
⏱ TIMER INTERRUPT — Lý Thuyết Chi Tiết

🔹 1. Khái niệm cơ bản

Timer interrupt là **ngắt do bộ định thời (Timer)** của vi điều khiển tạo ra định kỳ theo chu kỳ thời gian xác định trước.

- Bộ **Timer** hoạt động **độc lập với CPU** (asynchronous).
- Khi bộ đếm của Timer đạt đến giá trị định trước (Auto Reload Value), nó **phát sinh ngắt (Interrupt Request)**.
- CPU tạm dừng chương trình chính và **chuyển sang hàm phục vụ ngắt (ISR – Interrupt Service Routine)** để thực hiện tác vụ định kỳ.

👉 Ứng dụng phổ biến:

- Tạo **xung thời gian chính xác** (ví dụ mỗi 1ms, 10ms, 1s).
- Dùng làm **bộ định thời hệ thống (System Tick)**.
- Làm **nguồn kích hoạt ADC, PWM, hay sự kiện định kỳ**.
- Làm cơ sở cho **RTOS tick interrupt**.

🔹 2. Cấu trúc và nguyên lý hoạt động của Timer

🧩 Thành phần chính của một Timer

Thành phần	Chức năng
Clock Source (fCLK)	Nguồn xung clock cấp cho Timer
Prescaler (PSC)	Bộ chia tần số – giảm tần số clock xuống mức phù hợp
Counter (CNT)	Bộ đếm, tăng hoặc giảm theo từng xung clock
Auto Reload Register (ARR)	Giá trị cực đại, khi CNT đếm tới ARR → sinh ngắt và quay lại 0
Interrupt Enable bit	Cho phép hoặc cấm ngắt Timer

Thành phần	Chức năng
NVIC (Nested Vectored Interrupt Controller)	Quản lý và ưu tiên ngắt trong CPU Cortex-M

⚙️ **Quá trình hoạt động:**

1. Bộ đếm **CNT** tăng dần theo mỗi xung clock (đã được chia bởi Prescaler).
2. Khi CNT đạt tới giá trị **ARR**, nó:
    - Reset về 0 (nếu chế độ Auto Reload).
    - Sinh tín hiệu **Update Event**.
    - Gây ra **Timer Interrupt** (nếu ngắt được cho phép).
3. CPU nhảy tới **hàm ISR (Interrupt Service Routine)** tương ứng trong **bảng vector ngắt**.
4. Sau khi xử lý xong, CPU trở lại chương trình chính.

🔹 **3. Các loại Timer Interrupt trong vi điều khiển**

Loại Timer	Mô tả
Basic Timer	Đơn giản, chỉ có đếm thời gian và sinh ngắt (ví dụ: TIM6, TIM7 trong STM32).
General-purpose Timer	Có thể dùng để đếm, tạo PWM, Input Capture, Output Compare (TIM2–TIM5).
Advanced Timer	Hỗ trợ điều khiển motor, dead-time, trigger, DMA (ví dụ TIM1, TIM8).

🔹 **4. Chuỗi hoạt động của ngắt Timer**

1. Timer đếm từ 0 → ARR.
2. Khi đạt ARR → cờ ngắt (UIF – Update Interrupt Flag) được set.
3. Nếu bit cho phép ngắt (UIE) = 1 → gửi yêu cầu ngắt đến NVIC.
4. NVIC tạm dừng chương trình chính, gọi ISR.
5. Trong ISR:

- Xóa cờ UIF để tránh lặp.
- Thực hiện hành động mong muốn (VD: đếm thời gian, bật LED...).

6. Kết thúc ISR, CPU trở về chương trình chính.

5. Công thức tính toán Timer Interrupt

Công thức tổng quát:

$$f_{timer} = \frac{f_{clk}}{PSC + 1}$$
$$f_{interrupt} = \frac{f_{timer}}{ARR + 1}$$
$$T_{interrupt} = \frac{1}{f_{interrupt}}$$

Từ công thức chu kỳ, ta có tần số ngắt:

$$f_{\text{ngắt}} = \frac{f_{\text{CLK}}}{(PSC + 1) \times (ARR + 1)}$$
$$T_{\text{ngắt}} = \frac{(PSC + 1) \times (ARR + 1)}{f_{\text{CLK}}}$$

🎛️ **Ví dụ cụ thể (theo bài giảng):**

- **Clock nguồn:** $f_{clk} = 8$ **MHz**
- **Prescaler:** $PSC = 7999$

$$f_{timer} = \frac{8,000,000}{7999 + 1} = 1000 \text{ Hz}$$

- **ARR = 9**

$$f_{interrupt} = \frac{1000}{9 + 1} = 100 \text{ Hz}$$
$$T_{interrupt} = \frac{1}{100} = 10 \text{ ms}$$

✅ **Kết luận: Timer sẽ phát ngắt mỗi 10ms.**

🔹 7. Vai trò của NVIC trong Timer Interrupt

NVIC (Nested Vectored Interrupt Controller) chịu trách nhiệm:

- Nhận tín hiệu yêu cầu ngắt từ Timer.
- Kiểm tra mức ưu tiên (priority).
- Cho phép hoặc từ chối ngắt.
- Chuyển CPU đến địa chỉ ISR tương ứng.

Timer là **maskable interrupt** → có thể bật/tắt bằng phần mềm.

🔹 8. Lợi ích của việc dùng Timer Interrupt

Lợi ích	**Mô tả**
Tạo thời gian chính xác	Giúp MCU định thời gian sự kiện chính xác mà không cần delay()
Tiết kiệm CPU time	CPU có thể xử lý việc khác trong khi Timer tự chạy
Cơ sở cho RTOS tick	RTOS sử dụng Timer interrupt để tạo "tick" cho hệ thống
Lập lịch đa nhiệm	Cho phép xử lý nhiều tác vụ định kỳ
Đồng bộ hóa thiết bị	Dễ dàng tạo tín hiệu định kỳ (trigger cho ADC, PWM, DMA...)

🔹 9. Một số lỗi thường gặp & cách khắc phục

Lỗi	**Nguyên nhân**	**Cách khắc phục**
Timer không sinh ngắt	Chưa bật ngắt trong NVIC hoặc chưa gọi Start_IT()	Kiểm tra HAL_TIM_Base_Start_IT()
ISR bị gọi liên tục	Không xóa cờ ngắt (UIF)	Dùng __HAL_TIM_CLEAR_IT() hoặc HAL_TIM_IRQHandler()
Sai chu kỳ ngắt	PSC hoặc ARR tính sai	Kiểm tra công thức tính
Nhiễu / treo hệ thống	ISR chạy quá lâu	Giảm code trong ISR, chỉ đặt cờ, xử lý bên ngoài

⚙️ BẢNG SO SÁNH LỢI ÍCH VÀ TÁC HẠI CỦA TIMER INTERRUPT

Khía cạnh	Lợi ích (Ưu điểm)	Tác hại / Hạn chế (Nhược điểm)
1️⃣ Hiệu suất CPU	- CPU không phải chờ bằng delay() → tận dụng thời gian xử lý tác vụ khác.	- Nếu có quá nhiều ngắt hoặc tần suất ngắt cao → CPU bị quá tải, giảm hiệu suất.
2️⃣ Độ chính xác thời gian	- Tạo thời gian định kỳ chính xác dựa trên clock phần cứng.	- Sai số có thể xuất hiện nếu clock không ổn định hoặc ISR chạy quá lâu.
3️⃣ Đa nhiệm (Multitasking)	- Cho phép thực hiện nhiều công việc định kỳ song song (đếm thời gian, đo cảm biến, cập nhật LED...).	- ISR cần viết ngắn gọn, nếu quá dài có thể làm trễ hoặc mất ngắt khác.
4️⃣ Quản lý sự kiện	- Dễ xử lý các sự kiện định kỳ, như đọc cảm biến 10ms/lần, refresh LCD, PWM update...	- Cần quản lý cẩn thận cờ ngắt (flag), tránh bị "miss" hoặc trùng ngắt.
5️⃣ Tích hợp RTOS	- Là nền tảng tạo "system tick" cho hệ điều hành thời gian thực (RTOS).	- Nếu tick quá nhỏ (1ms) mà ISR nặng → hệ thống có thể bị trễ (latency).
6️⃣ Lập trình linh hoạt	- Có thể điều chỉnh thời gian ngắt dễ dàng bằng thay đổi PSC và ARR.	- Tính toán sai PSC/ARR dễ dẫn đến chu kỳ ngắt sai.
7️⃣ Ứng dụng thực tế	- Tạo bộ đếm thời gian, đồng hồ, PID control, ADC trigger, PWM sync...	- Khó debug hơn vì ngắt xảy ra bất kỳ lúc nào, phụ thuộc vào thời gian thực.
8️⃣ Tác động đến hệ thống	- Tăng tính phản ứng nhanh của hệ thống với sự kiện định kỳ.	- Nếu ưu tiên ngắt không hợp lý → gây "priority inversion" (ngắt thấp ưu tiên chặn ngắt cao).
9️⃣ Tiêu thụ năng lượng	- Có thể đưa MCU vào chế độ sleep giữa các ngắt → tiết kiệm năng lượng.	- Nếu ngắt xảy ra quá thường xuyên → CPU thức dậy liên tục → tốn năng lượng.

Khía cạnh	Lợi ích (Ưu điểm)	Tác hại / Hạn chế (Nhược điểm)
🔟 Độ tin cậy hệ thống	- Giúp chương trình chạy ổn định, tuần hoàn, dễ kiểm soát thời gian.	- ISR lỗi hoặc không xóa cờ UIF → hệ thống bị kẹt trong vòng ngắt vô hạn.

🧭 **Kết luận tổng quát:**

Tổng kết	Nhận xét
Lợi ích chính	Timer interrupt giúp MCU xử lý định kỳ chính xác, tăng hiệu suất CPU, tạo thời gian thực, và hỗ trợ lập lịch đa nhiệm.
Tác hại chính	Cần lập trình cẩn thận vì ISR hoạt động song song với main loop, dễ gây lỗi trễ, xung đột hoặc treo nếu không quản lý tốt.

🧩 PHÂN LOẠI TIMER INTERRUPT

🔹 1. Phân loại theo chức năng phần cứng của Timer

Loại Timer	Đặc điểm chính	Ứng dụng điển hình
1️⃣ **Basic Timer (TIM6, TIM7)**	- Chỉ có chức năng đếm thời gian và sinh ngắt (không hỗ trợ PWM, Input Capture). - Dễ cấu hình nhất. - Không có đầu ra ra chân (output pin).	- Tạo **ngắt định kỳ** (ví dụ mỗi 1ms, 10ms). - Dùng làm **System Tick**, **Trigger ADC** hoặc **Watchdog Timer**.
2️⃣ **General-Purpose Timer (TIM2–TIM5, TIM9–TIM14)**	- Có thể hoạt động ở nhiều chế độ: Counter, PWM, Input Capture, Output Compare. - Hỗ trợ ngắt khi đếm tràn, hoặc khi so sánh bằng giá trị định trước. - Có đầu ra để điều khiển ngoại vi.	- Tạo **PWM**, đo **tần số, chu kỳ, độ rộng xung**. - Tạo **ngắt định kỳ đa năng**. - Đồng bộ hóa với ADC, DMA.
3️⃣ **Advanced Timer (TIM1, TIM8)**	- Có thêm mạch **dead-time**, **break input**, **complementary output**. - Hỗ trợ **sinh ngắt phức tạp** (Update,	- **Điều khiển động cơ**, **biến tần**, **servo PWM**. - Ứng dụng yêu cầu độ chính xác và an toàn cao.

Loại Timer	Đặc điểm chính	Ứng dụng điển hình
	Compare, Trigger, Break). - Dùng cho ứng dụng công suất cao.	
Low-Power Timer (LPTIM1, LPTIM2)	- Tiêu thụ năng lượng thấp, hoạt động được ở chế độ Sleep/Stop. - Hoạt động bằng xung nội (LSI/LSE).	- **Đồng hồ thời gian thực** (RTC backup). - **Ứng dụng tiết kiệm năng lượng**.