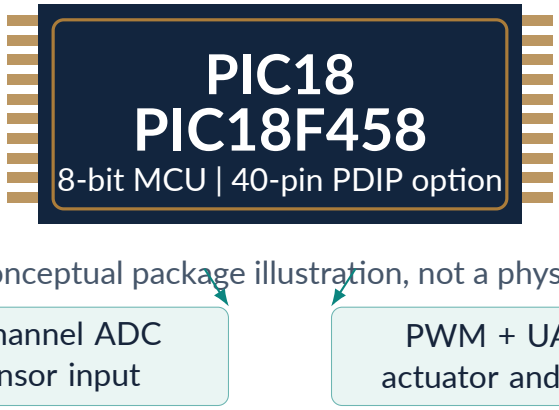
ZENITH STUDY HUB

Microprocessor & Micro-controller

UNIT IV | PIC18F458

Architecture, I/O, timers, PWM, ADC, interrupts and UART



Exactly matched to the official Unit IV syllabus: Introduction to PIC18F458; PIC18 Architecture; PIC18 I/O Ports & Configuration; PIC18 Timers & PWM; ADC in PIC18F458; PIC18 Interrupts & UART.

Study method: Learn the idea, inspect the diagram, derive the register formula, work a numerical example, then solve the self-check. Register-level listings are explicitly labelled pseudocode unless a native toolchain has compiled them.

Primary device: Microchip PIC18F458, DS41159E (2006).
No PIC18F4580-specific registers are substituted for this device.
Syllabus: SPPU PCC302-ROA, TE Robotics & Automation, 2024 Pattern.
Prepared by: Om Deshmukh **Revision:** 21 November 2026

How to use these notes

In simple words
The PIC18F458 is best learned as a sequence: *CPU makes decisions* → *pins receive inputs* → *timers schedule events* → *ADC measures analog sensors* → *PWM produces drive commands* → *UART communicates*. All the peripherals are controlled through special function registers (SFRs).

Contents

1 Introduction: what is PIC18F458? 3

2 PIC18 architecture: CPU, buses and memories 5

3 PIC18 I/O ports and configuration 8

4 Timers and PWM: generating time and power commands 10

5 Analog-to-digital conversion: measuring real sensors 13

6 Interrupts: event handling 17

7 UART: exchanging bytes over two wires 19

8 Integrated robotics case: sensor to PWM with UART logging 22

9 Revision tables, solved numericals and answer checkpoints 24

Official syllabus topic	Learning location
Introduction to PIC18F458	Section 1: purpose, precise device features, use cases
PIC18 architecture	Section 2: CPU, memory, WREG, BSR, instruction cycle
PIC18 I/O ports and configuration	Section 3: TRIS, PORT, LAT, analog multiplexing
PIC18 timers and PWM	Section 4: Timer0–3, Timer2, PR2, CCP/ECCP
ADC in PIC18F458	Section 5: multiplexing, sample/hold, 10-bit output
PIC18 interrupts and UART	Sections 6–7: priority, vectors, baud, RX errors

Engineering rule. A microcontroller pin cannot drive a motor or relay coil directly. Always design a protected driver stage with suitable power, fly-back protection and electrical isolation where needed. A software ISR is not a certified safety stop.

1 Introduction: what is PIC18F458?

In simple words
The PIC18F458 is a programmable 8-bit controller. It combines a small CPU, nonvolatile program memory, RAM, EEPROM, digital input/output pins and hardware peripherals. You can use it to measure a sensor, make a decision, and create a timing or communication signal without an external general-purpose computer.

1.1 Device facts that must be kept distinct

Property	PIC18F458 specifically
Core and instruction family	8-bit PIC18 enhanced RISC core; 75 instruction types
Program Flash	32 KiB (32,768 bytes), physically implemented
Data RAM	1,536 bytes (1.5 KiB), physically implemented
Data EEPROM	256 bytes, nonvolatile data storage
Maximum oscillator specification	DC to 40 MHz; typical instruction clock $f_{osc}/4$
Maximum instruction rate	Up to 10 million instruction cycles per second at 40 MHz
I/O banks	Ports A, B, C, D and E; pin functions are multiplexed
Timer units	Timer0, Timer1, Timer2 and Timer3
Analog inputs	Up to 8 channels, one 10-bit ADC via multiplexer
Communication	Addressable USART, MSSP, CAN and PSP interfaces
Packages	40-pin PDIP and 44-pin PLCC/TQFP options

Do not confuse: PIC18F458 is *not* PIC18F4580. The extra 0 changes the silicon family and can change control registers, oscillator choices and example code. The syllabus names the older PIC18F458, so these notes use DS41159E. *Source:* Microchip DS41159E, Table 1-1, pp. 7–9.

1.2 PIC versus a general microprocessor

A processor such as the 8086 executes instructions but typically needs externally connected memory and interfaces. The PIC18F458 includes the hardware needed for many embedded-control tasks on one chip. That does not mean it is automatically faster or better for every task. It means common I/O, timing, and measurement resources are already integrated.

1.3 A concrete robotics use

An automated gripper has a position sensor, motor-driver input, limit switch and serial diagnostic output. A PIC18F458 can use the ADC to measure position, a digital input to read the limit switch, PWM to command the driver and

UART to send a report. The actual motor power comes from a driver, not a PIC pin.

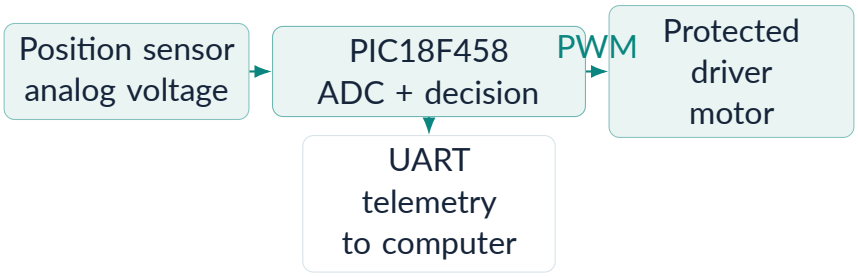


Figure 1. Original functional robotics diagram: ADC input → PIC processing → driver control; UART handles telemetry.

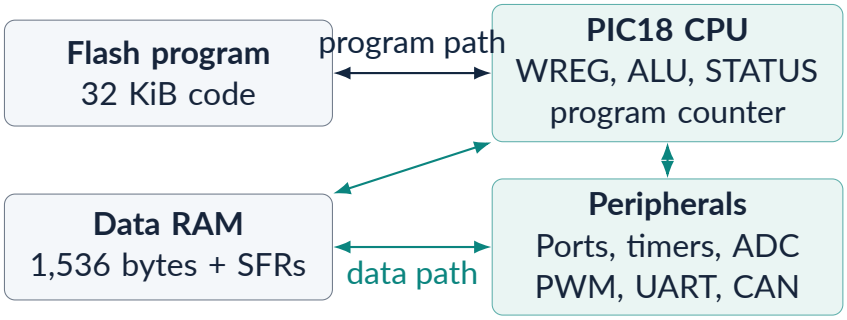
Exam template: Define a microcontroller, list the exact PIC18F458 specifications, draw the block-level system, and justify why ADC/PWM/interrupt hardware is useful in automation.

2 PIC18 architecture: CPU, buses and memories

In simple words
The PIC18 CPU fetches instructions from Flash and manipulates data held in RAM or SFRs. A Harvard-style design maintains logically distinct program and data paths. The CPU uses WREG for many computations, while the status register and a bank select register support control and addressing.

2.1 Harvard organization and an 8-bit datapath

Program instructions are nominally 16 bits wide in the PIC18 core, with some instructions requiring two instruction words. Data operations commonly use 8-bit register values. A 16-bit instruction word does not make the ALU a 16-bit general-purpose data processor. The PIC18F458 architecture includes instruction decoding, an 8-bit ALU, a hardware return stack and peripheral register interfaces.



Conceptual, source-based architecture; not an internal logic schematic

Figure 2. Two logical memory domains: program Flash feeds the CPU; RAM/SFR paths hold working data and peripheral state.

2.2 Important CPU register groups

Register	Role in ordinary language
WREG	Working register used for arithmetic, immediate values and many transfers
STATUS	Result flags, including zero, carry and digit carry
BSR	Bank Select Register: selects file-register bank in banked addressing
FSR0/1/2	Indirect pointers for data access through INDF forms
PC, PCLATH, PCLATU	Program-counter addressing and branch-related upper bits
STKPTR/TOS	Hardware return-stack depth/top-of-stack access
PRODH/PRODL	Hardware multiplier result storage for 8-by-8 multiply

2.3 The banked-addressing trap

SFRs and general data RAM share the *data address space*, but only installed RAM locations exist. A banked instruction forms an address using the BSR and its file-register operand. An ACCESS-type instruction uses a special access region and is not simply the same as BSR banking. Exact access-bank windows must be checked in the manufacturer data-memory map; do not assume any arbitrary bank number maps to physical SRAM.

STEP-BY-STEP EXAMPLE: Why can two instructions with the same file-

register number access different data?

Suppose a banked instruction addresses file offset 20H. With BSR=0, its logical file address is 020H. With BSR=2, the logical address is 220H. If that latter bank is not physically populated by RAM in the target device, it is *not* valid scratch memory. Always reconcile the logical address with the PIC18F458 implemented-memory map.

2.4 Program, data and EEPROM memory are different

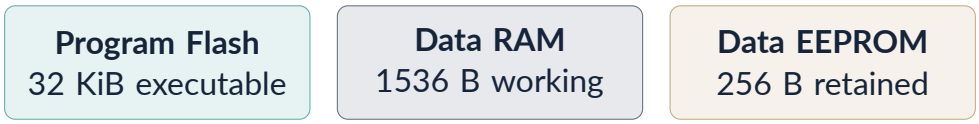


Figure 3. Memory types contrasted. A unit conversion (bits vs bytes) or a reset-behaviour assumption can change an engineering answer.

Flash stores firmware, SRAM stores temporary variables and most SFRs, and EEPROM retains small persistent user parameters through power-off. EEPROM writes are slower and have endurance limits. Interrupt vector addresses belong to *program* memory, not to RAM.

2.5 Instruction-cycle frequency and timing

For the ordinary PIC18 system clock, $T_{CY} = 4/f_{osc}$. At 20 MHz, one instruction cycle takes $0.2\,\mu$s; at 40 MHz, $0.1\,\mu$s. Many instructions require one cycle, while control-flow and multiword forms may require more; a program's execution time cannot be calculated by multiplying every source line by exactly one cycle.

STEP-BY-STEP EXAMPLE: Instruction cycle at 8 MHz

$T_{CY} = 4/(8\times10^6) = 0.5\,\mu$s. A 1000-cycle code segment takes about 0.5 ms *if* it actually uses 1000 device instruction cycles. Pipeline changes and interrupts modify exact wall-clock timing.

Worked answer: With an 8 MHz oscillator, the nominal instruction cycle frequency is 2 MHz and a cycle lasts 0.5 μs.

3 PIC18 I/O ports and configuration

In simple words
A pin can be an input, output or peripheral signal. The TRIS register chooses digital direction: 1 usually means input and 0 output. The PORT register reads current pin levels; LAT stores output latch state. Analog functions must be configured separately so the pin behaves as intended.

3.1 Five ports, but pins have alternative jobs

PIC18F458 implements PORTA through PORTE. Some pins connect to the ADC, oscillator, CAN, USART, parallel slave port or capture/PWM functions. Package pin count does not equal independent GPIO count. Analog defaults and peripheral assignments must be configured before digital I/O assumptions become valid.

Port	Typical examples of multiplexed functions
PORTA	AN0–AN4, voltage references, T0CKI, oscillator option on RA6
PORTB	External interrupt inputs, CAN receive/transmit, programming pins
PORTC	UART TX/RX, SPI/I^2C, CCP1; timer oscillator pins
PORTD	ECCP1/P1A–P1D and parallel slave port functions
PORTE	AN5–AN7 plus PSP control functions

Device distinction: For the 40-pin package, check the Microchip Table 1-2 pin list for the precise physical pin numbers. This figure is a functional summary, not a substitute for the official package pinout.

3.2 TRIS, PORT, LAT: a three-step mental model

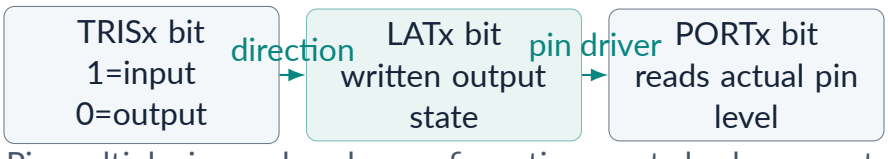


Figure 4. Direction, output latch, and pin input have different responsibilities; LAT avoids read-

modify-write hazards on outputs.

3.3 The read-modify-write problem

When firmware changes one output bit using a PORT read-modify-write operation, it may read a *physical pin* whose level differs from the intended latch value (for example, because of capacitive loading). Writing the read-back result can unintentionally change other outputs. On PIC18, use the relevant LAT register for modifying output-latch state where appropriate, and PORT when sampling real external input levels.

3.4 Digital input versus analog-channel configuration

The PIC18F458 ADC channel selection and analog/digital function are configured through ADCON0 and ADCON1 (with ADCS bits and PCFG bits). A TRIS bit alone does not force a multiplexed analog channel to become a fully configured digital input. Conversely, configuring AN0 as analog input requires a compatible analog selection and input direction, not merely reading PORTA.

Reset does not imply ready-to-use GPIO. Configure the oscillator, analog function, digital direction, latches, pull-up/driver circuits and alternate peripheral modes for the real board. Example snippets assume those decisions have already been made.

3.5 Worked I/O design: a status LED and limit sensor

Assume a properly protected LED driver is connected to an available digital output and a conditioned limit switch to a digital input. Initialization must (1) disable incompatible analog functions for those pins, (2) write the safe output latch state, (3) program TRIS direction, and (4) read the input from PORT while changing outputs through LAT. Avoid assuming a particular pin supports these roles without consulting the package pin table.

```
// REGISTER-LEVEL PSEUDOCODE, NOT COMPILED XC8:
configure_device_oscillator();
configure_selected_pins_as_digital();
LAT_output_bit = 0;          // safe output state
TRIS_output_bit = 0;         // configure output
TRIS_switch_bit = 1;         // configure input
```

```
while (1) {
    pressed = (PORT_switch_bit == ACTIVE_LEVEL);
    LAT_output_bit = pressed ? 1 : 0;
}
```

Exam trap: TRIS is direction, LAT is the intended driven value, PORT is a sampled pin value. An ADC channel needs configuration in addition to TRIS.

4 Timers and PWM: generating time and power commands

In simple words
A timer counts clock increments or external events; PWM repeatedly switches an output between high and low. Adjusting PWM duty cycle changes the fraction of each period spent high. A motor driver can use this signal to regulate applied average drive, but the PIC pin never powers the motor itself.

4.1 Four timer modules, different purposes

Module	Plain-language role
Timer0	8- or 16-bit timer/counter, with selectable clock source and prescaler
Timer1	16-bit timer/counter, with clock/oscillator options; can support capture/compare timing
Timer2	8-bit timed period using PR2, programmable prescaler and postscaler; common PWM timebase
Timer3	16-bit timer/counter associated with compare/capture choices, depending on device control bits

4.2 A universal counter-delay calculation

With timer tick frequency f_{tick} and N increments until rollover, an ideal duration is $t = N/f_{\text{tick}}$. For a 16-bit up-counter with preload P, $N = 65536 - P$. For the internal oscillator-derived path with prescaler S, $f_{\text{tick}} = f_{\text{osc}}/(4S)$. Hardware mode, synchronization and actual prescaler routing must match this example.

STEP-BY-STEP EXAMPLE: Timer0: nominal 5 ms at 8 MHz, prescale 1:2

The instruction clock is $8\,\text{MHz}/4 = 2\,\text{MHz}$. With prescaler $S = 2$, timer increments at $1\,\text{MHz}$ or one tick per $1\,\mu\text{s}$. Five milliseconds needs 5000 ticks. The 16-bit preload is $65536 - 5000 = 60536 =$ EC78H, so initial high/low bytes are EC and 78. The exact periodic interval must account for software reload and interrupt latency.

Worked answer: $P =$ EC78H for the stated 16-bit, 8 MHz oscillator, 1:2 prescale assumption.

4.3 Why Timer2 is important for PWM

Timer2 increments using an instruction-clock-derived source through its prescaler. PR2 holds the period comparison value, and T2CON configures prescaler, postscaler and on/off state. The postscaler is involved in the Timer2 interrupt path, *not* the fundamental CCP PWM output frequency.

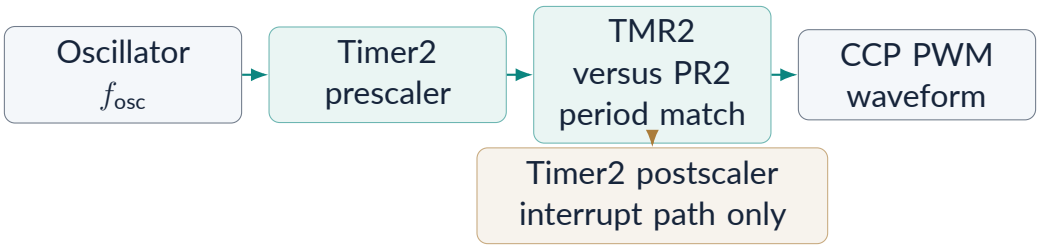


Figure 5. Timer2 prescaler controls PWM frequency through PR2; the postscaler affects interrupt scheduling, not the PWM period.

4.4 PIC18F458 CCP PWM frequency equation

The Microchip PIC18FXX8 datasheet specifies, in its CCP1 PWM section (DS41159E, §15.4.1),

$$T_{\mathrm{PWM}} = 4(\mathrm{PR2}+1)T_{\mathrm{OSC}}S, \qquad f_{\mathrm{PWM}} = \frac{f_{\mathrm{OSC}}}{4(\mathrm{PR2}+1)S},$$

where S is the Timer2 prescaler and $T_{\mathrm{OSC}} = 1/f_{\mathrm{OSC}}$. Use integer PR2 (0–255) and a supported prescale setting (1, 4 or 16). The CCP1 output is multiplexed with RC2, so TRISC2 must select its output role when using CCP1 PWM; enhanced CCP uses different pins and control bits.

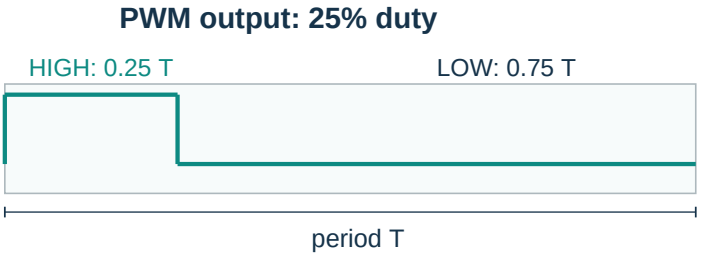


Figure 6. Original pulse-width modulation waveform. Duty changes pulse width while the PR2-derived period can stay fixed.

STEP-BY-STEP EXAMPLE: Find PWM frequency at 8 MHz, PR2=99 and prescaler=4

1. $\mathrm{PR2}+1 = 100$ timer-period counts.
2. $T_{\mathrm{OSC}} = 1/(8\times10^{6}) = 0.125\,\mu\mathrm{s}$.
3. $T_{\mathrm{PWM}} = 4\times100\times0.125\,\mu\mathrm{s}\times4 = 200\,\mu\mathrm{s}$.
4. $f_{\mathrm{PWM}} = 1/(200\,\mu\mathrm{s}) = 5000\,\mathrm{Hz} = 5\,\mathrm{kHz}$.

Worked answer: 5 kHz PWM. The Timer2 postscaler does not appear in this answer.

4.5 PWM duty-cycle register calculation

For standard CCP1 PWM, a ten-bit duty value is constructed from eight high bits CCPR1L and two low bits in $\mathrm{CCP1CON}\langle5:4\rangle$. The duty time obeys (DS41159E, §15.4.2)

$$T_{\mathrm{high}} = D\,T_{\mathrm{OSC}}S, \qquad D = 4\,\mathrm{CCPR1L} + \mathrm{DC1B}_{(1:0)}.$$

The nominal duty fraction is $D/[4(\text{PR2}+1)]$ while not exceeding the effective period. For an enhanced ECCP output, use the separate ECCP1 register fields described in §16, not CCP1CON by accident.

STEP-BY-STEP EXAMPLE: 25% duty at the same 5 kHz setup

The maximum period-count scale is $4(99+1) = 400$. A quarter-duty corresponds to $D = 100$. Decompose: CCPR1L $= \lfloor 100/4 \rfloor = 25 =$ 19H; the two low duty bits are 100 mod $4 = 0$. The high time is $100 \times (0.125\,\mu\text{s}) \times 4 = 50\,\mu$s. The complete period is 200 μs.

Worked answer: CCPR1L $=$ 19H; DC1B1:DC1B0=00; 25% duty; high time 50 μs.

4.6 Duty cycle is not motor speed

At fixed PWM frequency, raising duty generally increases the average drive applied by an appropriately designed motor driver. Actual shaft speed depends on supply, load, motor model, control strategy, back EMF and current limits. PWM on an unprotected pin is not a power stage, and the enhanced CCP feature is useful for more complex switching but still requires electrical safety provisions.

Three common mistakes: (1) using f_{osc} rather than $f_{\text{osc}}/4$ in timer tick calculations; (2) including the Timer2 *postscaler* in the PWM-frequency formula; and (3) treating the 10-bit duty register value as a direct percentage.

5 Analog-to-digital conversion: measuring real sensors

In simple words
An ADC converts an analog voltage into a number. On this PIC, several AN pins share a single 10-bit ADC through a multiplexer. Choose the input channel, allow acquisition time, run the conversion and read the resulting ten-bit value.

5.1 What 10-bit resolution means

Ten binary bits encode $2^{10} = 1024$ distinct output codes, from 0 to 1023. The ideal bin width is $\Delta V = (V_{REF+} - V_{REF-})/1024$. A frequently used *endpoint mapping* for an approximate conversion back to input voltage uses $V \approx V_{REF-} + \text{code} \cdot (V_{REF+} - V_{REF-})/1023$. These two denominators describe different conventions (quantization bin width versus endpoint interpolation); do not exchange them without stating the model.

STEP-BY-STEP EXAMPLE: Ideal resolution with a 5.000 V reference span

$\Delta V = 5.000/1024 = 0.0048828125$ V or approximately 4.883 mV per code bin. A sensor at 2.50 V would be around midscale, approximately code 512 under the simple ideal quantizer model. Real reference tolerance, source impedance, noise, ADC offset and nonlinearity affect measurements.

Worked answer: Ideal 10-bit bin width = 4.883 mV at a 5.000 V span. Midscale is near code 512; real accuracy is not guaranteed to be 4.883 mV.

5.2 Signal path from pin to result registers

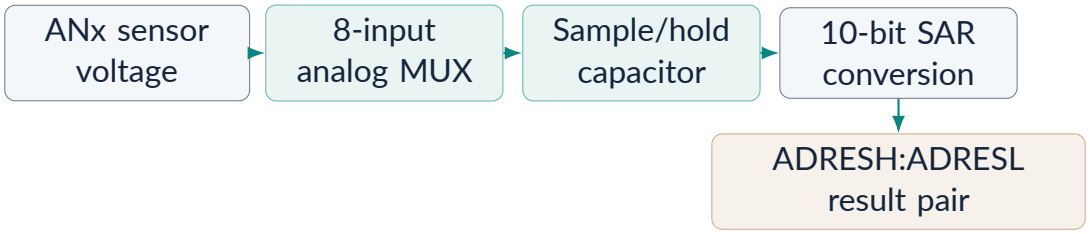


Figure 7. Original measurement pipeline; one multiplexer selects an AN input for the shared ADC converter and result registers.

5.3 ADCON0 and ADCON1: this PIC's actual arrangement

Unlike newer PIC variants that add ADCON2, the PIC18F458 ADC control fields are divided between ADCON0 and ADCON1. The DS41159E device register summary lists:

Register	Important bits on PIC18F458
ADCON0	ADCS1:ADCS0 (A/D clock), CHS2:CHS0 (channel), GO/DONE (conversion), ADON (enable)
ADCON1	ADFM (result alignment), ADCS2 (additional conversion-clock selection), PCFG3:PCFG0 (analog pin/reference configuration)
ADRESH/ADRESL	Ten-bit result spread across two 8-bit registers according to alignment

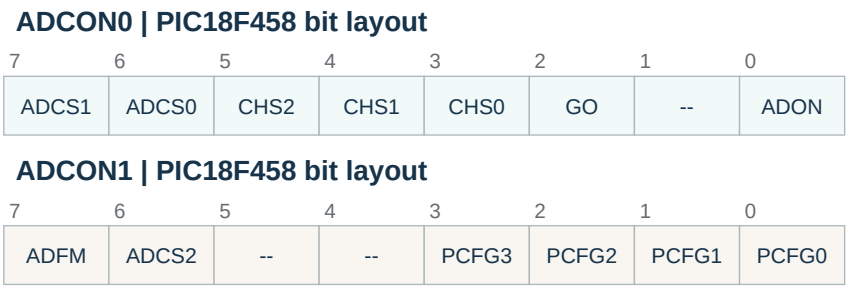


Figure 8. PIC18F458 ADCON0 and ADCON1 bit positions (from DS41159E register summary). GO abbreviates GO/DONE; dashes indicate unimplemented bits.

PIC-family trap: Do not copy a PIC18F4520/4580 tutorial that uses ADCON2 conversion settings into PIC18F458 code. This device's ADFM and ADCS2 are in ADCON1. Verify PCFG patterns in Microchip DS41159E before choosing reference pins and analog port behaviour.

5.4 Acquisition time versus conversion time

The selected analog pin charges the sample-and-hold capacitor through circuitry with finite impedance. After switching channels, a controller must allow sufficient *acquisition* time before setting GO/DONE. The ADC then performs the digital conversion using a specified A/D clock T_{AD}. These two phases are different. A code loop that changes channels and instantly starts conversion may produce unstable results even when the numeric formula is correct.

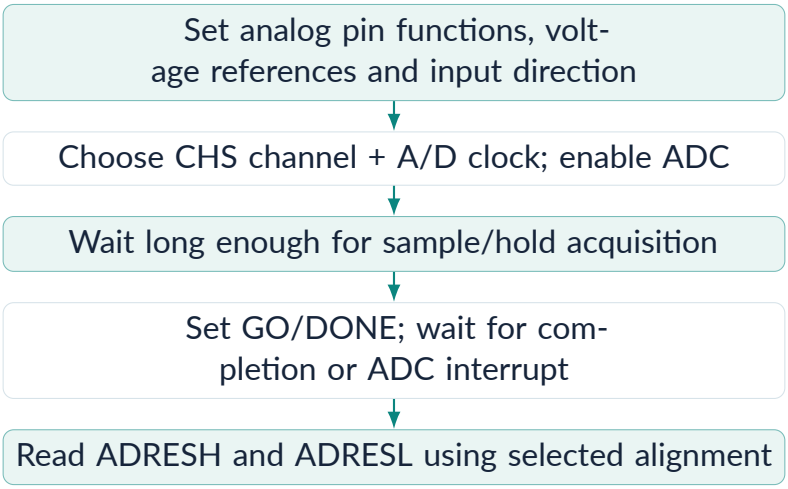


Figure 9. Five-step ADC conversion workflow; acquisition is necessary before conversion, especially after changing channels.

5.5 Worked sensor conversion and alignment

Assume an ideal 0–5 V input scale and 10-bit right-justified result. For an illustrative ADC reading `02A3H`, decimal $2(256) + 163 = 675$. Endpoint interpolation gives $V \approx 675(5/1023) = 3.2991$ V. If you instead use bin-width times code, $675(5/1024) = 3.2959$ V. The tiny difference reflects a stated convention, not a device fault. Under right justification, the code is built as $256(\text{ADRESH} \,\&\, 0x03) + \text{ADRESL}$.

Self-check: With the same reference span, find the code-bin width for a 3.3 V reference. **Answer:** $3.3/1024 = 0.00322266$ V or about 3.223 mV.

5.6 ADC pseudocode: deliberately not a compiled driver

```
// ALGORITHM: DEVICE-SPECIFIC REGISTER STEPS
set_TRIS_for_analog_input();
set_ADCON1_for_pin_map_reference_and_alignment();
set_ADCON0_for_channel_clock_and_ADON();
wait_for_acquisition_time_from_datasheet();
set_GO_DONE_bit();
while (GO_DONE_bit == 1) { /* conversion busy */ }
result = combine_ADRESH_ADRESL_as_configured();
```

This is a sequence for learning, not a complete XC8 source file. Board oscillator, analog source impedance, reference voltages, exact PCFG settings and the

target toolchain are not specified here. A datasheet-compliant hardware driver must provide them.

6 Interrupts: event handling

In simple words
An interrupt tells the CPU that a configured event needs attention. Instead of constantly polling every sensor, firmware can respond to timer overflow, ADC completion or a received serial character. Hardware flags, enable bits and interrupt priority rules determine which service routine runs.

6.1 PIC18F458 reset and interrupt vectors

The manufacturer's PIC18F258/458 program-memory map uses reset vector `0000H`, high-priority interrupt vector `0008H` and low-priority interrupt vector `0018H` when interrupt priorities are enabled. With interrupt priorities disabled, the high-priority vector serves as the common interrupt entry; software must still inspect each pending source.

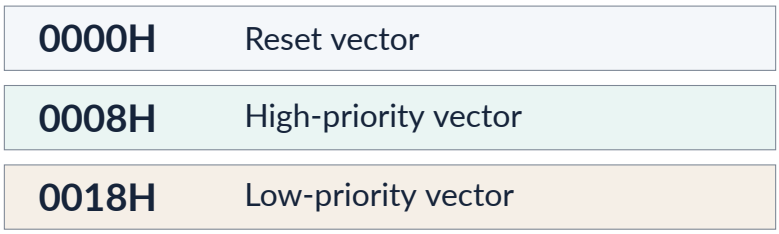


Figure 10. PIC18F458 program-memory vector addresses. These labels refer to code memory, not byte locations in data RAM.

6.2 Enable, flag and priority: three different jobs

Kind of bit	What it means
Flag (IF)	A hardware event occurred. A flag can set even if interrupts are disabled.
Enable (IE)	Software allows that particular source to request an interrupt.
Priority (IP)	Under priority mode, helps route an enabled request to high or low vector.
Global gate	Overall CPU permission; behaviour depends on RCON.IPEN.

Relevant controls span INTCON, INTCON2, INTCON3, PIRx, PIEx, IPRx and RCON. For example, PIC18F458 uses RCON.IPEN to select interrupt-priority operation; simply labelling every interrupt as high or low without setting IPEN is incomplete.

6.3 Safe interrupt response sequence

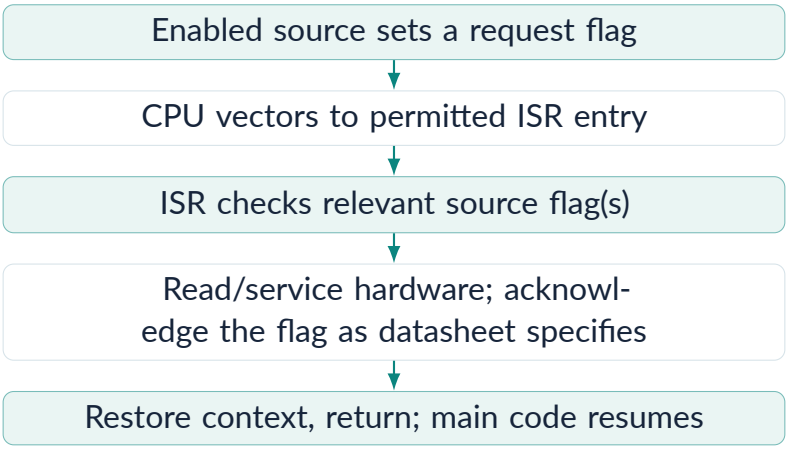


Figure 11. Source-aware ISR logic: peripheral flag handling must follow the source's datasheet rules, not a universal clear-bit recipe.

6.4 Why ISR code must be small

An ISR that waits on a UART buffer, performs extensive floating-point calculations or loops indefinitely can block more urgent work. Prefer acquiring the event, recording small data in a buffer, acknowledging the correct source, and returning. On priority-enabled PIC18, high-priority interrupts can preempt low-priority ISRs; careful state and context preservation remain essential.

6.5 Interrupt use case: sample a sensor periodically

Imagine Timer0 requests a periodic sample. The ISR records a `sample_due` flag and returns. The main loop notices that flag, starts an ADC conversion after appropriate acquisition and computes any slow control operation outside the ISR. This splits real-time timing from noncritical processing.

```
// CONCEPTUAL PSEUDOCODE - NOT COMPILED XC8
on_timer0_interrupt() {
    if (timer0_overflow_flag_is_set()) {
        acknowledge_timer0_source();
        sample_due = true;
    }
}
while (1) {
    if (sample_due) {
        sample_due = false;
        acquire_ADC_then_update_control();
    }
}
```

Flags are not interchangeable. Some PIC interrupt flags are cleared in software; others reflect hardware buffer status and clear by specific read/write operations. Follow the exact peripheral section. Never blindly write a zero to every IF bit.

7 UART: exchanging bytes over two wires

In simple words
UART turns each character into timed serial bits. PIC18F458 uses RC6/TX/CK for transmit and RC7/RX/DT for receive when configured for USART operation. Both ends must agree on speed, data length, parity/framing and electrical signalling.

7.1 Asynchronous 8N1 framing

A standard teaching frame has one start bit (low), eight data bits transmitted least-significant bit first, no parity and one stop bit (high). At 9600 baud,

each bit takes $1/9600 = 104.167\,\mu$s and the ten-bit frame takes approximately 1.04167 ms.

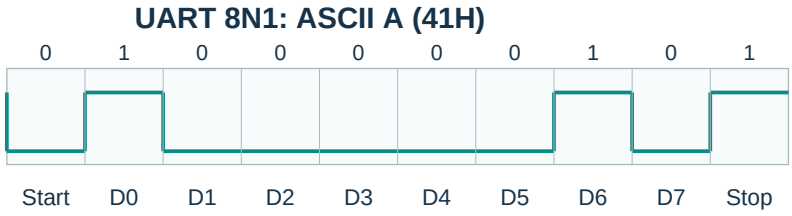


Figure 12. Original logic-level waveform for ASCII 41H; UART idle is high and the data bytes are transmitted LSB first.

7.2 This PIC's UART register roles

Register	Purpose and key fields
TXSTA	Transmitter: TXEN, SYNC, BRGH, TX9 and TRMT status
RCSTA	Receiver: SPEN, CREN, RX9, FERR and OERR
SPBRG	8-bit baud-rate divisor X on PIC18F458
TXREG	Write outgoing byte; transfers to hardware shift register
RCREG	Read received byte from receive buffer
PIR1/PIE1	TXIF/RCIF status and enable bits for UART-related interrupts

7.3 Baud rate: two asynchronous divisor modes

Microchip DS41159E Table 18-1 gives, in *asynchronous* mode (SYNC $= 0$),

$$\text{BRGH} = 0: \quad f_{\text{baud}} = \frac{f_{\text{OSC}}}{64(\text{SPBRG} + 1)},$$

$$\text{BRGH} = 1: \quad f_{\text{baud}} = \frac{f_{\text{OSC}}}{16(\text{SPBRG} + 1)}.$$

This unit uses the *PIC18F458 8-bit* SPBRG. A newer PIC18 with 16-bit BRG or different BRG bits uses a different setup; do not silently copy such a tutorial.

STEP-BY-STEP EXAMPLE: 9600 baud from a 16 MHz oscillator, BRGH=0

Set $X \approx 16{,}000{,}000/(64 \times 9600) - 1 = 25.0417$. The nearest 8-bit integer is 25. Then actual baud is $16{,}000{,}000/[64(26)] = 9615.3846$ baud. Percentage error is approximately $(9615.3846 - 9600)/9600 \times 100 = +0.1603\%$.

Worked answer: SPBRG $= 25$ at 16 MHz with BRGH=0, approximately 9615.38 baud ($+0.16\%$). This agrees with the manufacturer's worked example.

STEP-BY-STEP EXAMPLE: Choose BRGH=1 for the same oscillator

$X \approx 16{,}000{,}000/(16 \times 9600) - 1 = 103.1667$, so try $X = 103$. Actual baud is $16{,}000{,}000/[16(104)] = 9615.3846$ baud, the same small error in this particular case. The higher BRGH setting does not guarantee lower error for every oscillator/target pair; compare integer candidates.

7.4 Reception error: overrun and framing

FERR records a framing problem with a received character (for example, incorrect stop-bit condition). OERR indicates receive overflow: new serial reception can stop until firmware clears and re-enables continuous reception as described in the Microchip USART chapter. RCIF is associated with received-buffer occupancy; handle it by reading RCREG appropriately rather than assuming it behaves like a generic manually-clearable timer flag.

The transmitter's TXIF tells whether TXREG can accept new data, while TRMT reports the state of the transmit shift register. These are *not* identical meanings: TXREG can be free while the last bit is still in flight.

Electrical interface: RC6/RC7 are device logic-level signals, not RS-232 voltages. A computer USB connection requires a compatible USB–UART bridge; RS-232 uses an appropriate level translator. Common-ground and isolation decisions must be deliberate.

7.5 An illustrative data-transmission algorithm

```
// REGISTER-LEVEL ALGORITHM; NOT COMPILED CODE
choose_oscillator_and_verify_frequency();
set_SPBRG_from_oscillator_and_baud_formula();
select_async_mode_and_BRGH();
enable_serial_port_and_transmitter();
wait_until_TXREG_accepts_a_byte();
write_TXREG(0x41);  // letter A
wait_until_shift_register_is_empty_if_needed();
```

Quick question: Is an empty TXREG proof that the last stop bit has left RC6? **Answer:** No. Inspect TRMT when final physical transmission completion matters.

8 Integrated robotics case: sensor to PWM with UART logging

In simple words
Think of a gripper controller. A position sensor supplies an analog voltage. The ADC produces an approximate position code. Firmware compares that code with a target, updates the PWM command to the protected motor driver, and transmits diagnostic data over UART. A limit switch interrupts or disables motion if a boundary is reached.

8.1 Step-by-step control architecture

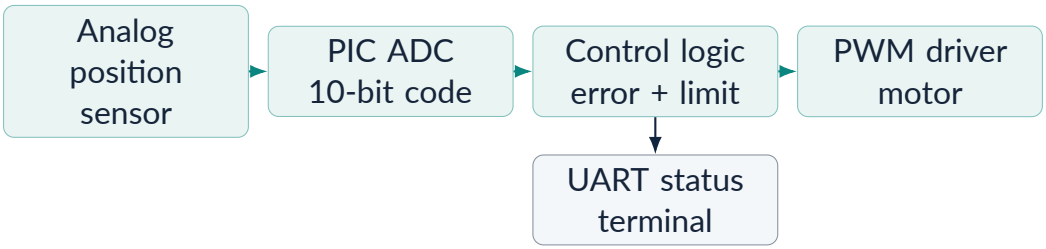


Figure 13. Source-based engineering example combining all Unit IV peripherals without suggesting the MCU pin powers the motor.

8.2 Worked scaling example

Suppose an ideal 10-bit ADC reads 600 with a 0–5 V span. Bin-width approximation gives $V \approx 600 \cdot 5/1024 = 2.9297$ V. A firmware controller might map the code 0–1023 onto a *limited* PWM command band rather than a direct 0–100% command; for example, a safeguarded 0–80% command requires sensor calibration, limits, and a documented control law. An equation such as $u = K_p(r - y)$ must be scaled and saturated, with actual control timing tested, before being used on hardware.

8.3 Engineering checklist

1. Confirm voltage levels, safe power-up state and multipurpose pin availability from the physical pinout.
2. Check TRIS/PORT/LAT and ADC PCFG settings against the target board.
3. Measure oscillator frequency; recalculate instruction cycles, timer reloads, PWM frequency and baud.
4. Acquire ADC voltage after adequate settling, with a known reference and source impedance.
5. Apply bounded PWM only to a driver input, including emergency-stop behaviour and fault handling.
6. Use short interrupt routines, store buffered state correctly and handle UART overrun.
7. Test sensor failure, disconnected wire, motor stall and watchdog/reset behaviour before live deployment.

Safety limitation: This is a conceptual educational controller. It is not a validated industrial safety system, certified firmware or tested hardware wiring diagram.

8.4 Sensor-to-output decision flow: avoiding an ambiguous pin setup

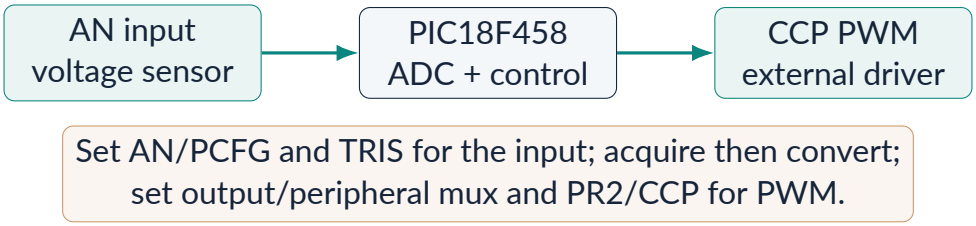


Figure 14. Original PIC18F458 integration sequence. The correct pin multiplexing and register bits must be confirmed against DS41159E before a physical build.

Important distinction: Setting TRIS for input does not automatically make a pin a correctly configured analog channel; the ADC and analog-function configuration also matter. Similarly, a timer output command does not

inherently enable a protected motor driver. PWM duty control and protective motor-current shutdown are separate design tasks.

9 Revision tables, solved numericals and answer checkpoints

9.1 Formula review

Quantity	Equation and assumption
Instruction-cycle time	$T_{CY} = 4/f_{OSC}$ for classical PIC18 clock path
16-bit timer preload	$P = 65536 - tf_{OSC}/(4S)$ for stated internal-clock and prescaler mode
Timer2 PWM period	$T_{PWM} = 4(\text{PR2} + 1)S/f_{OSC}$
Standard CCP duty fraction	$D/[4(\text{PR2} + 1)]$, with $D = 4\,\text{CCPR1L} + \text{DC1B}$
ADC quantization step	$\Delta V = (V_{REF+} - V_{REF-})/1024$ ideal ten-bit code width
UART low-speed baud	$f_{OSC}/[64(\text{SPBRG} + 1)]$, async BRGH=0
UART high-speed baud	$f_{OSC}/[16(\text{SPBRG} + 1)]$, async BRGH=1

9.2 Calculation A: select PR2 for 10 kHz PWM

Given $f_{OSC} = 16$ MHz and Timer2 prescaler $S = 4$, rearrange the PWM formula:

$$\text{PR2} = \frac{f_{OSC}}{4Sf_{PWM}} - 1 = \frac{16{,}000{,}000}{4 \cdot 4 \cdot 10{,}000} - 1 = 99.$$

Worked answer: PR2 $= 99 =$ 63H, with Timer2 1:4 prescale. This gives 10 kHz nominal PWM.

9.3 Calculation B: find ADC code-bin width

Given $V_{REF-} = 1.0$ V and $V_{REF+} = 4.3$ V, the span is 3.3 V. The ideal resolution is $3.3/1024 = 3.22265625$ mV/bin. Absolute input voltage must include the 1.0 V lower reference when converting a code back to volts.

Worked answer: $\Delta V = 3.223$ mV/bin, with offset reference 1.0 V.

9.4 Calculation C: UART frame duration

At 19200 baud with 8N1, there are 10 bits per character. One frame takes $10/19200 = 0.520833$ ms. An upper-bound continuous channel rate is $19200/10 = 1920$ 8-bit characters/s before software protocol overhead.

Worked answer: 0.5208 ms per 8N1 character; theoretical maximum 1920 characters/s.

9.5 Exam-style conceptual questions with answer hints

1. **PIC18F458 features:** Draw block diagram and give the program Flash, RAM, EEPROM, ADC input and timer counts. Use the verified device-specific values from Section 1.
2. **Harvard memory:** Describe why code Flash and RAM are logically separate. Contrast 16-bit instruction width with 8-bit data operations.
3. **TRIS/PORT/LAT:** Show a safe digital-input/output initialization sequence and explain why an output read-modify-write operation can be unsafe.
4. **Timer2 PWM:** Derive period from T_{OSC}, prescaler and PR2. Explain why the Timer2 postscaler is not in the formula.
5. **Duty register:** For PR2=99, compute the duty-field value for 50%. **Answer:** $D = 200$, so CCPR1L=50 (32H), DC1B=0.
6. **10-bit ADC:** Explain 1024 codes versus maximum numeric code 1023. Identify acquisition-time and reference-voltage effects.
7. **ADC control:** Name this PIC's ADCON0 and ADCON1 fields. Why is a PIC18F4580 ADCON2 tutorial unsafe to copy?
8. **Interrupts:** Distinguish flag, enable, priority and vector. Give 0000H,

0008H and 0018H and the role of RCON.IPEN.

9. **UART:** Derive SPBRG for 9600 baud at 16 MHz, BRGH=0. Explain the error and distinguish TXIF from TRMT.
10. **Robot controller:** Draw ADC sensor → PIC logic → PWM protected motor driver, including UART logging and limit-switch safety.

9.6 Common mistakes worth correcting before an exam

1. Writing *PIC18F4580* control-register values in a *PIC18F458* answer.
2. Calling 40 MHz a 40-MIPS instruction rate; the classical upper bound is 10 MIPS.
3. Using `PORT` latch writes interchangeably with `LAT` output-register writes.
4. Forgetting the analog pin selection when configuring TRIS.
5. Using the Timer2 postscaler to calculate PWM frequency.
6. Converting ADC code to voltage without stating V_{REF+}, V_{REF-} and quantization model.
7. Treating UART `TXIF` as proof that transmission's stop bit has finished.
8. Assuming every interrupt flag can be cleared by writing zero; check each source's behavior.

Source, figure and validation ledger

Syllabus authority: Savitribai Phule Pune University, TE Robotics & Automation 2024 Pattern, PCC302-ROA, Unit IV, supplied syllabus PDF.
Primary technical authority: Microchip Technology,

PIC18F248/258/448/458 Data Sheet, DS41159E. Manufacturer reference Microchip official PDF (DS41159E). Key device table §1; CPU and memory §4; I/O §9; Timer2 §13; CCP PWM §15; ECCP §16; USART §18; A/D §20. This is the *PIC18FXX8* manual, not a PIC18FXX80 manual.
All embedded figures: Original explanatory TikZ vector diagrams, redrawn from verified manufacturer-level behavior with labels and explanations. No

copyrighted manufacturer figure is directly reproduced; no external raster-photo licence needs to be assigned to the PDF.
Numerical examples: Author-developed and independently recomputed using Python arithmetic in the release QA. Quantization conventions, oscillator assumptions and clock prescalers are explicitly stated.
Limitations: Register-level listings are *illustrative pseudocode*, not native compiled XC8 or assembly. No physical PIC18F458 board, simulator or MPLAB target test has been executed; an independent technical reviewer should approve firmware-ready examples before live deployment.

Rapid revision: PIC18F458 register and pin map

Register quick-reference

Register group	Reason you use it
WREG, STATUS, BSR	CPU calculations and banked register access
TRISx, PORTx, LATx	Set direction, sample inputs, drive output latches
T0CON, T1CON, T2CON, T3CON	Choose timer modes and start selected counters
PR2, CCPR1L, CCP1CON	Specify standard CCP PWM period and duty value
ADCON0, ADCON1, ADRESH/L	Select analog channel, conversion and result
RCON, INTCON, PIRx/PIEx/IPRx	Interrupt priority, requests, enables and flags
TXSTA, RCSTA, SPBRG, TXREG, RCREG	USART configuration, baud, transmit and receive

Selected alternate pin functions

Signal alias	Function you should recognize
RA0/AN0; RA1/AN1	Analog channels zero and one
RE0/AN5	Another ADC input channel
RC2/CCP1	Standard CCP1 capture/compare/PWM output
RD4/ECCP1/P1A	Enhanced CCP output function
RC6/TX/CK	USART transmit or synchronous clock
RC7/RX/DT	USART receive or synchronous data
RB0/INT0; RB1/INT1	External interrupt inputs
RB2/CANTX/INT2	CAN TX or external interrupt input
RB3/CANRX	CAN receive function

These are signal aliases, not package pin numbers. The board design must use the precise package pin table in Microchip DS41159E, Table 1-2.

Final exam check: Can you independently derive one PWM example, one timer reload, one ADC voltage, and one UART baud configuration? For each answer, state the oscillator and hardware assumptions; that matters more than memorizing a single hexadecimal value.