctly. Some embedded systems use on-chip oscillators that only require an external resonator (quartz crystal or ceramic resonator) to generate accurate timing. In contrast, others may use external oscillator ICs for precise or high-frequency clock generation. The stability and frequency of the oscillator directly affect instruction timing, peripheral coordination, and overall system performance. Stabilization circuits such as oscillator start-up timers, can delay system release from reset until the oscillator output is stable.

4.4 Real Time Clock (RTC)

A *Real Time Clock (RTC)* is a dedicated timing device that maintains accurate calendar and clock time, even across power cycles or deep sleep modes, often thanks to a backup battery

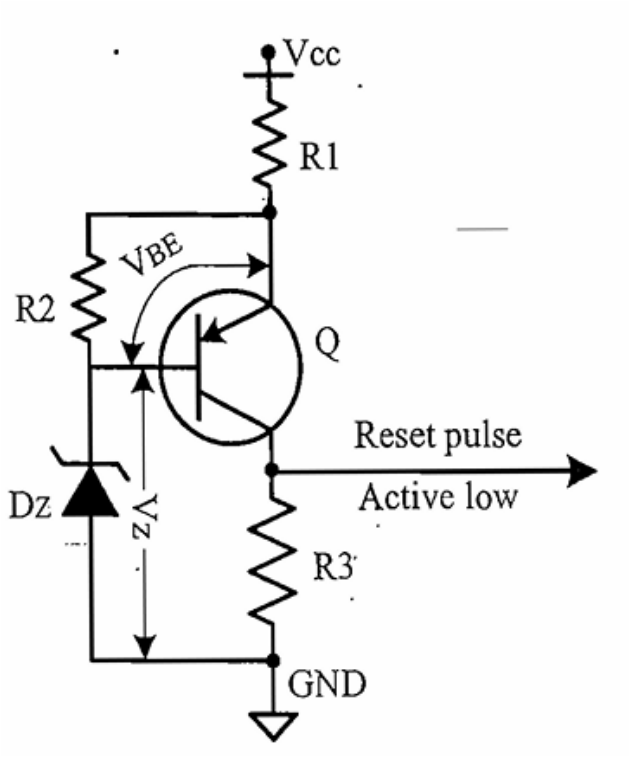

Figure 8: Brown-out protection circuit.

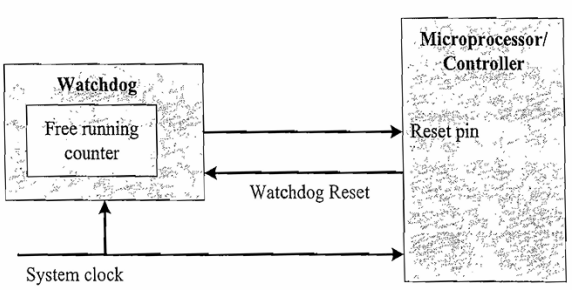

Figure 9: Watchdog Timer

or low-power oscillator. Refer to Fig. 11. Unlike the processor's main system clock, which is focused on instruction timing, an RTC counts seconds, minutes, hours, and sometimes calendar dates. RTC modules include a controller, a low-frequency crystal oscillator (typically 32.768 kHz), and timekeeping registers. These circuits are essential in systems requiring accurate time stamps, scheduled events, alarms, and power-efficient sleep/wake cycles. Even when the main MCU is powered down to save energy, the RTC continues to keep time.

4.5 Watchdog Timer

A watchdog timer (WDT) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service (reset) it. The Watchdog Timer is clocked from a separate On-chip Oscillator, which runs at 1 MHz.

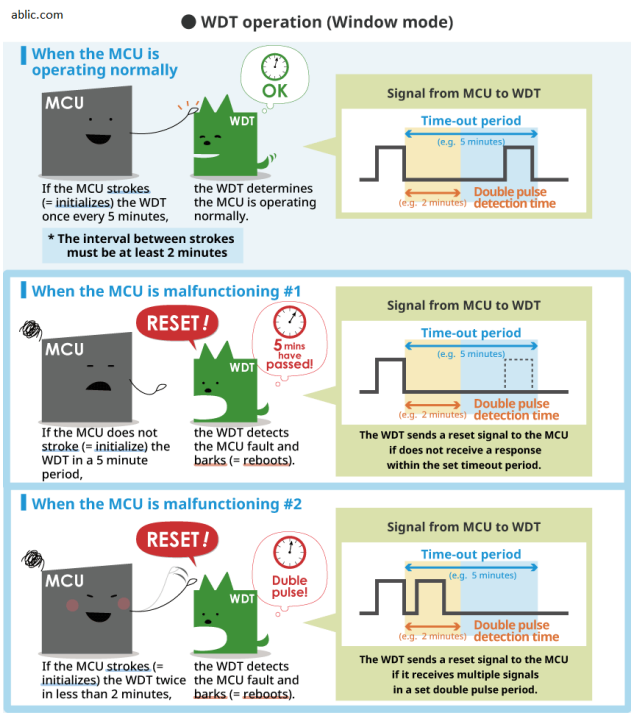

Figure 10: Watchdog Timer

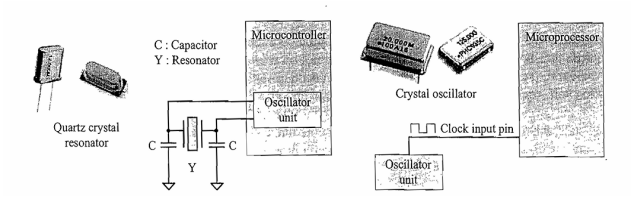

Figure 11: Oscillator circuit using quartz crystal

It is also used to detect system malfunctions and recover from them autonomously. Refer to Fig. 9. It functions as a countdown timer that must be periodically reset (often called *feeding* or *kicking* the watchdog) by normal system software to prevent a timeout. If a software fault, infinite loop, or hardware error prevents the timer from being restarted, the watchdog will elapse and generate a timeout signal. This timeout can trigger corrective actions such as a processor reset or entering a fail-safe state, thereby restoring the system to a known good condition. Watchdog timers can be internal to microcontrollers or implemented as external supervisor ICs; their time intervals may be fixed or programmable, and advanced designs include multistage or windowed watchdog modes for enhanced fault detection. They are critical in systems where human intervention is impractical or where reliability is essential.