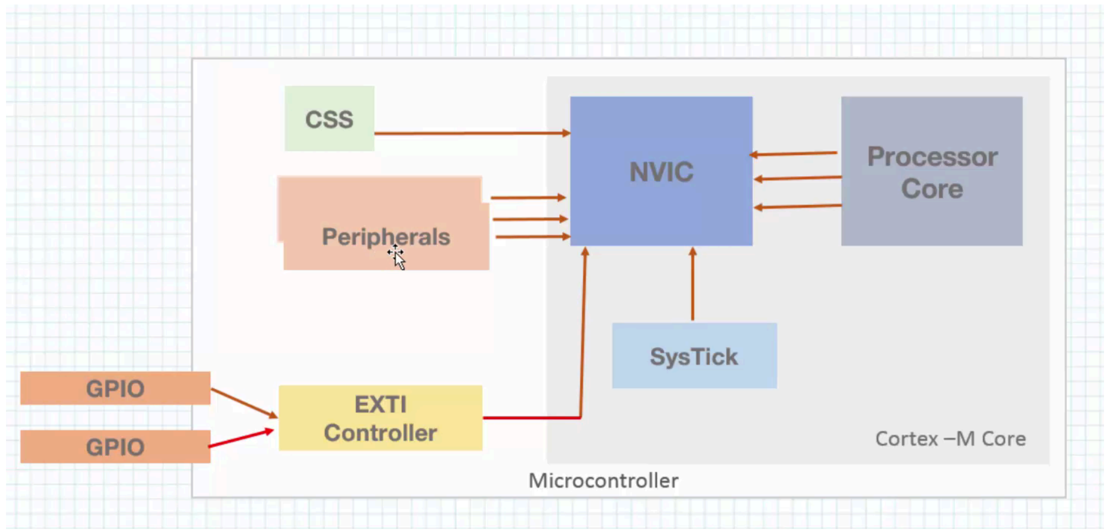
Interrupts

Interrupts vs Polling

The function that gets called when the interrupt occurs is called the Interrupt Service Routine (ISR) or the interrupt handler.

NVIC (Nested Vector Interrupt controller): Dedicated hardware in cortex microcontroller for handling interrupts.

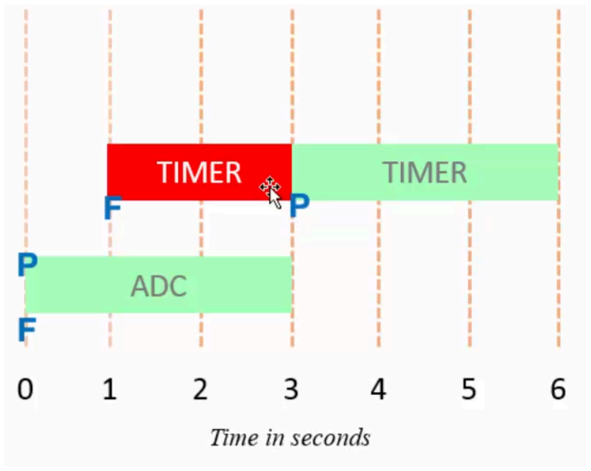


Interrupts by the processor are called exceptions. Interrupts from outside the processor core are known as hardware exceptions/Interrupt requests.

The Vector table contains the addresses of all the interrupt handlers and exception handlers.

For the GPIO pins PA0 to PH0 are connected to one multiplexer pin.

Pending state vs active state for interrupts



Red: Pending state ; **Green**: Active ; **P:** Pending state cleared ; **F**: Interrupt fired

Assume that the ADC interrupt has a higher priority than the timer interrupt.

1. The interrupt is fired at t=0 and the pending state is cleared as there is no other interrupt
2. The timer interrupt fires at t=1
3. As ADC interrupt has higher priority, it remains in the pending state.
4. At t=3, ADC interrupt completes. Since there is no other interrupt with a higher priority, the pending state of the timer interrupt is cleared and it becomes active.

Interrupt priority
Priority of an interrupt: Lower number = higher priority

The priority of each interrupt is defined using the Interrupt priority register (IPR).
Each Interrupt request (IRQ) uses 8-bits in a single IPR register. So one IPR allows us to configure the priorities of 4 different IRQs.

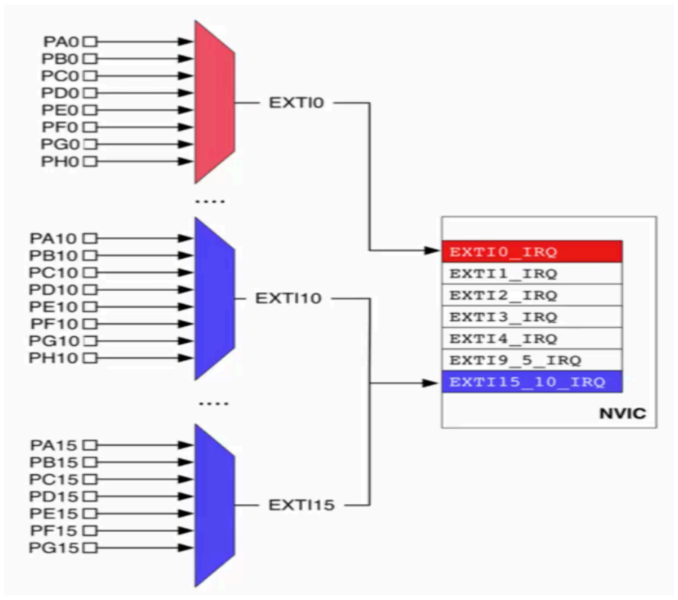
To find the IPR number, we divide the IRQ number by 4, the remainder will determine which byte it is in the IPR register.

IPR can also be divided into sub-priorities.

EXTI Interrupt lines
GPIO pins are connected to EXTI lines. *(Refer to the reference manual for the External interrupt/event GPIO mapping diagram)*.

Pins 10-15 share the same IRQ in NVIC and thus share the same interrupt routine. The same goes for pins 5-9 as well.

SYSCFG is used to select the interrupt GPIO pin. EXTI is used for configuring the interrupt itself.



Developing a GPIO interrupt driver

Objective

Write bare-metal C code to trigger an interrupt when the user button on the STM32 Nucleo board is pressed, causing the LED to toggle its state.

GPIO initialisation

1. The user button is connected to port C, pin13 (PC13).
2. Enable the clock for GPIO C and set the mode to input.

Interrupt initialisation

1. Enable the clock for the SYSCFG module. Use the external interrupt configuration register (SYSCFG_EXTICRx) for port and pin selection.
2. Unmask EXT13 using the interrupt mask register (EXTI_IMR).
3. Configure interrupt for falling edge trigger using the falling trigger selection register (EXTI_RTSR).
4. Enable IRQ for EXT13 in NVIC.
5. Enable global interrupts.

Developing a UART interrupt driver

Objective

Write bare metal C code to trigger the interrupt for the UART module based on keyboard input from a laptop. The interrupt routine must trigger a function call that turns the user LED on the STM32 Nucleo board on/off based on the input.

The **USART2** module on the STM32F4 will be used for this project as it is connected to USB. The module is configured to operate as **UART.**

GPIO initialisation

1. Enable the clock for PA3 and set it to alternate function mode. PA3 is Rx pin for USART2.

Refer to USART interrupts in the reference manual.

USART module and interrupt initialisation

1. Enable clock for USART2 module (APB1 clock bus).
2. Set baud rate
    a. Define a function to calculate the baud rate divisor.
    b. In another function set the BRR in USART2 register to the divisor.
3. Enable the receiver using CR1 (Control register 1).
4. Received data ready to be read interrupt event: enable bit: RXNEIE.
5. Enable IRQ for USART2.
6. Enable the UART module using CR1 (Control register 1).

Facts about ISRs

1. ISR cannot return a value.
2. ISR cannot be passed parameters.