BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM
KHOA ĐIỆN - ĐIỆN TỬ
BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG

BÁO CÁO GIỮA KÌ

Học kỳ I/2025-2026

HỌC PHẦN: HỆ THỐNG NHÚNG

SYSTEM TIMER (SySTick)

GVHD: Ths. Huỳnh Hoàng Hà
Danh sách sinh viên:

1: Đoàn Minh Duy Bình	MSSV : 23139005
2: Hoàng Ngọc Dung	MSSV : 23139006
3: Trần Hữu Dương	MSSV : 23139009
4: Nguyễn Trần Minh Đức	MSSV : 23139012
5: Cao Như Ý	MSSV : 23139052

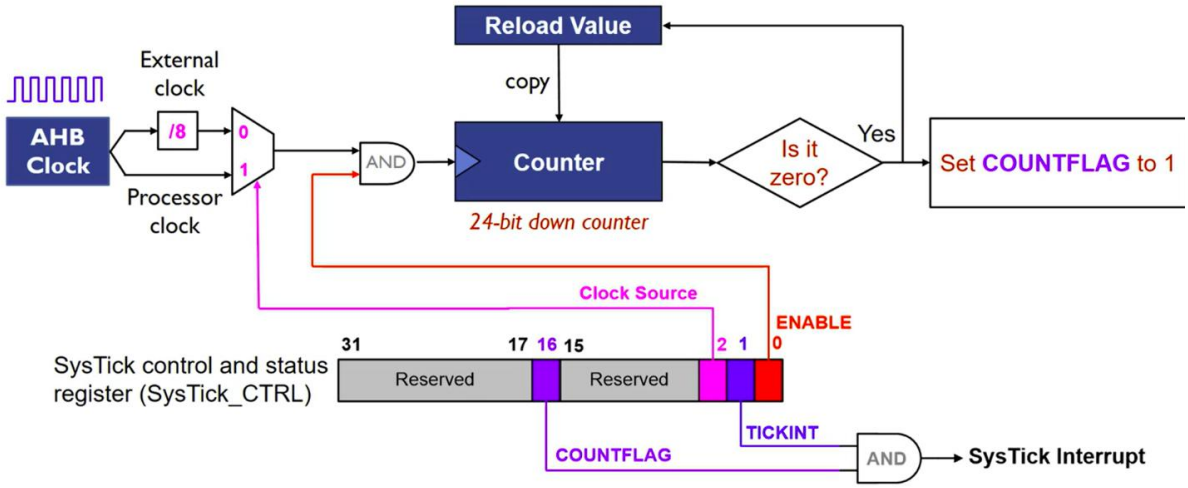
***TP. Hồ Chí Minh*, tháng 10 năm 2025**

MỤC LỤC

1.1. Khá quát về System Timer (SysTick)

1.1.1. Khái niệm

SysTick (System Tick Timer) là một bộ đếm thời gian 24 bit độc lập được tích hợp trong các vi điều khiển sử dụng kiến trúc ARM Cortex-M (dòng vi xử lý). Bộ đếm này sẽ giảm từ giá trị reload xuống 0 và tạo ra một ngắt SysTick khi giá trị này đạt 0. Sau đó, bộ đếm sẽ nạp lại giá trị reload và tiếp tục đếm từ giá trị đó.



Hình 1: Sơ đồ hoạt động của SysTick

1.1.2. Chức năng

- System Tick Timer thường được được dùng để tạo hàm delay với độ chính xác cao, thay thế cho các hàm delay với độ chính xác tương đối dùng vòng lặp for/while.
- Nhiệm vụ tạo ra ngắt SysTick theo chu kỳ thời gian cố định.

1.1.3. Ứng dụng

- Cung cấp khoảng ngắt định kỳ cho hệ điều hành thời gian thực RTOS
- Được sử dụng cho các mục đích ngắt có tính chu kỳ
- Tạo một khoảng thời gian trễ (delay)

Ví dụ:

- Đo thời gian trôi qua: dùng để cài đặt hàm trễ.
- Thực hiện tác vụ định kỳ: ví dụ polling trạng thái ngoại vi hoặc đọc dữ liệu đầu vào theo chu kỳ.
- Trong hệ điều hành: SysTick phục vụ bộ lập lịch CPU để quản lý đa nhiệm.

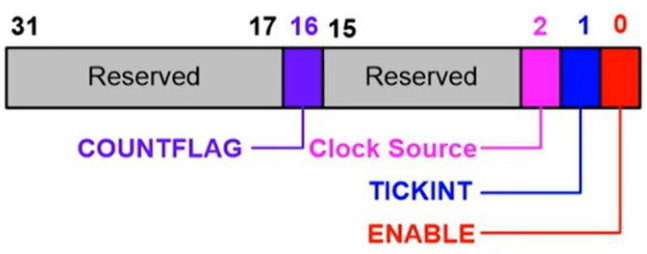
1.2. Cơ chế hoạt động của SysTick

Dựa vào hình 1 thể nguồn cách thức hoạt động Systick:

1. Nguồn clock đầu vào:
    - SysTick có thể chọn từ Processor Clock (AHB Clock) hoặc External Clock = AHB Clock / 8.
    - Bit Clock Source trong thanh ghi SysTick_CTRL dùng để chọn.
2. Bộ đếm 24-bit (Counter):
    - Là bộ đếm giảm (down counter) 24-bit.
    - Được nạp giá trị từ thanh ghi Reload Value.
    - Mỗi chu kỳ clock, bộ đếm giảm 1.
3. Khi Counter = 0:
    - Bit COUNTFLAG được set = 1 trong SysTick_CTRL để báo hiệu đã hết chu kỳ.
    - Bộ đếm được nạp lại từ Reload Value và tiếp tục giảm.
4. Tạo ngắt SysTick:
    - Nếu bit ENABLE = 1 (bật SysTick) và bit TICKINT = 1 (cho phép ngắt), thì khi Counter về 0 → NVIC sẽ nhận yêu cầu ngắt SysTick.
    - CPU sẽ nhảy vào SysTick_Handler() để xử lý.
5. Thanh ghi SysTick_CTRL (Control & Status Register):
    - ENABLE (bit 0): bật/tắt SysTick.
    - TICKINT (bit 1): bật/tắt ngắt SysTick.
    - CLKSOURCE (bit 2): chọn nguồn clock (Processor clock hoặc External clock).
    - COUNTFLAG (bit 16): **cờ báo hiệu khi Counter đếm từ 1 → 0.**

Đây là sơ đồ khối cho thấy cách SysTick tạo ra ngắt định kỳ trong ARM Cortex-M, dựa trên bộ đếm giảm 24-bit và các bit điều khiển trong thanh ghi SysTick_CTRL.

1.2.1. SysTick_CTRL (Control & Status Register)



Một số chức năng của thanh ghi CRTL: Bật/tắt SysTick, chọn nguồn xung clock (HCLK hoặc HCLK/8), kiểm tra cờ tràn (COUNTFLAG), cho phép/ngắt ngắt SysTick.

Các bit trong thanh ghi:

- Bit 0 – ENABLE (R/W): bật/tắt SysTick.
  - 0 → Counter dừng.
  - 1 → Counter chạy.
- Bit 1 – TICKINT (R/W): bật/tắt ngắt SysTick.
  - 0 → Không tạo ngắt khi counter về 0.
  - 1 → Tạo ngắt SysTick khi counter về 0.
- Bit 2 – CLKSOURCE (R/W): chọn nguồn clock.
  - 0 → Dùng clock tham chiếu bên ngoài (thường = HCLK/8).
  - 1 → Dùng clock của CPU (HCLK).
- Bit 16 – COUNTFLAG (RO): kiểm tra counter đã về 0 hay chưa.
  - 0 → Counter chưa về 0 kể từ lần đọc trước.
  - 1 → Counter vừa về 0 (cờ này tự clear khi đọc).
- Các bit khác [31:17], [15:3]: Reserved (không dùng).

1.2.2. SysTick_LOAD (Reload Value Register)

- Bit [31:24] – Reserved (Không dùng luôn bằng 0)
- Bit [23:0] – RELOAD : Giá trị nạp vào bộ đếm (CVR) mỗi khi counter đếm về 0

=> Đây chính là chu kỳ lặp của SysTick.

- Giá trị tối đa: 0x00FF.FFFF (16.777.215) : 24 bit
- Bộ đếm giảm dần từ giá trị RELOAD về 0.
- Ghi giá trị RELOAD = 0 sẽ vô hiệu hóa SysTick, không phụ thuộc vào bit TICKINT.
- Khoảng thời gian giữa hai ngắt SysTick:

$$Interval = (RELOAD + 1)\ x\ Chu\ kỳ\ Clock$$

- Nếu có 100 chu kỳ clock giữa hai ngắt SysTick → RELOAD = 99.

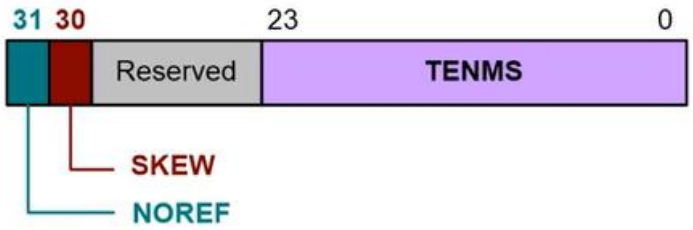
$$T_s = 100\ \times T_c \rightarrow (RELOAD + 1)\ \times T_c = 100\ \times T_c \rightarrow RELOAD = 100 - 1 = 99$$

1.2.3. SysTick_VAL (Current Value Register)

Hình 2: Bits [31:0] – CURRENT (giá trị của bộ đếm tại thời điểm lấy mẫu)

- Khi đọc → trả về giá trị hiện tại của bộ đếm.
- Khi bộ đếm chuyển từ 1 xuống 0 → tạo ra một ngắt.
- Khi ghi vào SysTick_VAL → sẽ xóa bộ đếm và đưa COUNTFLAG về 0.
- Điều này khiến bộ đếm sẽ nạp lại (reload) ở chu kỳ clock tiếp theo.
- Tuy nhiên, không kích hoạt ngắt SysTick.

1.2.4. SysTick_CALIB (Calibration Register)



Thanh ghi chỉ đọc (Read-only register) . Trong đóTENMS (10 ms): chứa giá trị reload, giá trị này tương ứng với chu kỳ 10 ms. Nhằm cung cấp giá trị hiệu chỉnh theo clock chuẩn 10ms và dùng để tạo thời gian chuẩn theo datasheet.

Các bit trong thanh ghi:

- Bit 30 – SKEW
    - 0: Giá trị hiệu chuẩn 10ms (TENMS) là chính xác.
    - 1: Giá trị hiệu chuẩn 10ms không chính xác, do tần số clock.
- Bit [29:24] – Reserved (không dùng).
- Bit [23:0] – TENMS: Chứa giá trị reload tương ứng với 10ms (tức 100 Hz) theo clock mặc định.
- Bit 31 – NOREF
    - 0: Có clock tham chiếu (reference clock) được implement.
    - 1: Không có clock tham chiếu.

1.4. Cách thay đổi ngắt thời gian

1.4.1. Công thức chu kỳ ngắt SysTick

$$T_s = (LOAD + 1) \times T_c$$

- Ý nghĩa: tính thời gian giữa hai lần ngắt SysTick liên tiếp.
- Trong đó:
  - LOAD: giá trị nạp lại vào bộ đếm (0 → 0xFFFFFF).
  - T_c: chu kỳ của clock nguồn (nghịch đảo của tần số clock).
- Đây là công thức tổng quát, cho biết nếu ta nạp LOAD bao nhiêu thì khoảng thời gian ngắt sẽ là bao nhiêu.

Ví dụ:

- Clock = 80 MHz → T_clock = 12.5 ns.
- LOAD = 799,999 →

$$T_s = (799{,}999 + 1) \times 12.5ns = 10msT$$

1.4.2. Công thức tính giá trị RELOAD

$$RELOAD = (Clock\ /\ Tần\ số\ mong\ muốn) - 1$$

Systick bản chất là 1 bộ đếm lùi 24bit. Khi ta nạp vào thanh ghi LOAD 1 giá trị N (N là reload value) thì thanh ghi Current Value sẽ bắt đầu đếm từ N->0

=> Mất đúng (N+1)* chu kỳ clock thì counter mới đếm về 0, sinh ngắt

Muốn thay đổi tốc độ → chỉ cần thay đổi RELOAD hoặc thay đổi nguồn clock.

Ví dụ:

- Clock = 80 MHz.
- Muốn ngắt mỗi 10 ms → f_mong_muon = 100 Hz.
- RELOAD = (80,000,000 / 100) – 1 = 799,999.

1.4.3. Mối quan hệ giữa hai công thức

- Chu kỳ ngắt:

$$T_s = (LOAD + 1) \times T_{clock}$$

- Tần số ngắt:

$$f_s = \frac{1}{T_s} = \frac{Clock}{LOAD + 1}$$

- Tính RELOAD (LOAD):

$$LOAD = \frac{Clock}{f_s} - 1$$

Clock là tần số nguồn (ví dụ 80 MHz), LOAD là giá trị nạp vào bộ đếm SysTick, còn f_SysTick là tần số ngắt mong muốn. Công thức (1) được sử dụng để tính chu kỳ ngắt từ giá trị RELOAD, trong khi công thức (2) được dùng để tính RELOAD dựa trên chu kỳ hoặc tần số ngắt mong muốn. Vì vậy:

- Nếu tăng tần số CPU → chu kỳ SysTick ngắn hơn, ngắt xảy ra nhanh hơn.
- Nếu giảm tần số CPU → chu kỳ SysTick dài hơn, ngắt chậm hơn.
- Muốn giữ thời gian ngắt cố định (ví dụ 1 ms), thì phải tính lại RELOAD mỗi khi thay đổi tần số.

1.5. SysTick và thư viện HAL

1.5.1. Phân tích chi tiết biến và dòng liên quan đến SysTick

Trong quá trình sử dụng HAL trên dòng vi điều khiển STM32, hệ thống **SysTick** đóng vai trò quan trọng trong việc tạo ra các ngắt định kỳ để phục vụ hệ điều hành thời gian thực hoặc các tác vụ định thời khác. Dưới đây là phân tích chi tiết về các biến và hằng số liên quan đến SysTick được sử dụng trong thư viện HAL:

Biến / Hằng số	Ý nghĩa và chức năng
__IO uint32_t uwTick	- __IO là macro từ CMSIS biểu thị biến kiểu volatile, dùng để truy cập an toàn trong các ngữ cảnh có ngắt (Interrupt Service Routine). - uwTick là biến đếm hệ thống (global tick), được tăng mỗi khi ngắt SysTick xảy ra (thường trong hàm HAL_IncTick()).
uint32_t uwTickPrio = (1UL << __NVIC_PRIO_BITS)	- __NVIC_PRIO_BITS là số bit dùng để biểu diễn độ ưu tiên trong NVIC. - Sau khi hàm HAL_InitTick() khởi tạo xong, uwTickPrio sẽ nhận giá trị bằng TickPriority. - Mục đích: thiết lập độ ưu tiên của ngắt SysTick trong NVIC. Nếu giá trị không hợp lệ, hệ thống sẽ dùng mặc định khi gọi HAL_InitTick().
HAL_TickFreqTypeDef uwTickFreq = HAL_TICK_FREQ_DEFAULT	- uwTickFreq là biến định nghĩa số **millisecond** được cộng vào uwTick sau mỗi lần ngắt SysTick. - Biến này là kiểu enum được định nghĩa trong HAL, gồm các giá trị sau: • HAL_TICK_FREQ_1KHZ → 1U (1ms/tick, tickbase 1kHz) • HAL_TICK_FREQ_100HZ → 10U (10ms/tick)

	• HAL_TICK_FREQ_10HZ → 100U (100ms/tick) • HAL_TICK_FREQ_DEFAULT mặc định là HAL_TICK_FREQ_1KHZ.

Lưu ý: Các định nghĩa và macro trên được xây dựng dựa trên thư viện CMSIS – một tập hợp các chuẩn do ARM phát triển nhằm cung cấp quyền truy cập thống nhất tới các thanh ghi và tài nguyên phần cứng của nhân Cortex-M (ví dụ như SysTick, NVIC...).
CMSIS là một tập hợp các thư viện chuẩn hóa do ARM phát triển nhằm:

- Chuẩn hóa truy cập đến các thanh ghi cấp thấp (register-level)
- Tạo ra nền tảng thống nhất giữa các hãng vi điều khiển dùng nhân Cortex-M
- Dễ dàng tích hợp và tái sử dụng mã nguồn (HAL, RTOS, DSP, NN, v.v.)

NVIC là một phần tích hợp trong CPU giúp quản lý và xử lý các ngắt (interrupts):

- Tự động ưu tiên
- Cho phép ngắt lồng nhau
- Phản ứng nhanh với ngắt có độ ưu tiên cao hơn
- HAL_InitTick(TICK_INT_PRIORITY): khởi tạo nguồn timebase (mặc định là SysTick). Cấu hình nguồn tạo thời gian hệ thống (timebase), mặc định là sử dụng bộ đếm SysTick. Đây là thành phần rất quan trọng vì liên quan đến các hàm như HAL_Delay() và thời gian hệ thống. Một lưu ý đặc biệt là HAL không kiểm tra giá trị trả về từ HAL_InitTick() — do đó nếu hàm này thất bại (ví dụ do giá trị reload quá lớn), chương trình vẫn tiếp tục chạy và có thể gây ra lỗi treo khi dùng HAL_Delay().
- HAL_MspInit(): Gọi tới hàm khởi tạo MSP (MCU Support Package), nơi người dùng có thể định nghĩa các cấu hình ngoại vi cụ thể như bật xung nhịp, cấu hình GPIO, DMA, v.v.

1.5.2. Các hàm HAL liên quan đến Systick

Trong môi trường STM32, thư viện HAL (Hardware Abstraction Layer) cung cấp một tập hợp các hàm hỗ trợ người lập trình dễ dàng cấu hình và sử dụng bộ định thời hệ thống SysTick. Các hàm này bao gồm khả năng cấu hình tần số ngắt, khởi tạo, trì hoãn (delay) chương trình, và quản lý tạm dừng hoặc khôi phục tick.

a) HAL_Init(void): Khởi tạo toàn bộ thư viện HAL, bao gồm cấu hình ưu tiên ngắt, hệ thống time base (mặc định là SysTick).

```
HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
HAL_InitTick(TICK_INT_PRIORITY);
HAL_MspInit();
```

Dòng lệnh	Mô tả
HAL_NVIC_SetPriorityGrouping(...)	Cấu hình mức độ phân nhóm ưu tiên ngắt của hệ thống NVIC. Không trực tiếp liên quan SysTick nhưng ảnh hưởng đến cách phân chia bit ưu tiên.
HAL_InitTick(TICK_INT_PRIORITY)	Khởi tạo SysTick làm nguồn thời gian hệ thống (time base). Nếu cấu hình thất bại (ví dụ reload quá lớn), HAL vẫn tiếp tục chạy, tiềm ẩn nguy cơ treo nếu gọi HAL_Delay().
HAL_MspInit()	Gọi hàm người dùng để cấu hình ngoại vi như GPIO, Clock, DMA...

b) __weak HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)

Hàm HAL_InitTick() có nhiệm vụ khởi tạo bộ SysTick để tạo nguồn thời gian hệ thống (time base) cho thư viện HAL. Nội dung quan trọng của hàm gồm hai phần: cấu hình 1. SysTick và thiết lập độ ưu tiên ngắt.

1. Cấu hình SysTick

```
if (HAL_SYSTICK_Config(SystemCoreClock / (1000U / uwTickFreq)) > 0U)
{
  return HAL_ERROR;
}
```

Biểu thức SystemCoreClock / (1000U / uwTickFreq):

- SystemCoreClock: tần số xung nhịp CPU (Hz).
- 1000U / uwTickFreq: mẫu số thể hiện số lần ngắt trên mỗi giây.
- Kết quả của phép chia cho biết số chu kỳ clock cần đếm trước khi sinh ra một ngắt SysTick.

Ví dụ: với SystemCoreClock = 72MHz, uwTickFreq = 1 (1ms):

72_000_000 / 1000 = 72_000 → LOAD = 71_999.

Với uwTickFreq = 10 (100Hz):

72_000_000 / 100 = 720_000 → LOAD = 719_999.

Kiểm tra giới hạn:

Nếu giá trị vượt giới hạn 24-bit → HAL_SYSTICK_Config() trả lỗi → HAL_ERROR.

2. Cấu hình độ ưu tiên ngắt SysTick:

```
if (TickPriority < (1UL << __NVIC_PRIO_BITS))
{
  HAL_NVIC_SetPriority(SysTick_IRQn, TickPriority, 0U);
  uwTickPrio = TickPriority;
}
else
{
  return HAL_ERROR;
}
```

Ý nghĩa: Xác định mức ưu tiên cho ngắt SysTick trong bộ điều khiển NVIC.

- __NVIC_PRIO_BITS: số bit dùng cho cấu hình mức ưu tiên (tùy dòng chip, thường là 4 bit).
- Nếu giá trị TickPriority hợp lệ, hàm HAL_NVIC_SetPriority() sẽ được gọi để:
- Gán mức ưu tiên cho SysTick_IRQn.
- Cập nhật biến toàn cục uwTickPrio lưu lại giá trị này.

Nếu TickPriority vượt giới hạn cho phép, hàm trả về lỗi (HAL_ERROR).

c) __weak void HAL_IncTick(void)

Tăng biến thời gian hệ thống uwTick mỗi lần xảy ra ngắt SysTick.

uwTick += uwTickFreq;

Ý nghĩa:

- Nếu uwTickFreq = 1 → uwTick tăng 1 mỗi mili-giây.
- Có thể override để thay đổi hành vi trong main.c.

d) __weak uint32_t HAL_GetTick(void)

Trả về giá trị thời gian hệ thống (uwTick) tính bằng đơn vị mili-giây.

```
__weak uint32_t HAL_GetTick(void)
{
    return uwTick;
}
```

Hàm trả về giá trị hiện tại của biến toàn cục uwTick.

Vì uwTick được tăng đều đặn trong hàm HAL_IncTick() (mặc định gọi trong SysTick Handler), giá trị trả về của HAL_GetTick() chính là thời gian hệ thống (system time) tính theo đơn vị millisecond.

Cơ chế hoạt động:

```
uint32_t tickstart = HAL_GetTick();
uint32_t wait = Delay;

if (wait < HAL_MAX_DELAY) {
  wait += (uint32_t)(uwTickFreq);
}

while ((HAL_GetTick() - tickstart) < wait);
```

Ghi chú quan trọng:

Nếu HAL_SuspendTick() đã được gọi hoặc SysTick chưa được cấu hình đúng → uwTick không tăng → chương trình treo.

Không nên gọi trong ISR có ưu tiên cao hơn SysTick → có thể gây kẹt chương trình.

e) __weak void HAL_SuspendTick(void)

Chức năng: HAL_SuspendTick(): Vô hiệu hóa ngắt SysTick (SysTick vẫn đếm nhưng không sinh ngắt).

Cơ chế:

CLEAR_BIT(SysTick->CTRL, SysTick_CTRL_TICKINT_Msk); // Suspend

f) __weak void HAL_ResumeTick(void)

Chức năng: Bật lại ngắt SysTick.

Cơ chế: SET_BIT(SysTick->CTRL, SysTick_CTRL_TICKINT_Msk); // Resume

- Lưu ý: Nếu gọi HAL_Delay() sau khi HAL_SuspendTick(), chương trình có thể bị treo. HAL không thao tác tới bit ENABLE → counter vẫn hoạt động dù ngắt bị tắt.

g) HAL_StatusTypeDef HAL_SetTickFreq(HAL_TickFreqTypeDef Freq)

Chức năng: Cho phép thay đổi tần số tick hệ thống (biến uwTickFreq) trong khi chương trình đang chạy. Cụ thể, hàm thực hiện các bước sau:

1. Kiểm tra tham số

assert_param(IS_TICKFREQ(Freq));

- Đảm bảo giá trị truyền vào Freq hợp lệ (ví dụ HAL_TICK_FREQ_10HZ, HAL_TICK_FREQ_100HZ, HAL_TICK_FREQ_1KHZ).
- Nếu không hợp lệ, chương trình sẽ dừng tại assert_param() (trong chế độ Debug).

2. Lưu tạm giá trị cũ

backup prevTickFreq = uwTickFreq;

uwTickFreq = Freq;

- Lưu lại giá trị tick frequency hiện tại (prevTickFreq) để có thể khôi phục nếu cấu hình mới thất bại.
- Gán uwTickFreq bằng giá trị mới Freq.

3. Tái cấu hình SysTick

status = HAL_InitTick(uwTickPrio);

- Gọi lại HAL_InitTick() để tính toán lại giá trị reload của SysTick dựa trên uwTickFreq mới.
- Nếu cấu hình thành công → status = HAL_OK.
- Nếu thất bại (ví dụ do giá trị reload vượt quá khả năng 24-bit của bộ đếm SysTick) → status = HAL_ERROR.

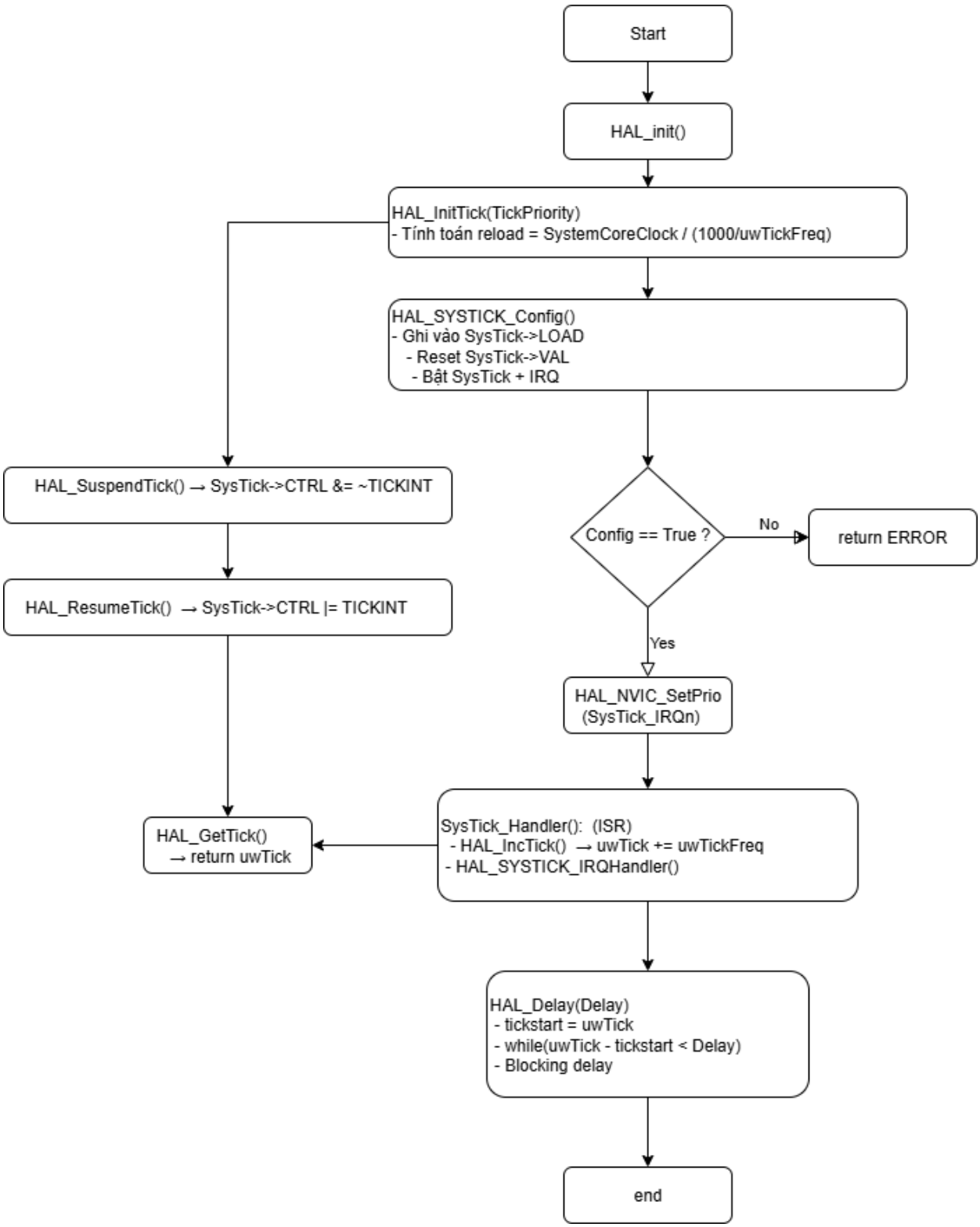
4. Khôi phục khi lỗi

```
if (status != HAL_OK) {
   uwTickFreq = prevTickFreq;
}
```

- Nếu không cấu hình được SysTick với tần số mới, hàm khôi phục lại uwTickFreq bằng giá trị cũ.
- Điều này đảm bảo hệ thống tiếp tục hoạt động ổn định.

f) Sơ đồ Luồng hàm SysTick trong HAL (cơ bản trong báo cáo đề cập)

Sơ đồ mô tả cơ chế hoạt động của **SysTick trong HAL STM32**. Khi hàm HAL_Init() được gọi, hệ thống sẽ khởi tạo time base thông qua HAL_InitTick(), tính toán giá trị reload và cấu hình SysTick bằng HAL_SYSTICK_Config(). Nếu thành công, NVIC được thiết lập độ ưu tiên cho ngắt SysTick; nếu thất bại, hàm trả về lỗi. Mỗi lần SysTick hết hạn sẽ sinh ngắt và gọi SysTick_Handler(), trong đó HAL_IncTick() tăng biến thời gian hệ thống uwTick. Biến này có thể được truy cập qua HAL_GetTick() hoặc dùng trong HAL_Delay() để tạo delay dạng blocking. Ngoài ra, có thể tạm dừng hoặc tiếp tục ngắt SysTick bằng HAL_SuspendTick() và HAL_ResumeTick(). Như vậy, SysTick đóng vai trò trung tâm trong việc duy trì thời gian hệ thống cho các hàm HAL.



1.5.3. Các thành phần phần cứng (SysTick registers)

Trong bộ vi điều khiển STM32, SysTick là một bộ định thời (timer) có sẵn trong nhân Cortex-M, thường được sử dụng để tạo các ngắt định kỳ phục vụ cho việc quản lý thời gian. Dưới đây là các thành phần thanh ghi và hàm xử lý liên quan đến SysTick:

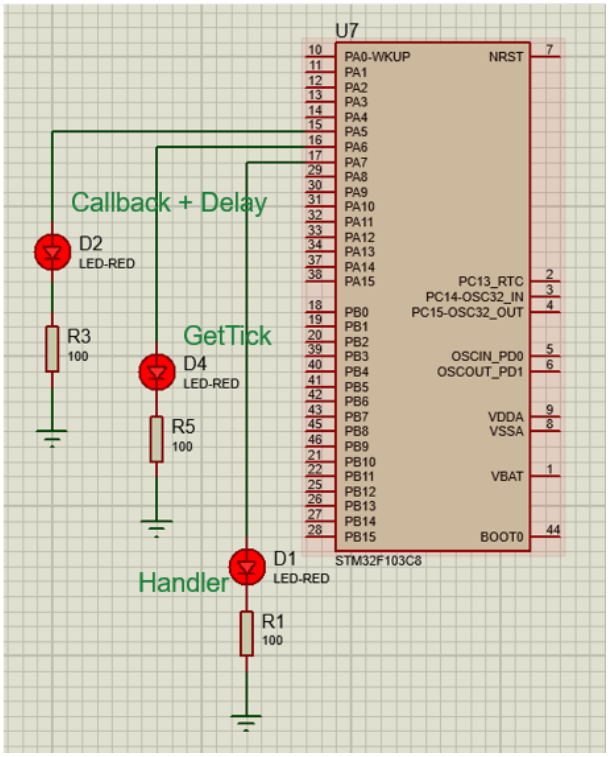
Hàm / Thanh ghi	Chức năng
SysTick_Config(uint32_t ticks)	Hàm thuộc CMSIS, dùng để cấu hình giá trị reload, kích hoạt bộ đếm và bật ngắt SysTick. Hàm này chủ yếu dùng trong lập trình bare-metal (không sử dụng HAL).
SysTick_Handler(void)	Hàm xử lý ngắt SysTick gốc. Trong thư viện HAL, hàm này sẽ gọi đến HAL_IncTick() và HAL_SYSTICK_IRQHandler() để cập nhật biến thời gian và xử lý ngắt.
SysTick->CTRL	Thanh ghi điều khiển SysTick. Bao gồm các bit điều khiển như: - Enable: Bật/tắt SysTick. - TickInt: Cho phép tạo ngắt khi đếm về 0. - Clock source: Chọn nguồn xung nhịp. - COUNTFLAG: Cờ báo hiệu khi SysTick đếm xong.
SysTick->LOAD	Thanh ghi nạp lại (reload value). Khi bộ đếm đếm từ giá trị này về 0 sẽ sinh ra ngắt và bắt đầu lại chu kỳ mới.
SysTick->VAL	Giá trị hiện tại của bộ đếm. Có thể được reset về 0 để khởi động lại chu kỳ đếm từ đầu.
SysTick->CALIB	Thanh ghi hiệu chỉnh chuẩn (calibration). Cung cấp thông tin hiệu chỉnh dựa vào xung nhịp HSI hoặc HSE. Được dùng trong chế độ tự hiệu chỉnh (auto-calibration) nhưng không sử dụng trong HAL.

Lưu ý: Các thanh ghi và hàm trên thuộc **CMSIS (Cortex Microcontroller Software Interface Standard)** – bộ thư viện chuẩn hóa do ARM phát triển để truy cập các phần tử phần cứng của nhân Cortex-M.

1.6. Code ví dụ tham khảo

1.6.1. Sử dụng thư viện HALL

Mô phỏng: Bài tập SYSTICK chớp-tắt led cho STM32 (dòng STM32F103C08) để so sánh tốc độ sử dụng các hàm như hình mô phỏng sau:



a) Hàm hàm Delay() bằng ngắt:

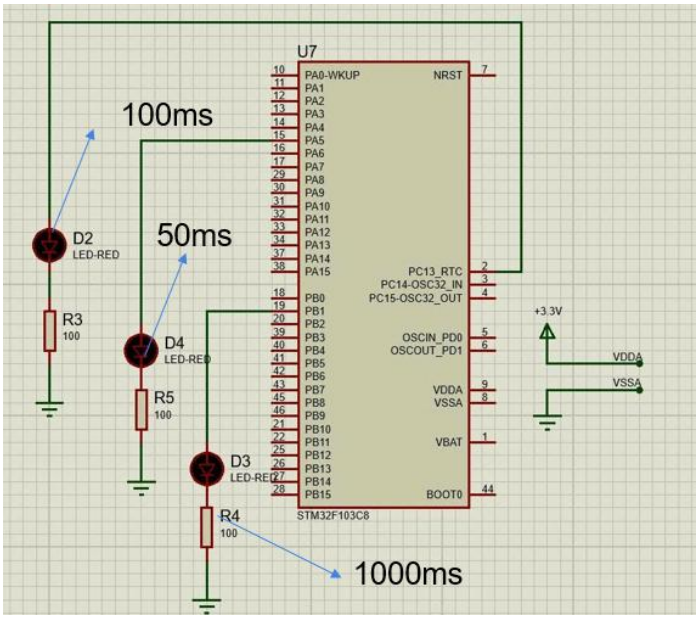
```
volatile int32_t TimeDelay;

int main (void) {
   SysTick_Initialize(1000);     // Chu kỳ ngắt = 1000 chu kỳ clock
   Delay(100);                   // Tạo trễ 100 tick
   ...
}

void SysTick_Handler (void) {     // Trình phục vụ ngắt SysTick
   if (TimeDelay > 0)            // Ngăn không cho TimeDelay âm
      TimeDelay--;               // Giảm biến đếm toàn cục TimeDelay
}

void Delay (uint32_t nTime) {
   // nTime: chỉ định độ dài thời gian trễ
   TimeDelay = nTime;            // Gán giá trị ban đầu cho TimeDelay
   while (TimeDelay != 0);       // Vòng lặp bận (busy wait) cho đến khi TimeDelay về
}
```

1.6.2. SysTick viết hàm sử dụng thanh ghi

Mô phỏng mạch chạy thực tế:

Bài tập SYSTICK chớp-tắt led cho STM32

(dòng STM32F103C08) để so sánh tốc độ



a) Khai báo

```
#include "main.h"
/* SysTick delay kiểu thanh ghi (1 ms) */
static void Delay_1_Ms(void);
static void Delay_Ms(uint32_t u32Delay);
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* 1 ms delay với SysTick @ HCLK = 8 MHz (HSI) */
```

b) Hàm Delay

```
static void Delay_1_Ms(void){
  /* SysTick là counter 24-bit đếm xuống theo HCLK khi CLKSOURCE = 1 */
  SysTick->LOAD = 8U * 1000U - 1U;  /* 8000 tick = 1 ms @ 8 MHz */
  SysTick->VAL  = 0U;               /* reset current value */
  SysTick->CTRL = (1U << 2)         /* CLKSOURCE = HCLK */
                | (1U << 0);        /* ENABLE = 1, không bật interrupt */
  /* Đợi COUNTFLAG (bit 16) bật lên khi đếm xong (hết 1ms) */
  while ((SysTick->CTRL & (1U << 16)) == 0U) {
    /* wait */}
  /* Tắt SysTick sau mỗi lần delay (giữ “sạch”) */
  SysTick->CTRL = 0U;}
```

```
/* Delay theo ms bằng cách lặp 1 ms,Để delay N ms → gọi Delay_1_Ms() lặp lại
N lần. */
```

```
static void Delay_Ms(uint32_t u32Delay)
{
  while (u32Delay--) {
    Delay_1_Ms();
  }
}
```

b) Hàm Main chính

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();  // Khởi tạo HAL, SysTick (1ms), reset trạng thái peripheral
  SystemClock_Config(); // Cấu hình clock (ở đây giữ HSI = 8 MHz)
  MX_GPIO_Init();// Cấu hình GPIOB, GPIOC làm output push-pull
  /* Mức khởi tạo:
     - PC13: set 1 (LED on-board active-low -> tắt)
     - PB1, PB2: kéo xuống 0 */
 GPIOC->ODR |=  (1U << 13); // PC13 = 1 → LED off (LED on-board active-low)
  GPIOB->ODR &= ~(1U << 1); // PB1 = 0
  GPIOB->ODR &= ~(1U << 2); // PB2 = 0
 while (1) // vong lap tao 3 nhip khac nhau cho 3 chan
  {
    /* --- PB1: toggle mỗi 50 ms --- */
    GPIOB->ODR ^= (1U << 1);  // PB1 thay đổi trạng thái mỗi 50 ms (ON-OFF-
ON…).
    Delay_Ms(50);
    /* --- PB2: toggle mỗi 5 ms --- */
    GPIOB->ODR ^= (1U << 2);
    Delay_Ms(5);
    GPIOC->ODR ^= (1U << 13);
    Delay_Ms(1000); // LED on-board chop cham 1 lan/s
  }
}
```

c) Hàm khởi tại với nhiệm thiết lập bộ định thời SysTick của ARM Cortex-M để tạo ra ngắt định kỳ.

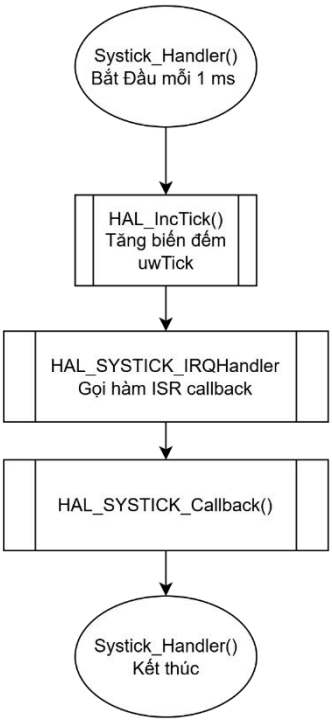
```
void SysTick_Initialize (uint32_t ticks) {
    SysTick->CTRL = 0;                          // Tắt SysTick
    SysTick->LOAD = ticks - 1;                  // Gán giá trị reload

    // Đặt mức ưu tiên ngắt của SysTick thấp nhất (tức là giá trị ưu tiên
lớn nhất)
    NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);

    SysTick->VAL = 0;                           // Xóa giá trị bộ đếm hiện tại
```

```
    // Chọn clock cho bộ xử lý: 1 = clock CPU; 0 = clock ngoài
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE;

    // Bật ngắt SysTick, 1 = bật, 0 = tắt
    SysTick->CTRL |= SysTick_CTRL_TICKINT;

    // Bật SysTick
    SysTick->CTRL |= SysTick_CTRL_ENABLE;
}
```

Trước hết, thanh ghi điều khiển (**CTRL**) được xóa để tắt SysTick, đảm bảo an toàn khi cấu hình. Sau đó, giá trị tải lại (**LOAD**) được gán bằng ticks - 1, quyết định khoảng thời gian giữa hai ngắt liên tiếp. Hàm **NVIC_SetPriority()** được gọi để đặt mức ưu tiên ngắt của SysTick xuống thấp nhất, tránh ảnh hưởng đến các ngắt quan trọng khác. Thanh ghi giá trị hiện tại (**VAL**) được reset về 0 để bắt đầu đếm từ đầu. Tiếp đó, code chọn nguồn clock cho SysTick (thường là clock của CPU), bật chức năng ngắt SysTick (**TICKINT**) để cho phép CPU xử lý khi bộ đếm về 0, và cuối cùng kích hoạt SysTick (**ENABLE**) để bắt đầu hoạt động.

1.7. Ứng dụng vào dự án cuối kì

Sử dụng API điều khiển servo . Ví dụ: quá trình điều khiển động cơ cho cá ăn. Mỗi khi có lệnh cho ăn thì sẽ thực hiện đếm lên 1000 tick tương ứng với 1s



Hình 3: Sơ đồ ngắt Systick sử dụng API.

- DS3231 (RTC): giữ thời gian thực (giờ, phút, giây) và cung cấp cho MCU khi đọc.
- SysTick (trên MCU): tạo xung nhịp định kỳ (ví dụ 1 ms), giúp MCU thực hiện kiểm tra thời gian theo chu kỳ.
- Servo SG90: nhận tín hiệu PWM từ MCU xoay trục, điều khiển cơ cấu đổ thức ăn.
- Cơ chế hoạt động tổng quát
    - SysTick → tạo nhịp 1 giây → MCU đọc DS3231.
    - MCU → so sánh giờ ăn → nếu trùng khớp → MCU xuất PWM.
    - Servo → quay góc → đổ thức ăn → quay về vị trí ban đầu.

```
void HAL_SYSTICK_IRQHandler(void)
{
 HAL_SYSTICK_Callback();
}
void HAL_SYSTICK_Callback(void)
{
   // Ví dụ: quay động cơ trong 1s
   static uint32_t counter = 0;
   if(counter >= 1000) {  // 1000 * 1ms = 1s
    feed.IsFeed = 3;
       counter = 0;
   }
   if (feed.IsFeed == 2) {
       counter++;
   }
}
```

1.8. Tài liệu tham khảo

[1] Embedded Systems and Deep Learning. (n.d.). Lecture 12: System Timer (SysTick) [Video]. YouTube. https://www.youtube.com/watch?v=aLCUDv_fgoU

[2] M. (2010). Cortex-M3 devices generic user guide (Rev. r2p1). ARM Ltd. https://developer.arm.com/documentation/dui0552/a

[3] STMicroelectronics, "Description of STM32L0 HAL and low-layer drivers," UM1749, ST, 2017. [Online]. Available: https://www.st.com/resource/en/user_manual/um1749-description-of-stm32l0-hal-and-low-layer-drivers-stmicroelectronics.pdf. [Accessed: 29-Sep-2025].

[4] giunzz, "STM32_GK," GitHub Repository, 2025. [Online]. Available: https://github.com/giunzz/STM32_GK. [Accessed: 29-Sep-2025].

[5] giunzz, "Systick.zip," GitHub Repository Data_Science, 2025. [Online]. Available: https://github.com/giunzz/Data_Science/blob/main/Systick.zip. [Accessed: 29-Sep-2025].