24EC504 – Unit II – Part 1: Timers, Interrupts, ADC and DAC

Core Peripheral Glossary

Peripheral	Full form	Purpose
GPIO	General Purpose Input Output	Read switches, control LEDs
Timer	Timer / Counter	Delays, measure time, PWM
UART	Universal Asynchronous Receiver Transmitter	Serial communication
SPI	Serial Peripheral Interface	High-speed link to sensors and displays
I²C	Inter-Integrated Circuit	Two-wire link to sensors
ADC	Analog-to-Digital Converter	Analog to digital
DAC	Digital-to-Analog Converter	Digital to analog voltage
PWM	Pulse Width Modulation	Motor speed, LED brightness
Watchdog timer	–	Resets the system if it hangs

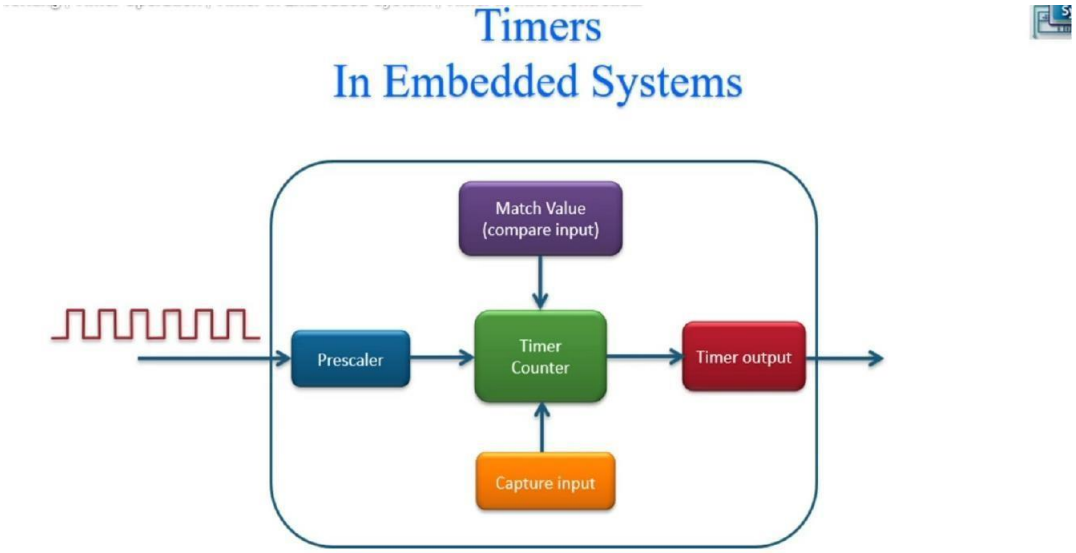
1. Timers, Counters and PWM Generation

1.1 What is a Timer?

A timer is a hardware peripheral that counts clock cycles, measures time, generates precise delays and triggers events at regular intervals. It runs independently of the main program, so the CPU can do other work while it counts. Depending on configuration, the same block can be a delay generator, an event counter or a PWM generator.

Block flow: Clock source → Prescaler → Timer counter → Match (compare) value → Output / Capture.

Why timers are useful: counting external events, accurate delays, task scheduling, PWM, capturing the time of an event, generating interrupts, measuring external events and synchronizing system components.



Building blocks

- **Clock source:** feeds the prescaler.
- **Prescaler:** divides the clock down to a usable counting rate.
- **Timer counter (TCNT/CNT):** increments on every tick.
- **Counting modes:** up, down, and up/down (modulo).
- **Match/compare value:** the Auto-Reload Register (ARR).
- **Overflow:** event when the counter reaches its maximum or match value.
- **Interrupts:** generated on overflow or compare match.

Timer modes: simple timer, interrupt-based, PWM, watchdog; timer events Output Compare (OC) and Input Capture (IC); auto-reload.

Applications: automation, energy metering, building control, safety systems, water heating, cooking appliances, traffic lights, fitness equipment, alarm systems.

1.2 Pulse Width Modulation (PWM)

PWM controls the average power delivered to a load by varying the ON time (width) of a digital pulse while keeping the frequency constant. It is generated by hardware timers. The output alternates HIGH and LOW at a fixed frequency, and changing the duty cycle gives different analog-equivalent voltage or power.

$$\text{Duty Cycle (\%)} = \frac{T_{on}}{T} \times 100$$

$$\text{PWM Frequency} = \frac{\text{Input Clock}}{(\text{Prescaler} + 1) \times \text{Counter Period}}$$

Applications: LED brightness control, DC motor speed, servo position, power electronics, battery charging, audio signal generation.

1.3 STM32 Timers and Counters

STM32 timers are highly configurable and enable time-based operations, precise delays, PWM generation and event counting.

Counting principle (example): with Fsys = 72 MHz and prescaler 2048, TCNT increments every 28.44 µs. A 16-bit counter overflowing at 65536 gives an interrupt every $28.44 \times 10^{-6} \times 65536 \approx 1.864$ s. For exactly 1 s, change the maximum count: ARR = 35155.

General STM32 timer formulas:

$$F_{timer} = \frac{F_{clock}}{PSC + 1} \qquad T_{tick} = \frac{PSC + 1}{F_{clock}}$$

$$T_{overflow} = \frac{(PSC + 1)(ARR + 1)}{F_{clock}}$$

The overflow-time formula is the most important STM32 timer formula and is used in nearly every timer calculation. To hit a target period, solve it for ARR.

How the registers work: ARR is loaded with a value (max 65535 for 16-bit). Each incoming pulse increments CNT. When CNT overflows ARR, an interrupt is generated, CNT resets and the cycle repeats. The prescaler divides the incoming pulse frequency.

Timer groups:

Timers	Type	Channels
TIM1, TIM8	Advanced-control	6 independent
TIM2–TIM5	General-purpose	4 independent
TIM12 / TIM13, TIM14	General-purpose	2 / 1
TIM15 / TIM16, TIM17	General-purpose	2 / 1
TIM6, TIM7	Basic	None; used as time base or DAC trigger

Modes: timer, counter, input capture, PWM, advanced. Not every timer supports every mode, so check the datasheet.

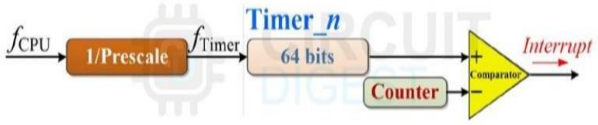
CubeMX/HAL workflow:

1. Select the timer instance (for example TIM2) and set the clock source to Internal Clock.
2. Set the prescaler and counter period (auto-reload).
3. Enable the timer interrupt in the NVIC tab.
4. Generate the initialization code.
5. Start the timer with `HAL_TIM_Base_Start_IT(&htim2);`
6. Handle the event in the callback `HAL_TIM_PeriodElapsedCallback`, for example `HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);`

Counter mode counts external events, such as pulses from a push-button. Example: TIM2 in counter mode with preload 20 (interrupt after 20 pulses), input on PA0 with an internal pull-down.

1.4 ESP32 Timer and Timer Interrupts

Timer interrupts make timed events happen to the millisecond. They behave like external interrupts but are triggered by a timer: the current instruction completes, the ISR runs, and execution resumes where it left off. The ESP32 timer is a prescaler feeding a 64-bit counter that raises an interrupt.



Execution flow: `timerBegin()` configures the timer → `timerAttachInterrupt()` attaches the ISR → the timer counts in the background → on expiry it raises an interrupt → the program runs the ISR and resumes.

```
\text{Timer speed (Hz)} = \frac{\text{Timer clock (MHz)}}{\text{Prescaler}}
```

At 80 MHz, prescaler 1 gives 80 MHz; prescaler 80 gives 1 MHz.

Example: LED blink with a timer interrupt

```
#define LED 21
hw_timer_t *My_timer = NULL;
void IRAM_ATTR onTimer(){ digitalWrite(LED, !digitalRead(LED)); }
void setup() {
  pinMode(LED, OUTPUT);
  My_timer = timerBegin(0, 80, true);          // 1 MHz tick
  timerAttachInterrupt(My_timer, &onTimer, true);
  timerAlarmWrite(My_timer, 1000000, true);    // 1 s, auto-reload
  timerAlarmEnable(My_timer);
}
void loop() { }
```

STM32 vs ESP32 Timers

Feature	STM32	ESP32
Primary focus	Industrial control, automation, precise PWM	RTOS task scheduling, wireless timing
Bit width	16-bit or 32-bit	64-bit
Motor control	Advanced hardware, dedicated dead-time insertion	Software-driven, via MCPWM peripheral

Feature	STM32	ESP32
Encoder support	Native hardware decoder in most timers	Separate PCNT (pulse counter) block

2. Interrupt Handling and Nested Interrupts

2.1 What is an Interrupt?

An interrupt is a signal to the CPU requesting immediate attention. The CPU suspends the current program, runs a special routine called the Interrupt Service Routine (ISR), then resumes.

Interrupt = Stop current task → Execute ISR → Resume previous task

Polling (no interrupts)	With interrupts
CPU continuously checks all peripherals	CPU works on the main program
Wastes processing time	Peripheral interrupts only when necessary
Slow response	Faster response
High power consumption	Efficient CPU use

Interrupt process: main program runs → interrupt request occurs → CPU completes the current instruction → saves the Program Counter → jumps to the ISR → ISR executes → RETI executes → CPU resumes the main program.

2.2 Types of Interrupts

- **Hardware interrupts:** from hardware devices, either external (outside the chip) or internal (on-chip peripherals).
- **Software interrupts:** generated by software instructions, used for exceptions and operating-system services.

Hardware interrupt type	Description	Examples
Maskable	Can be enabled/disabled by software; lower priority; normal peripherals	Timer, UART, ADC
Non-maskable (NMI)	Cannot be disabled; highest priority; emergencies	Power failure, watchdog timeout, memory error

2.3 CPU Response to an Interrupt

The CPU finishes the current instruction, saves the PC and processor registers, stops normal execution, jumps to the ISR, executes it, and returns to the main program.

2.4 Interrupt Service Routine (ISR)

An ISR is a function that runs automatically when an interrupt occurs. It should have a short execution time, executes automatically, ends with RETI (Return from Interrupt), and serves only one interrupt source.

2.5 Interrupt Priority and Nested Interrupts

Priority decides which interrupt is served first when several occur together. A **nested interrupt** happens when a higher-priority interrupt interrupts a lower-priority ISR.

Interrupt	Priority
Power failure	Highest
Emergency stop	High
UART	Medium
Timer	Low

Cortex-M3 Exceptions and the NVIC

An exception interrupts normal execution: peripheral interrupts, reset, memory access fault, undefined instruction fault and hardware fault.

- Exceptions 1–15 are system exceptions; 16 and above are external interrupts (IRQs). There are 240 interrupts in total, numbered 16–255 in the vector table.
- The **NVIC** controls all exceptions, sets the priority of each and handles it.
- **Pre-emption priority:** decides whether an interrupt can occur while the processor is already running another handler.
- **Sub-priority:** used only when two exceptions with the same pre-empt priority occur together; the one with higher sub-priority (lower value) is handled first.

STM32 EXTI (external interrupt): up to 20 edge detectors (connectivity-line) or 19 (others). Each line can be configured for trigger type (pulse or pending) and edge (rising, falling or both), masked independently, and has a pending register for status.

ESP32 Interrupt Examples

- **Button debouncing:** an ISR starts a timer or ignores further interrupts briefly to remove noisy multiple triggers.
- **PWM:** interrupts can change PWM parameters dynamically.
- **Low-power management:** the ESP32 sleeps and interrupts wake it, for example when a motion sensor fires.
- **Sensor data acquisition:** interrupts time sensor reads precisely.
- **Real-time communication:** interrupts ensure timely handling of streaming or network events.
- **Error handling:** interrupts detect hardware malfunctions or unexpected data.

External interrupts come from sources on the pins, such as buttons and sensors. Benefits: immediate response, minimal latency, lower power (sleep until needed) and simpler code.

3. ADC and DAC Interfacing

3.1 Why ADCs Matter

- The real world is analog: voice, sensor readings and bio-signals such as ECG and EEG vary continuously.
- Digital systems need discrete binary data to store, compute and transmit.
- The ADC is the bridge between them.

An **ADC** is a digital circuit that converts an analog signal into digital data: Analog signal → ADC → Digital output.

Five stages of an ADC

Stage	What happens
1. Input	The continuous analog signal is presented to the ADC.
2. Sampling	The signal is measured at regular instants, set by the sampling rate.
3. Quantization	Each sample is assigned a discrete level; the gap to the true value is quantization error.
4. Encoding	Levels become binary numbers; the bit count sets resolution, accuracy and precision.
5. Output	The final digital signal is ready to store, process or transmit.

Performance parameters

$$SNR\,(dB) = 10 \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$$

Higher SNR means the ADC captures smaller signals clearly in noise. **Bandwidth** is the range of frequencies the ADC can sample accurately, set by the sampling rate.

Nyquist–Shannon rule: sampling rate ≥ 2 × maximum signal frequency. A 150 kHz ADC is limited to signals below 75 kHz to avoid aliasing.

Types of ADC

Type	Speed	Resolution	Cost	Typical use
Flash	Very high	Low–medium	High	Video, optical storage
Semi-flash	High	Medium	Medium	Balanced speed/cost
SAR	Medium	Medium–high	Low	Consumer and medical instruments

Type	Speed	Resolution	Cost	Typical use
Sigma-delta	Low–medium	Very high	Low–medium	Audio, precision sensing
Pipelined	High	Medium–high	Medium	Communications, medical equipment

- **Flash (direct):** fastest, GHz sampling; a bank of parallel comparators (n-bit needs $2^n - 1$); larger, costlier, higher power.
- **Semi-flash:** two parallel flash converters (one for MSBs, one for LSBs); compact and cost-effective, slightly slower.
- **SAR:** samples and holds, then binary-searches against an internal DAC, setting each bit from MSB to LSB.
- **Sigma-delta:** oversamples above Nyquist, pushes quantization noise to high frequencies, filters it with a low-pass filter, then decimates.
- **Pipelined:** coarse-then-refined conversion over several stages with digital correction.

ADC applications: audio and signal processing, data acquisition, communication and radio, automotive sensors, consumer electronics, and biomedical signals (ECG, EEG, heart rate, blood pressure, SpO_2).

3.2 DAC (Digital-to-Analog Converter)

A DAC converts a digital input into a continuous analog voltage or current. The number of input bits is a power of two (2, 4, 8, 16, …); more bits give finer, smoother steps. Digital input → DAC → Analog output.

Resolution: 2^N discrete levels for N bits. An 8-bit DAC gives 256 steps; a 16-bit DAC gives 65,536.

Weighted-resistor DAC	R-2R ladder DAC
Binary-weighted resistors (R, 2R, 4R, …) feeding an inverting summing amplifier	Ladder of only two resistor values, R and 2R
Simple concept, hard to scale	Easy to scale; industry standard

Disadvantages of weighted-resistor DAC: the resistor spread grows exponentially with bits (8-bit needs up to 128:1), and precisely making such a range becomes difficult and costly.

Why R-2R is preferred: only R and 2R are needed however many bits; adding a bit means adding one more identical section.

DAC selection parameters: resolution, accuracy, operating speed, power consumption and noise performance.

DAC uses: audio amplifiers, video encoders, data acquisition, calibration and measuring instruments, motor control, DSPs and telecommunication.