Embedded Systems

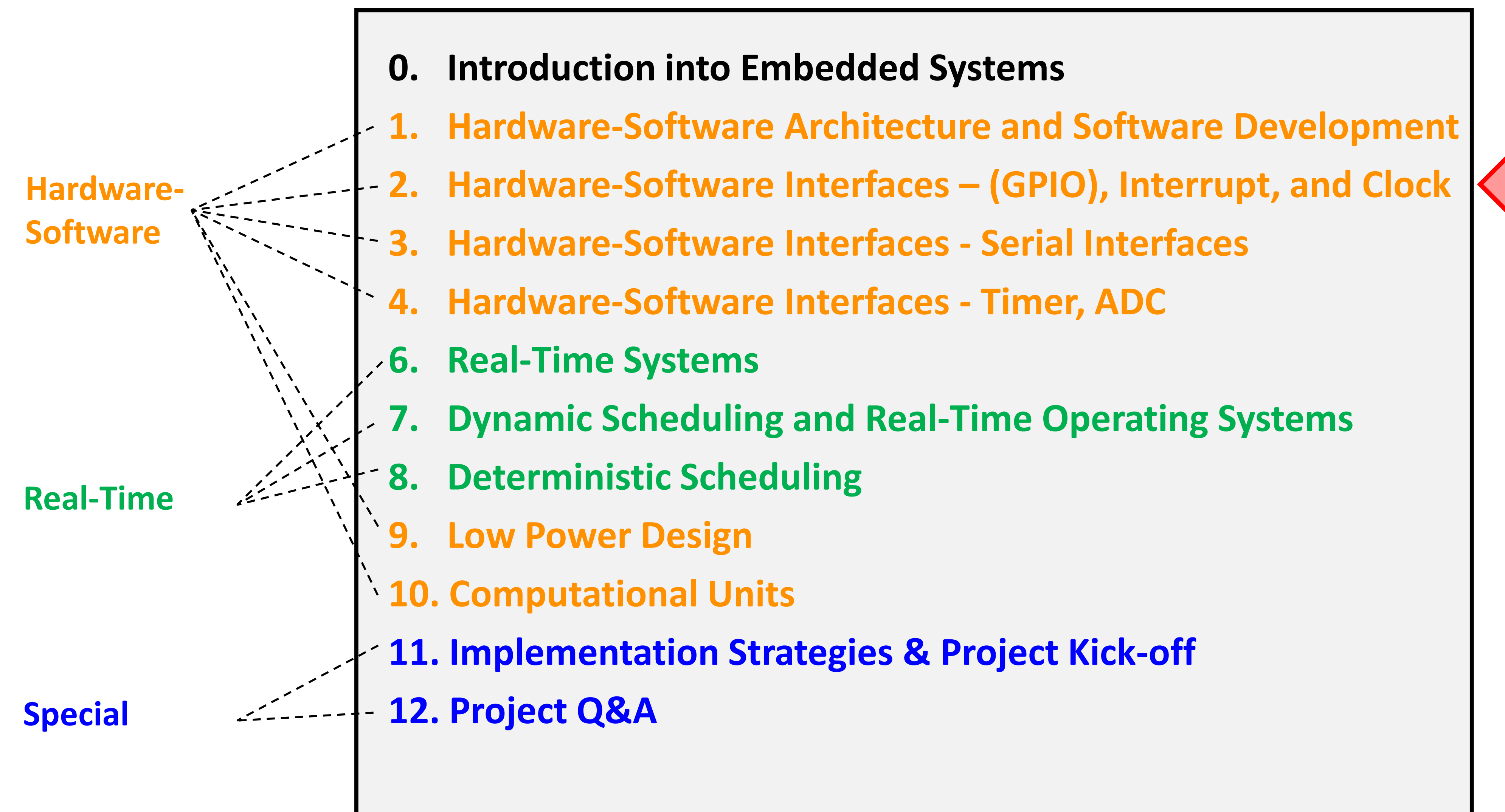
Lecture 2
Hardware-Software Interfaces –GPIO, Interrupt, and Clock

© Michele Magno

D-ITET Center for Project-Based Learning

Source the Slides or adaptation from:
Embedded Systems. P. Marwedel

Where We are



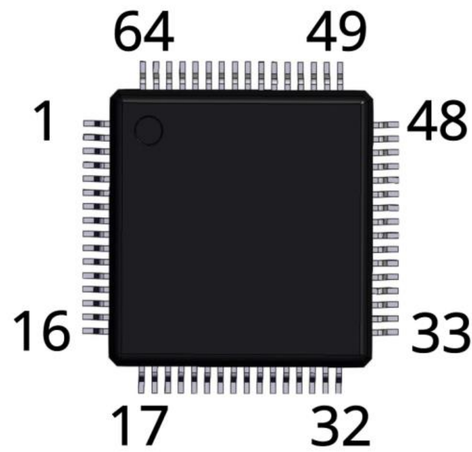
GPIO (General-Purpose Input-Output) Recap

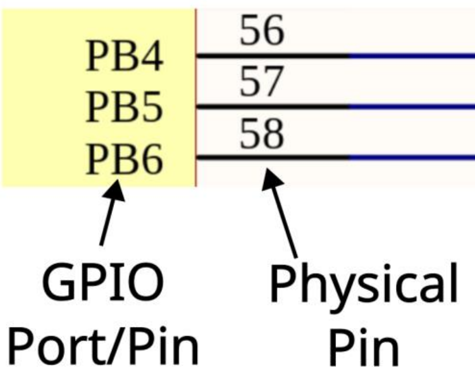
Remember those Pins?

- **MCUs feature a number of General-Purpose Input/Output (GPIO) pins.**

- **They are grouped into ports:**
  - Each port is identified by a letter (A-H).
  - Each port features up to 16 GPIO pins.

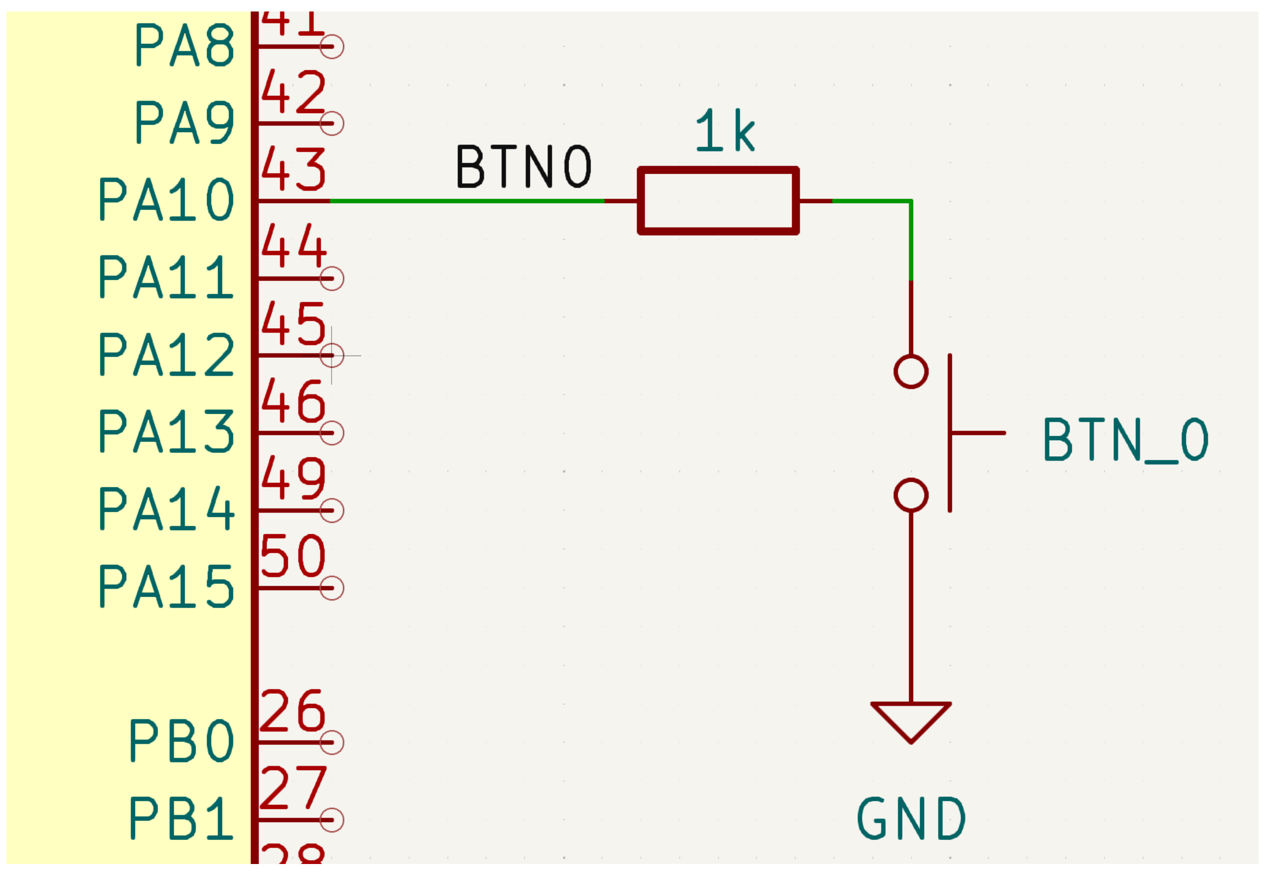
- **They can be configured in a variety of ways:**
  - Input/Output/Analog/Alternate (Peripheral)
  - Push-Pull/Open Drain
  - Internal Pull-Ups/Pull-Downs



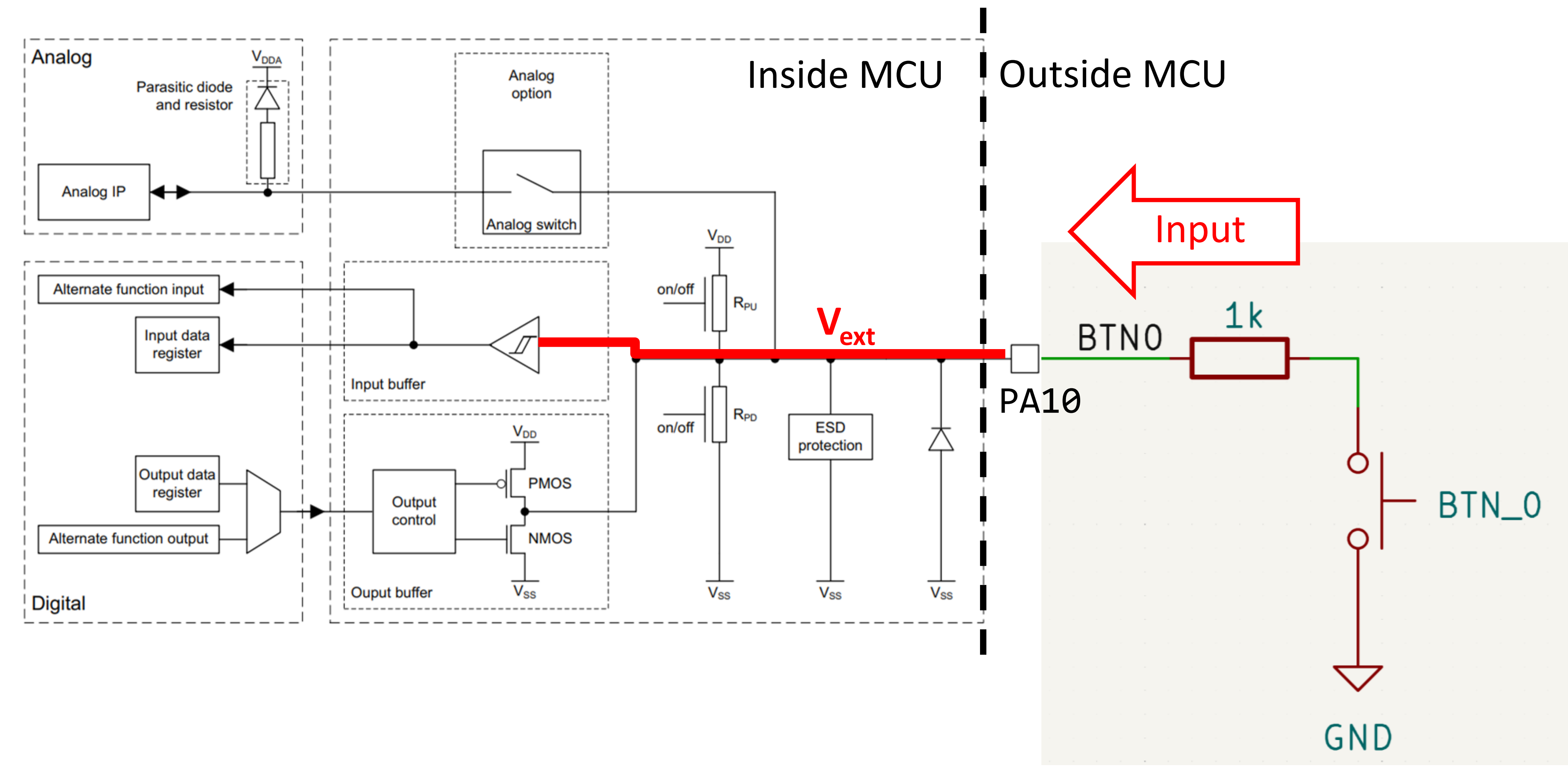


GPIO: Inputs Recap

- **Let's focus on a reading the state of a button using GPIO Input:** `BTN_0` on pin `PA10`

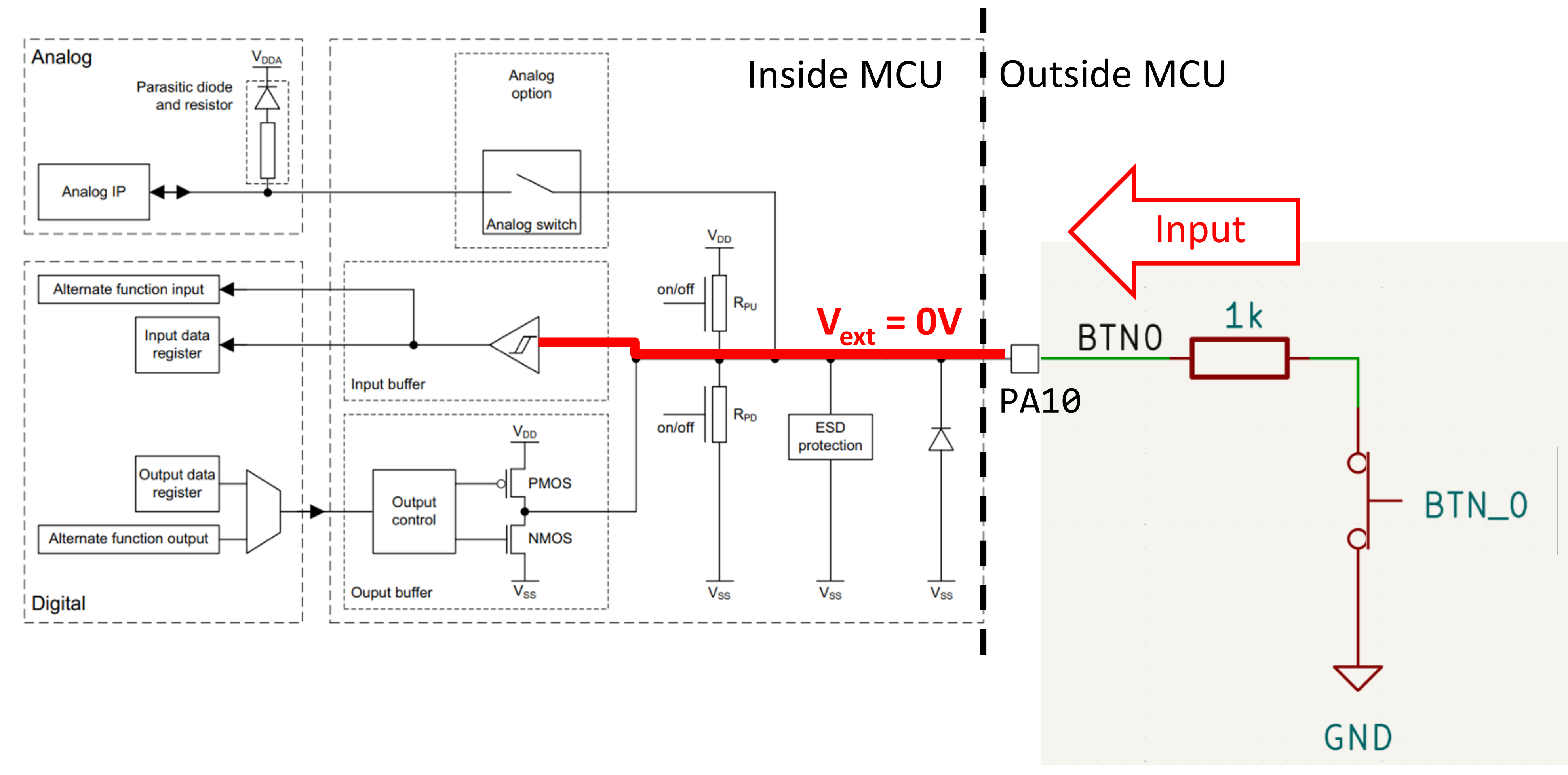


GPIO: Inputs Recap



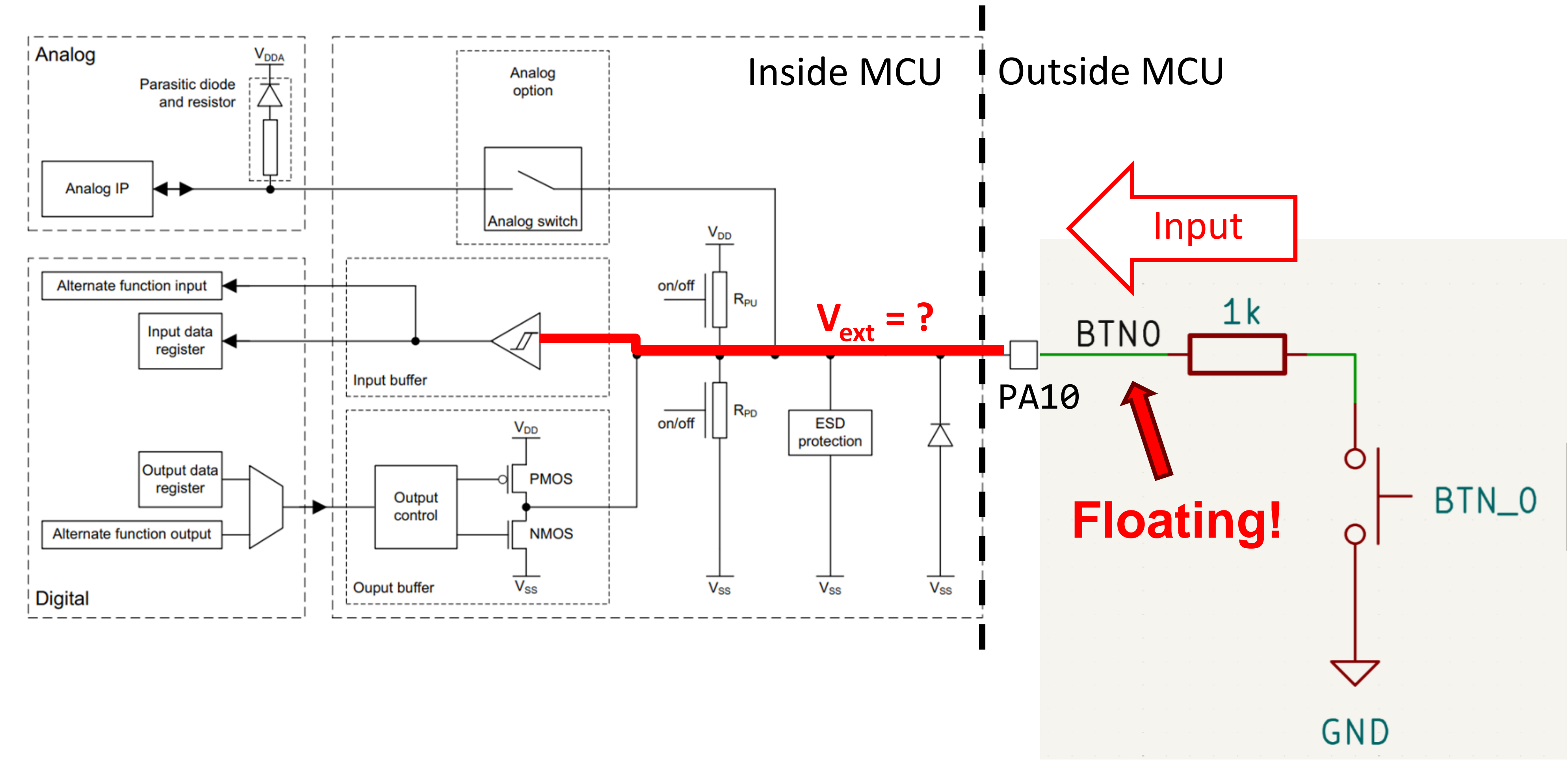
We must configure the Pin as Input

GPIO: Inputs Recap



When Button is pressed: V_{ext} = GND = 0b0

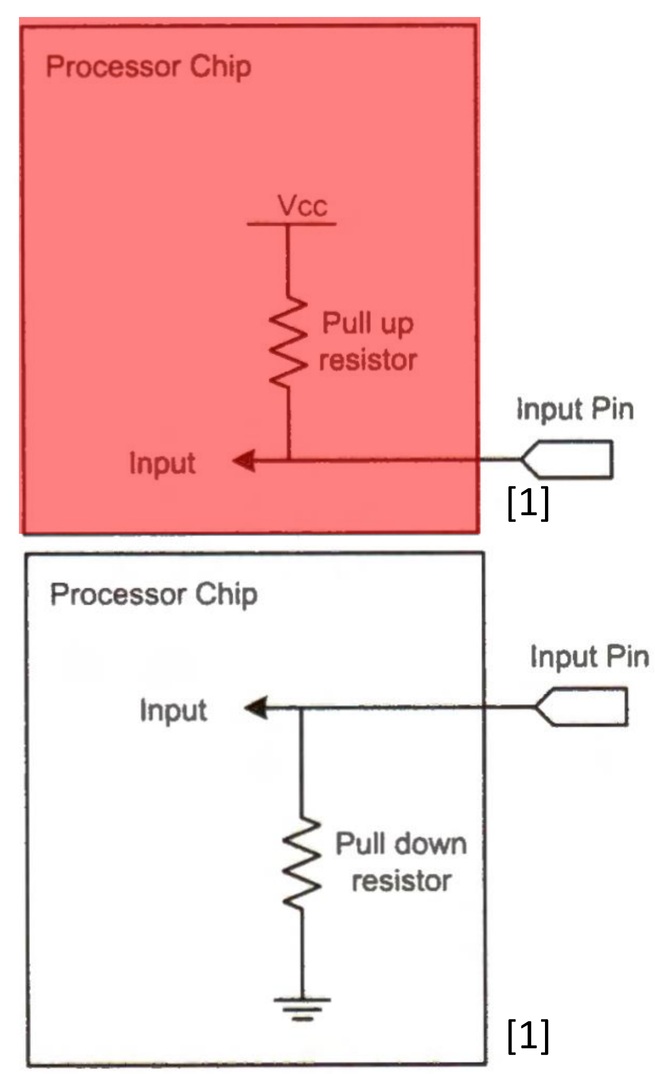
GPIO: Inputs Recap



When Button is not-pressed: V_{ext} = Floating

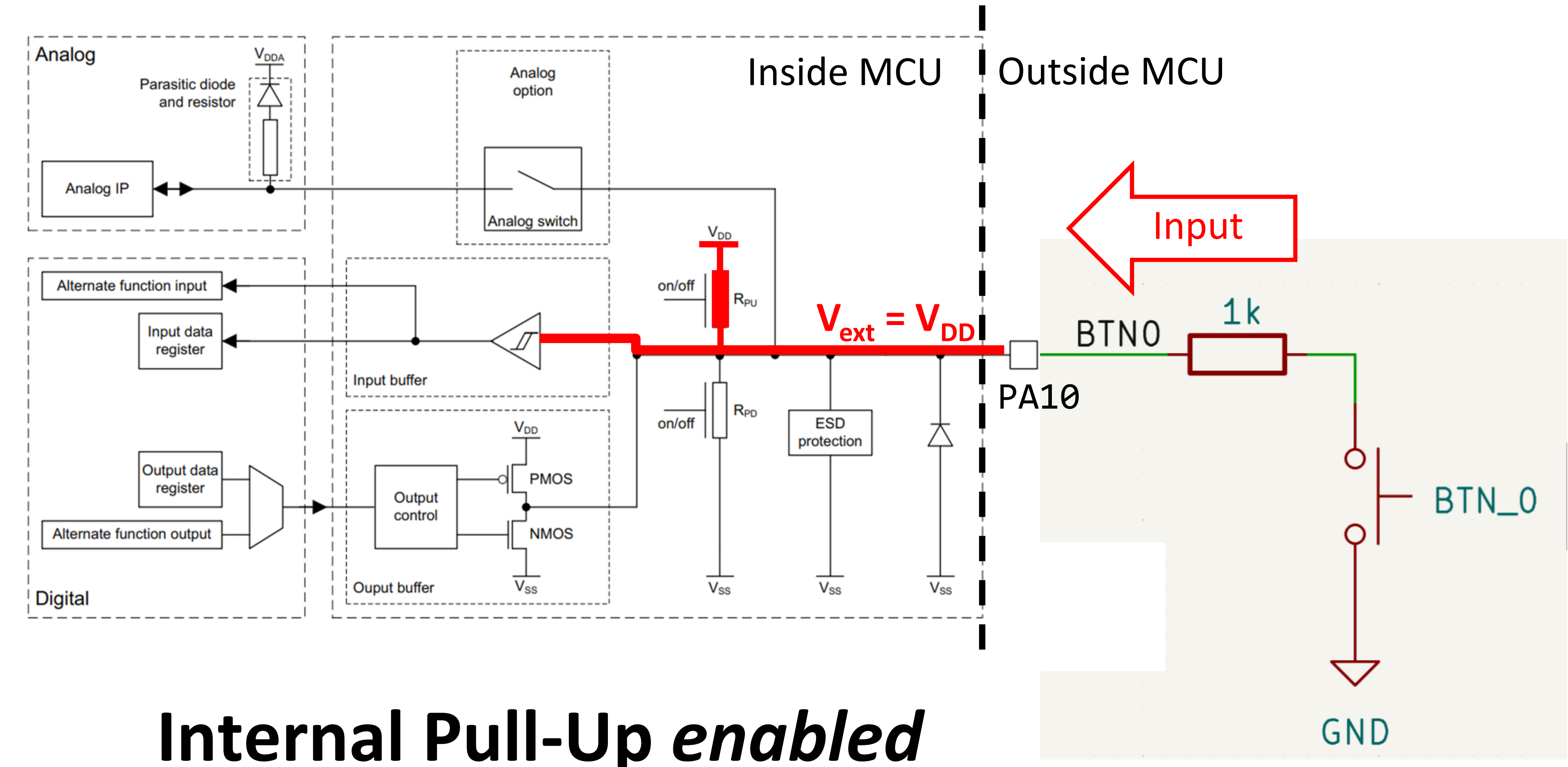
GPIO Operating Modes – Input Modes

- *Input with internal pull-up resistor*: steady-state value is «high», or «logic 1». Pin is connected to internal power supply via a resistor.
- *Input with internal pull-down resistor*: steady-state value is «low», or «logic 0». Pin is connected to ground via a resistor.



[1] *Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 356*

GPIO: Inputs Recap



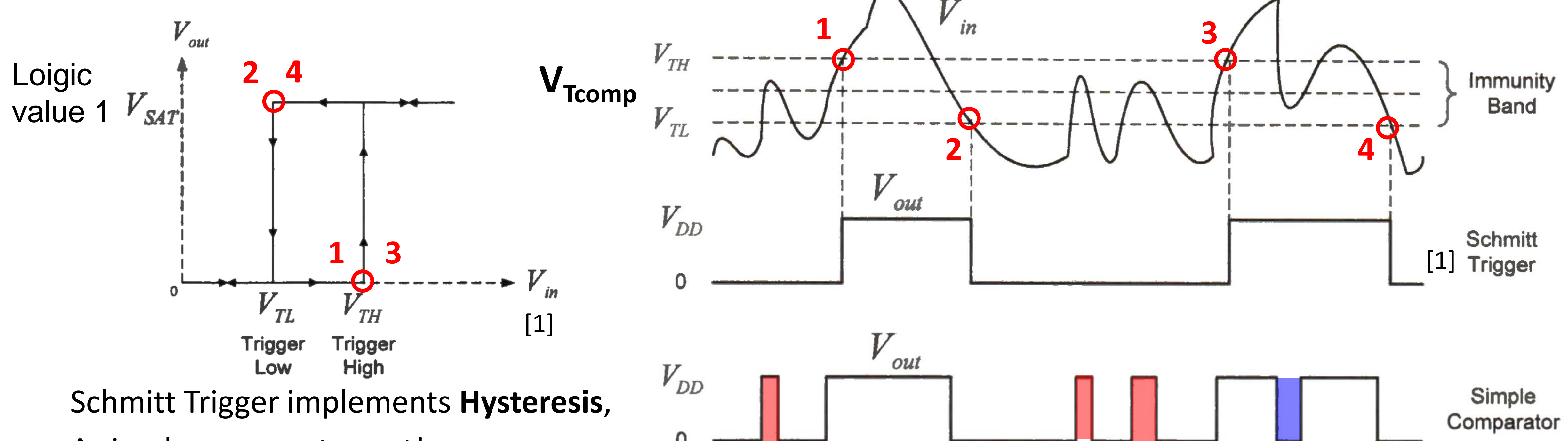
Internal Pull-Up *enabled*

When Button is not-pressed: $V_{ext} = V_{DD}$ = 0b1

GPIO: Inputs

- Even with pull-up/pull-down, voltage on the pin can be noisy.

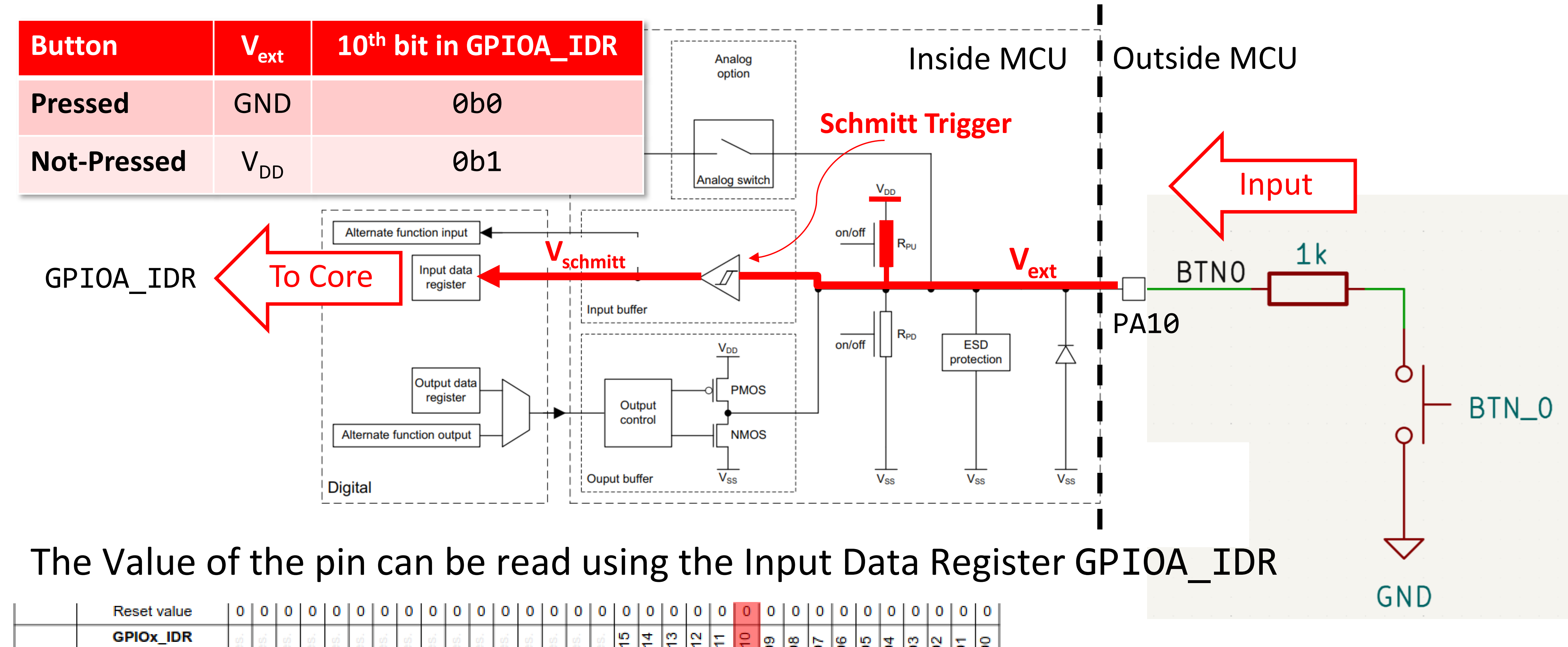
For digital inputs, each GPIO contain a *Schmitt Trigger* to convert noisy or slow signal edges into a clean edge.



Schmitt Trigger implements **Hysteresis**,
A simple comparator not!

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 359

GPIO: Inputs



Button	V_{ext}	10th bit in GPIOA_IDR
Pressed	GND	0b0
Not-Pressed	V_{DD}	0b1

The Value of the pin can be read using the Input Data Register GPIOA_IDR

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

GPIO: Polling

- *Polling*: Repeatedly checking the state of the `GPIOA_IDR` register in a loop
  - Easy to implement
  - CPU Intensive – wasted CPU cycles
  - Poor Scalability
  - Higher Power consumption
  - Higher Latency

Button	V_{ext}	10th bit in GPIOA_IDR
Pressed	GND	0b0
Not-Pressed	V_{DD}	0b1

```
while (1) {
    // Check if the 10th bit is set
    if(GPIOA_IDR & (1 << 10)){
        // 10th bit is 0b1
        button_not_pressed();
    } else {
        // 10th bit is 0b0
        button_pressed();
    }
}
```

GPIO: Polling

Can we do better?

Interrupts

‘Interrupt’ Defined

“An interrupt is a hardware-invoked function call” [1]

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 249

Waiting for an Event: Family Vacation

Polling

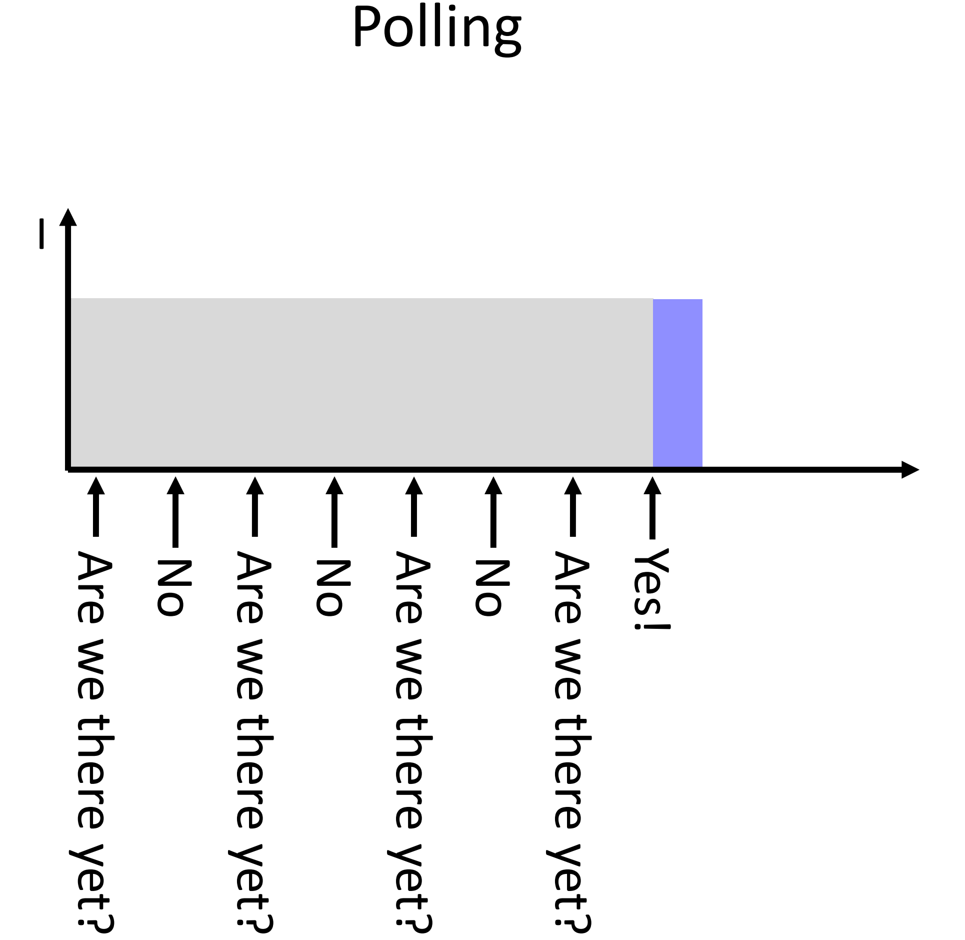
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?

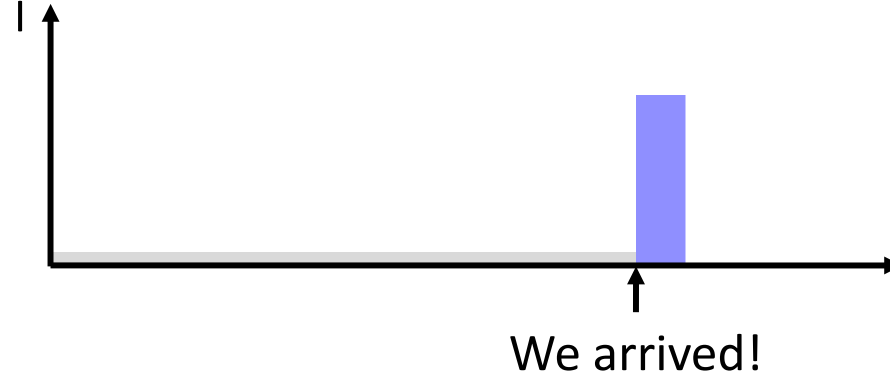
Interrupts

Wake me up when we get there...

An engineering example...

Waiting for an Event





Waiting for an Event: Button Push

Polling

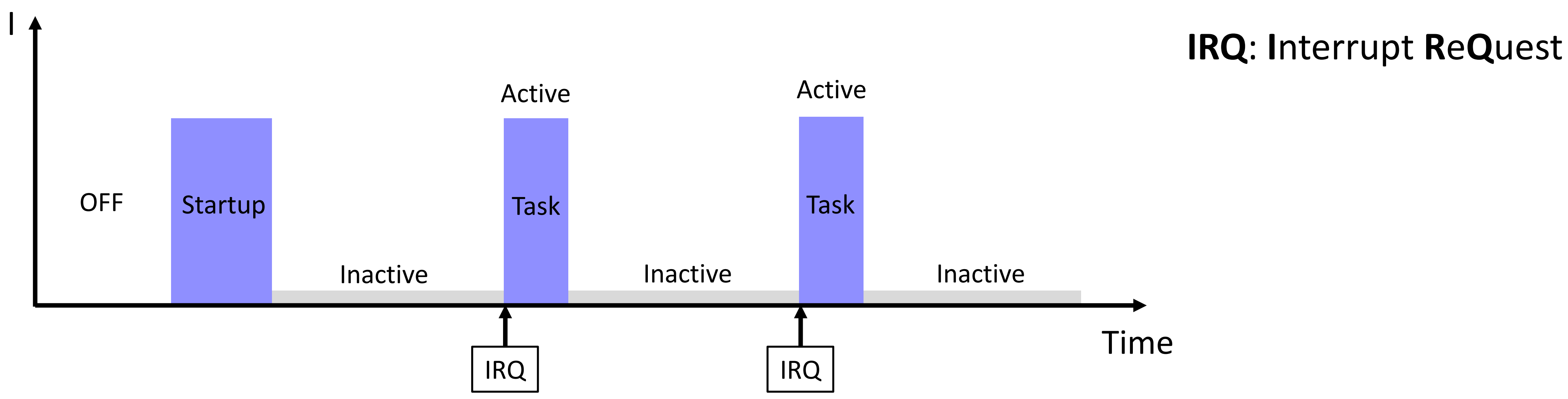
Interrupts

```c
while (1) {
    // Poll the pin state
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5) == 1) {
        // Pin is high, do something
    }
}
```

```c
// Interrupt service routine for GPIOA Pin 5
void EXTI9_5_IRQHandler(void){
  // Pin IRQ triggerd, do something
}
```

< 0.1% CPU Load

Typical Application Profile with Embedded Systems



IRQ: **I**nterrupt **R**e**Q**uest

- **OFF** – power is not applied to MCU
- **Startup Initialization**– MCU performs configuration (peripherals, clocks, …)
- **Inactive** – MCU is in low power mode to reduce power consumption
- **Active** – MCU is in normal mode and performs tasks

Interrupts

- *External Interrupts* are interrupts coming from a peripheral such as:
  - A button being pressed to start a function
  - Sensor informs that new data is ready to be read out
- *Internal Interrupts* are triggered from inside the Microcontroller
  - Illegal use of instructions (divide by zero, stack overflow,…), often called trap
  - A timer that waits to turn off a light automatically

Interrupts

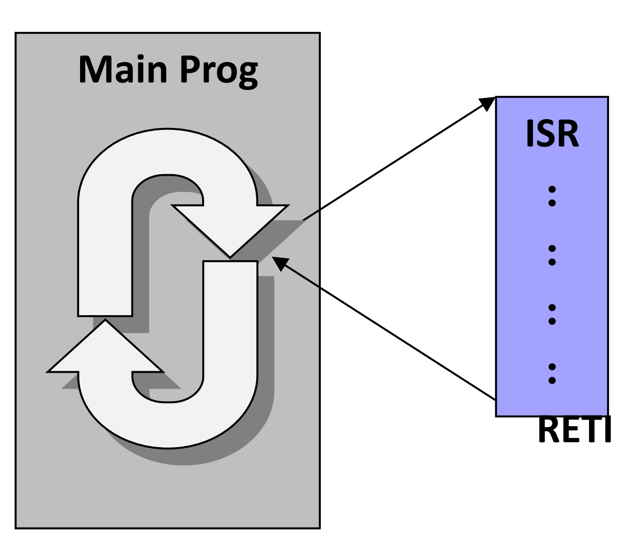
A way to respond to an external/internal event (i.e., flag being set) without polling

How it works:

- H/W senses flag being set
- Automatically transfers control to s/w that "services" the interrupt
- Code that handles interrupt is called
  - *Interrupt Service Routine (ISR)*
- When done, H/W returns control to wherever it left off

Advantages:

- Transparent to user
- Cleaner code
- MCU doesn't waste time polling
- Latency not dependent on polling period



Foreground / Background Scheduling

```
main() {
    //Init
    initGPIO();
    initClocks();
    registerISRs();
    ...

    while(1){
    background
    or
    Low Power Mode
    }
}

ISR1
  get data
  process

ISR2
  set a flag
```

System Initialization

- The beginning part of main() is usually dedicated to setting up your system

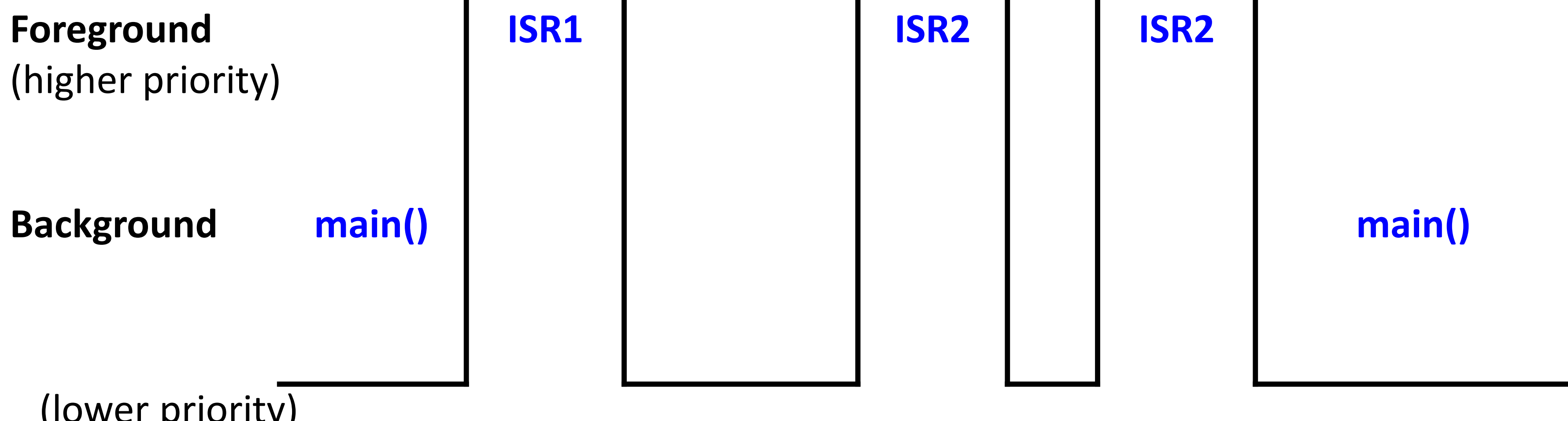
Background

- Most systems have an endless loop that runs ‘forever’ in the background
- In this case, **Background** implies that it runs at a lower priority than **Foreground**
- In microcontrollers, the background loop often contains a **Low Power Mode** command – this sleeps the CPU/System until an interrupt event wakes it up

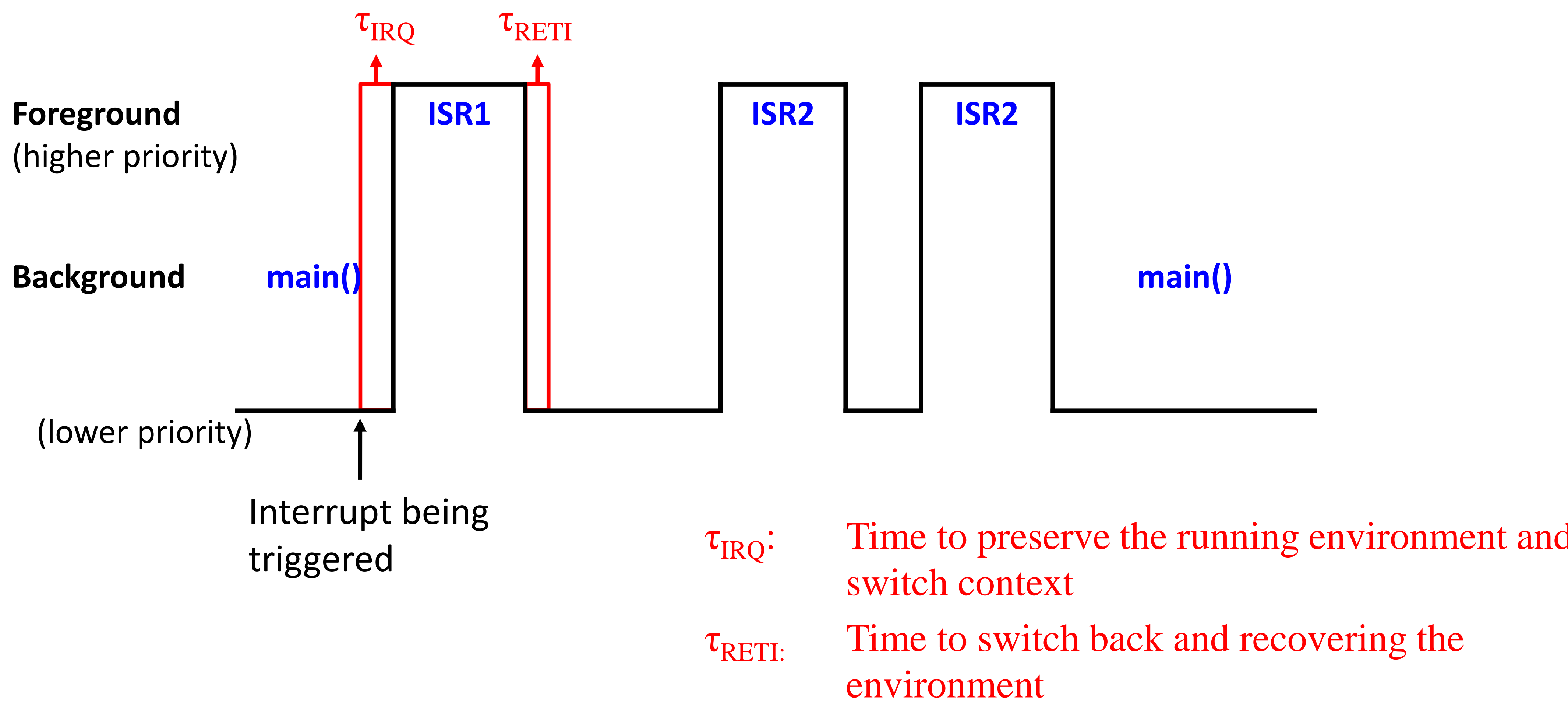
Foreground

- **Interrupt Service Routine** (ISR) runs in response to enabled hardware interrupt
- These events may change modes in Background – such as waking the CPU out of low-power mode
- ISR’s, by default, are not interruptible
- Some processing may be done in ISR, but it’s usually best to *keep them short*

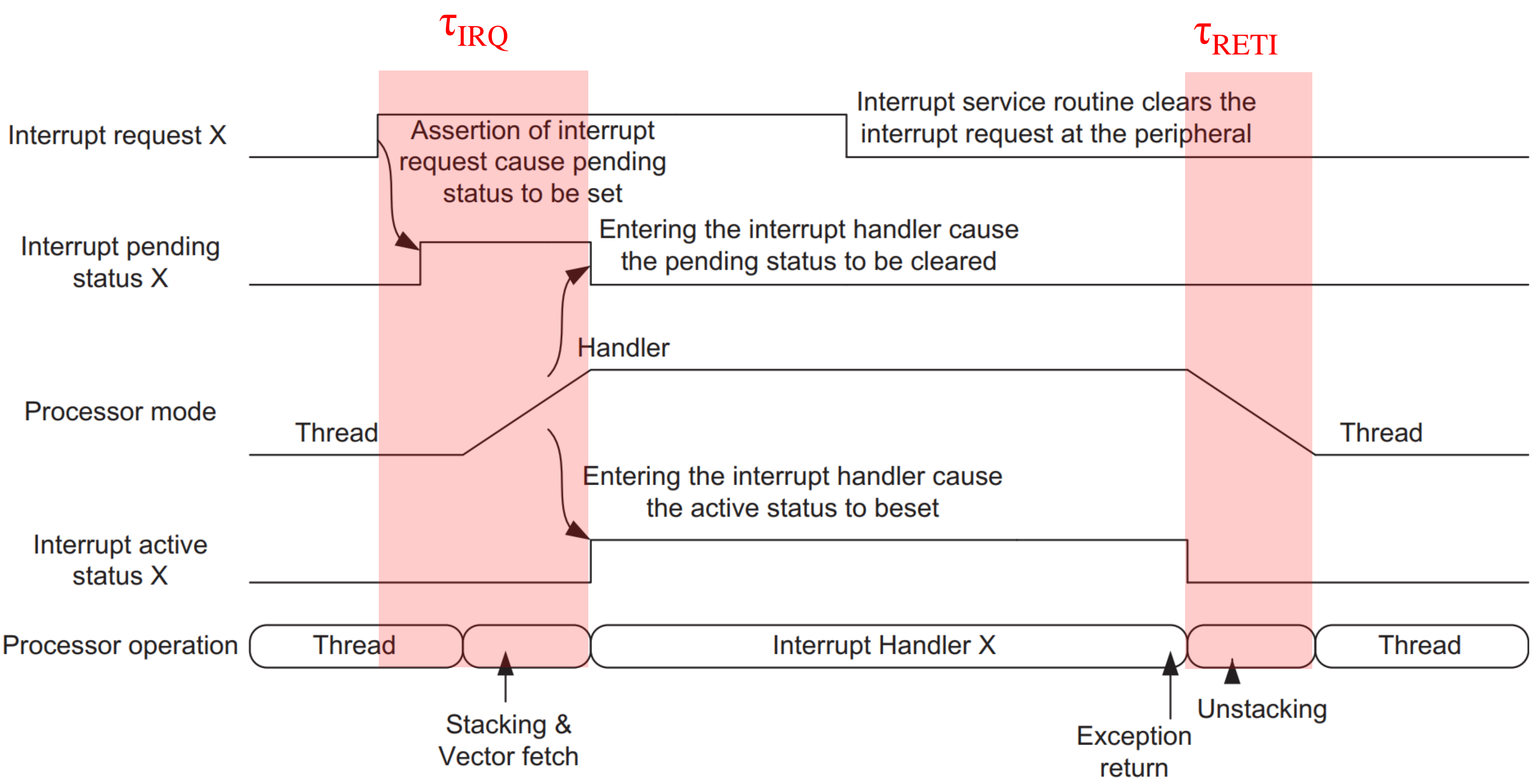
Foreground / Background (States)



Interrupt – Switching Context



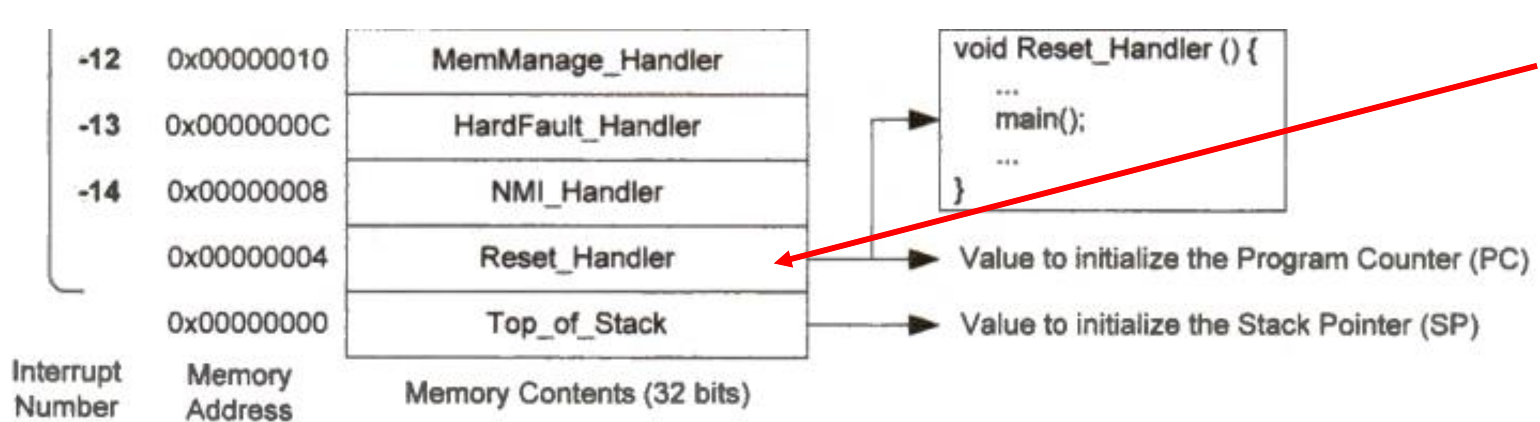
Interrupt – Switching Context



Interrupt Vector Table on ARM Cortex-M

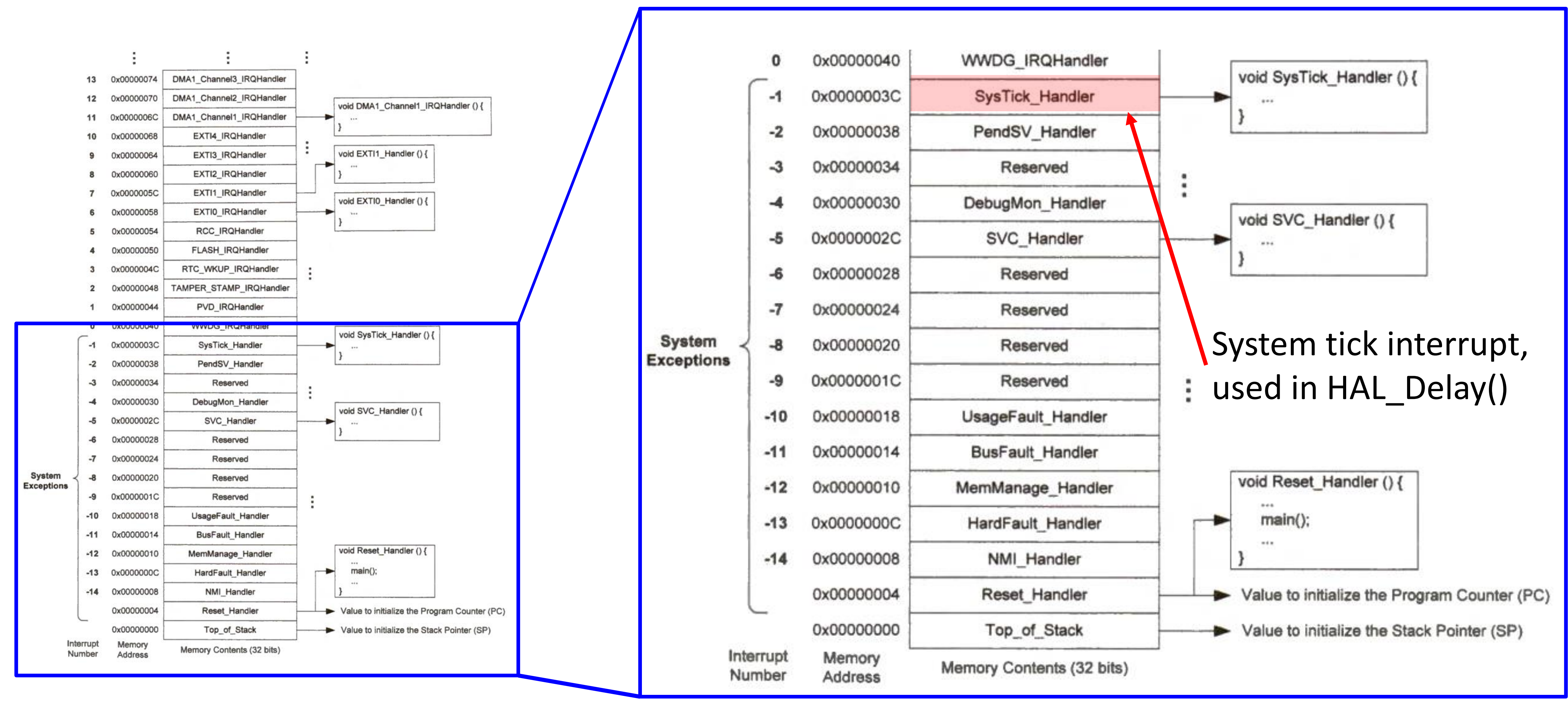
Each type of interrupt has one associated ISR at a predefined address. The Cortex-M processor *stores the memory addresses of all ISR* in the *Interrupt Vector Table.*

- The first 15 entries are system interrupts
- Therefore, table maps interrupt numbers N to the N+16th entry
- The table starts at the fixed memory address of 0x0000 0000
- Each entry stores one address of 32 bit

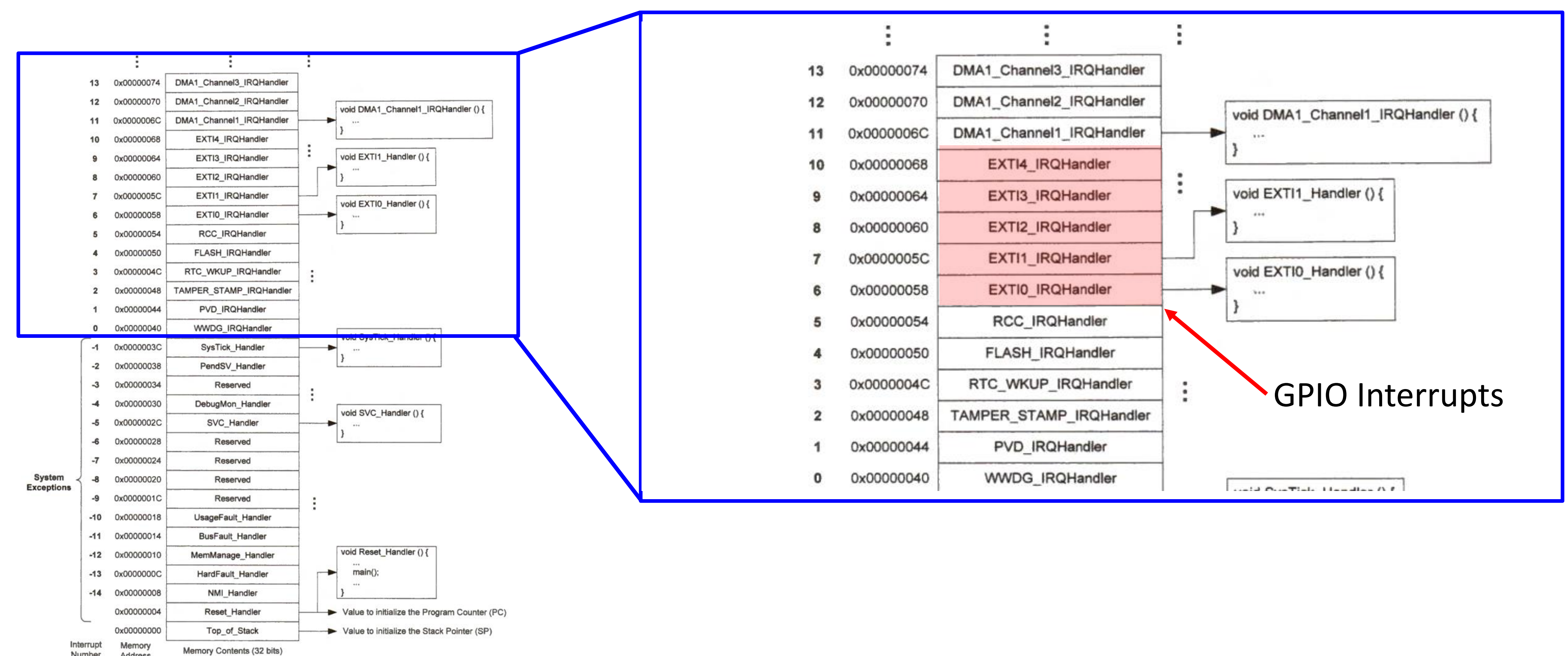


The address 0x00000004 contains the start address of the reset handler

Interrupt Vector Table – Example



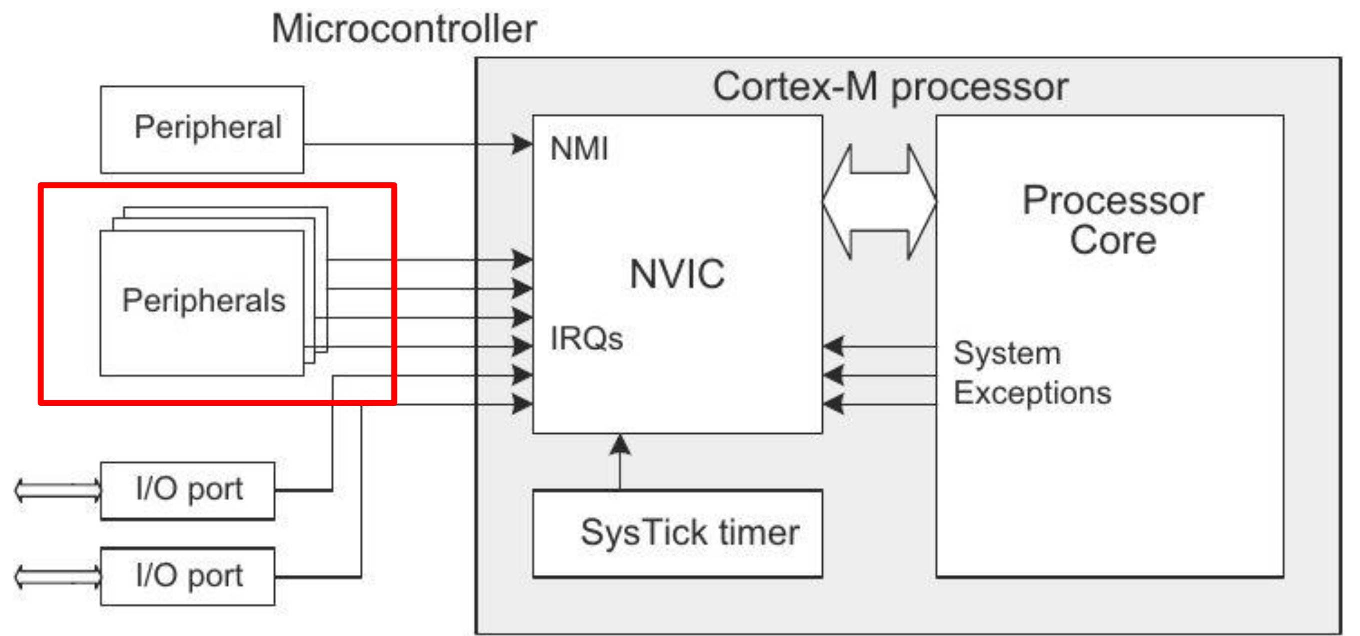
Interrupt Vector Table – Example



ARM Cortex-M – Nested Vector Interrupt Controller

The *Nested Vector Interrupt Controller (NVIC):*

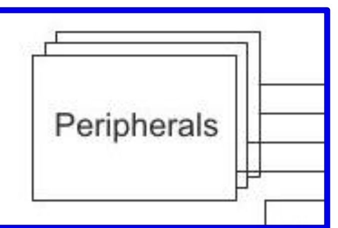
- **Manages** interrupts and their **priority levels**
- A large number of **maskable interrupt channels**
- Low-latency exception and interrupt handling (**12 Cycles, can change with architecture**)
- Power management control



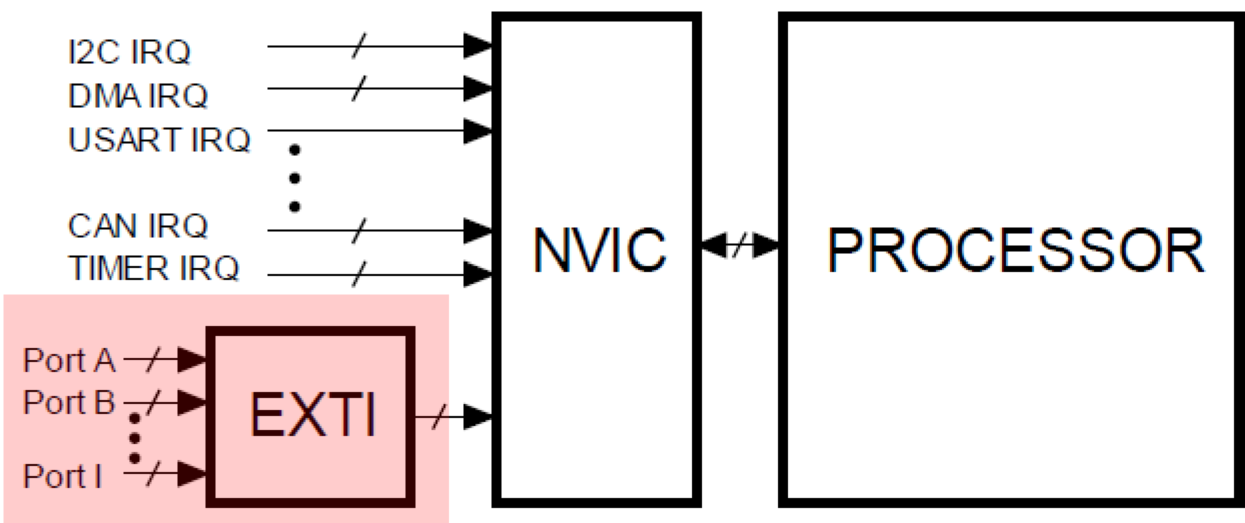
[1]

[1] The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors p.230

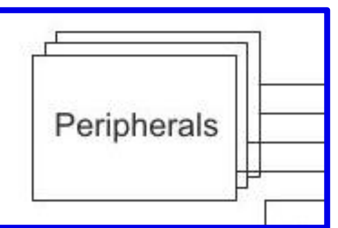
EXTI



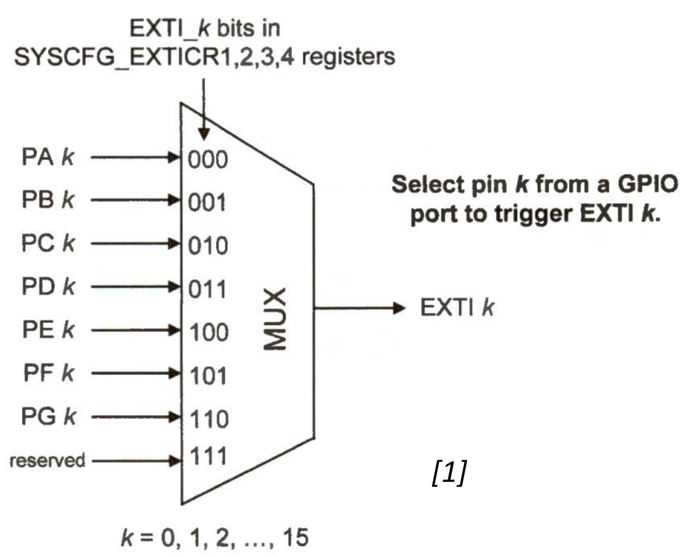
- Connects the external interrupts to the NVIC
- The EXTI (EXTernal Interrupt/Event) controller consists of up to 40 edge detectors for generating event/interrupt requests on STM32L47x/L48x devices.
- Each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising, falling, or both).



EXTI



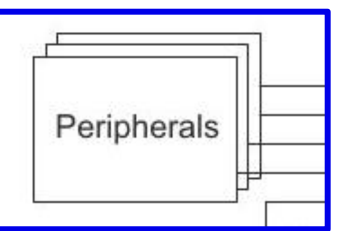
- 16 external interrupts, named EXTI0, EXTI1,..EXTI15, each with one edge detector associated to one GPIO pin
- MCUs has more than 16 GPIO pins, how does this work?
- All pins with the same pin number are connected on the same EXTI line (eg. Pin_2 Port A and Pin_2 Port C share the same EXTI2)



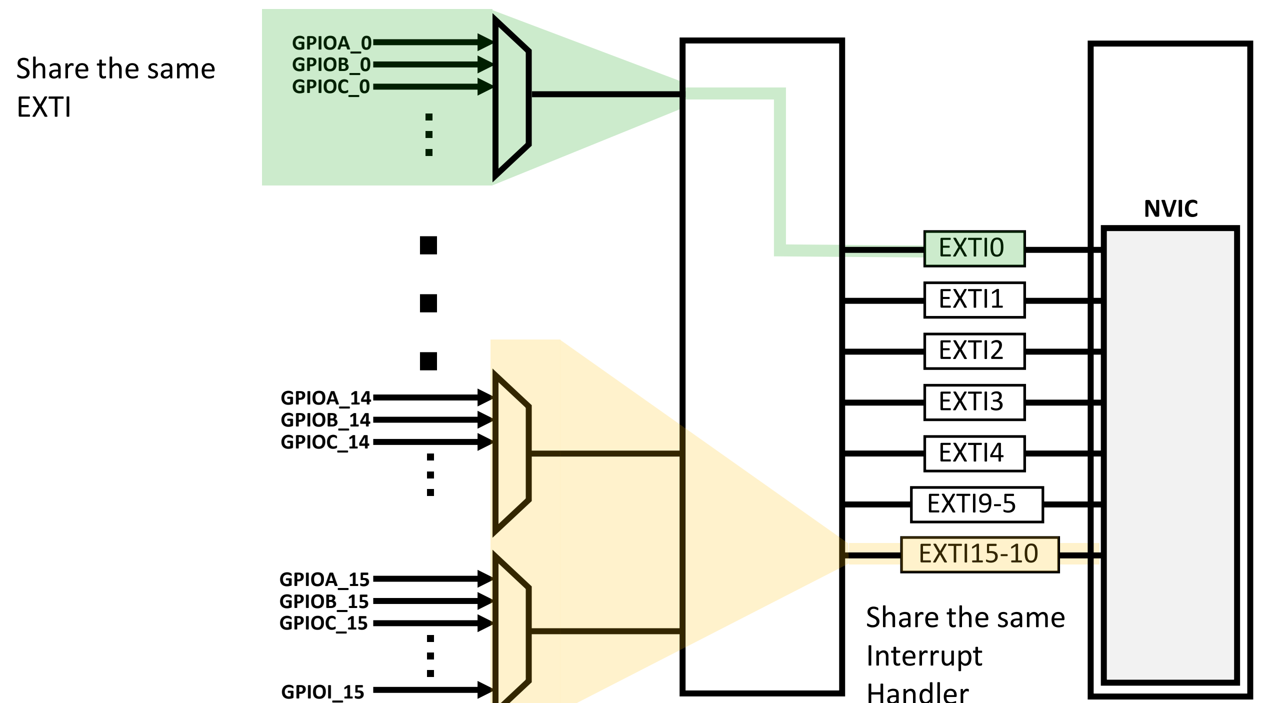
[1]

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 273

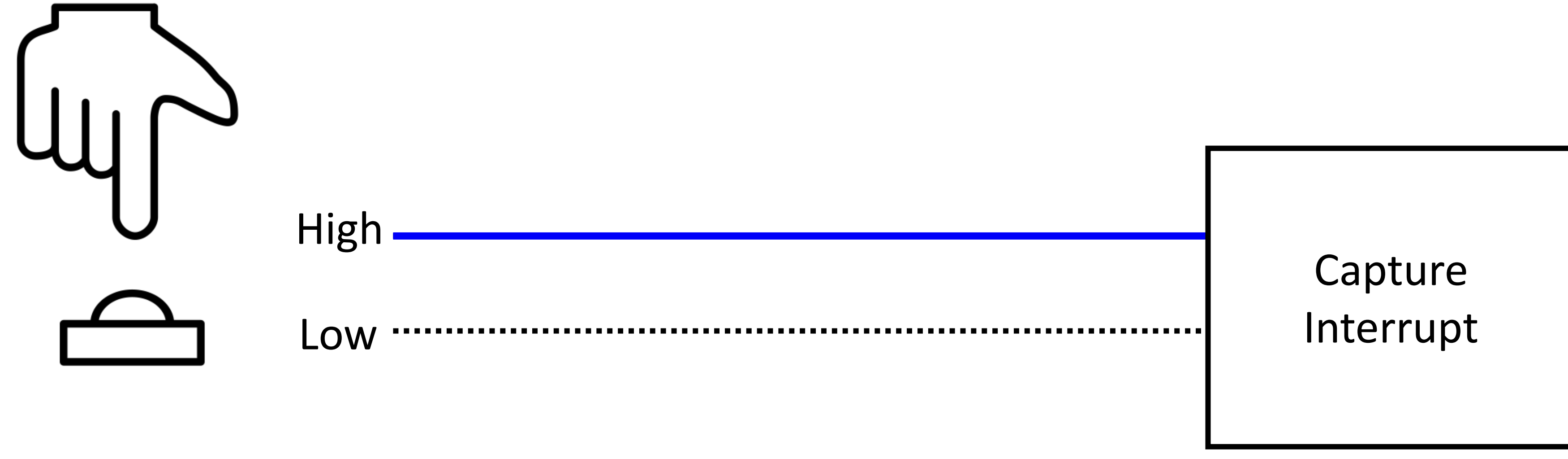
EXTI



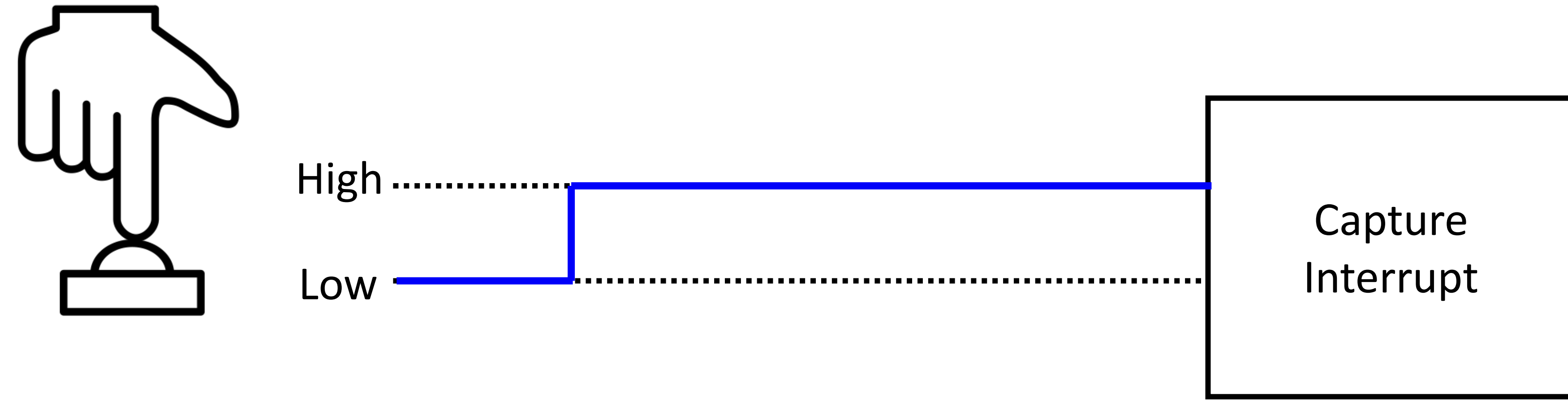
- The Interrupt handlers for each EXTI are defined in the Interrupt vector table
- Some EXTI may share the same interrupt handler
  - For STM32L4 EXTI10 to EXTI15 have the same interrupt number and therefore same handler
  - Interrupt handler can check which EXTI register triggers interrupt



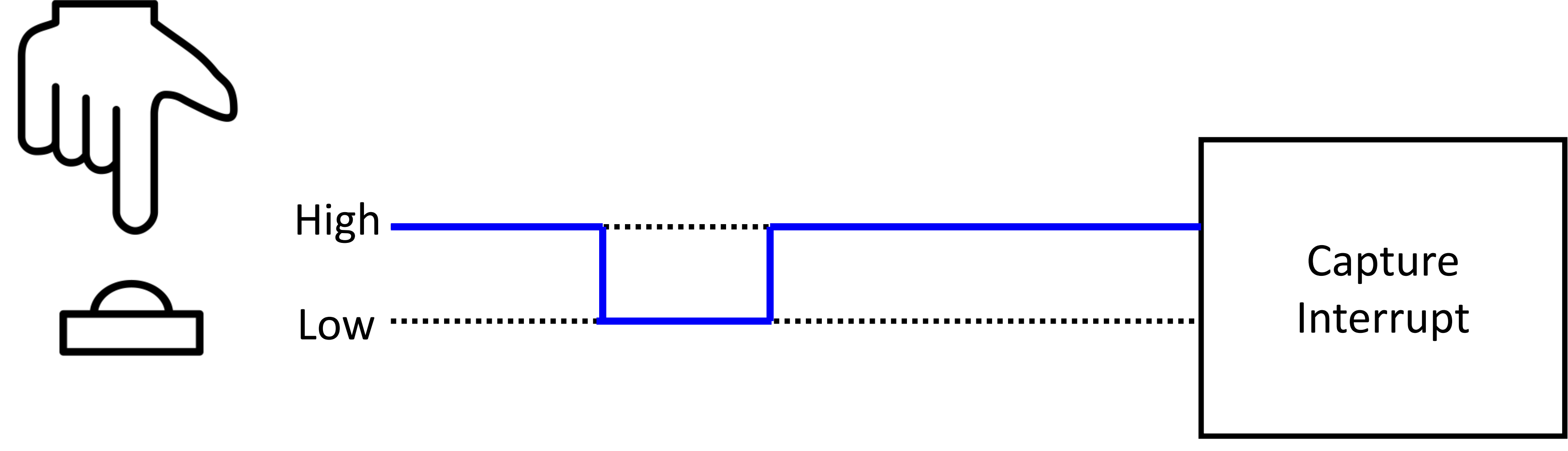
EXTI Module Example: From Pin to NVIC



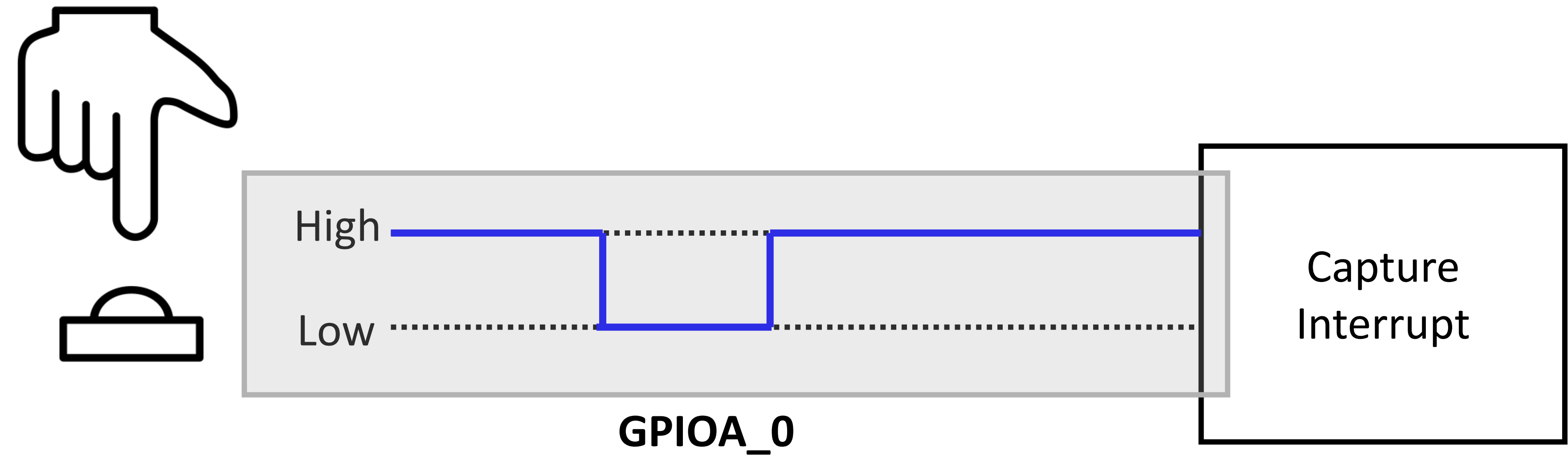
EXTI Module Example: From Pin to NVIC



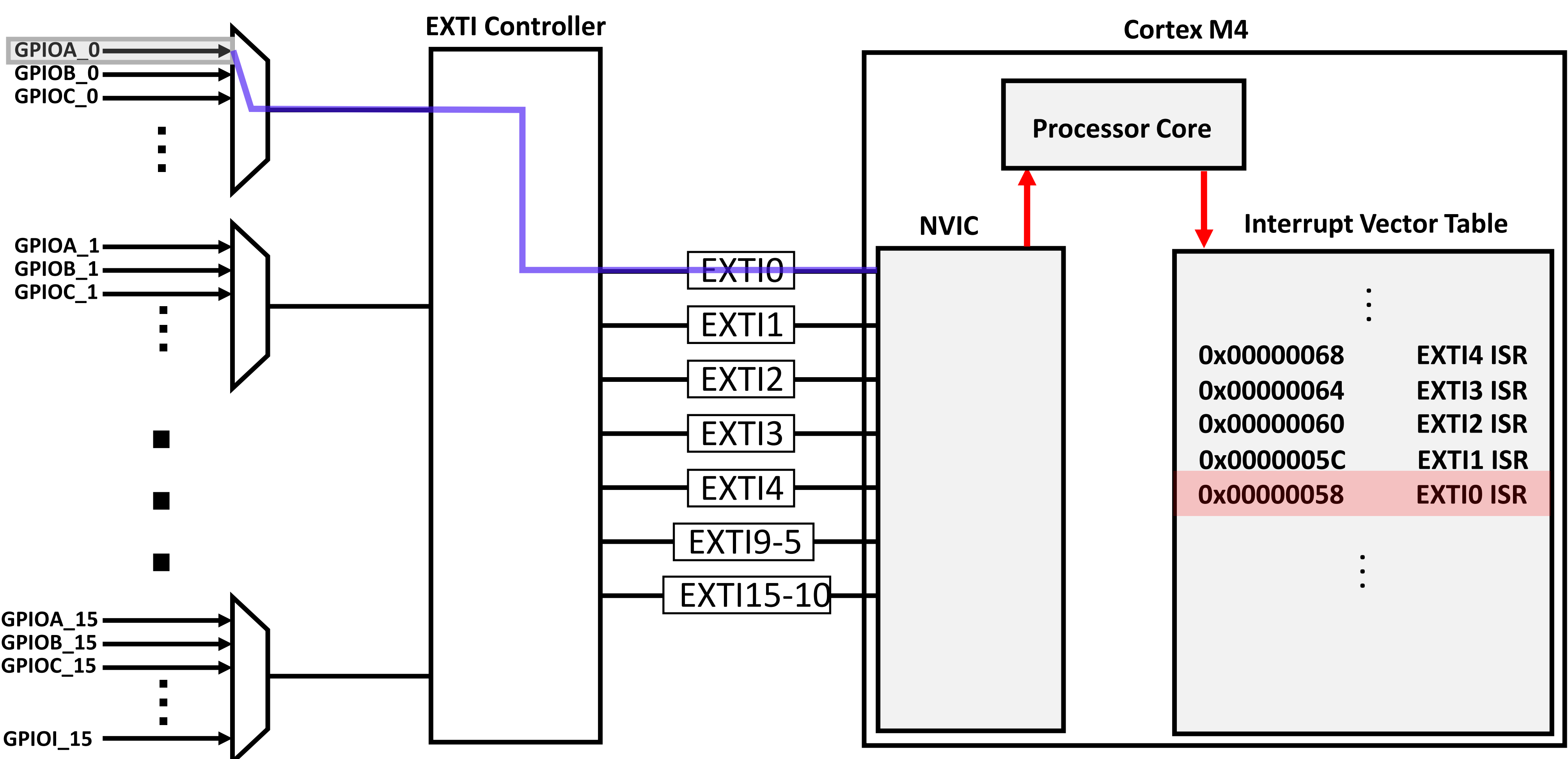
EXTI Module Example: From Pin to NVIC



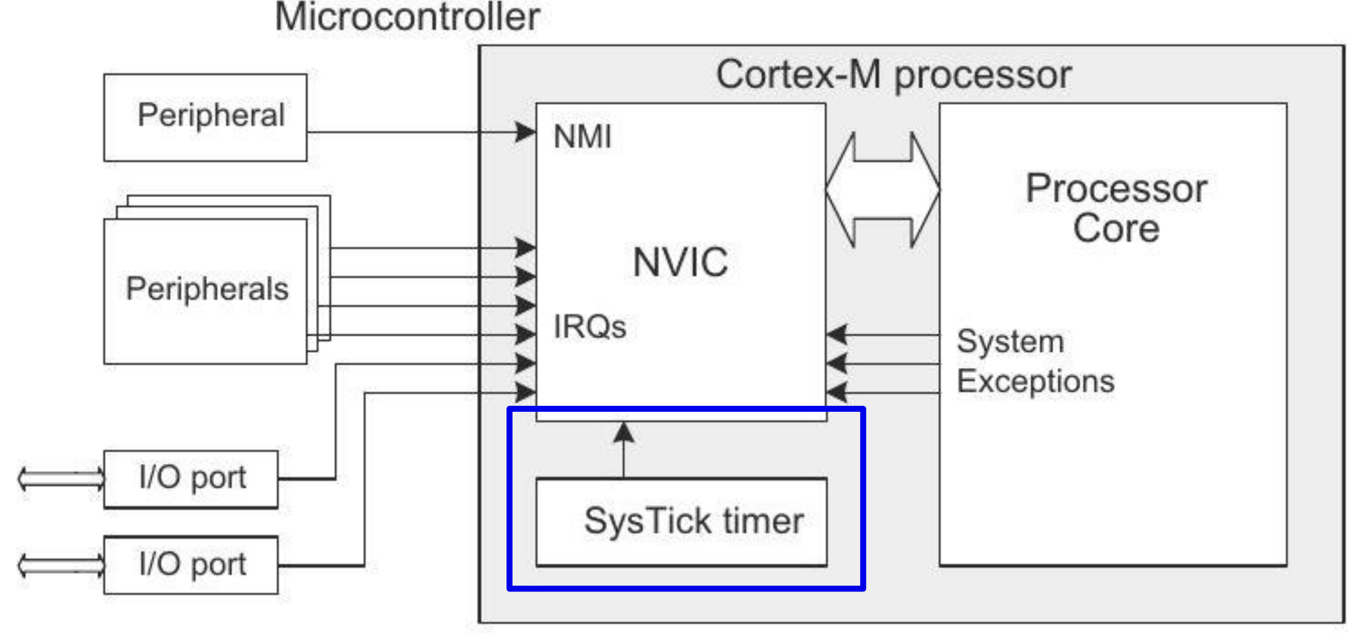
EXTI Module Example: From Pin to NVIC



EXTI Module Example: From Pin to NVIC

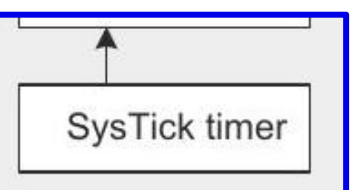


SysTick Timer



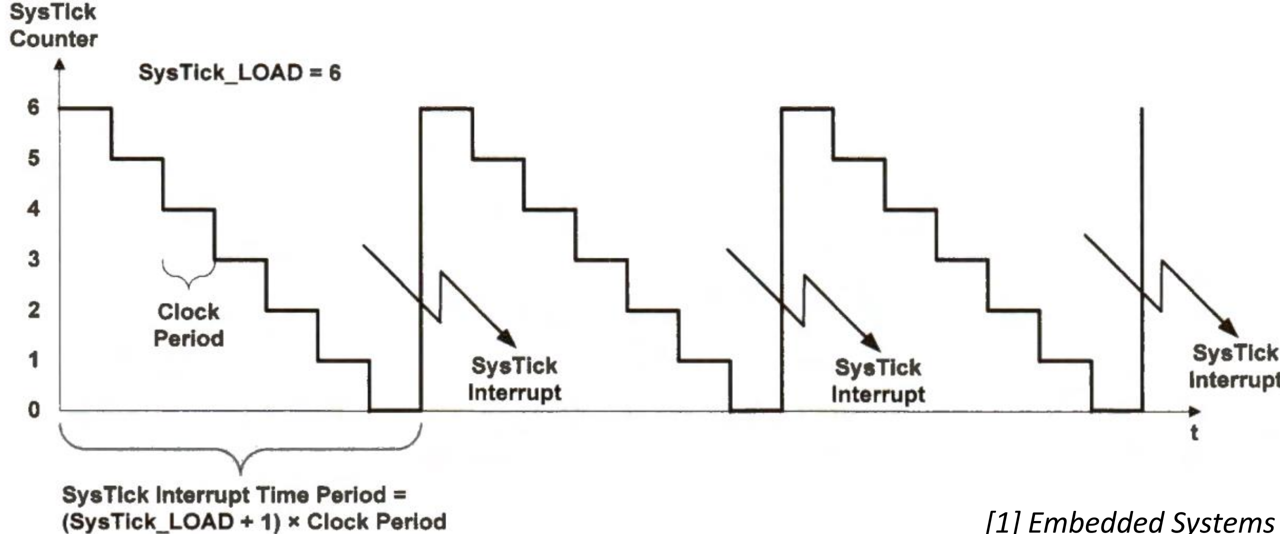
[1] The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors p.230

SysTick Timer



The *SysTick timer* is a simple 24-bit down-counting counter which can generate interrupts periodically at a configurable time interval.

- Timer counts from N-1 to 0 and generates an interrupt once it reaches 0
- Timer is not stopped when processor is halted. It still generates interrupts while debugging
- Used for example for HAL_Delay()

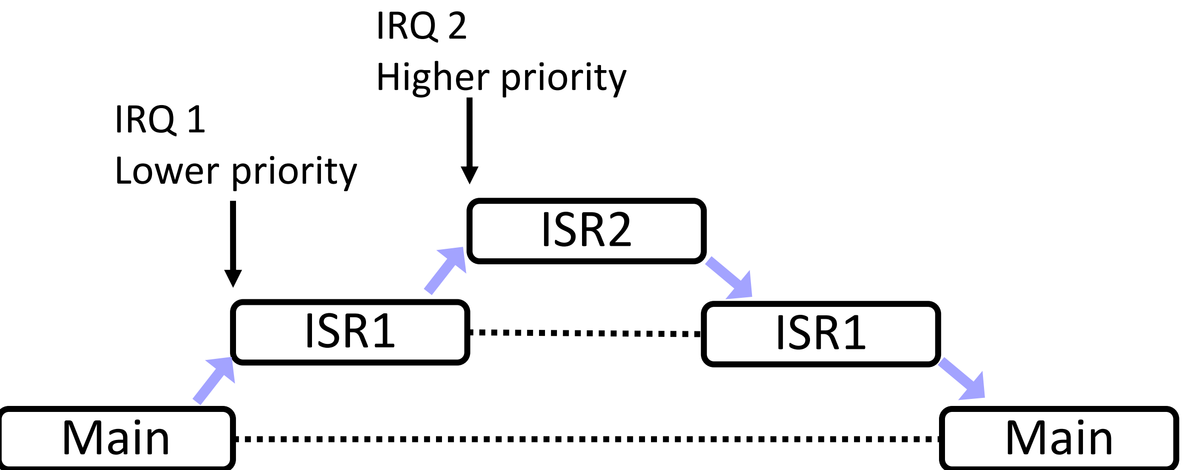


[1] *Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 268*

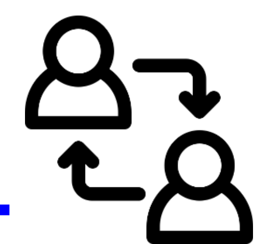
Interrupt Priorities

The *interrupt priority* determines the order in which interrupts are serviced. System interrupts have usually higher priority than interrupts from peripherals

- Priority Byte is split up into
  - Preemption Priority: determines if other ISR can be preempted
  - Sub-Priority Number: determines the order if multiple pending interrupts have same preemptive priority. Not used for preemption



Interaction: Interrupts

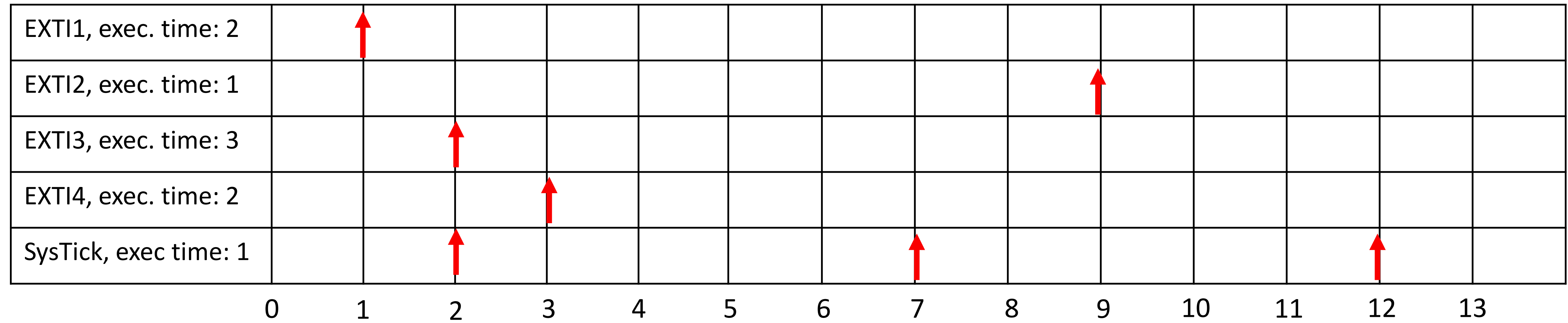


Draw the timeline for these tasks by following their interrupt priorities (Preemptive Priority and Sub-priority Number). Each task has it’s own execution time, which is defined below.

Hint: Lower number leads to a higher priority!

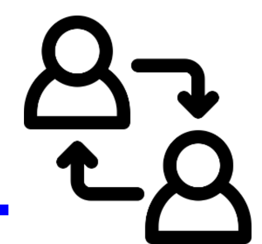
- *Preemption Priority*: determines if other ISR can be preempted
- *Sub-Priority Number*: determines the order if multiple pending interrupts have same preemptive priority. Does not lead to a preemption!

Name	Preemptive Priority	Sub-Priority Number
EXTI1_IRQHandler	2	3
EXTI2_IRQHandler	2	1
EXTI3_IRQHandler	2	2
EXTI4_IRQHandler	2	1
SysTick_Handler	1	2



Icon by Freepik

Interaction: Interrupts

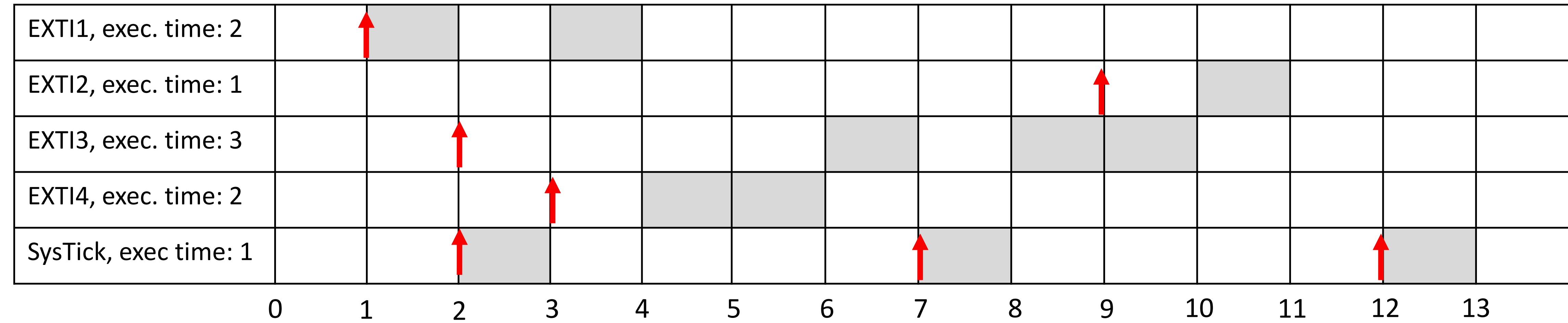


Draw the timeline for these tasks by following their interrupt priorities (Preemptive Priority and Sub-priority Number). Each task has it's own execution time, which is defined below.

Hint: Lower number leads to a higher priority!

- *Preemption Priority*: determines if other ISR can be preempted
- *Sub-Priority Number*: determines the order if multiple pending interrupts have same preemptive priority. Does not lead to a preemption!

Name	Preemptive Priority	Sub-Priority Number
EXTI1_IRQHandler	2	3
EXTI2_IRQHandler	2	1
EXTI3_IRQHandler	2	2
EXTI4_IRQHandler	2	1
SysTick_Handler	1	2



Icon by Freepik

Issues with Interrupts

Let’s increment a counter in the ISR and press the button:

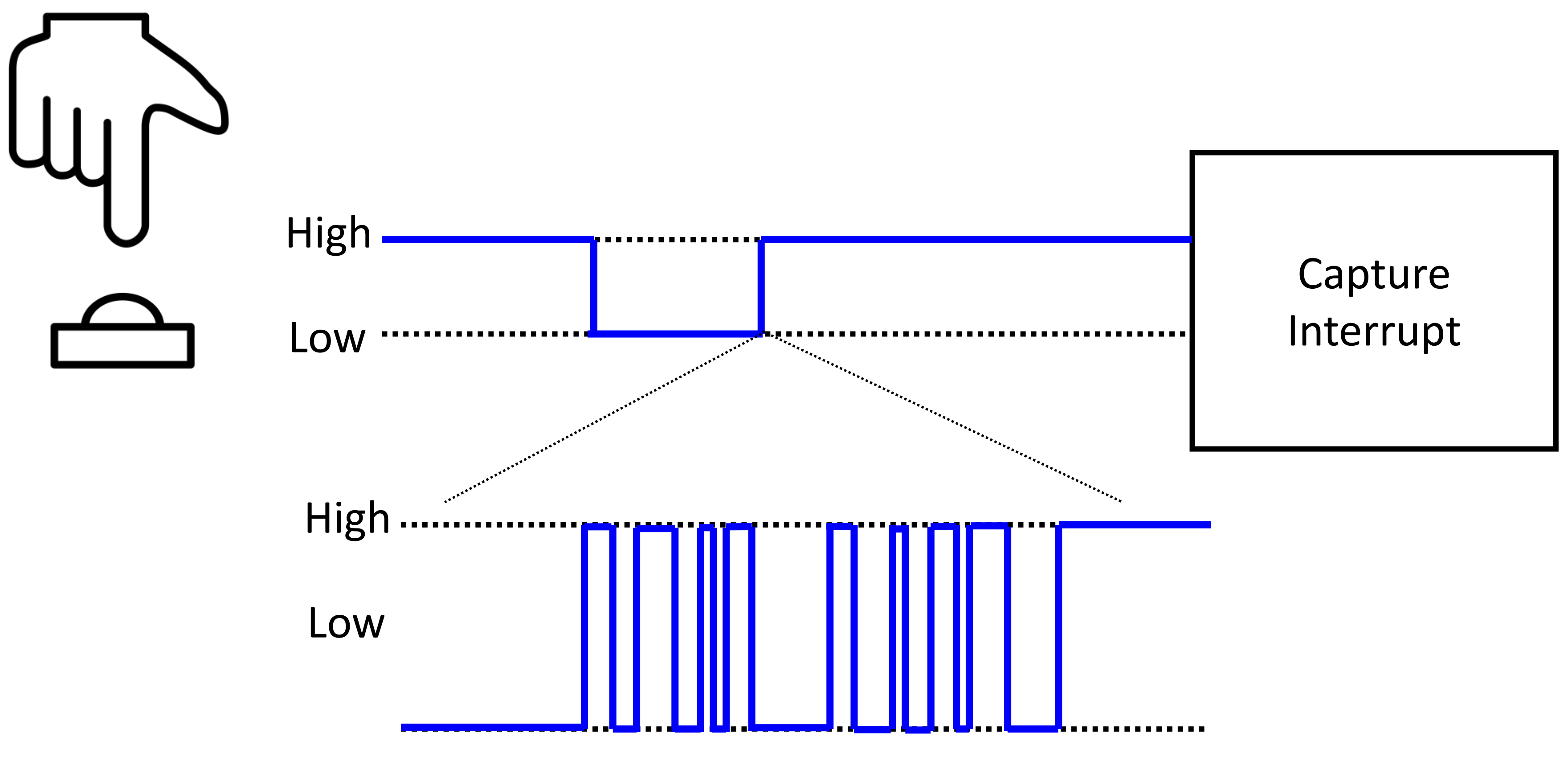
```
volatile uint64_t buttonPressCount = 0;

// Interrupt service routine for Button
void buttonISR(void){
  buttonPressCount++;
}
```

After pressing the button once, `buttonPressCount == 102`!

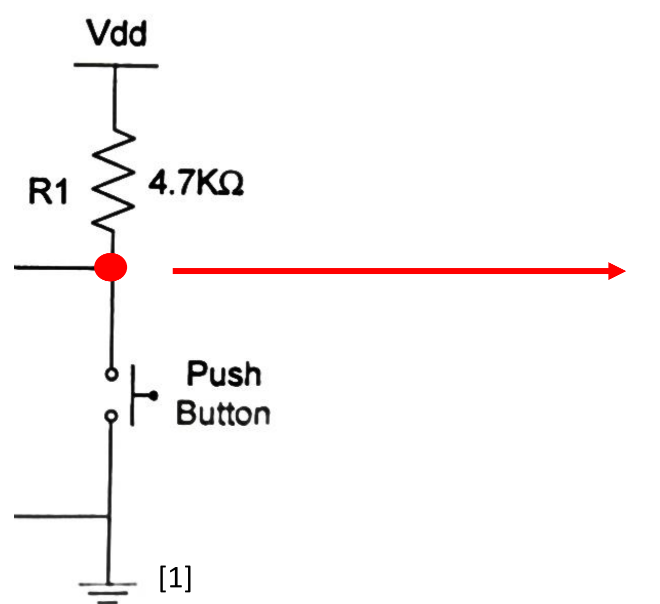
Issues with Interrupts

- Microcontrollers are Fast!

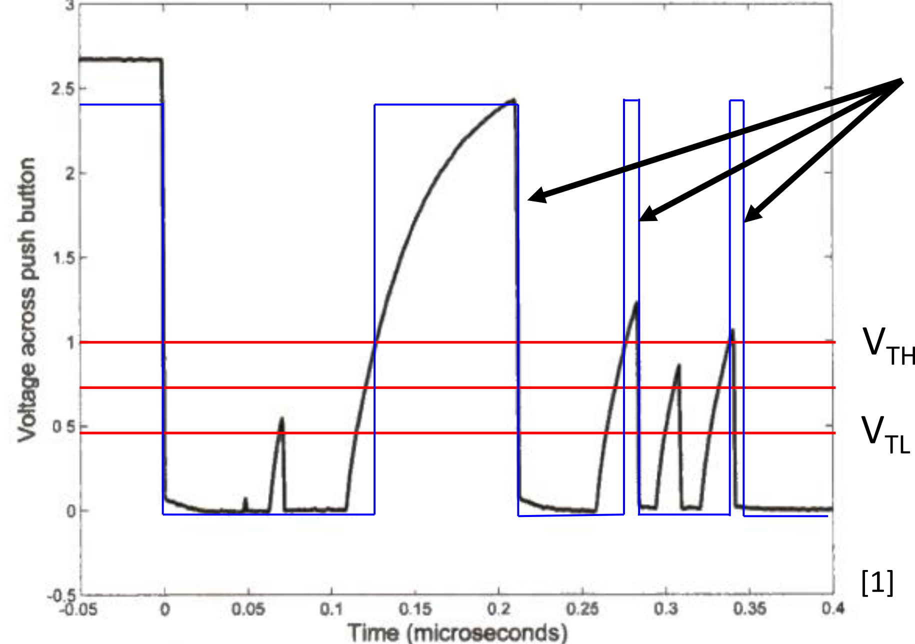


Button without Debouncing

Buttons are hardware components, and their mechanic has the *tendency to bounce*.
Buttons need debouncing to prevent multiple action detection.



Button without debouncing



Schmitt-Trigger not sufficient, several events detected!

V_{TH}

V_{TL}

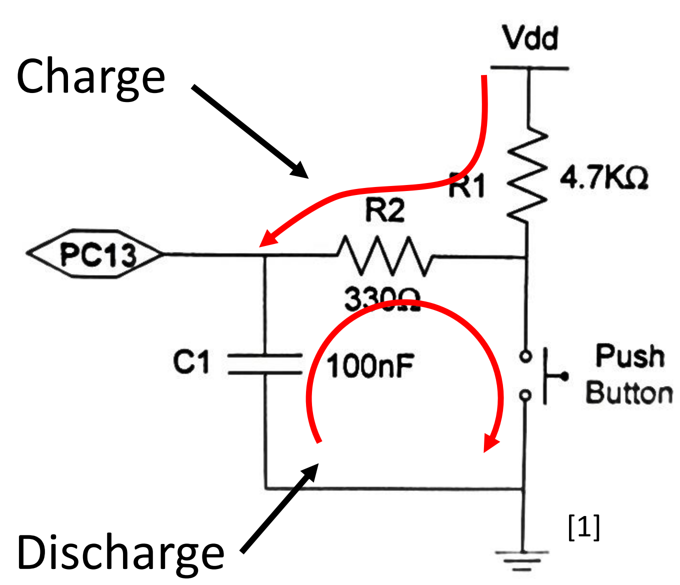
When pressing the button the two metal contacts rebound, and multiple button presses may be measured

Either hardware or software debouncing needed

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 362

Button With Debouncing

Hardware Debouncing filter signal spikes using an RC filter



Button with hardware debouncing

Default state:
- Switch open
- C1 fully charged to Vdd
- PC13 high

Pressed:
- C1 discharges through small R2
- If rebound: C1 starts recharging slowly through both resistors

Release:
- C1 charges through R1 and R2
- C1 returns to fully charged default state

Fast discharge and slow recharge eliminates bounces!

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 371

Clocks

Clocks

Synchronous circuits with logic gates are formed by transistors with *limited speed*

- There are times in which the gate have an undefined state
- We need a clock to only do operations when logic gates have a defined state

A microcontroller has *different clocks for various purposes*:

- CPU speed
- RTC (Real time clock) to keep track of calendar time/date
- Communication rate with another device
- Sampling of an analog signal

Different clock speeds can be derived from the internal and external clock sources of the microcontroller-

Clock Accuracy and Jitter

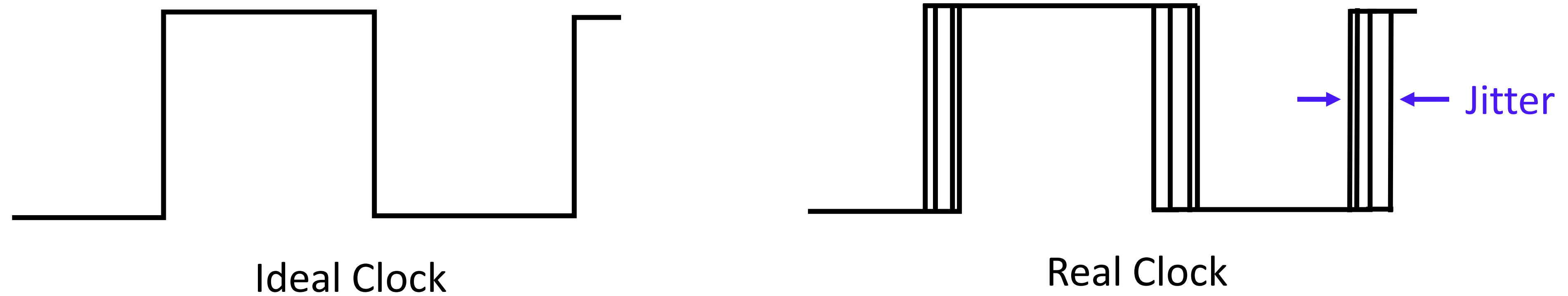
The *clock accuracy* defines the drift of the clock

- Clock is dependent on temperature
- RTC accuracy defined in parts per million (PPM)

$$PPM = \frac{ActualFrequency - \text{Specified}Frequency}{\text{Specified}Frequency} * 10^6$$

$$\text{Accuaracy } [\%] = \frac{PPM}{10000}$$

The *clock jitter* are short term phase variation from the theoretical position in time



Clocks on STM32L4

Three internal clock sources:

- High-speed 16 MHz RC oscillator (HSI16)
- Multi-speed RC oscillator (MSI)
- Low-speed 32 kHz RC oscillator (LSI)

Two external oscillators

- High speed 4-48 MHz oscillator (HSE)
- Low speed 32.768 kHz oscillator (LSE)

Three Phase-Locked Loops (PLL) to multiply internal clock frequencies, where N and M are integers given by the PLL's hardware structure:

Clocks on STM32L4: Accuracy

Three internal clock sources:

- High-speed 16 MHz RC oscillator (HSI16)
- Multi-speed RC oscillator (MSI)
- Low-speed 32 kHz RC oscillator (LSI)

	MSI (100 kHz to 48 MHz)		HSI16 (16 MHz)
	MSI mode (w/o LSE)	PLL mode (w/ 32.768 kHz LSE)	
Accuracy (typ.)	Over [0 - 85 C]: +/- 1.55 % Over [1.62 - 3.6 V]: +0.8/-4.5 %	Average accuracy = LSE accuracy Jitter < 0.25%	Over [0-85 C]: +/- 0.8 % Over [1.62 - 3.6 V]: +0.1/-0.2 %

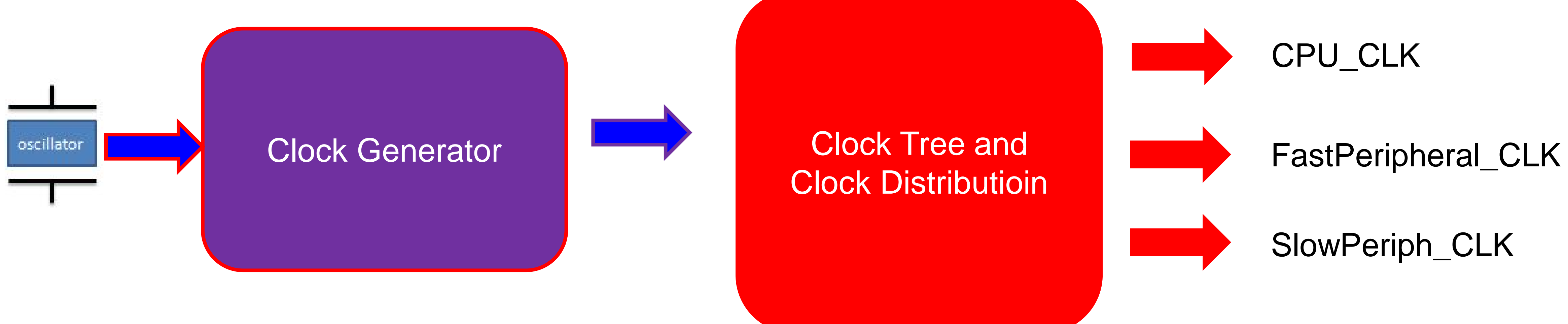
Clock Sources on The NUCLEO-L476RG

STM32L4 with internal RC clocks

External Crystal (LSE)

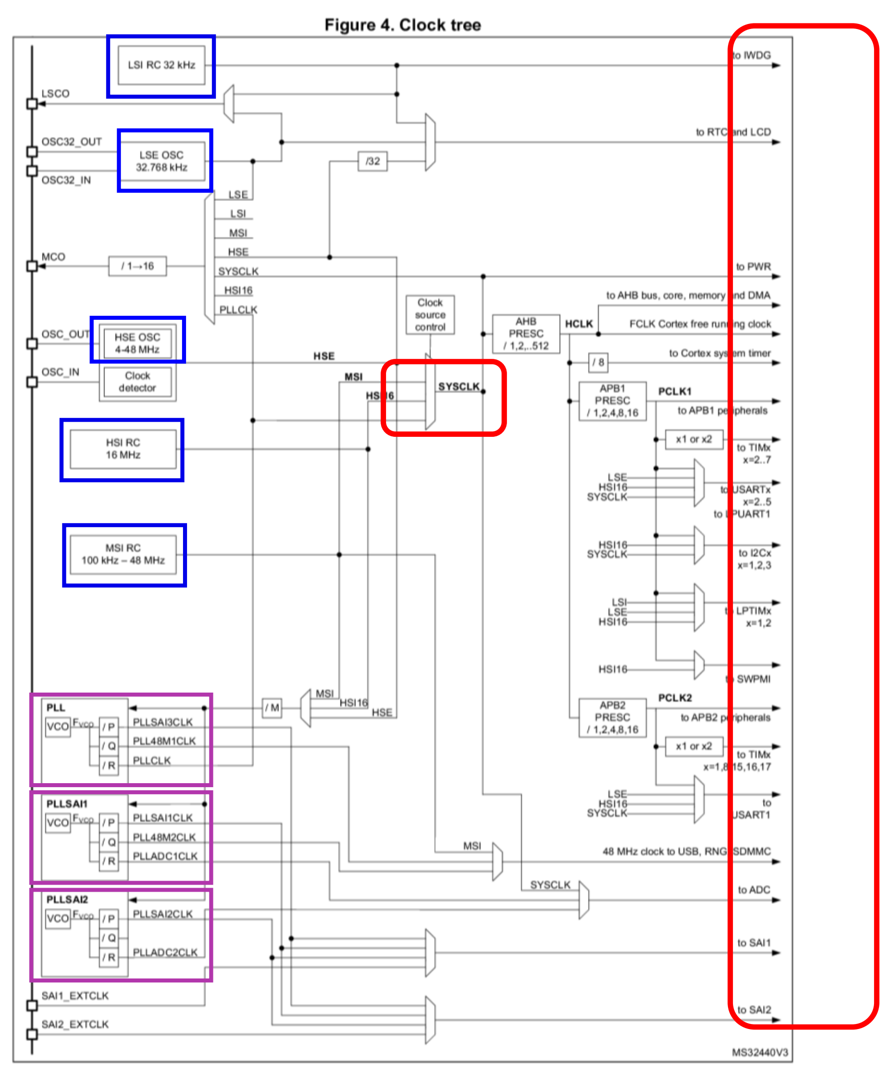
Optional second external crystal

Oscillator VS Clocks to subsystems



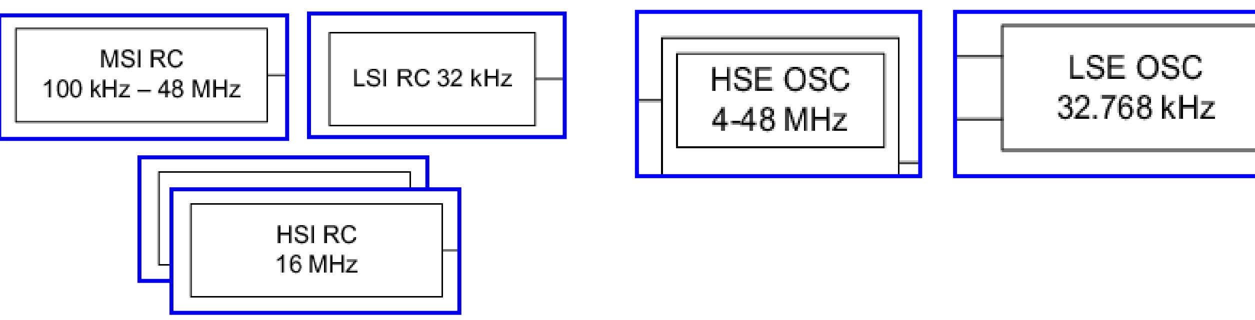
Oscillator: Provides the initial stable clock signal; can be internal or external (crystal, RC oscillator, etc.).
Clock Generator (PLL/FLL): Multiplies or divides the base frequency to generate various clock signals for different system components.
Clock Distribution: Delivers clock signals to different parts of the embedded system (e.g., CPU, peripherals, memory).

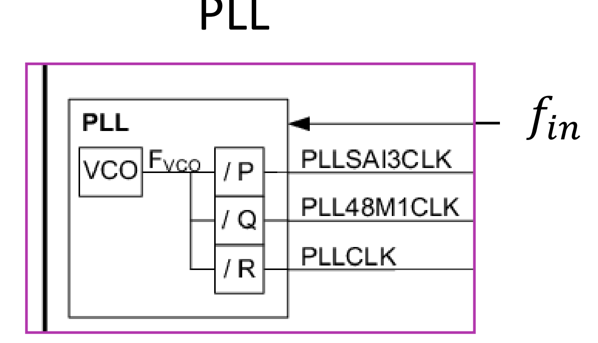
STM32L4 Clock Tree



Internal Oscillators

External Oscillators



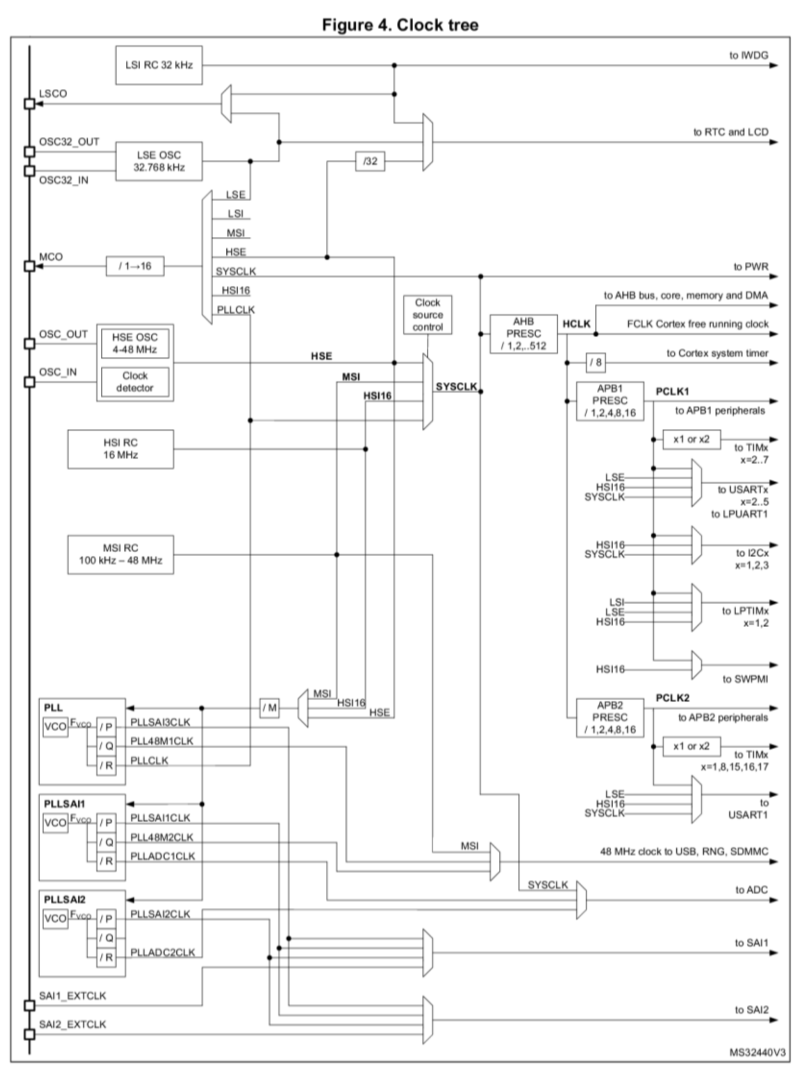


f_{in} Usually comes from Oscillators

Limit Voltage-Controlled Oscillator (VCO) Range: Design the VCO to operate within a limited, predefined frequency range, which simplifies frequency stability requirements.

where N, P, Q, and R are integers given by the PLL's hardware structure:

STM32L4 Clock Tree

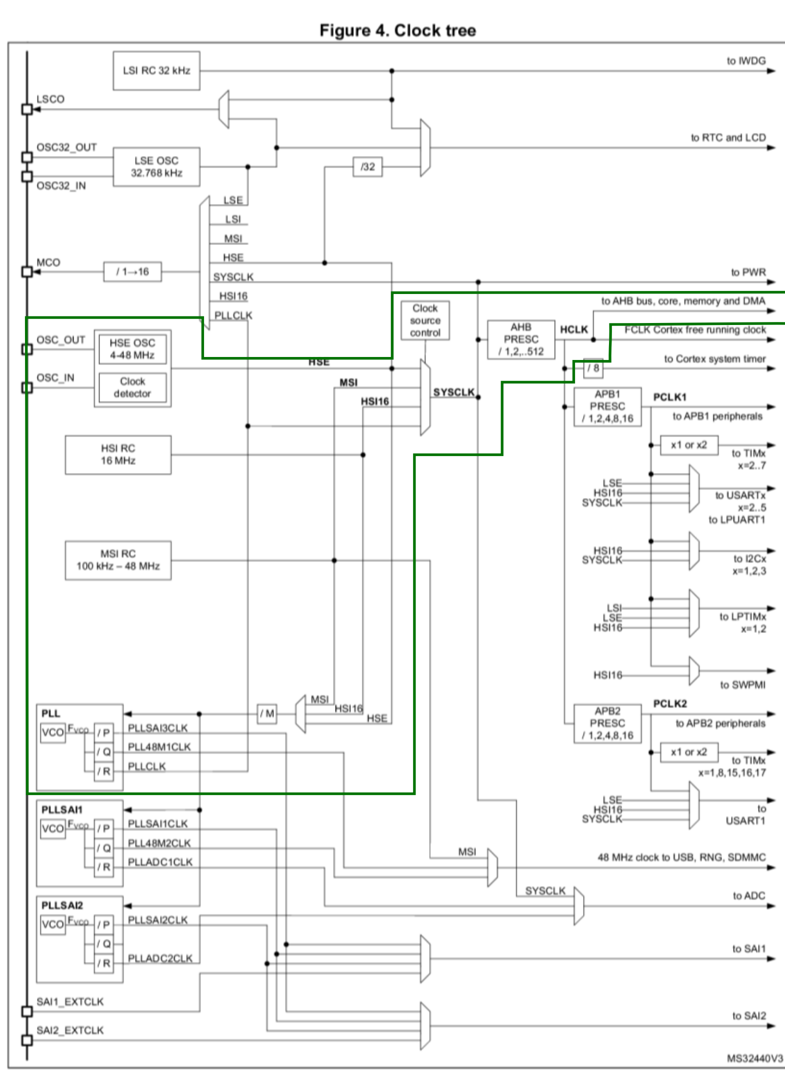


With these different internal clock sources and PLLs, different clock speeds can be generated

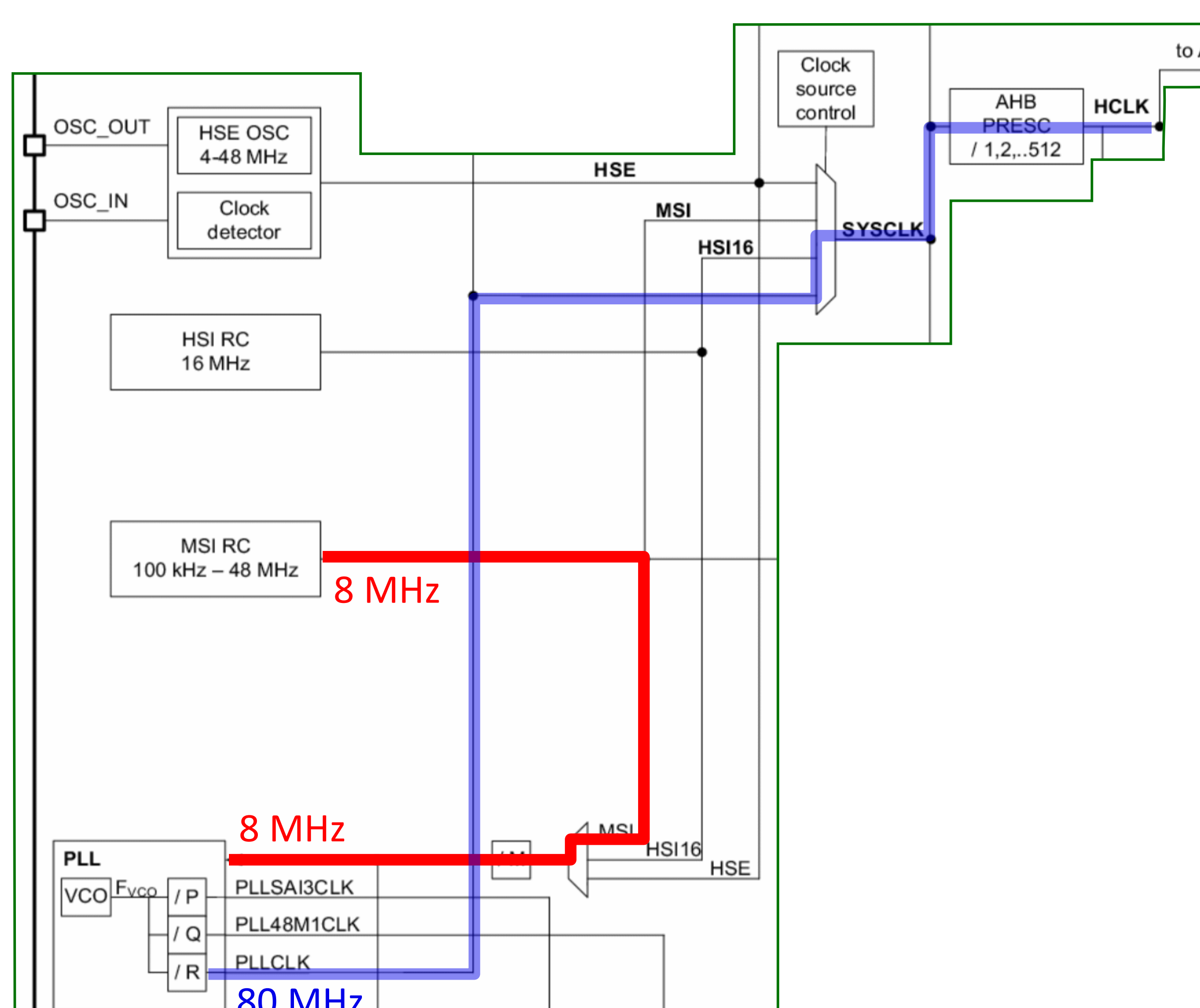
Example:

- Real time clock 1Hz
- SYSCLK 80 MHz
- 48 MHz for USB
- 44.1 kHz for microphone

STM32L4 Clock Tree: Core Frequency SYSCLK

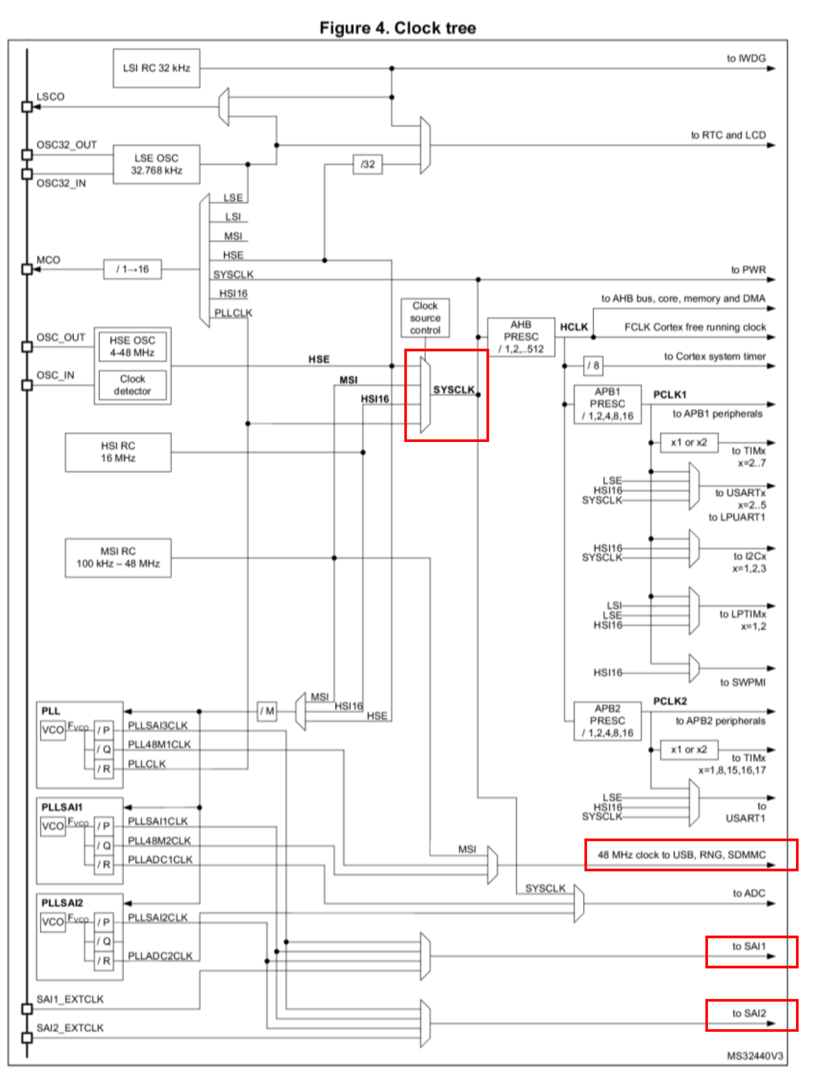


STM32L4 Clock Tree: Core Frequency SYSCLK



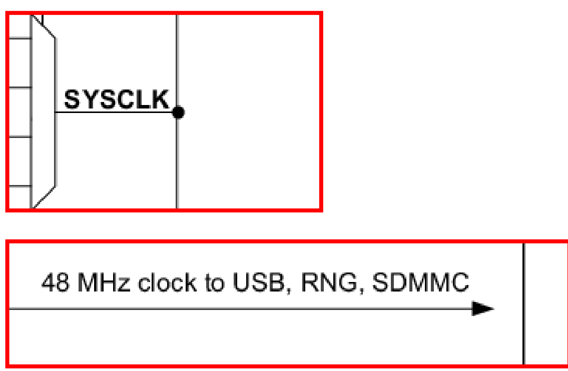
- System clock (SYSCLK) can be derived from HSI16, MSI or HSE
- Example SYSCLK=80 MHz:
  - MSI goes into PLL (to increase it)
  - PLLCLK is chosen as SYSCLK

STM32L4 Clock Tree: Audio Clock

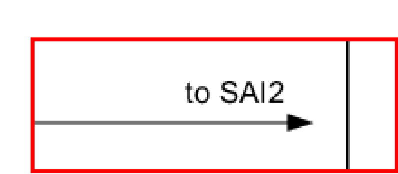


Goal:

- 80 MHz as SYSCLK
- 48 MHz at USB
- 2 audio peripherals in SAI1 and SAI2 (**S**erial **A**udio **I**nterfaces)
  - 11.29 MHz to subsample to 44.1 kHz
  - 49.14 MHz to subsample to 192 kHz

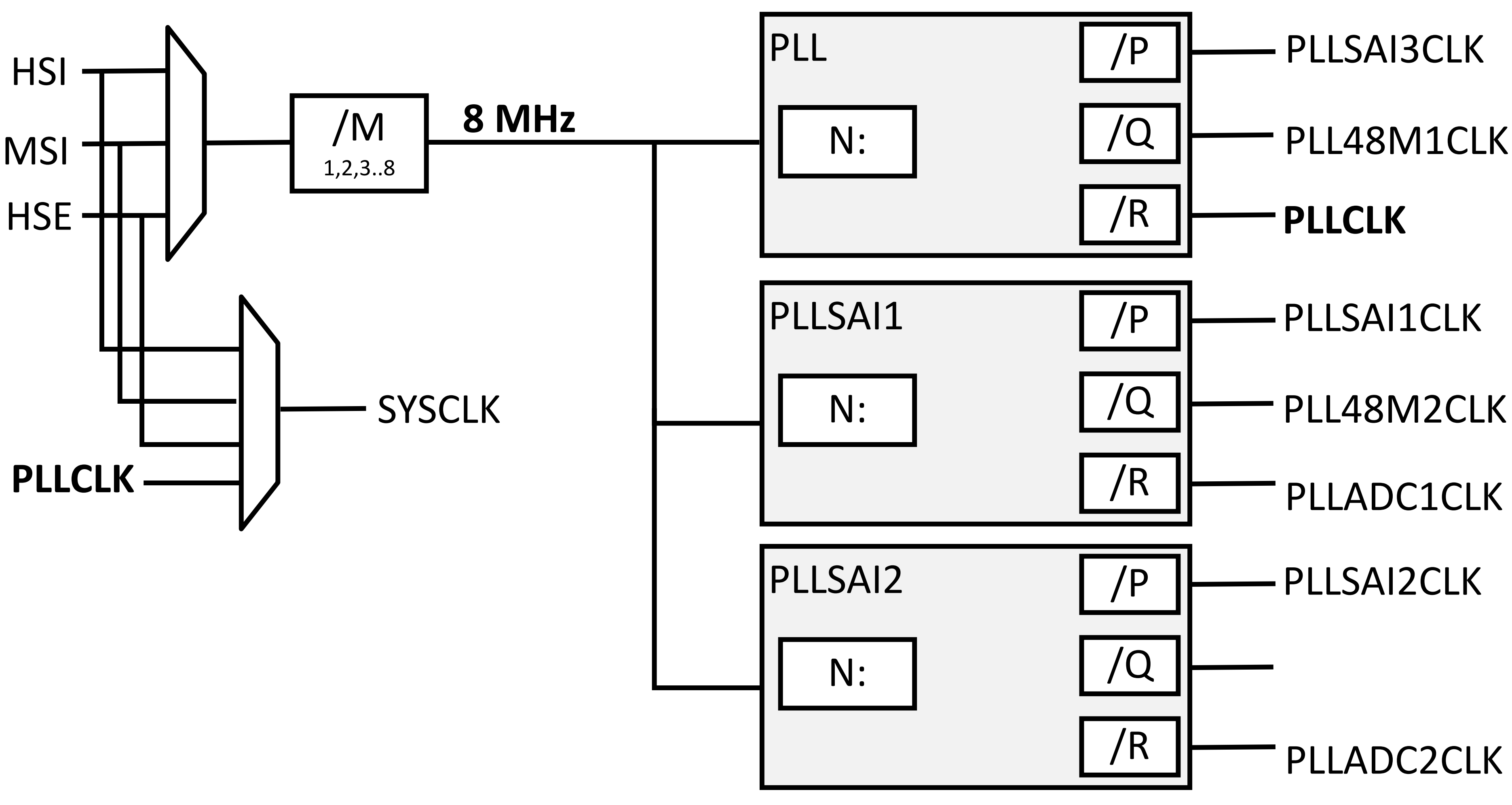


to SAI1

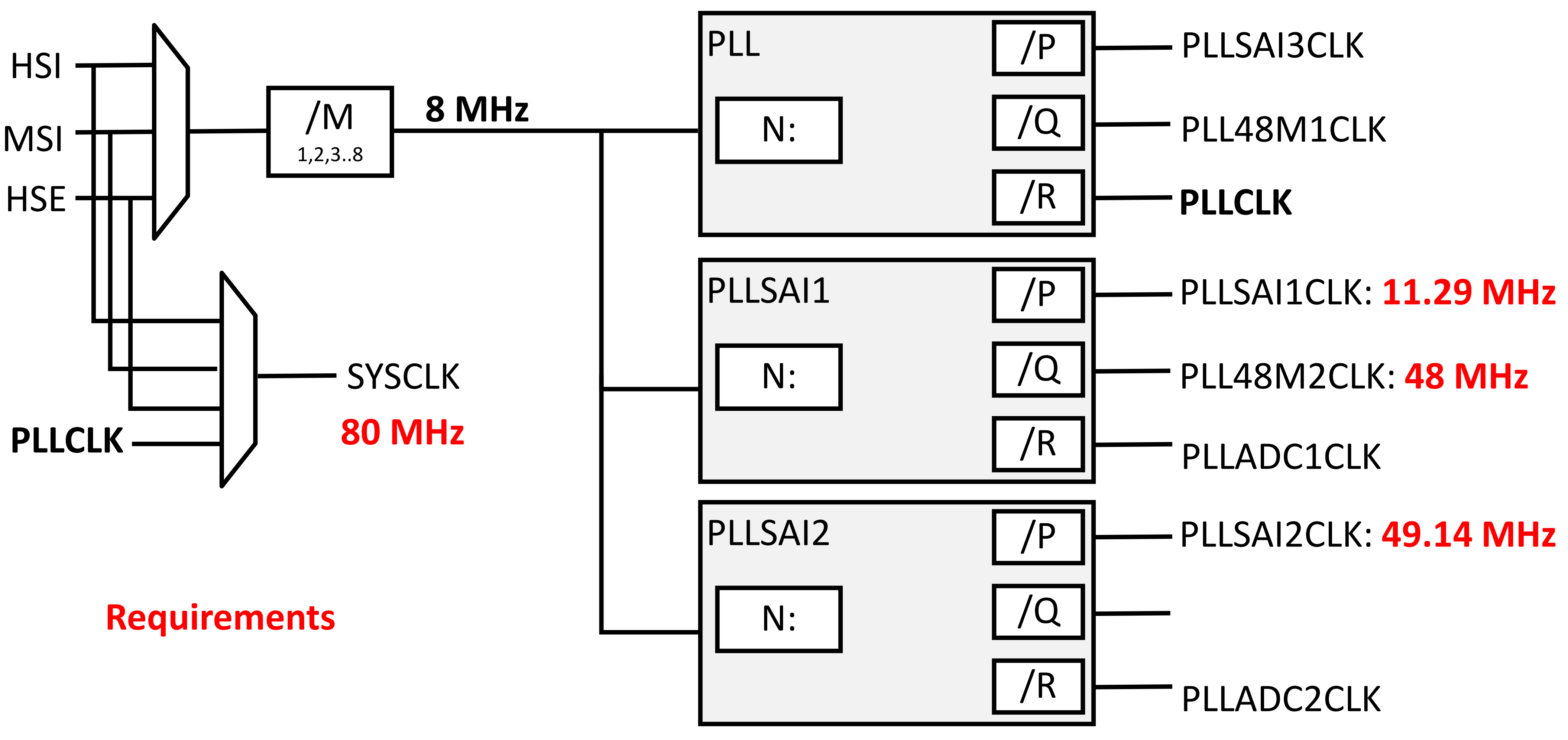


Use PLLs to achieve all the frequency requirements

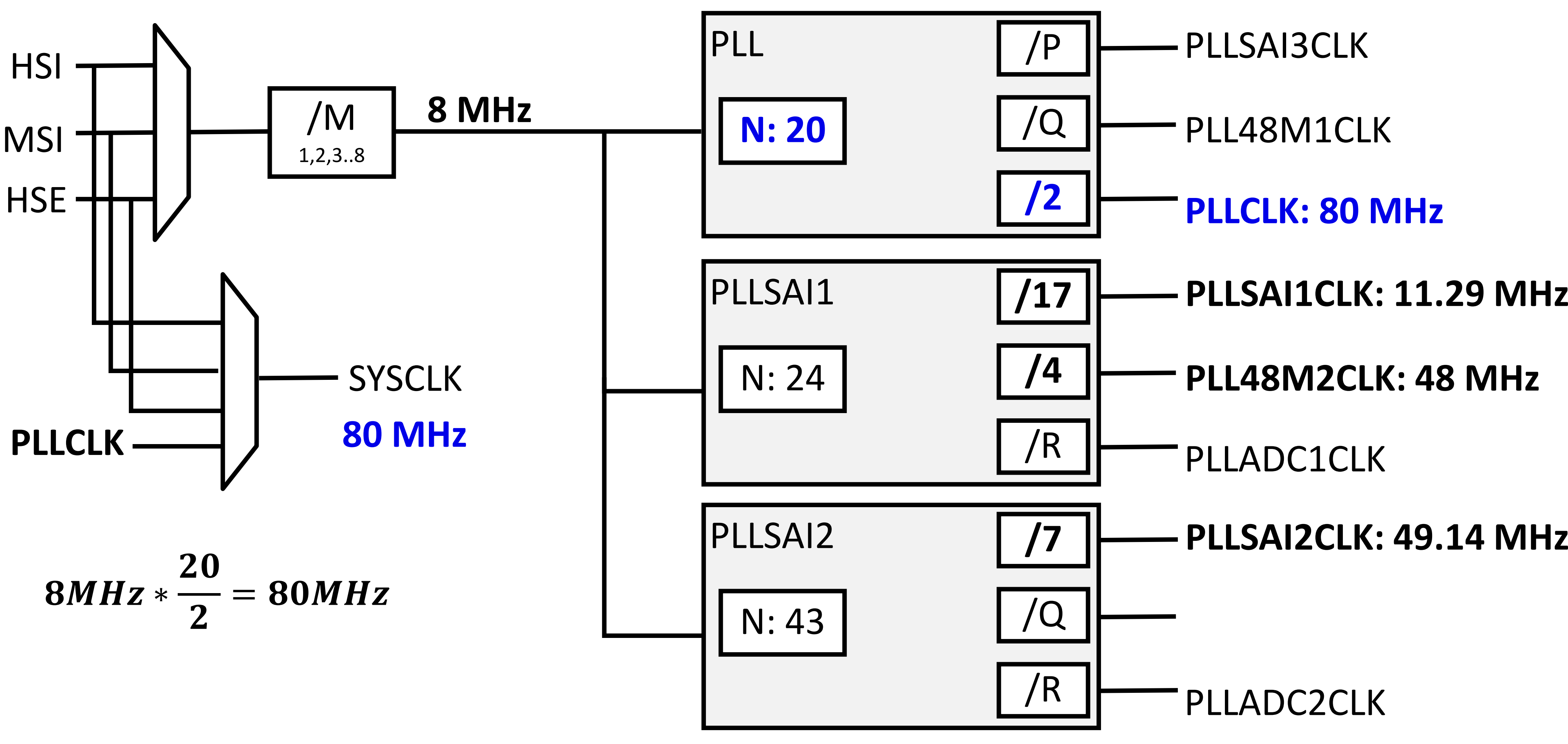
Simplified Clock Tree STM32L4 Audio Clock



Simplified Clock Tree STM32L4 Audio Clock

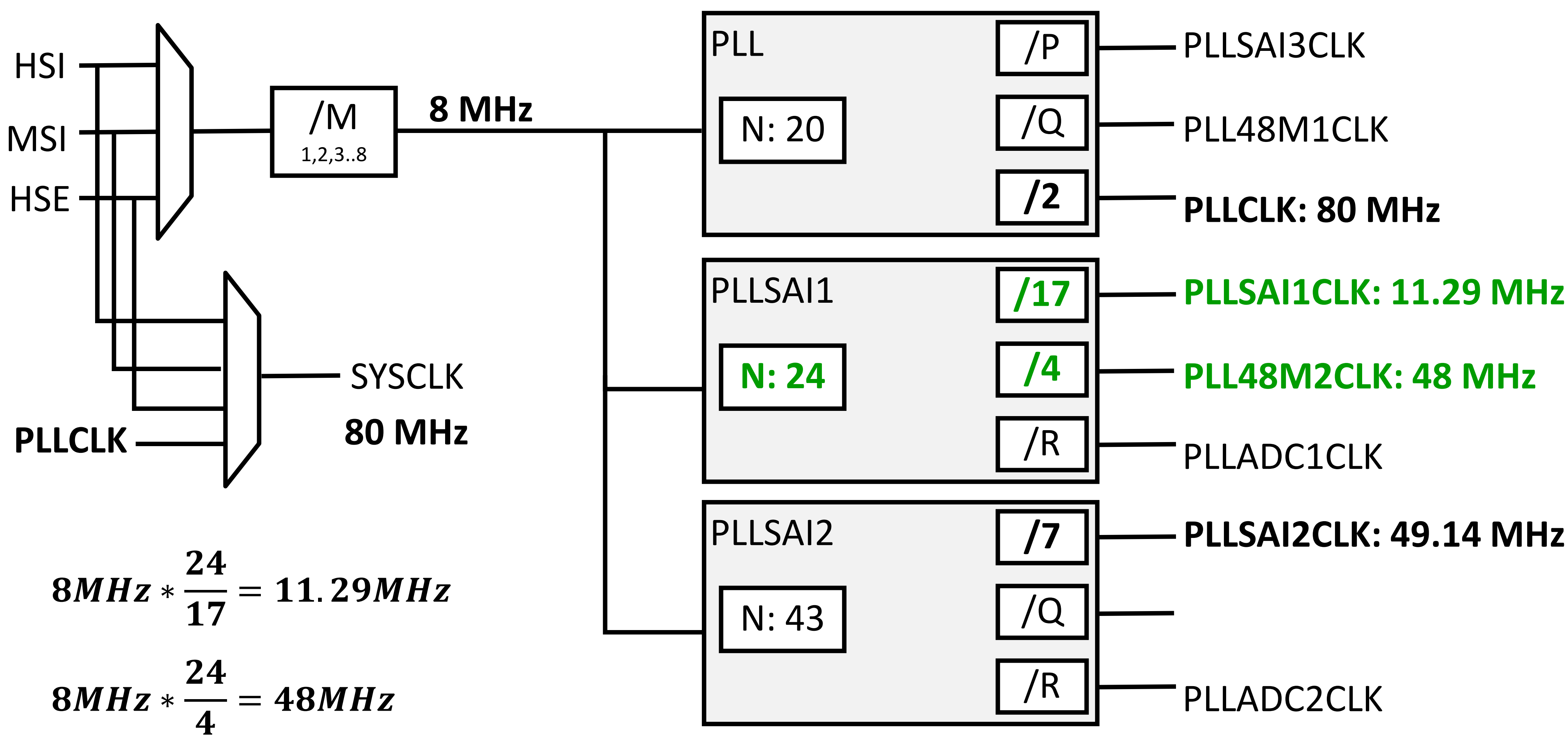


Simplified Clock Tree STM32L4 Audio Clock



$$8MHz * \frac{20}{2} = 80MHz$$

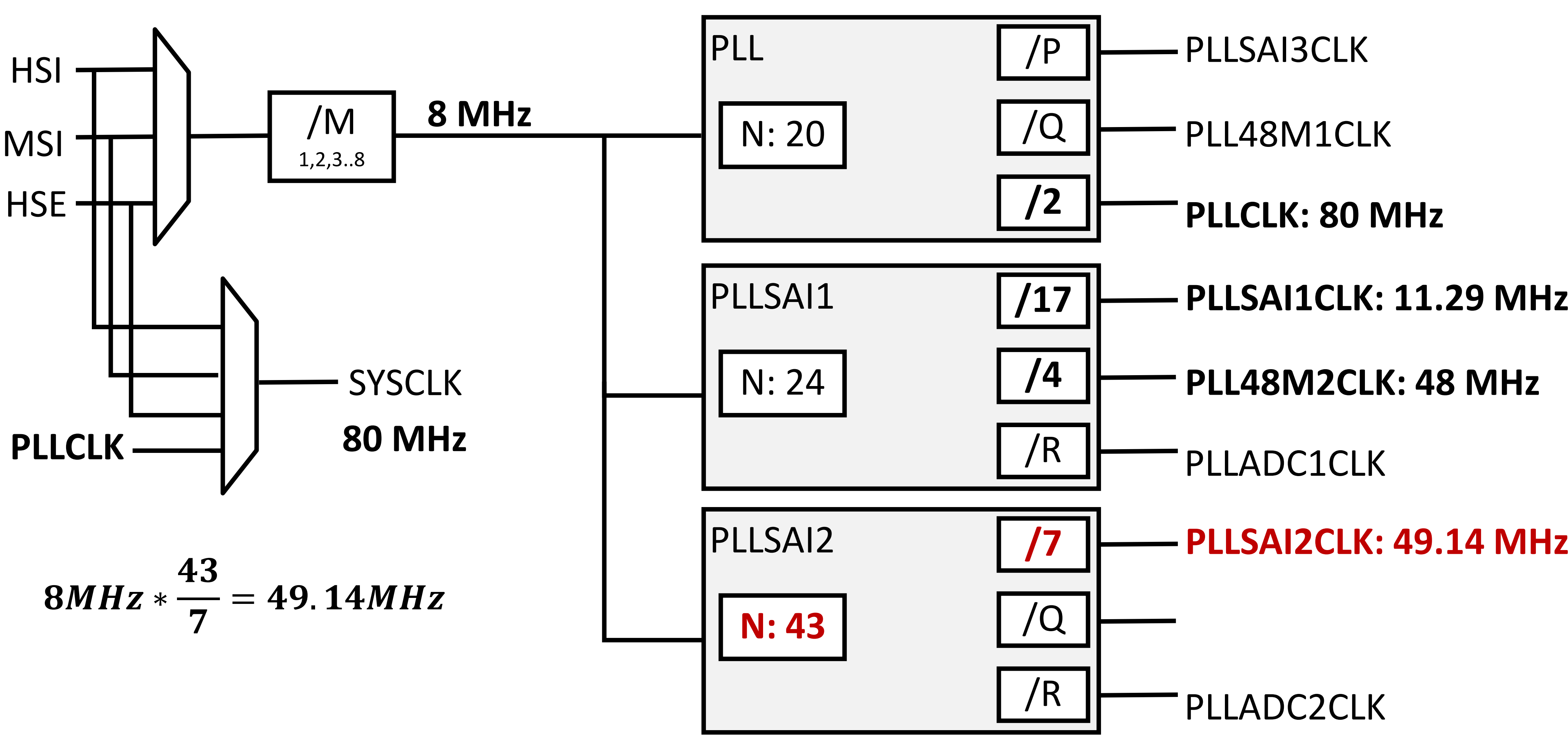
Simplified Clock Tree STM32L4 Audio Clock



$$8MHz * \frac{24}{17} = 11.29MHz$$

$$8MHz * \frac{24}{4} = 48MHz$$

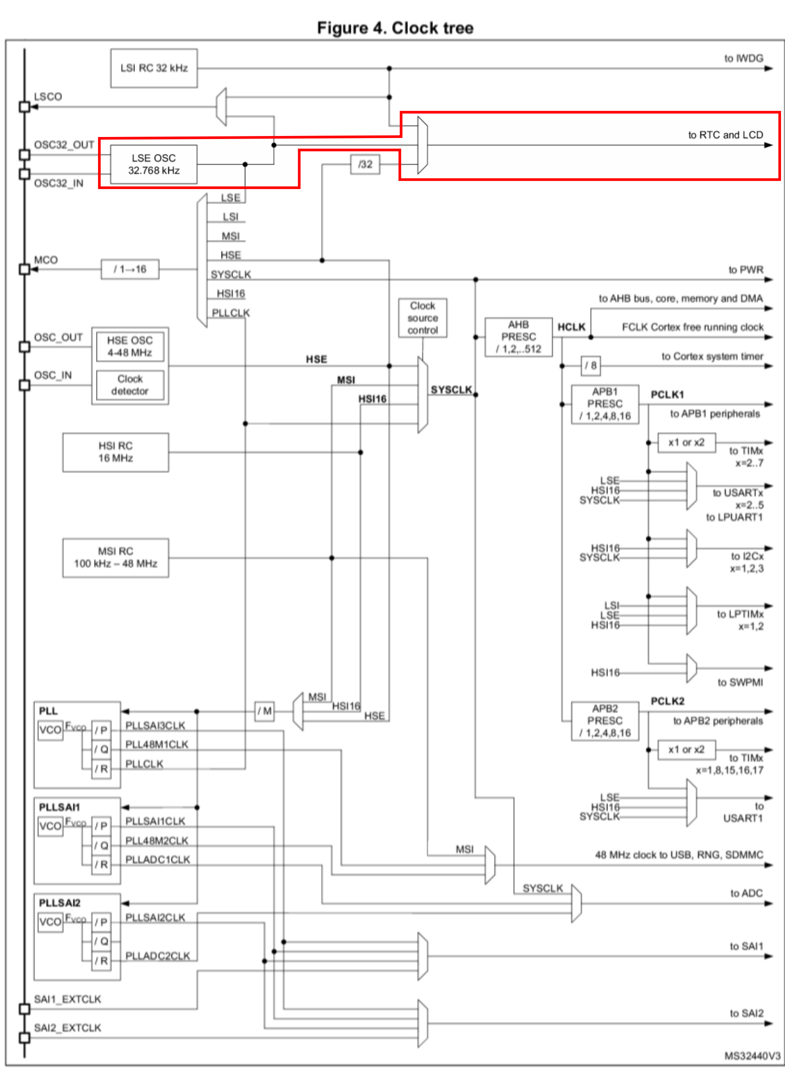
Simplified Clock Tree STM32L4 Audio Clock

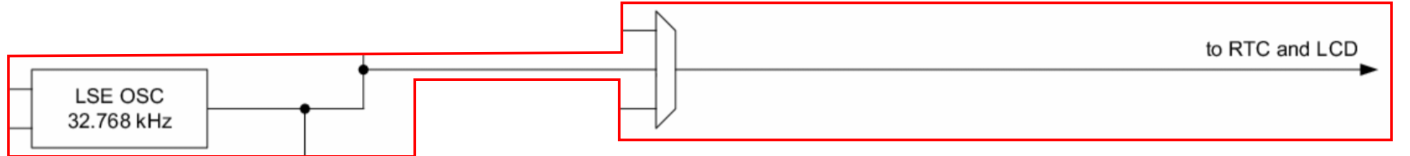


$$8MHz * \frac{43}{7} = 49.14MHz$$

STM32L4 Clock Tree: RTC

Figure 4. Clock tree





- For a real-time clock we want 1 Hz
- LSE (32.768 kHz=2^{15} Hz) suitable clock as it can be divided to get 1 Hz

Comparison MSI and HSI16

	MSI (100 kHz to 48 MHz)		HSI16 (16 MHz)
	MSI mode (w/o LSE)	PLL mode (w/ 32.768 kHz LSE)	
Accuracy (typ.)	Over [0 - 85 C]: +/- 1.55 % Over [1.62 - 3.6 V]: +0.8/-4.5 %	Average accuracy = LSE accuracy Jitter < 0.25%	Over [0-85 C]: +/- 0.8 % Over [1.62 - 3.6 V]: +0.1/-0.2 %
Consumption (typ.)	100 kHz : 0.6 µA 800 kHz : 1.9 µA 1 MHz : 4.7 µA 8 MHz : 18.5 µA 16 MHz : 62 µA 48 MHz : 155 µA		150 µA
Startup time (typ.)	100 kHz : 10 µs 48 MHz : 2.5 µs	5% final frequency : 0.5 ms 1% final frequency : 1.5 ms max	1 µs

Clock: Latency and Energy

- Higher clock speed result in higher current and therefore more energy
- Execution task is linearly dependent on frequency

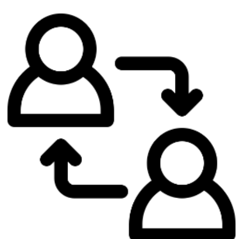
⇨ Trade-off **energy vs. latency** especially in real time constraints

- More detailed analysis in Power and Energy Lecture

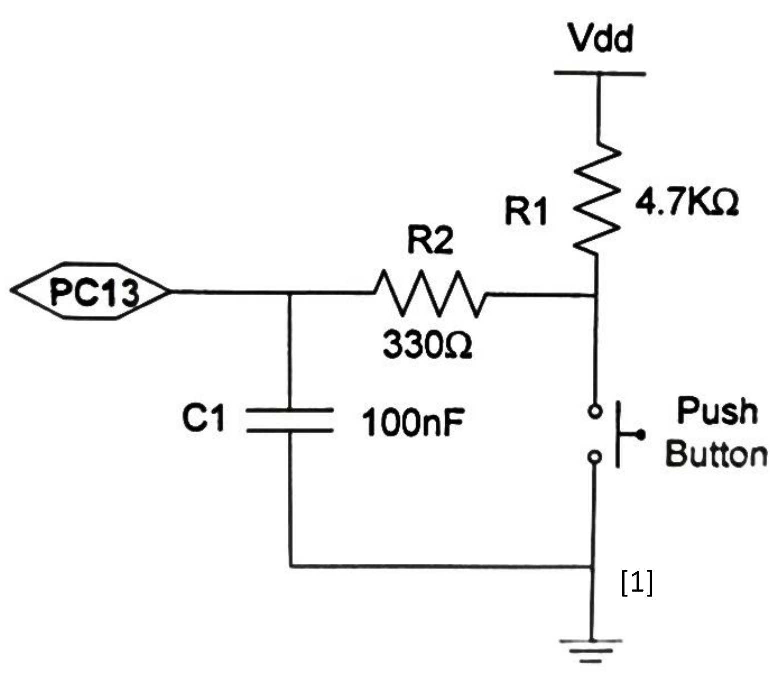
What Did You Learn?

- ✓ Peripherals
- ✓ GPIO Recap
  - Pull-up/Pull-down
  - Schmitt-Trigger
  - Polling
- ✓ Interrupts
  - EXTI, SysTick
  - NVIC, Priorities
- ✓ Clocks
  - Speed, Jitter, Accuracy, Power

Home-Optional Interaction: Button Press



How long is the delay for a button press and button release?



6.3.14 I/O port characteristics

General input/output characteristics

Unless otherwise specified, the parameters given in *Table 70* are derived from tests performed under the conditions summarized in *Table 23: General operating conditions*. All I/Os are designed as CMOS- and TTL-compliant (except BOOT0).

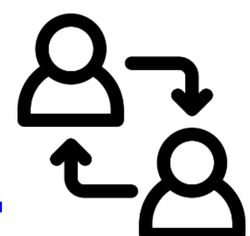
Table 70. I/O static characteristics

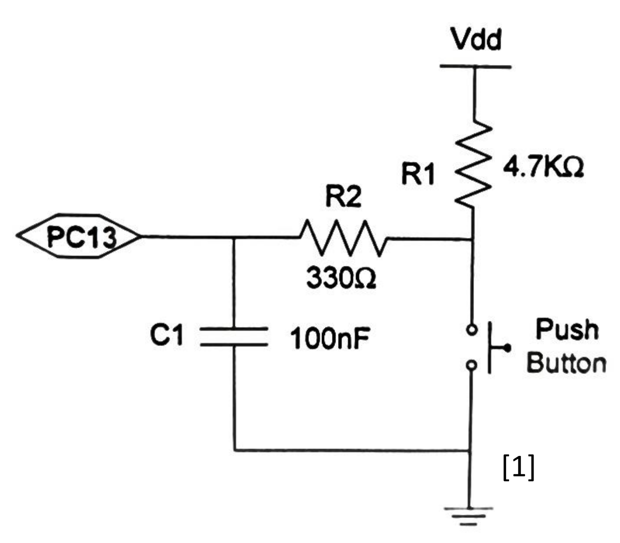
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}(1)	I/O input low level voltage except BOOT0	1.62 V<V_{DDIOx}<3.6 V	-	-	0.3xV_{DDIOx}(2)	V
	I/O input low level voltage except BOOT0	1.62 V<V_{DDIOx}<3.6 V	-	-	0.39xV_{DDIOx}-0.06(3)	
	I/O input low level voltage except BOOT0	1.08 V<V_{DDIOx}<1.62 V	-	-	0.43xV_{DDIOx}-0.1(3)	
	BOOT0 I/O input low level voltage	1.62 V<V_{DDIOx}<3.6 V	-	-	0.17xV_{DDIOx}(3)	
V_{IH}(1)	I/O input high level voltage except BOOT0	1.62 V<V_{DDIOx}<3.6 V	0.7xV_{DDIOx}(2)	-	-	V
	I/O input high level voltage except BOOT0	1.62 V<V_{DDIOx}<3.6 V	0.49xV_{DDIOX}+0.26(3)	-	-	

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 371

Icon by Freepik

Home-Optional Interaction: Button Press





Button with debouncing

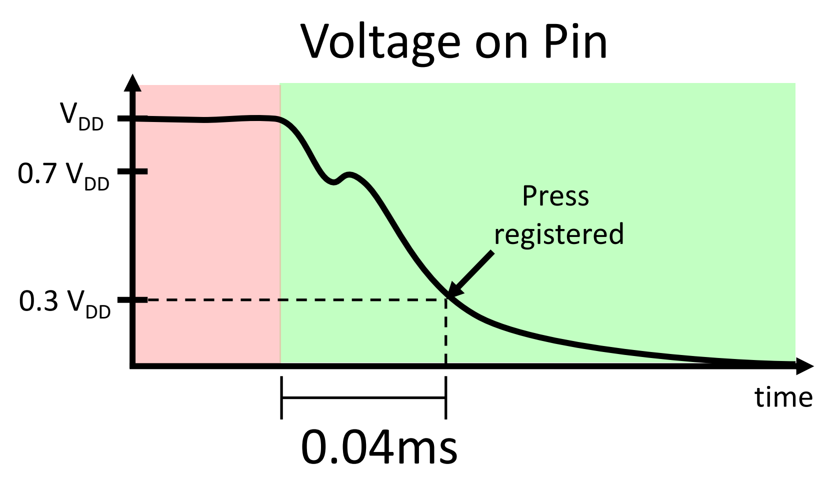
$$\tau = R_2 C_1 = 0.033ms$$

Delay: $V_{Pin} = V_{DD} e^{-\frac{t}{\tau}}$

$$0.3V_{DD} = V_{DD} e^{-\frac{t}{\tau}}$$

$$t = -\ln(0.3)\tau$$

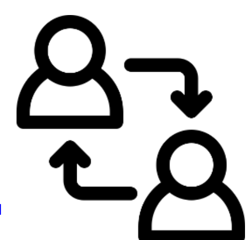
$$t = \mathbf{0.04\ ms}$$

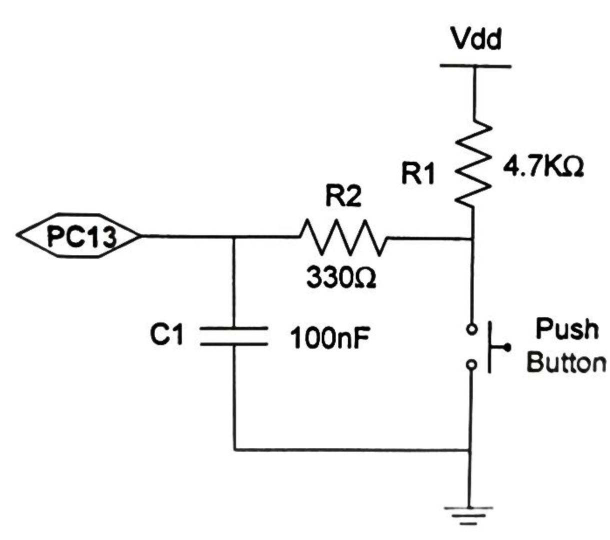


[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 371

Icon by Freepik

Home-Optional Interaction: Button Press





Button with debouncing [1]

$$\tau = (R_1 + R_2)C_1 = 0.503\ ms$$

Delay: $V_{Pin} = V_{DD}(1 - e^{-\frac{t}{\tau}})$

$$0.7V_{DD} = V_{DD}(1 - e^{-\frac{t}{\tau}})$$

$$t = -\ln(0.3)\tau$$

$$t = \mathbf{6\ ms}$$

Voltage on Pin

V_{DD}

0.7 V_{DD}

0.3 V_{DD}

Release registered

time

6ms

Button pressed

Button released

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 371

Icon by Freepik