Embedded Systems

By Chetna

Introduction to Embedded Systems

- An **embedded system** is a specialized computer system designed to perform **one specific task** or a small set of tasks within a larger system.
  Examples: washing machines, microwave ovens, cars, medical devices, smart watches.
- Unlike general-purpose computers, embedded systems are:
- Task-specific
- Often real-time
- Limited in memory and power
- Designed for reliability and efficiency

History of Embedded Systems

- **Early Beginnings (1940s–1950s): Foundations**
- The idea of embedded systems began with **early electronic control systems**.
- During **World War II**, embedded-like systems were used in:
  - Missile guidance systems
  - Radar systems
- These systems were built using:
  - Vacuum tubes
  - Discrete components
- They were **large, expensive, and power-hungry**, but they showed that computers could control machines.

Transistor Era (1950s–1960s)

- The invention of the **transistor** replaced vacuum tubes.
- Systems became:
  - Smaller
  - More reliable
  - Less power-consuming
- Embedded control systems began appearing in:
  - Industrial machines
  - Telephone switching systems
- Still very expensive and used mainly by governments and large industries.

Integrated Circuits (1960s–1970s)

- **Integrated Circuits (ICs)** allowed many components to fit on one chip.
- This led to:
- Reduced size and cost
- Improved performance
- Embedded systems started being used in:
- Medical equipment
- Industrial automation
- Software began to play a more important role, often written in **assembly language**.

Microprocessor Revolution (1970s)

- In 1971, Intel introduced the **first microprocessor (Intel 4004)**.
- This was a major breakthrough because:
  - A full CPU was on a single chip
  - Systems became programmable
- Embedded systems could now be designed more flexibly.
- Used in:
  - Calculators
  - Early consumer electronics

Microcontroller Era (1980s)

- **Microcontrollers** combined:
  - CPU
  - Memory (RAM & ROM)
  - Input/Output peripherals on a single chip.
- This made embedded systems:
  - Cheaper
  - Smaller
  - Easier to design
- Widely used in:
  - Home appliances
  - Automobiles
  - Industrial control
- Programming languages like **C** became popular.

Real-Time Embedded Systems (1990s)

- Systems began requiring **real-time performance**, where tasks must complete within strict time limits.
- **Real-Time Operating Systems (RTOS)** were developed.
- Embedded systems expanded into:
  - Airplanes
  - Robotics
  - Telecommunications
- Networking capabilities were added.
- **Focus**: Timing accuracy and reliability.

System-on-Chip (SoC) and Consumer Boom (2000s)

- **System-on-Chip (SoC)** technology integrated:
  - Processor
  - Memory
  - Graphics
  - Communication modules
- Enabled advanced products like:
  - Smartphones
  - Digital cameras
  - Gaming consoles
- Embedded Linux and Android appeared.
- Devices became:
  - More powerful
  - More user-friendly

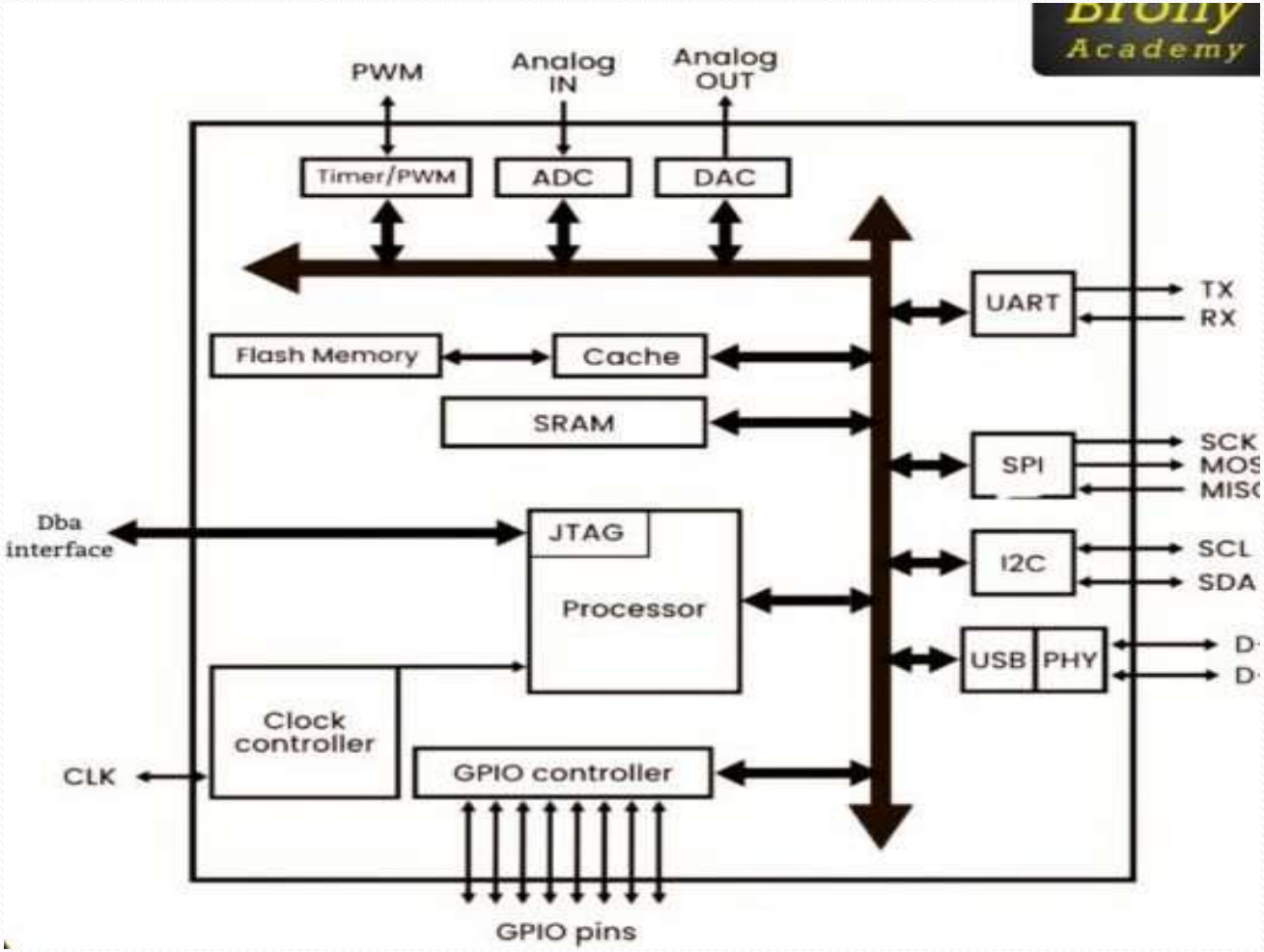
Internet of Things (IoT) Era (2010s–Present)

- Embedded systems became **connected to the internet**.
- This led to **IoT (Internet of Things)**.
- Examples:
  - Smart homes
  - Wearable devices
  - Smart cities
- Features include:
  - Wireless communication
  - Cloud connectivity
  - Sensors and AI integration
- Emphasis on:
  - Security
  - Energy efficiency

Current Trends and Future Direction

- Embedded systems today use:
  - Artificial Intelligence (AI)
  - Machine Learning (ML)
  - Edge computing
- Used in:
  - Autonomous vehicles
  - Medical diagnostics
  - Space technology
- Future embedded systems will be:
  - More intelligent
  - More secure
  - More energy-efficient

Embedded System Architecture



Cont.

- **Processing Unit (Microcontroller/Microprocessor)**
  - Acts as the brain of the system, executing instructions and managing data flow.
  - Embedded systems often use microcontrollers (MCUs) or microprocessors (MPUs) optimized for low power consumption and high integration.
  - Examples: ARM Cortex, PIC, and AVR series.
- **Memory Subsystem**
  - Divided into volatile (RAM) and non-volatile (ROM, Flash) memory.
  - RAM stores temporary data and variables during operation, while ROM or Flash holds firmware and program code.
  - Embedded systems often use memory-efficient techniques to operate under tight resource constraints.

Cont.

- **Power Supply Unit (PSU)**
  - Provides regulated power to all components, ensuring stable and efficient operation.
  - Power management strategies, including sleep modes and energy-saving techniques, are crucial for battery-operated systems.
- **Clock and Timing Circuits**
  - Provide the necessary timing signals to synchronize the operation of various components.
  - Use crystal oscillators or internal clock sources to maintain precise timing.

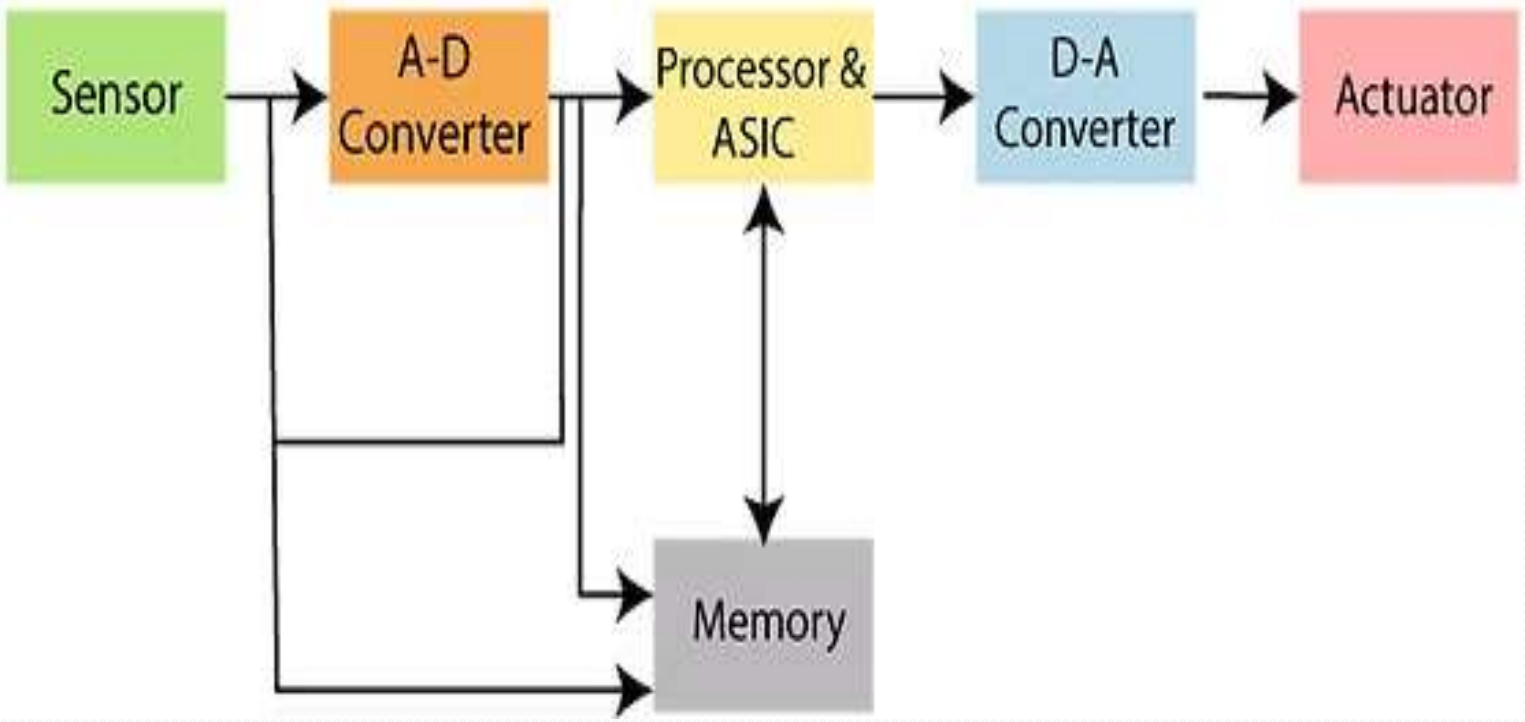
Cont.

- **Input/Output (I/O) Interfaces**
  - Facilitate communication between the embedded system and external devices such as sensors, actuators, displays, and user interfaces.
  - Examples: GPIO (General Purpose Input/Output), UART, SPI, and I2C protocols.
- **Communication Interfaces**
  - Allow the system to exchange data with other devices or networks using wired or wireless protocols.
  - Examples: Ethernet, Wi-Fi, Bluetooth, CAN bus, and Zigbee.

Cont.

- **Sensors and Actuators**
  - Sensors gather real-world data (e.g., temperature, pressure, light) and convert it into electronic signals.
  - Actuators convert electronic signals into physical actions (e.g., motor movement, valve control).

Functional Structure of Embedded System



Sensor

- **Function:** A **sensor** detects physical quantities from the environment.
- Examples of physical quantities:
  - Temperature
  - Pressure
  - Light
  - Speed
  - Humidity
- **Output:** The sensor produces an **analog signal** (continuous signal).
- Example: A temperature sensor produces a voltage proportional to temperature.

A–D Converter (Analog-to-Digital Converter)

- **Function:**
- Converts the **analog signal** from the sensor into a **digital signal**.
- Digital signals are in binary form (0s and 1s), which processors can understand.
- **Importance:**
- Most processors cannot directly process analog signals.
- ADC samples the signal and converts it into numerical values.
- **Example:**
- Analog voltage: 2.5 V
- Digital output: 10101100 (binary)

Processor & ASIC

- This is the **heart (brain)** of the embedded system.
- **Processor (Microcontroller / Microprocessor):**
- Executes the program stored in memory.
- Performs:
  - Data processing
  - Decision making
  - Control operations
- Uses programming languages like **C or Embedded C**.

Cont.

- **ASIC (Application-Specific Integrated Circuit):**
- Special hardware designed for a specific task.
- Improves:
  - Speed
  - Efficiency
  - Power consumption
- **Functions performed:**
- Reads digital data from ADC
- Processes data according to software logic
- Sends control signals to DAC or actuator
- ⬙ **Example**:
  If temperature > set value → turn ON cooling fan.

Memory

- **Function:**
- Stores:
  - Program code (firmware)
  - Input data
  - Output data
  - Intermediate results
- **Types of Memory:**
- **ROM / Flash** – Stores program permanently
- **RAM** – Stores temporary data during execution
- **EEPROM** – Stores small data like calibration values
- **Connection:**
- Memory communicates **bidirectionally** with the processor.

D–A Converter (Digital-to-Analog Converter)

- **Function:**
- Converts **digital output** from the processor into an **analog signal**.
- Required when the actuator needs analog input.
- **Example:**
- Digital signal: 11001010
- Analog output: Corresponding voltage or current
- ⬙ **Key role**: Converts processor decisions into real-world control signals.

Actuator

- **Function:**
- Converts electrical signals into **physical action**.
- Acts on the environment.
- **Examples:**
- Motor
- Relay
- LED
- Heater
- Valve
- **Output:**
- Physical movement or change.
- ⬙ **Example**:
- Motor rotates
- Heater warms
- Valve opens/closes

Working of Embedded System

- Sensor measures a physical parameter.
- ADC converts the analog signal to digital.
- Processor reads the digital data.
- Processor processes data using program stored in memory.
- Decision is made based on logic.
- Digital output is converted to analog using DAC.
- Actuator performs the required physical action.

Real-Life Example: Automatic Temperature Control System

- **Sensor**: Temperature sensor
- **ADC**: Converts temperature voltage to digital value
- **Processor**: Compares temperature with set value
- **Memory**: Stores control program
- **DAC**: Generates control signal
- **Actuator**: Fan or heater

Introduction to PIC Microcontroller

- **PIC** stands for **Peripheral Interface Controller**.
  A PIC microcontroller is a **single-chip embedded controller** developed by **Microchip Technology** that contains:
- CPU (processor)
- Program memory
- Data memory
- Input / Output (I/O) ports
- Timers, interrupts, and communication modules
- PIC microcontrollers are widely used in **embedded systems** due to their **simplicity, low cost, low power consumption, and reliability**.

History of PIC Microcontroller

- **Origin (1970s)**
- PIC was originally developed by **General Instrument (GI)** in the **1970s**.
- The first PIC devices were designed to work as **simple peripheral controllers** for GI's CP1600 microprocessor.
- They were mainly used for:
  - Keyboard control
  - Display interfacing
  - I/O operations

Development by Microchip Technology (1989)

- In **1989**, **Microchip Technology** acquired the PIC microcontroller technology from General Instrument.
- Microchip redesigned PICs to be:
  - Fully programmable
  - Stand-alone microcontrollers
- The famous **PIC16 series** was introduced.
- ⬙ This marked the beginning of **modern PIC microcontrollers**.

Evolution and Growth (1990s)

- PIC microcontrollers gained popularity due to:
  - **RISC architecture**
  - **Harvard architecture**
  - Simple instruction set
- Programming was initially done in **assembly language**.
- Later, **C compilers** became available.
- Popular series introduced:
- **PIC10 / PIC12** – Small, low-pin devices
- **PIC16** – Mid-range, widely used
- **PIC18** – High-performance 8-bit PICs
- **2.4 Advanced PIC Families (2000s–Pr**

Advanced PIC Families (2000s–Present)

- Microchip expanded PIC into:
- **PIC24** – 16-bit microcontrollers
- **dsPIC** – Digital Signal Controllers (DSP + MCU)
- **PIC32** – 32-bit microcontrollers
- Modern PICs support:
- USB
- CAN
- Ethernet
- Low-power and IoT applications

Features of PIC Microcontroller

- **Architecture Features**
- **(a) Harvard Architecture**
- Separate memory for:
  - Program (Flash/ROM)
  - Data (RAM)
- Allows simultaneous instruction fetch and data access.
- Results in **faster execution**.
- **(b) RISC Architecture**
- Reduced Instruction Set Computer
- Simple and fewer instructions (≈35 for PIC16)
- Most instructions execute in **one clock cycle**.

Cont.

- **Memory Organization**
- **Program Memory (Flash/ROM)**: Stores program code.
- **Data Memory (RAM)**: Stores variables.
- **EEPROM**: Stores non-volatile data.
- **I/O Ports**
- Multiple **bidirectional I/O ports**.
- Each pin can be configured as:
  - Input
  - Output
- Controlled using special registers (TRIS registers).

Cont.

- **Timers and Counters**
- Built-in timers:
  - Timer0
  - Timer1
  - Timer2
- Used for:
  - Delay generation
  - Event counting
  - PWM control

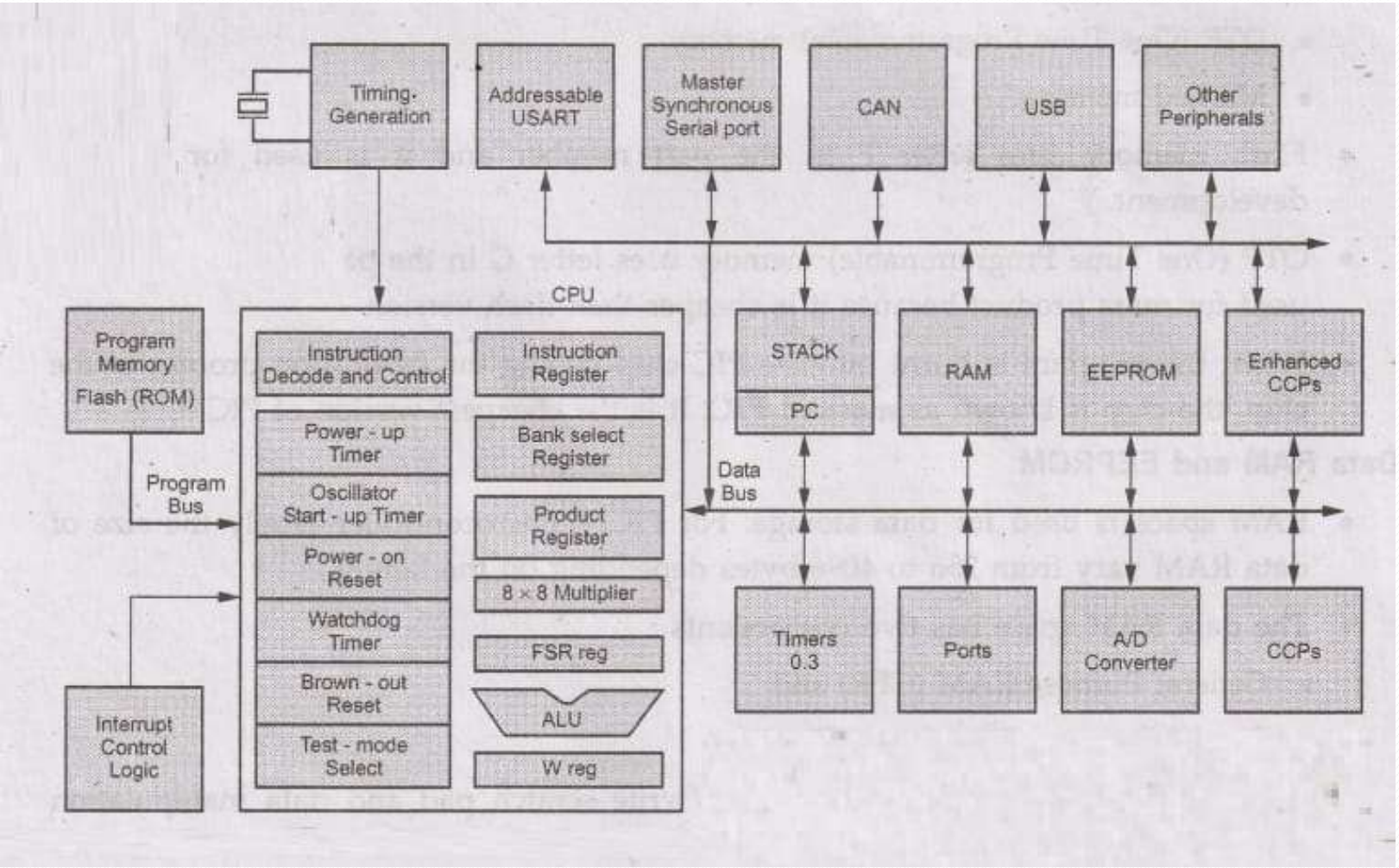
Cont.

- **Interrupt System**
- Supports **hardware interrupts**.
- Enables quick response to:
  - External events
  - Timer overflows
  - Communication events
- **Communication Interfaces**
- PIC microcontrollers support standard communication protocols:
- **USART** – Serial communication
- **SPI** – High-speed data transfer
- **I²C** – Inter-device communication
- **USB, CAN** (in advanced PICs)

Cont.

- **Analog Features**
- **ADC (Analog-to-Digital Converter)** for sensor input.
- **Comparators**
- **PWM modules** for motor control.
- **Low Power Consumption**
- Designed for battery-operated systems.
- Power-saving modes:
  - Sleep mode
  - Idle mode
- Very suitable for portable and IoT devices.
- **High Reliability**
- Flash memory allows:
  - In-circuit programming (ICSP)
  - Easy updates
- Watchdog Timer (WDT) improves system reliability.

PIC Block Diagram



Cont.

Harvard architecture PIC18 uses Harvard architecture in which program and data are accessed from separate memories using separate buses.

ALU

The PIC18 contains 8-bit ALU. It is capable of performing arithmetic operations such as add, subtract, multiply, shift and logical operations.

Multiplier 8 × 8

An 8 x 8 hardware multiplier is included in the ALU of the PIC18 devices.

By making the multiply a hardware operation, it completes in a single instruction cycle. This is an unsigned multiply that gives a 16-bit result. The result is stored in the 16-bit product register pair (PRODH: PRODL).

Cont.

- **Program ROM**
- ROM is used to store programs and hence the name. It is also known as code ROM. For PIC18 microcontroller family the size of program RAM vary from 4 K are to 128 K depending on the family. In PIC18 family microcontrollers, program ROM is available in different memory types:
- Flash memory
- OTP (One Time Programmable) memory

Cont.

- **Masked memory**
- Flash memory uses letter F in the part number and it is used for product development.
- OTP (One Time Programmable) memory uses letter C in the part number and it is used for mass product because it is cheaper than flash version.
- When the program is burnt into the PIC chip during the fabrication process of the chip, the chip is known as masked PIC. It is the cheapest version of PIC.

Cont.

- **Data RAM and EEPROM**
- RAM space is used for data storage. For PIC18 microcontroller family the size of data RAM vary from 256 to 4096 bytes depending on the family.
- The data RAM space has two components:
- General Purpose RAM (GPR) and
- Special Function Registers (SFRs)
- The RAM GPR space is used for read/write scratch pad and data manipulation and is divided into memory banks of 256 bytes of each. Thus the size of GPR for PIC18 family is always in multiples of 256 bytes.

Cont.

- The SFRS are used by the CPU and peripheral modules for controlling the desired operation of the device.
- The size of EEPROM on the chip vary with the family and it is used to store critical data that does not frequent access.
- **Interrupt Sources**
- PIC18 devices support multiple interrupt sources.
- Each interrupt can be assigned with low/high priority.

Cont.

- **I/O Pins**
- In PIC18 microcontroller family I/O pins vary from 16 to 72 depending on the family.
- **Capture/Compare/PWM (CCP) Module**
- The CCP (Capture/Compare/PWM) module contains a 16-bit register that can operate as a 16-bit capture register, as a 16-bit compare register or as a PWM duty cycle register.

Cont.

- **Enhanced Capture/Compare/PWM (CCP) Module**
- The operation of the ECCP module is identical to that of the CCP module. The ECCP module, on the other hand, has Enhanced PWM functionality capability. and auto-shutdown
- **CAN Module**
- The Controller Area Network (CAN) module is a serial interface, useful for communicating with other peripherals or microcontroller devices.
- This interface/protocol was designed to allow communications within noisy environments.

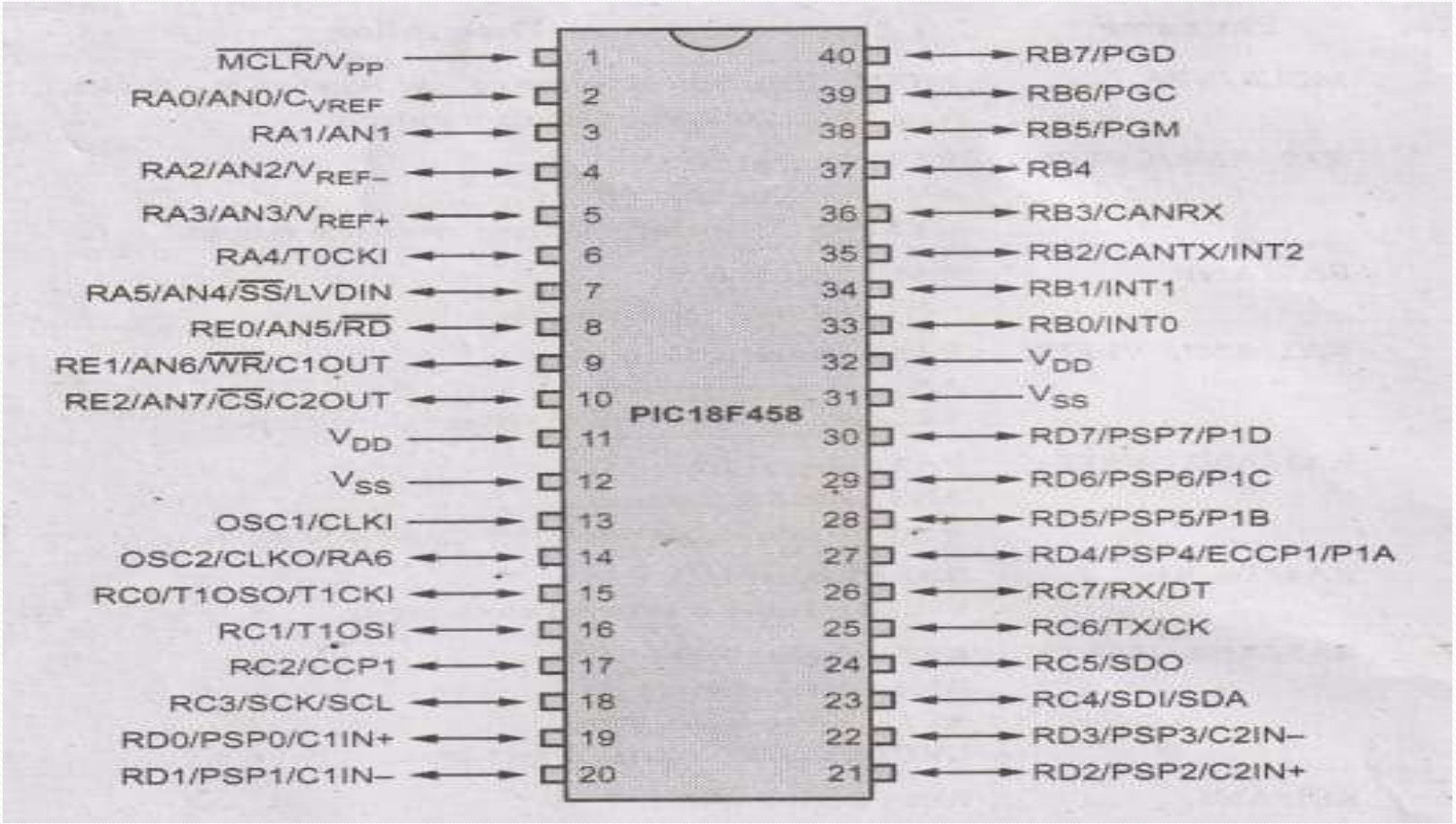
Cont.

- **ADC**
- ADC is 10-bit and the number of ADC channels in each chip varies from 5 to 16.
- **Timers**
- PIC18 has 4/5 timers along with watch dog timer.
- **Master Synchronous Serial Port (MSSP) module**
- The Master Synchronous Serial Port (MSSP) module is a serial interface useful for communicating with other peripheral or microcontroller devices. It has two modes of operation:
- 3-wire SPI (Supports all 4 SPI modes)
- I2 C Master and Slave mode

Cont.

- **Universal Synchronous Asynchronous Receiver Transmitter (USART) module**
- The USART is a serial interface that can be configured in the following modes:
- Asynchronous (full-duplex)
- Synchronous - Master (half-duplex)
- Synchronous - Slave (half-duplex)

Pin Diagram of PIC18F458



PIC18F458 – Every Pin Uses & Functions (40-pin DIP)

- **PIN 1**
- MCLR / VPP
- Use: Reset & Programming
- Function:
- Resets the microcontroller when pulled LOW
- VPP provides programming voltage during ICSP

PORT A (Pins 2–7) – Analog + Digital + Timer

- **Pin 2 – RA0 / AN0 / CVREF**
- Use: Input / Output / ADC / Comparator
- Function:
- Digital I/O
- Analog input channel AN0
- Comparator voltage reference output
- **Pin 3 – RA1 / AN1**
- Use: ADC & Digital I/O
- Function:
- Digital input/output
- Analog input AN1

Cont.

- **Pin 4 – RA2 / AN2 / VREF−**
- Use: ADC Reference
- Function:
- Analog input AN2
- Negative reference voltage for ADC
- **Pin 5 – RA3 / AN3 / VREF+**
- Use: ADC Reference
- Function:
- Analog input AN3
- Positive reference voltage for ADC

Cont.

- **Pin 6 – RA4 / T0CKI**
- Use: Timer Control
- Function:
- Digital I/O
- External clock input for Timer0
- **Pin 7 – RA5 / AN4 / SS / LVDIN**
- Use: ADC / SPI / Voltage Monitor
- Function:
- Analog input AN4
- SPI Slave Select
- Low Voltage Detect input

PORT E (Pins 8–10) – Control + Analog

- **Pin 8 – RE0 / AN5 / RD**
- Use: Control Signal
- Function:
- Analog input AN5
- Read signal for Parallel Slave Port
- **Pin 9 – RE1 / AN6 / WR / C1OUT**
- Use: Write Control / Comparator
- Function:
- Analog input AN6
- Write signal for PSP
- Comparator-1 output

Cont.

- **Pin 10 – RE2 / AN7 / CS / C2OUT**
- Use: Chip Select / Comparator
- Function:
- Analog input AN7
- Chip Select for PSP
- Comparator-2 output

Cont.

- **POWER SUPPLY**
- **Pin 11 – VDD**
- Use: Power Supply
- Function:
- +5V DC supply
- **Pin 12 – VSS**
- Use: Ground
- Function:
- 0V reference

Cont.

- **OSCILLATOR (Clock)**
- **Pin 13 – OSC1 / CLKI**
- Use: Clock Input
- Function:
- External crystal or clock input
- **Pin 14 – OSC2 / CLKO / RA6**
- Use: Clock Output
- Function:
- Crystal oscillator output
- Can act as digital I/O

Cont.

- **PORT C (Pins 15–20, 23–24) – Timers, PWM, Serial**
- **Pin 15 – RC0 / T1OSO / T1CKI**
- Use: Timer1
- Function:
- Timer1 oscillator output
- Timer1 external clock input
- **Pin 16 – RC1 / T1OSI**
- Use: Timer1
- Function:
- Timer1 oscillator input

Cont.

- **Pin 17 – RC2 / CCP1**
- Use: PWM / Capture / Compare
- Function:
- CCP module for PWM generation
- **Pin 18 – RC3 / SCK / SCL**
- Use: SPI / I²C Clock
- Function:
- SPI clock
- I²C clock

Cont.

- **Pin 19 – RC4 / SDI / SDA**
- Use: SPI / I²C Data
- Function:
- SPI data input
- I²C data line
- **Pin 20 – RC5 / SDO**
- Use: SPI Output
- Function:
- SPI data output

Cont.

- **Pin 23 – RC6 / TX / CK**
- Use: UART Transmission
- Function:
- UART transmit pin
- Synchronous clock output
- **Pin 24 – RC7 / RX / DT**
- Use: UART Reception
- Function:
- UART receive pin
- Synchronous data input

Cont.

- **PORT D (Pins 21–22, 25–28) – Parallel Slave Port & PWM**
- **Pin 21 – RD2 / PSP2 / C2IN+**
- Use: Comparator
- Function:
- Comparator-2 positive input
- PSP data line
- **Pin 22 – RD3 / PSP3 / C2IN−**
- Use: Comparator
- Function:
- Comparator-2 negative input
- PSP data line

Cont.

- **Pin 25 – RD4 / PSP4 / ECCP1 / P1A**
- Use: PWM Output
- Function:
- PWM channel output
- **Pin 26 – RD5 / PSP5 / P1B**
- Use: PWM Output
- Function:
- PWM output

Cont.

- **Pin 27 – RD6 / PSP6 / P1C**
- Use: PWM Output
- Function:
- PWM output
- **Pin 28 – RD7 / PSP7 / P1D**
- Use: PWM Output
- Function:
- PWM output

Cont.

- **POWER (Right Side)**
- **Pin 31 – VSS**
- Use: Ground
- **Pin 32 – VDD**
- Use: +5V supply
- **PORT B (Pins 33–40) – Interrupts, CAN,Programming**
- **Pin 33 – RB0 / INT0**
- Use: External Interrupt
- Function:
- Interrupt 0 input

Cont.

- **Pin 34 – RB1 / INT1**
- Use: External Interrupt
- Function:
- Interrupt 1 input
- **Pin 35 – RB2 / INT2 / CAN TX**
- Use: CAN Communication
- Function:
- CAN transmit pin

Cont.

- **Pin 36 – RB3 / CAN RX**
- Use: CAN Communication
- Function:
- CAN receive pin
- **Pin 37 – RB4**
- Use: Interrupt-On-Change
- Function:
- Triggers interrupt on pin change

Cont.

- **Pin 38 – RB5 / PGM**
- Use: Programming Mode
- Function:
- Low voltage programming enable
- **Pin 39 – RB6 / PGC**
- Use: Programming Clock
- Function:
- ICSP clock line
- **Pin 40 – RB7 / PGD**
- Use: Programming Data
- Function:
- ICSP data line

I/O PORTS

- Depending on the device selected, there are up to five general purpose I/O ports available on PIC18FXX8 devices. Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin. Each port has three registers for its operation:
  - TRIS register (Data Direction register)
  - PORT register (reads the levels on the pins of the device)
  - LAT register (output latch)
- The data latch (LAT register) is useful for read-modify write operations on the value that the I/O pins are driving.

PORTA, TRISA and LATA Registers

- PORTA is a 7-bit wide, bidirectional port. The corresponding Data Direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin). On a Power-on Reset, these pins are configured as inputs and read as '0'.

  Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch.

Cont.

- Read-modify-write operations on the LATA register read and write the latched output value for PORTA.
- The RA4 pin is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/ T0CKI pin is a Schmitt Trigger input and an open-drain output. All other RA port pins have TTL input levels and full CMOS output drivers.
- The other PORTA pins are multiplexed with analog inputs and the analog VREF+ and VREF- inputs. The operation of each pin is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register 1). On a Power-on Reset, these pins are configured as analog inputs and read as ‘0’.
- The TRISA register controls the direction of the RA pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set, when using them as analog inputs.

Cont.

FIGURE 9-1: RA3:RA0 AND RA5 PINS BLOCK DIAGRAM

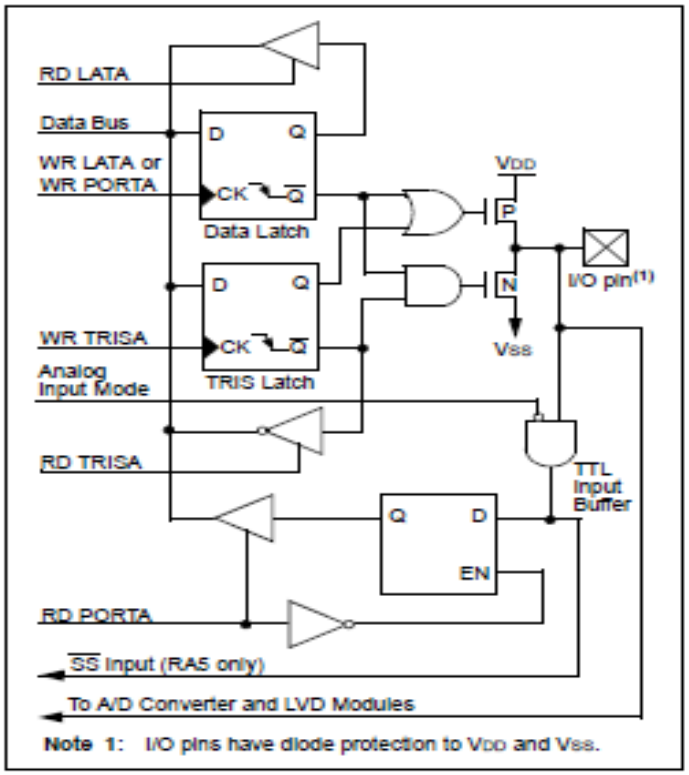
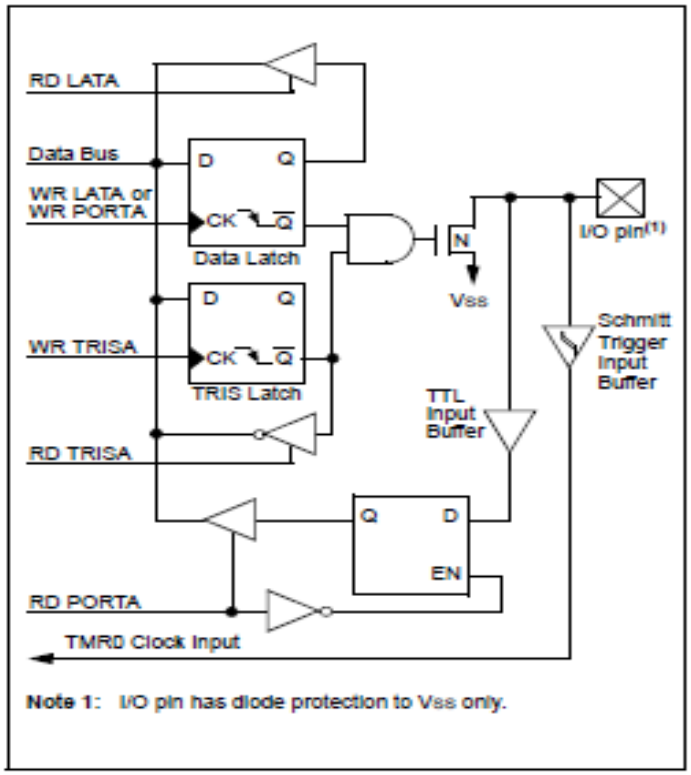
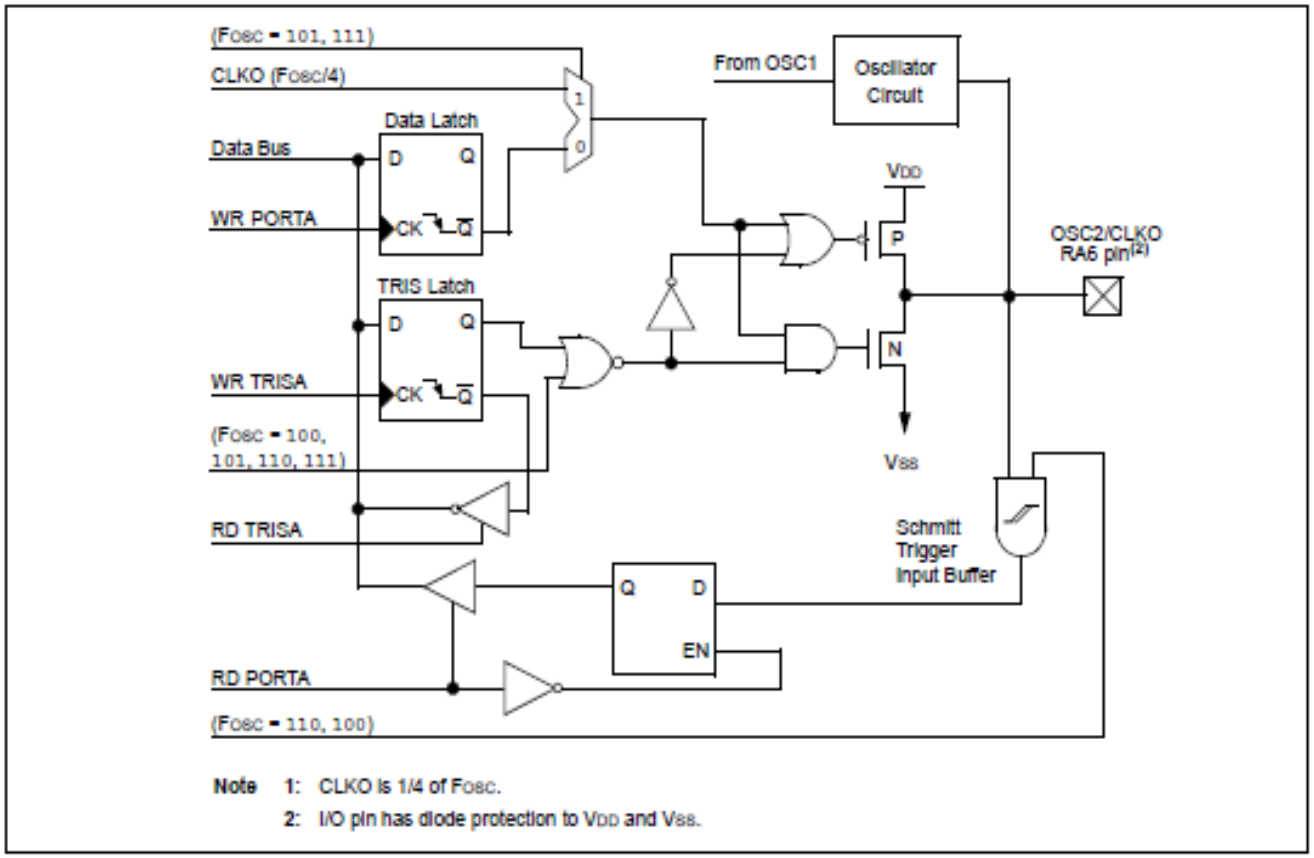


FIGURE 9-2: RA4/T0CKI PIN BLOCK DIAGRAM



Cont.

FIGURE 9-3: OSC2/CLKO/RA6 PIN BLOCK DIAGRAM



Note 1: CLKO is 1/4 of FOSC.

2: I/O pin has diode protection to VDD and VSS.

Cont.

TABLE 9-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0/CVREF	bit 0	TTL	Input/output, analog input or analog comparator voltage reference output.
RA1/AN1	bit 1	TTL	Input/output or analog input.
RA2/AN2/VREF-	bit 2	TTL	Input/output, analog input or VREF-.
RA3/AN3/VREF+	bit 3	TTL	Input/output, analog input or VREF+.
RA4/T0CKI	bit 4	ST/OD	Input/output, external clock input for Timer0, output is open-drain type.
RA5/AN4/$\overline{SS}$/LVDIN	bit 5	TTL	Input/output, analog input, slave select input for synchronous serial port or Low-Voltage Detect input.
OSC2/CLKO/RA6	bit 6	TTL	Oscillator clock output or input/output.

Legend: TTL = TTL input, ST = Schmitt Trigger input, OD = Open-Drain

TABLE 9-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	-00x 0000	-uuu uuuu
LATA	—	Latch A Data Output Register							-xxx xxxx	-uuu uuuu
TRISA	—	PORTA Data Direction Register							-111 1111	-111 1111
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	uu-- uuuu

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by PORTA.

PORTB, TRISB and LATB

- **Registers**
- PORTB is an 8-bit wide, bidirectional port. The corresponding Data Direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin). Read-modify-write operations on the LATB register, read and write the latched output value for PORTB.

Cont.

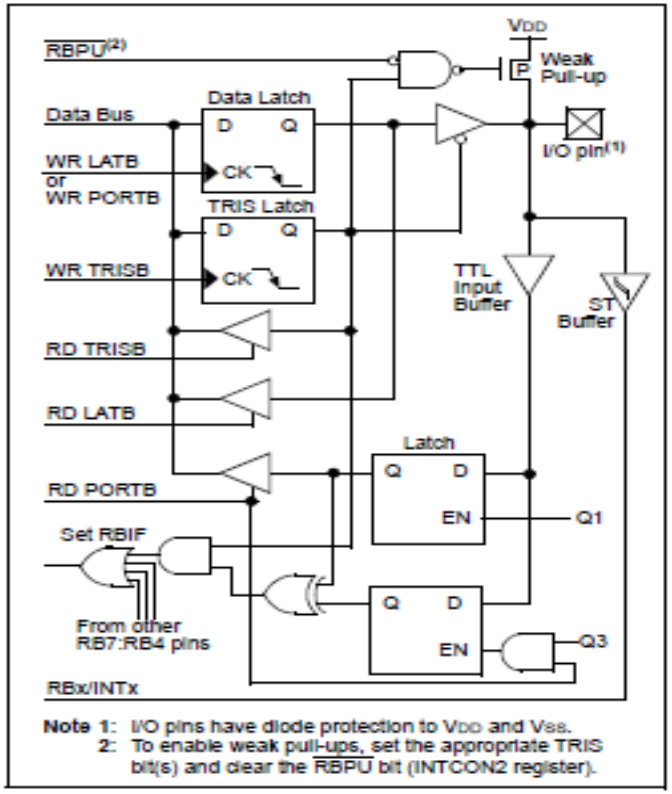
- Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (INTCON2 register). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset..
- Four of the PORTB pins (RB7:RB4) have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are ORed together to generate the RB Port Change Interrupt with Flag bit RBIF (INTCON register).
- This interrupt can wake the device from Sleep. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:
- a) Any read or write of PORTB (except with the MOVFY instruction). This will end the mismatch condition.
- b) Clear flag bit RBIF

Cont.

- A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.
- The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.
- Note
- 1: While in Low-Voltage ICSP mode, the RB5 pin can no longer be used as a general purpose I/O pin and should not be held low during normal operation to protect against inadvertent ICSP mode entry.
- 2: When using Low-Voltage ICSP Program-ming (LVP), the pull-up on RB5 becomes disabled. If TRISB bit 5 is cleared, thereby setting RB5 as an output, LATB bit 5 must also be cleared for proper operation.

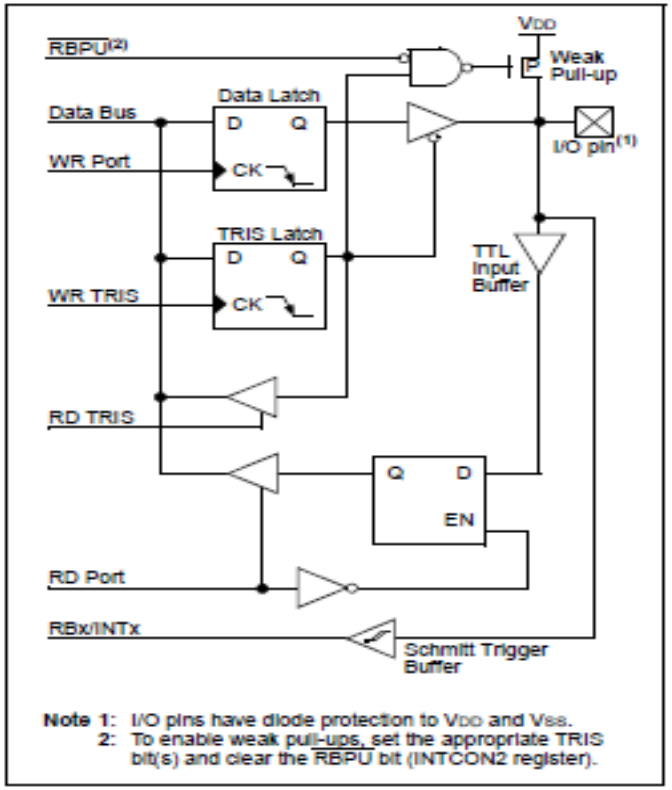
Cont.

FIGURE 9-4: RB7:RB4 PINS BLOCK DIAGRAM



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (INTCON2 register).

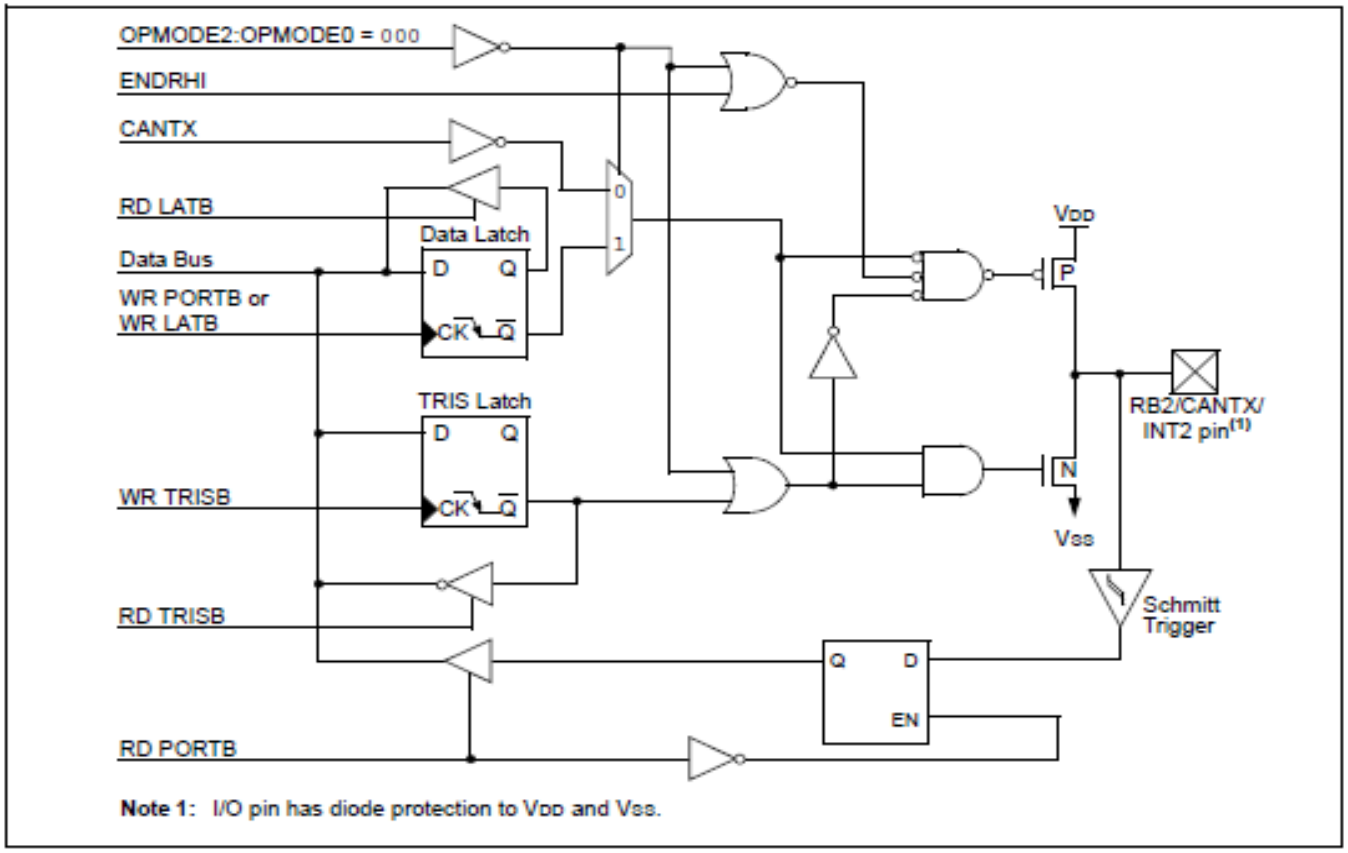
FIGURE 9-5: RB1:RB0 PINS BLOCK DIAGRAM



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (INTCON2 register).

Cont.

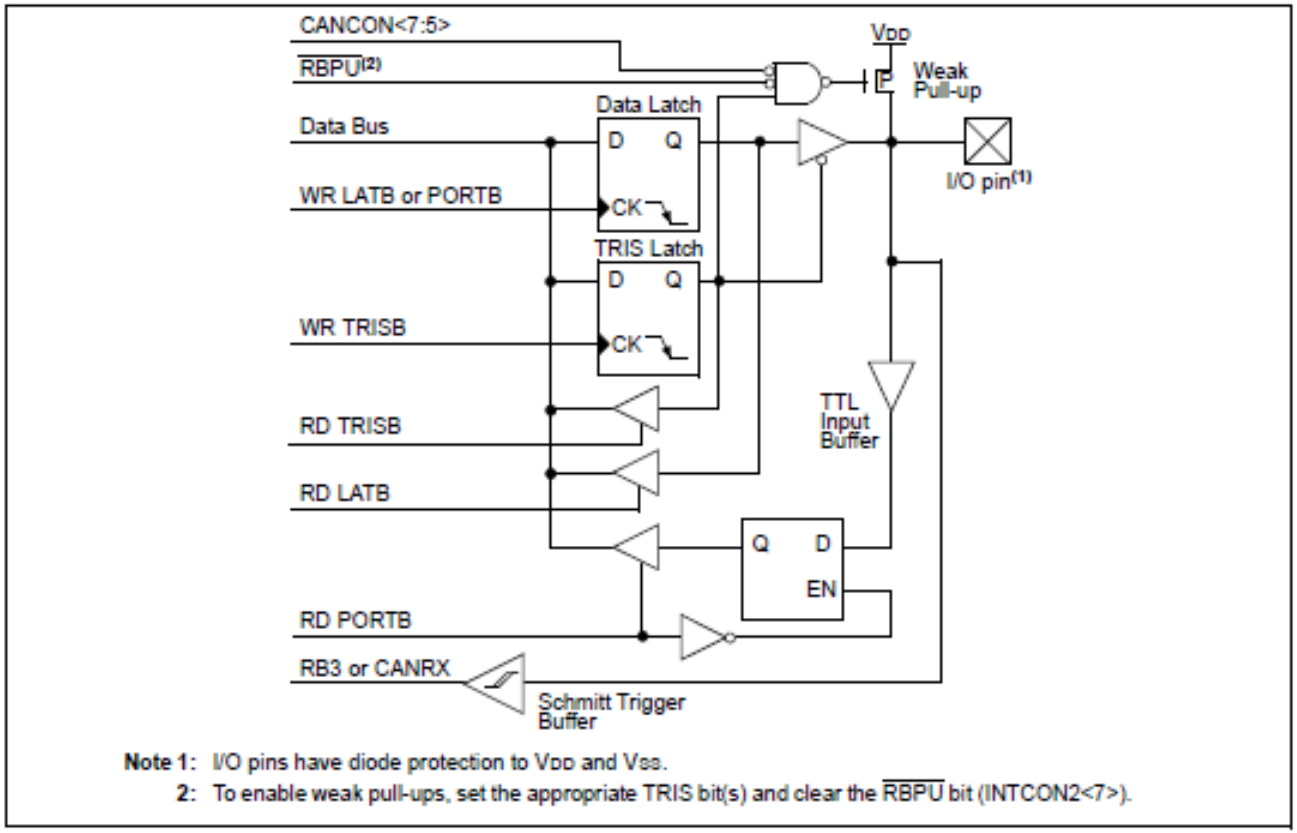
FIGURE 9-6: RB2/CANTX/INT2 PIN BLOCK DIAGRAM



Note 1: I/O pin has diode protection to VDD and VSS.

Cont.

FIGURE 9-7: RB3/CANRX PIN BLOCK DIAGRAM



Note 1: I/O pins have diode protection to VDD and VSS.

2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (INTCON2<7>).

Cont.

TABLE 9-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT0	bit 0	TTL/ST(1)	Input/output pin or external interrupt 0 input. Internal software programmable weak pull-up.
RB1/INT1	bit 1	TTL/ST(1)	Input/output pin or external interrupt 1 input. Internal software programmable weak pull-up.
RB2/CANTX/ INT2	bit 2	TTL/ST(1)	Input/output pin, CAN bus transmit pin or external interrupt 2 input. Internal software programmable weak pull-up.
RB3/CANRX	bit 3	TTL	Input/output pin or CAN bus receive pin. Internal software programmable weak pull-up.
RB4	bit 4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5/PGM	bit 5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Low-voltage serial programming enable.
RB6/PGC	bit 6	TTL/ST(2)	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit 7	TTL/ST(2)	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

TABLE 9-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
LATB	LATB Data Output Register								xxxx xxxx	uuuu uuuu
TRISB	PORTB Data Direction Register								1111 1111	1111 1111
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	—	—	TMR0IP	—	RBIP	111- -1-1	111- -1-1
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	11-1 0-00

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

PORTC, TRISC and LATC Registers

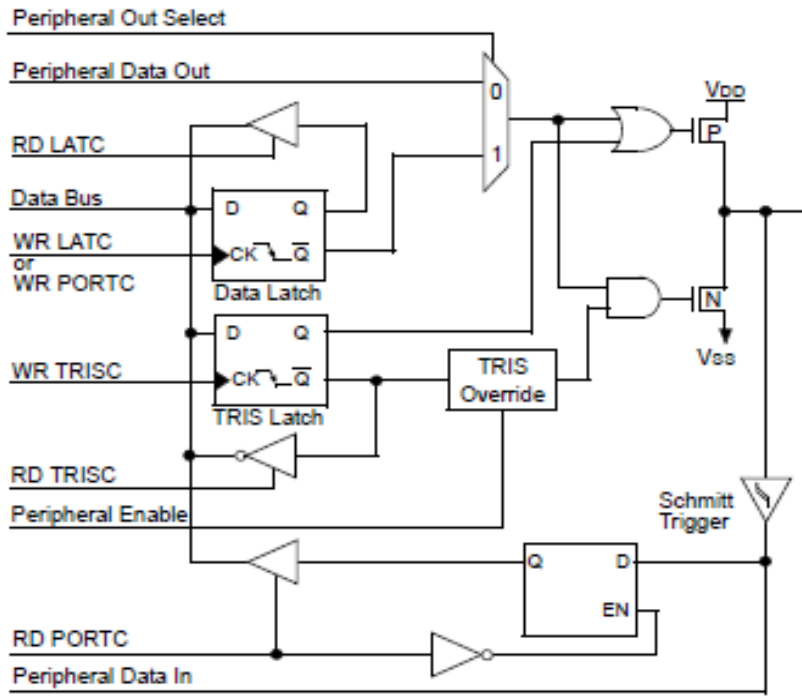
- PORTC is an 8-bit wide, bidirectional port. The corresponding Data Direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e.. put the contents of the output latch on the selected pin).
- Read-modify-write operations on the LATC register, read and write the latched output value for PORTC.
- PORTC is multiplexed with several peripheral functions (Table 9-5). PORTC pins have Schmitt Trigger input buffers.

Cont.

- When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.
- The pin override value is not loaded into the TRIS register. This allows read-modify-write of the TRIS register, without concern due to peripheral overrides.

Cont.

FIGURE 9-8: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE)



TRIS OVERRIDE

Pin	Override	Peripheral
RC0	Yes	Timer1 Oscillator for Timer1/Timer3
RC1	Yes	Timer1 Oscillator for Timer1/Timer3
RC2	No	—
RC3	Yes	SPI™/I²C™ Master Clock
RC4	Yes	I²C Data Out
RC5	Yes	SPI Data Out
RC6	Yes	USART Async Xmit, Sync Clock
RC7	Yes	USART Sync Data Out

Note 1: I/O pins have diode protection to VDD and VSS.

Cont.

TABLE 9-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit 0	ST	Input/output port pin, Timer1 oscillator output or Timer1/Timer3 clock input.
RC1/T1OSI	bit 1	ST	Input/output port pin or Timer1 oscillator input.
RC2/CCP1	bit 2	ST	Input/output port pin or Capture 1 input/Compare 1 output/ PWM1 output.
RC3/SCK/SCL	bit 3	ST	Input/output port pin or synchronous serial clock for SPI™/I^2C™.
RC4/SDI/SDA	bit 4	ST	Input/output port pin or SPI data in (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit 5	ST	Input/output port pin or synchronous serial port data output.
RC6/TX/CK	bit 6	ST	Input/output port pin, addressable USART asynchronous transmit or addressable USART synchronous clock.
RC7/RX/DT	bit 7	ST	Input/output port pin, addressable USART asynchronous receive or addressable USART synchronous data.

Legend: ST = Schmitt Trigger input

TABLE 9-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
LATC	LATC Data Output Register								xxxx xxxx	uuuu uuuu
TRISC	PORTC Data Direction Register								1111 1111	1111 1111

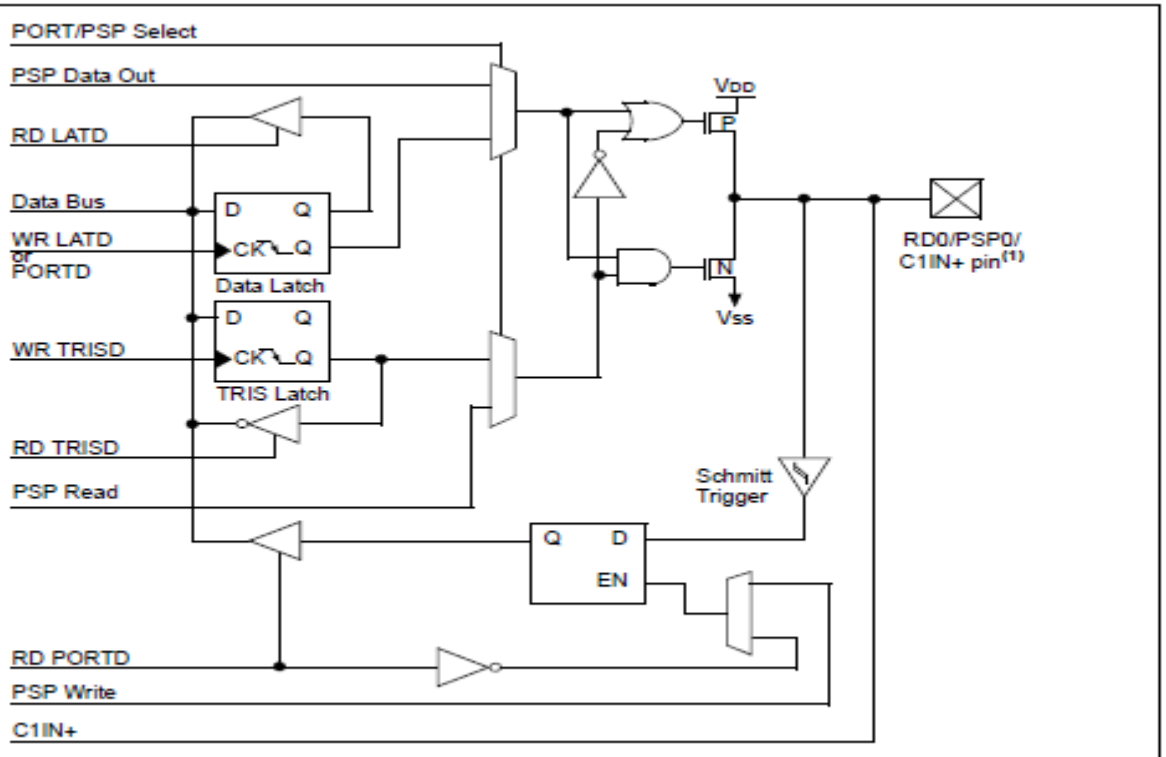
Legend: x = unknown, u = unchanged

PORTD, TRISD and LATD Registers

- Note:
- This port is only available on the PIC18F448 and PIC18F458.
- PORTD is an 8-bit wide, bidirectional port. The corresponding Data Direction register for the port is TRISD. Setting a TRISD bit (= 1) will make the corresponding PORTD pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISD bit (0) will make the corresponding PORTD pin an output (i.e., put the contents of the output latch on the selected pin).
- Read-modify-write operations on the LATD register read and write the latched output value for PORTD.
- PORTD uses Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.
- PORTD can be configured as an 8-bit wide, micro-processor port (Parallel Slave Port or PSP) by setting the control bit PSPMODE (TRISE<4>). In this mode, the input buffers are TTL. See Section 10.0 "Parallel Slave Port" for additional information.
- PORTD is also multiplexed with the analog comparator module and the ECCP module.

Cont.

FIGURE 9-9: PORTD BLOCK DIAGRAM IN I/O PORT MODE



Note 1: I/O pins have diode protection to VDD and VSS.

Cont.

TABLE 9-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0/C1IN+	bit 0	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 0 or C1IN+ comparator input.
RD1/PSP1/C1IN-	bit 1	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 1 or C1IN- comparator input.
RD2/PSP2/C2IN+	bit 2	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 2 or C2IN+ comparator input.
RD3/PSP3/C2IN-	bit 3	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 3 or C2IN- comparator input.
RD4/PSP4/ECCP1/P1A	bit 4	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 4 or ECCP1/P1A pin.
RD5/PSP5/P1B	bit 5	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 5 or P1B pin.
RD6/PSP6/P1C	bit 6	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 6 or P1C pin.
RD7/PSP7/P1D	bit 7	ST/TTL[1]	Input/output port pin, Parallel Slave Port bit 7 or P1D pin.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 9-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
LATD	LATD Data Output Register								xxxx xxxx	uuuu uuuu
TRISD	PORTD Data Direction Register								1111 1111	1111 1111
TRISE	IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0	0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

PORTE, TRISE and LATE Registers

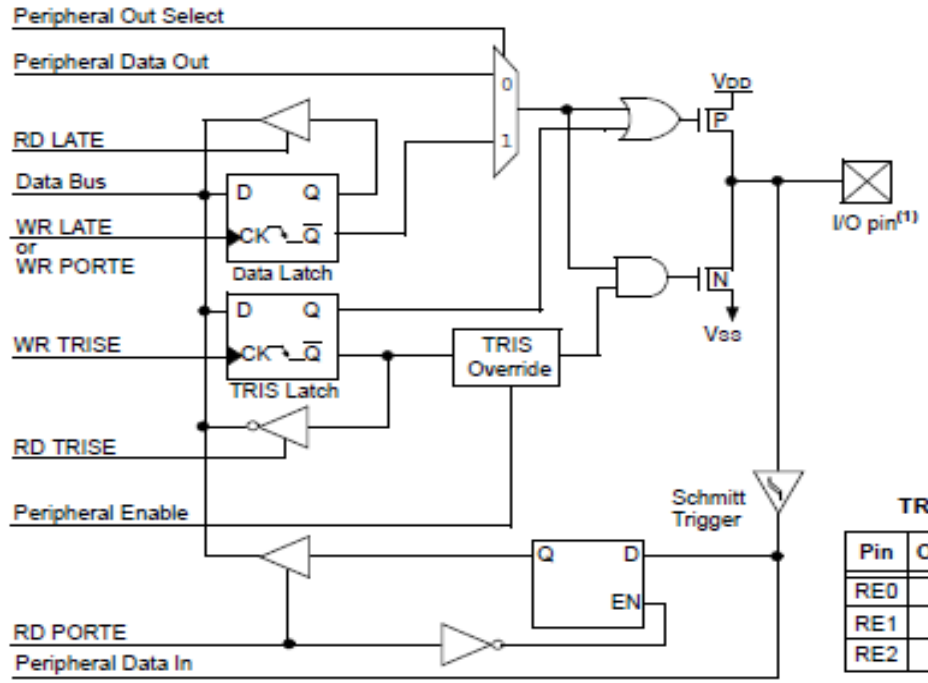
- Note:
- This port is only available on the PIC18F448 and PIC18F458
- PORTE is a 3-bit wide, bidirectional port. PORTE has three pins (RED/AN5/RD, RE1/AN6/WR/C1OUT and RE2/AN7/CS/C2OUT) which are individually config-urable as inputs or outputs. These pins have Schmitt Trigger input buffers.
- Read-modify-write operations on the LATE register, read and write the latched output value for PORTE.
- The corresponding Data Direction register for the port is TRISE. Setting a TRISE bit (1) will make the corresponding PORTE pin an input (i.e.. put the corre-sponding output driver in a high-impedance mode). Clearing a TRISE bit (= 0) will make the corresponding PORTE pin an output (i.e., put the contents of the output latch on the selected pin).

Cont.

- The TRISE register also controls the operation of the Parallel Slave Port through the control bits in the upper half of the register. These are shown in Register 9-1.
- When the Parallel Slave Port is active, the PORTE pins function as its control inputs. For additional details, refer to Section 10.0 "Parallel Slave Port".
- PORTE pins are also multiplexed with inputs for the A/D converter and outputs for the analog comparators. When selected as an analog input, these pins will read as '0's. Direction bits TRISE<2:0> control the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

Cont.

FIGURE 9-10: PORTE BLOCK DIAGRAM



TRIS OVERRIDE

Pin	Override	Peripheral
RE0	Yes	PSP
RE1	Yes	PSP
RE2	Yes	PSP

Note 1: I/O pins have diode protection to VDD and VSS.

Cont.

REGISTER 9-1: TRISE REGISTER

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0
bit 7							bit 0

bit 7 **IBF**: Input Buffer Full Status bit
1 = A word has been received and waiting to be read by the CPU
0 = No word has been received

bit 6 **OBF**: Output Buffer Full Status bit
1 = The output buffer still holds a previously written word
0 = The output buffer has been read

bit 5 **IBOV**: Input Buffer Overflow Detect bit (in Microprocessor mode)
1 = A write occurred when a previously input word has not been read (must be cleared in software)
0 = No overflow occurred

bit 4 **PSPMODE**: Parallel Slave Port Mode Select bit
1 = Parallel Slave Port mode
0 = General Purpose I/O mode

bit 3 **Unimplemented**: Read as '0'

bit 2 **TRISE2**: RE2 Direction Control bit
1 = Input
0 = Output

bit 1 **TRISE1**: RE1 Direction Control bit
1 = Input
0 = Output

bit 0 **TRISE0**: RE0 Direction Control bit
1 = Input
0 = Output

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Cont.

TABLE 9-9: PORTE FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/AN5/$\overline{RD}$	bit 0	ST/TTL(1)	Input/output port pin, analog input or read control input in Parallel Slave Port mode.
RE1/AN6/$\overline{WR}$/C1OUT	bit 1	ST/TTL(1)	Input/output port pin, analog input, write control input in Parallel Slave Port mode or Comparator 1 output.
RE2/AN7/$\overline{CS}$/C2OUT	bit 2	ST/TTL(1)	Input/output port pin, analog input, chip select control input in Parallel Slave Port mode or Comparator 2 output.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 9-10: SUMMARY OF REGISTERS ASSOCIATED WITH PORTE

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
TRISE	IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0	0000 -111	0000 -111
PORTE	—	—	—	—	—	Read PORTE pin/ Write PORTE Data Latch			---- -xxx	---- -uuu
LATE	—	—	—	—	—	Read PORTE Data Latch/ Write PORTE Data Latch			---- -xxx	---- -uuu
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTE.

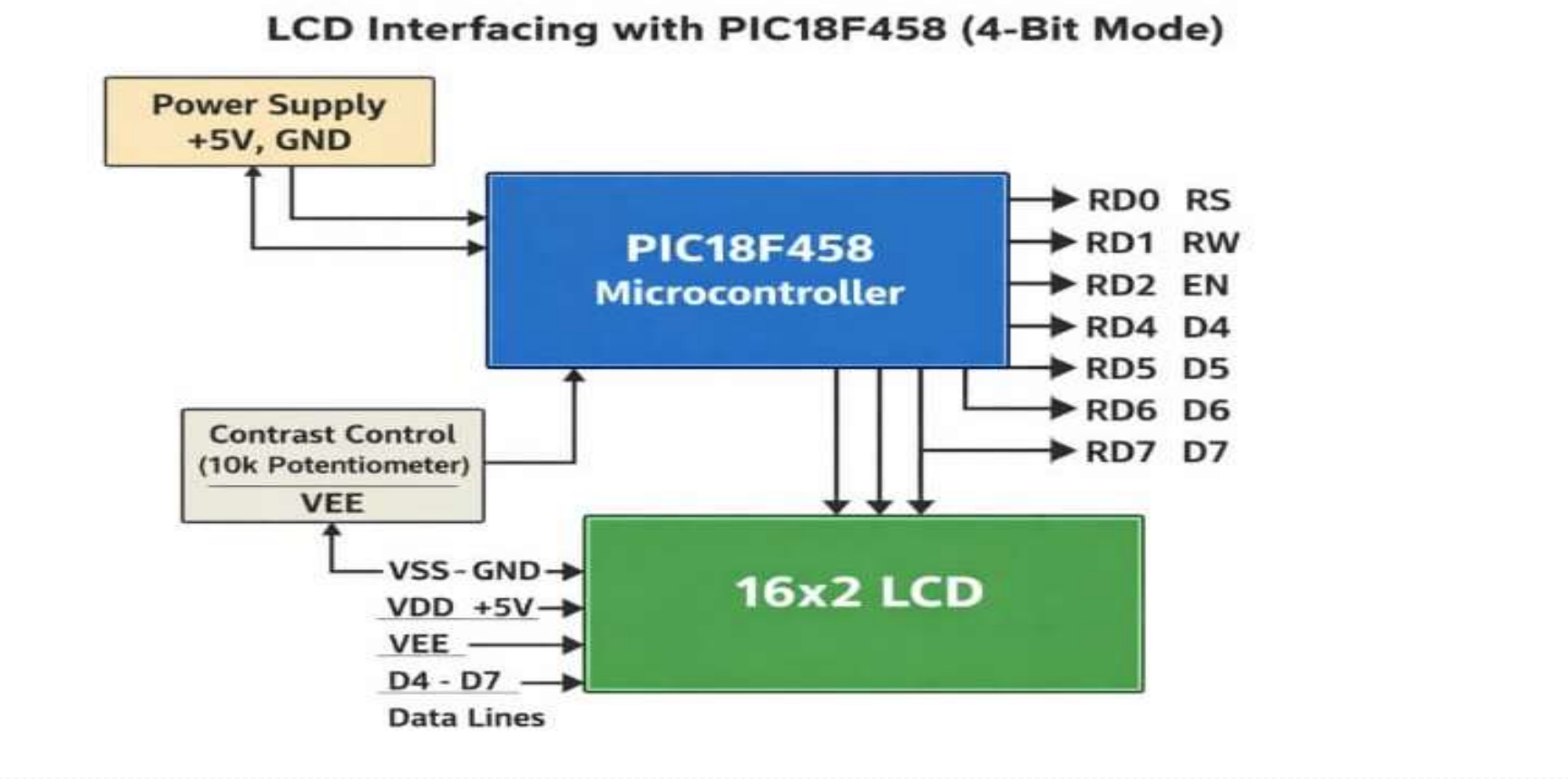
The Configuration Registers

- PIC18F452 microcontrollers have a set of configuration registers (PIC16-series microcontrollers had only one configuration register). Configuration registers are programmed during the programming of the flash program memory by the programming device.

LCD Interfacing with PIC18F458

- A Liquid Crystal Display (LCD) is commonly used to display text, numbers, and symbols in embedded systems.
- The most widely used LCD is 16×2 alphanumeric LCD, which can display 16 characters per line and 2 lines.
- PIC18F458 interfaces with LCD to display:
- Sensor values
- Status messages
- Menu options

LCD Interfacing with PIC18F458



LCD Registers

- (a) Command Register (RS = 0)
- Used to send instructions such as:
- Clear display
- Cursor ON/OFF
- Set LCD mode
- (b) Data Register (RS = 1)
- Used to send characters to be displayed.

Interfacing Modes

- (A) 8-Bit Mode
- Uses D0–D7 (8 data pins)
- Faster
- Uses more PIC pins
- (B) 4-Bit Mode (Most Common)
- Uses D4–D7 only
- Saves PIC pins
- Slightly slower
- ☞ PIC18F458 usually uses 4-bit mode

Working Principle

- 1. PIC sends command to initialize LCD
- 2. LCD is set in 4-bit mode
- 3. PIC sends higher nibble first, then lower nibble
- 4. Enable pin is pulsed to latch data
- 5. Characters are displayed

Advantages of LCD Interfacing

- Low power consumption
- Easy to program
- Clear visual output
- Widely used in embedded systems

Applications

- Digital meters
- Industrial control panels
- Medical instruments
- Embedded projects

Keyboard Interfacing with PIC18F458

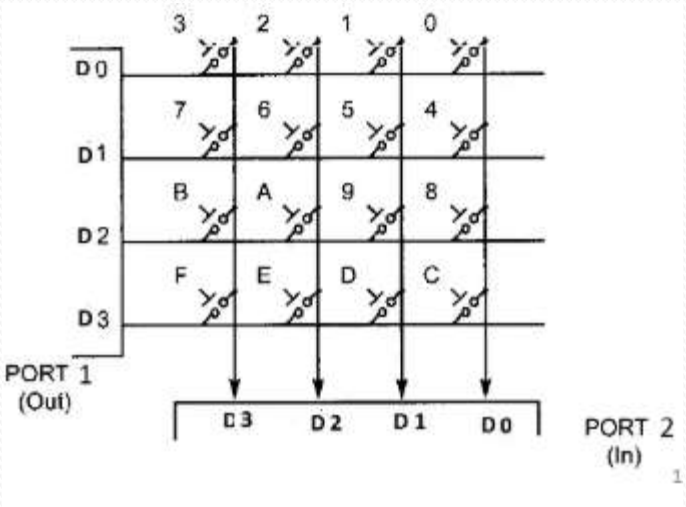
- A keyboard is an input device used to enter numeric or alphanumeric data into a microcontroller system.
- In embedded systems, a matrix keyboard (4×4 or 4×3) is commonly used to reduce the number of I/O pins.
- PIC18F458 reads key presses and processes them for:
- Password entry
- Menu selection
- Calculator systems
- Control applications

Types of Keyboards

- (A) Numeric Keyboard
- Contains digits 0–9
- Example: 4×3 keypad
- (B) Alphanumeric Keyboard
- Contains A–Z, 0–9, symbols
- Example: PC keyboard (rare in MCUs)
- ☞ Most common: 4×4 Matrix Keyboard

4×4 Matrix Keyboard Structure

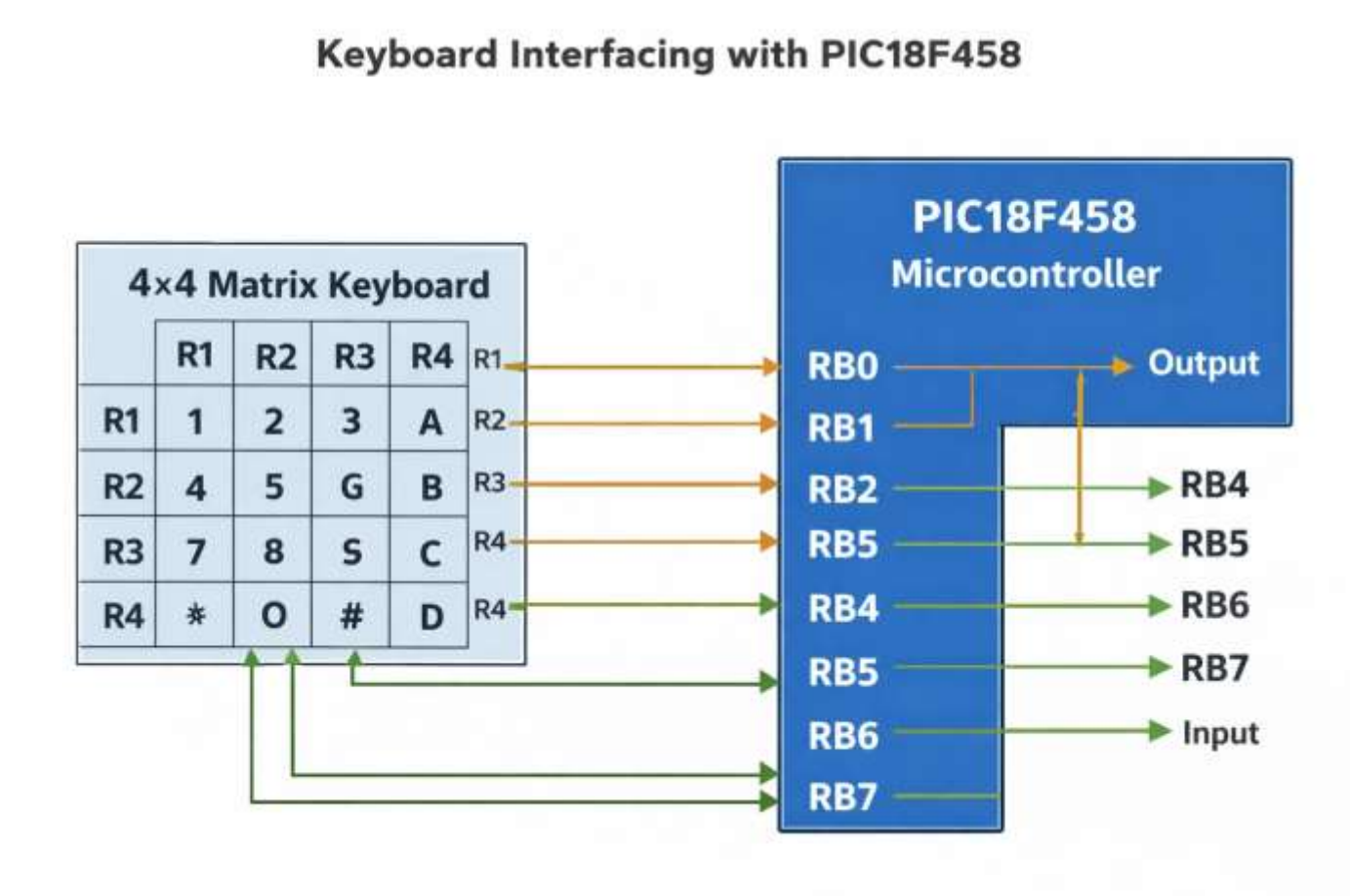
- Keyboards are organized in a matrix of rows and columns
- The CPU accesses both rows and columns through ports.
- 4 Rows (R1–R4)
- 4 Columns (C1–C4)
- Total keys = 16



Cont.

- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed.
- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns.
- If the data read from columns is D3D0 = 1111, no key has been pressed and the process continues till key press is detected.
- If one of the column bits has a zero, this means that a key press has occurred. For example, if D3 - D0 = 1101, this means that a key in the D1 column has been pressed.
- After detecting a key press, microcontroller will go through the process of identifying the key.

Cont.

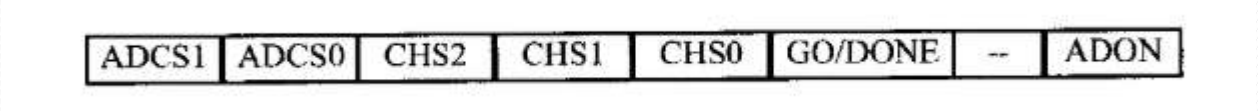

Keyboard Interfacing with PIC18F458
4×4 Matrix Keyboard
R1 R2 R3 R4
R1 1 2 3 A
R2 4 5 G B
R3 7 8 S C
R4 * O # D
R1
R2
R3
R4
R4
PIC18F458
Microcontroller
RB0
RB1
RB2
RB5
RB4
RB5
RB6
RB7
Output
RB4
RB5
RB6
RB7
Input

ADC Interfacing with PIC18F458

- As the ADC is widely used in data acquisition, in recent years an increasing number of microcontrollers have an on-chip ADC peripheral, just like timers and USART. An on-chip ADC eliminates the need for an external ADC connection, which leaves more pins for other I/O activities. The vast majority of the PIC18 chips come with 8 channels of ADC, and some PIC18s have as many as 16 channels of ADCs.
- PIC18 ADC features programming
- The ADC peripheral of the PIC18 has the following characteristics:
- (a) It is a 10-bit ADC.
- (b) It can have 5 to 15 channels of analog input channels, depending on the family member. In PIC18, pins RA0-RA7 of PORTA are used for the 8 analog channels.
- (c) The converted output binary data is held by two special function registers called ADRESL (A/D Result Low) and ADRESH (A/D Result High).

ADCON0 register

- The ADCON0 register is used to set the conversion time and select the analog input channel among other things. In order to reduce the power consumption of the PIC18, the ADC feature is turned off when the microcontroller is powered up.



- We turn on the ADC with the ADON bit of the ADCON0 register. The other important bit is the GO/DONE bit. We use this bit to start conversion and monitor it to see if conversion has ended.

DAC Interfacing with PIC18F458

- The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals.
- The DAC0808 is an 8-bit monolithic digital-to-analog converter (DAC).
- featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with ±5V supplies.
- No reference current (IREF) trimming is required for most applications since the full scale output current is typically ±1 LSB of 255 IREF/256.
- Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8-bit zero accuracy for IREF≥2 mA.
- The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

Block Diagram of DAC 0808

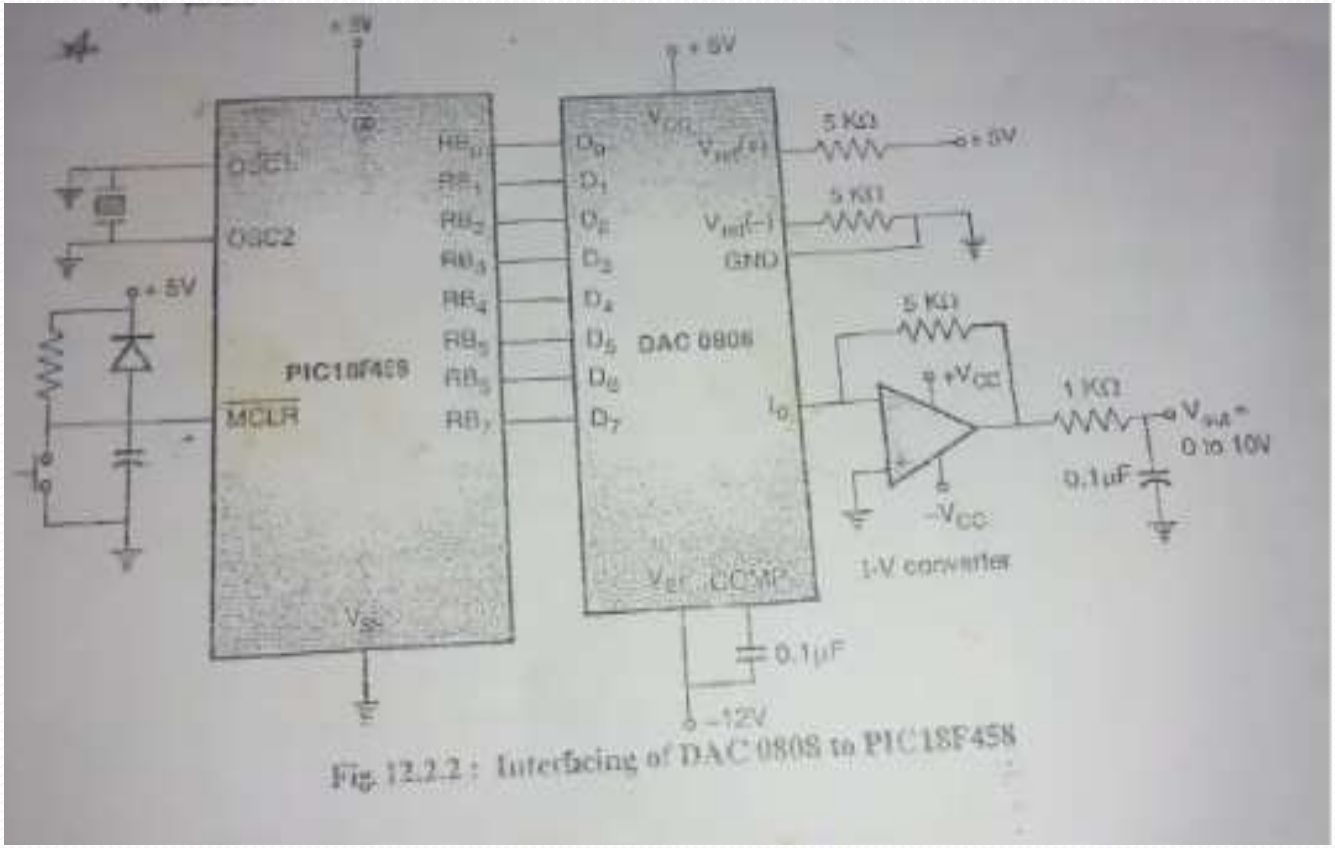


Fig. 12.2.2 : Interfacing of DAC 0808 to PIC18F458

Working Principle (Step-by-Step)

- 1. PIC18F458 sends an 8-bit digital value to PORTB
- 2. PORTB data appears at DAC0808 pins D0–D7
- 3. DAC0808 converts digital data into proportional current
- 4. Op-amp converts this current into analog voltage
- 5. Filter smoothens the output
- 6. Final output voltage varies from 0 to 10 V

Applications

- Waveform generation (sine, ramp, triangular)
- Motor speed control
- Audio signal generation
- Data acquisition systems
- Industrial control systems

Advantages

- High speed
- Simple interfacing
- Good resolution (8-bit)
- Low cost

Relay Interfacing with PIC18F458

- A **relay** is an **electromechanical switch** that lets a low-voltage microcontroller control a **high-voltage or high-current load** like a lamp, motor, or appliance.
  When you energize the coil with the right voltage, it closes or opens the contact between **Common (COM)** and **Normally Open (NO)** or **Normally Closed (NC)** contact terminals.

Why You Need a Driver Circuit?

- A PIC18F458 pin **cannot directly drive a relay** because:
- A relay coil typically needs **50 mA or more**, while a PIC GPIO can only source/sink ~25 mA safely.
  The relay coil produces **back-EMF (voltage spikes) when switched off**, which can damage the microcontroller if not protected.
- So instead of connecting the relay directly to the PIC pin, you use:
- A **transistor (e.g., BC547 or ULN2003)** as a switch
  A **flyback diode** (e.g., 1N4148 or 1N4007) to protect against back-EMF

Pin / Component Roles

Component	Role
PIC pin (e.g., **RB1**)	Sends logic 0/1 to switch relay
Transistor (e.g., BC547)	Amplifies current so relay coil can energize
Resistor (1 K)	Limits base current into transistor
Flyback diode	Prevents voltage spikes from coil entering PIC
Relay coil	Gets energized when transistor switches on
COM / NO / NC	Connects or disconnects load (AC/DC) when relay actuates

How It Works (Step by Step)?

- **PIC outputs HIGH** (logic 1) on a port pin (e.g., RB1).
- Current flows through base resistor into the transistor, turning it **ON**.
- Transistor saturates, allowing current through the relay coil from supply to ground.
- The relay coil energizes, switching the contact from NC to NO.
- A **diode** across the coil absorbs the coil's back-EMF when transistor switches **OFF**. **Driving a Load**
- The load (like a **230 V AC lamp**) is connected separately through the relay's NO and COM contacts; **the microcontroller never directly handles the high voltage**.
- Always follow safety practices (e.g., fuses, proper insulation) when dealing with mains.

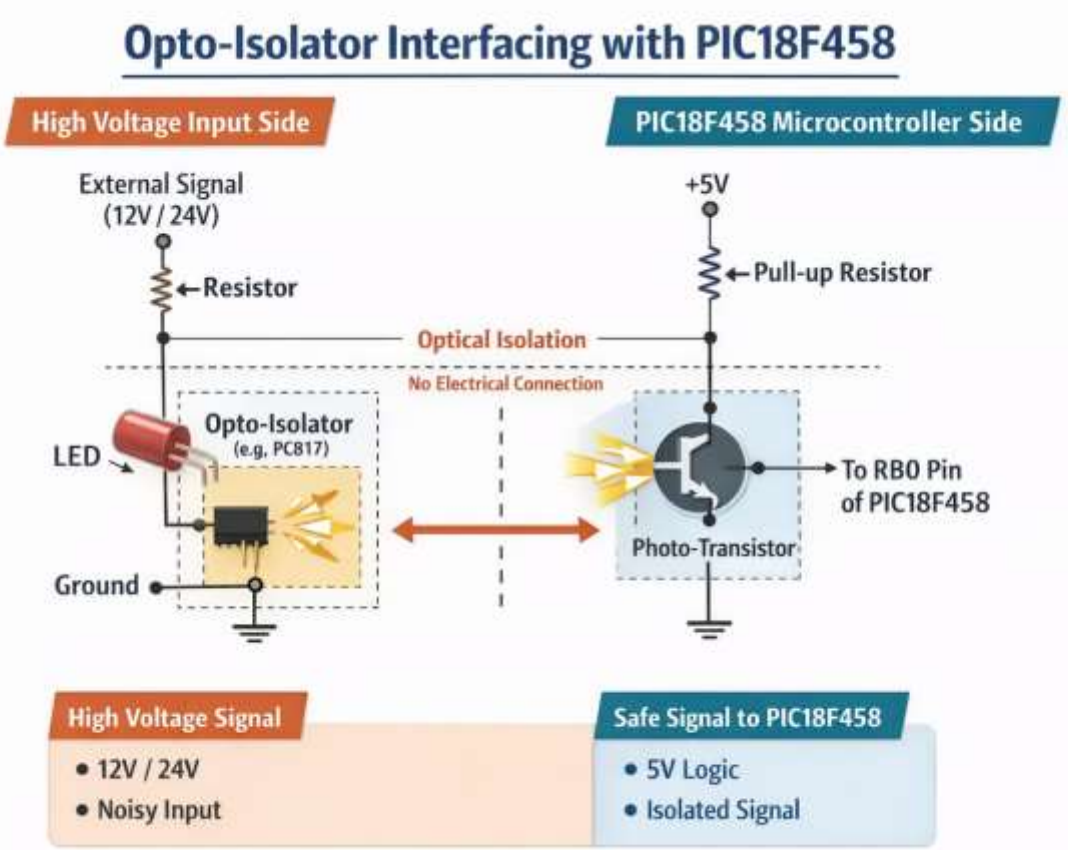
Optoisolators interfacing with PIC18F458

- An **opto-isolator** (opto-coupler) is a device that transfers a signal **using light**, not a direct electrical connection.
- Inside it has:
- **LED (input side)**
- **Phototransistor / photodiode / triac (output side)**
- Because there is **no electrical connection**, it protects the microcontroller from:
- High voltage
- Noise
- Ground loops
- Spikes from relays, motors, AC lines

Cont.

- Common opto-isolators:
- **PC817** (very common)
- **4N25 / 4N35**
- **MOC3021** (for TRIAC / AC loads)
- **Why Use Opto-Isolator with PIC18F458?**
- PIC18F458 operates at **5 V logic**, and its I/O pins can handle only small currents.
- Opto-isolators are used when:
- Input signal is **high voltage**
- Input comes from **AC / industrial sensors**
- Output controls **relays, motors, or noisy circuits**

Optoisolators interfacing with PIC18F458



Circuit Diagram Explanation.

- Input Side (High Voltage Side)
- External signal is applied to the LED through a current-limiting resistor.
- When input signal is HIGH, LED glows.
- Output Side (PIC Side)
- Light from LED activates photo-transistor.
- Transistor conducts → PIC input pin goes LOW or HIGH (depending on connection).
- PIC reads this as a logic signal.

Interfacing with PIC18F458 (Example)

Opto-Isolator Pin	Connected To
LED Anode	External Signal via resistor
LED Cathode	Ground (external)
Collector	+5V via pull-up resistor
Emitter	PIC GND
Collector Output	PIC18F458 RB0 pin

Stepper Motor Interfacing with PIC18F458

- A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor.
- The rotary motion occurs in a step-wise manner from one equilibrium position to the next.
- Stepper Motors are used very wisely in position control systems like printers, disk drives, process control machine tools, etc.
- The basic two-phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding.
- The excitation of any one winding generates a North Pole. A South Pole gets induced at the diametrically opposite side.
- The rotor magnetic system has two end faces. It is a permanent magnet with one face as South Pole and the other as North Pole.
- The Stepper Motor windings A1, A2, B1, B2 are cyclically excited with a DC current to run the motor in clockwise direction. By reversing the phase sequence as A1, B2, A2, B1, anticlockwise stepping can be obtained.

Cont.

- Step Angle
- The number of steps required to complete one full rotation depends on the step angle of the stepper motor. The step angle can vary from 0.72 degrees to 15 degrees per step. Depending on that 500 to 24 steps may be required to complete one rotation.
- In position control applications the selection on motor should be based on the minimum degree of rotation that is required per step.

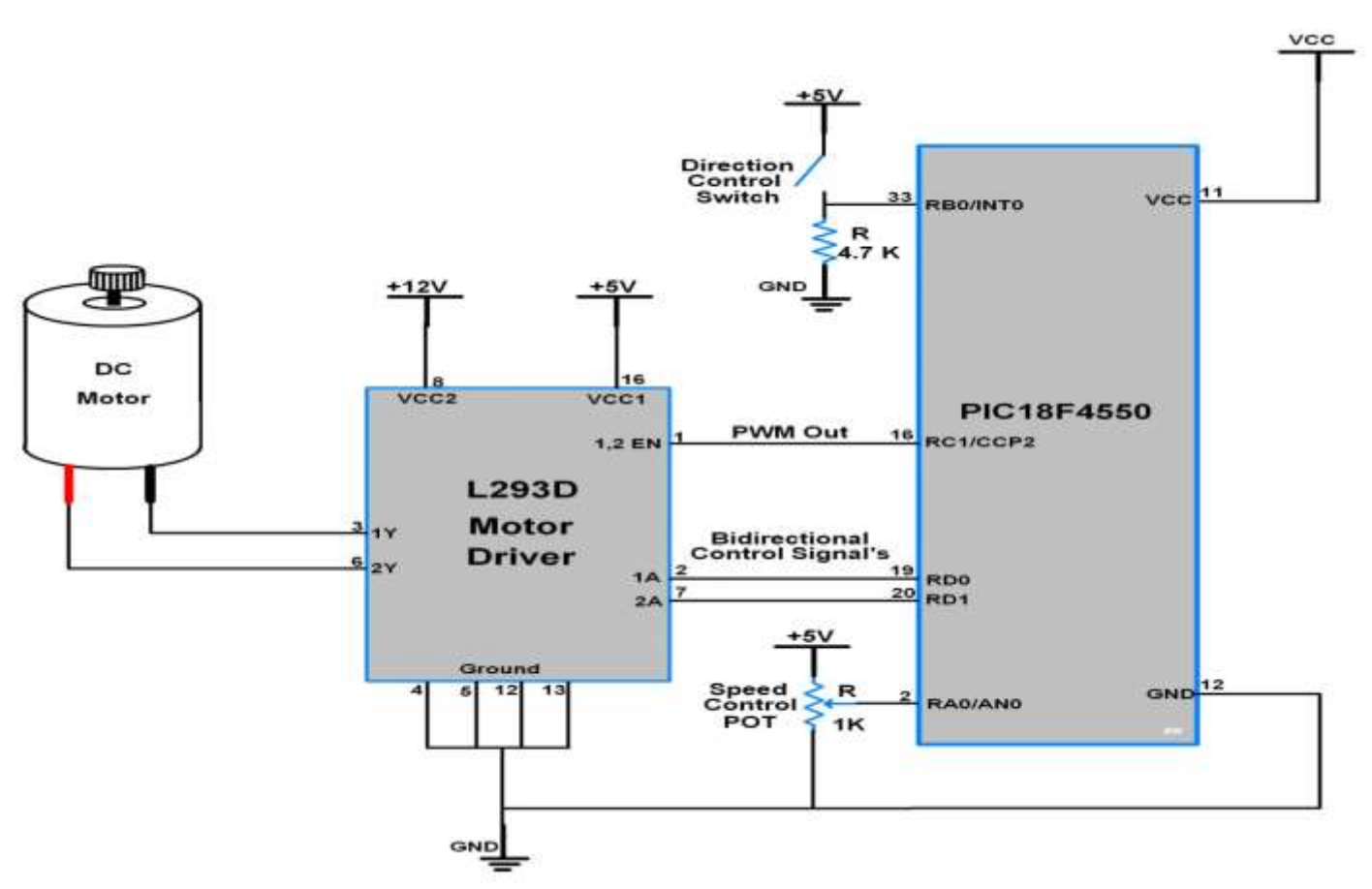
DC motor interfacing with PIC18F4550

- C motor converts electrical energy in the form of Direct Current into mechanical energy.
- In the case of the motor, the mechanical energy produced is in the form of a rotational movement of the motor shaft.
- The direction of rotation of the shaft of the motor can be reversed by reversing the direction of Direct Current through the motor.

Cont.

- The motor can be rotated at a certain speed by applying a fixed voltage to it. If the voltage varies, the speed of the motor varies.
- Thus, the DC motor speed can be controlled by applying varying DC voltage; whereas the direction of rotation of the motor can be changed by reversing the direction of current through it.
- For applying varying voltage, we can make use of the PWM technique.
- For reversing the current, we can make use of H-Bridge circuit or motor driver ICs that employ the H-Bridge technique or other any other mechanisms.

Cont.


VCC
+5V
Direction
Control
Switch
33
RB0/INT0
VCC
11
R
4.7 K
GND
+12V
+5V
DC
Motor
8
VCC2
16
VCC1
PIC18F4550
1,2 EN
1
PWM Out
16
RC1/CCP2
L293D
Motor
Driver
3
1Y
6
2Y
Bidirectional
Control Signal's
1A
2
19
RD0
2A
7
20
RD1
+5V
Ground
4
5
12
13
Speed
Control
POT
R
1K
2
RA0/AN0
GND
12
GND

- **Control the speed of the DC Motor using PIC18F4550**
- Here, we are going to interface the DC motor with a PIC18F4550 microcontroller. In which we will control the DC motor speed by using POT connected to ADC of PIC18F4550 and direction by using a switch.
- We are going to use the L293D motor driver IC to control the DC motor movement in both directions. It has an in-built H-bridge motor drive.

Cont.

- As shown in the above figure we have connected 1KΩ Potentiometer at ADC channel 0 of PIC18F4550 to change the speed of the DC motor.
- One toggle switch is connected to the INT0 pin which controls the motor rotating direction.
- PORTD is used as an output control signal port. It provides control to motor1 input pins of the L293D motor driver which rotates the motor clockwise and anticlockwise by changing their terminal polarity.