EE 304
Embedded Systems

Mehmet Bozdal

ABDULLAH GÜL ÜNİVERSİTESİ

Agenda

- General Purpose Input Output (GPIO)
- Interrupts

Previous Lab

```c
while (1) {
    GPIOD->ODR ^= (1 << 0) ; // Toggle pins 0
    delay(499999); // Introduce a delay

    // Check the state of pin 4 of Port D
    if ((GPIOD->IDR & GPIO_IDR_IDR4) == GPIO_IDR_IDR4) {
        delay(100); // Small delay

        // Depending on the current system clock source, switch to a different source
        if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_HSI) {
            initClockHSE();
            GPIOD->ODR = (1 << 5); // Set PD5 to indicate HSE as active
        }
        else if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_HSE) {
            initClockPLL();
            GPIOD->ODR = (1 << 6); // Set PD6 to indicate PLL as active
        }
        else if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_PLL) {
            initClockHSI();
            GPIOD->ODR = (1 << 7); // Set PD7 to indicate HSI as active
        }
    }
}
```

Any issues?

- Switch bounce?
- Software debounce (wait ~50ms)

Behavior of button click

- Does it catch all the clicks correctly?
- What happens if I click while the delay?

Polling (Synchronous Approach)

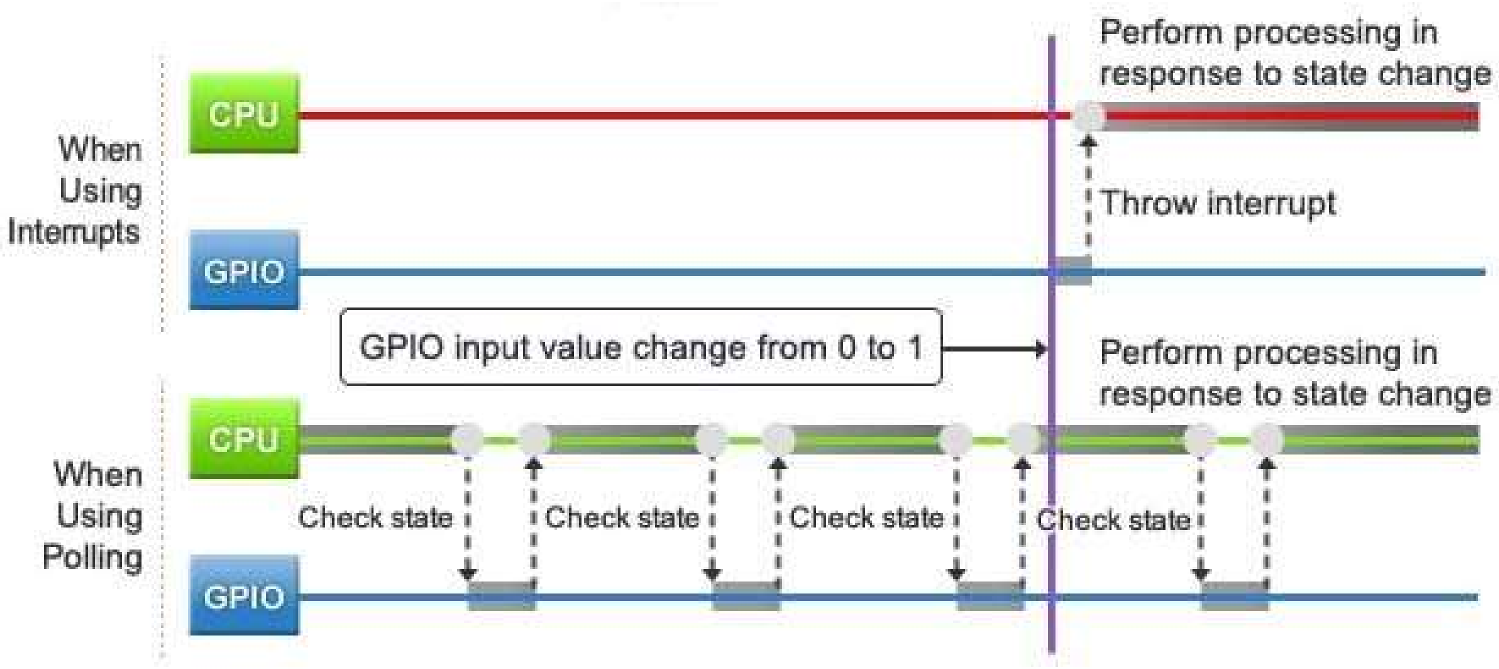
- Polling is a synchronous method where the program explicitly checks the status of a particular input or event at regular intervals.
- It involves repeatedly querying a hardware device or a flag to determine if a specific condition has occurred(consumes processing power during this time).
- Polling is relatively straightforward to implement and understand, making it a suitable choice for simple applications and for situations.

Interrupt (Asynchronous Approach)

- Interrupts are asynchronous events triggered by hardware or external events.
- When an interrupt occurs, the processor suspends its current task to handle the interrupt (priority-based), and then returns to the original task.
- Interrupts provide real-time responsiveness to external events, as they are serviced immediately when the event occurs, without the need for periodic polling.

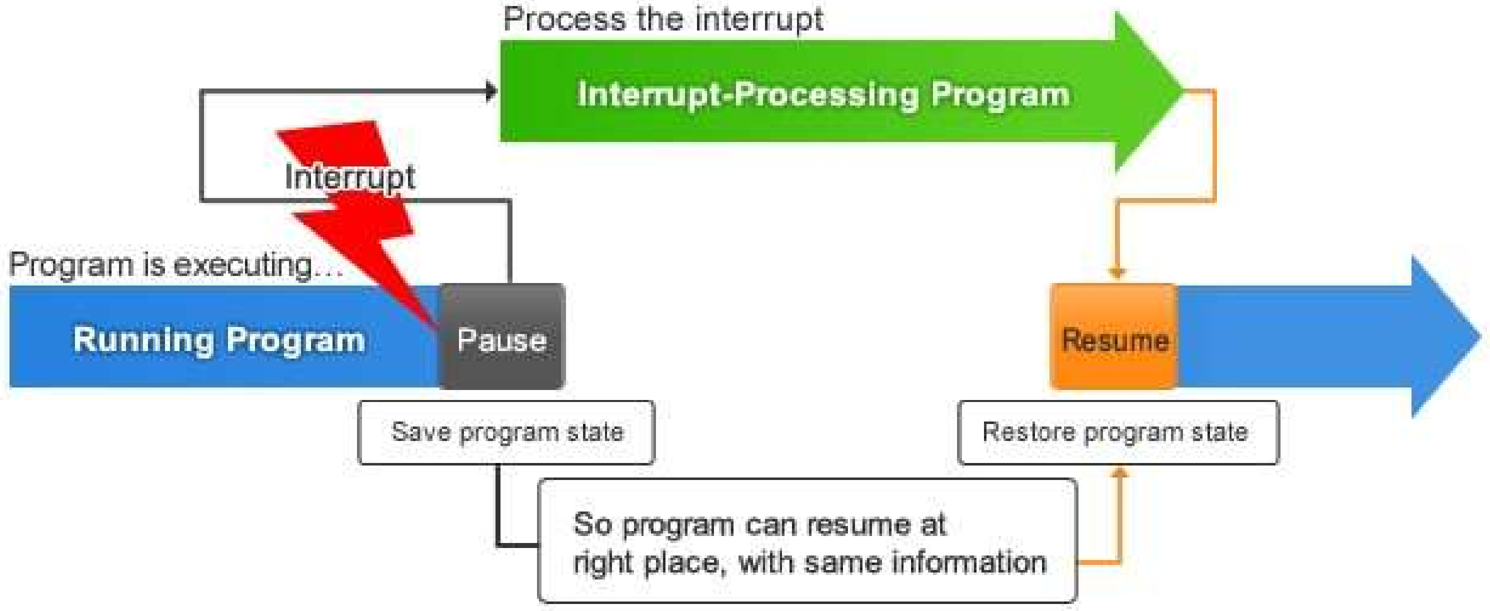
AGÜ

Interrupts vs. Polling


Perform processing in response to state change
When Using Interrupts
CPU
Throw interrupt
GPIO
GPIO input value change from 0 to 1
Perform processing in response to state change
When Using Polling
CPU
Check state
Check state
Check state
Check state
GPIO

Source: https://www.renesas.com/kr/en/support/engineer-school/mcu-programming-peripherals-04-interrupts

Interrupt Processing Flow



Source: www.renesas.com

Steps to Execute an Interrupt

When an interrupt gets active, the microcontroller goes through the following steps:

- The microcontroller closes the currently executing instruction and saves the address of the next instruction (PC) on the stack.
- It also saves the current status of all the interrupts internally (i.e., not on the stack).
- It jumps to the memory location of the interrupt vector table that holds the address of the interrupts service routine.

Steps to Execute an Interrupt

- The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the location where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then, it start to execute from that address.

Interrupt Service Routine (ISR)

- **ISR (or interrupt handler)** is the software module that is executed when the hardware requests an interrupt.
- There may be one large ISR that handles all requests (polled interrupts), or many small ISRs specific for each potential source of interrupt (vectored interrupts).
- **Except for the SysTick interrupt, the ISR software must explicitly clear the trigger flag that caused the interrupt (acknowledge).**
- Although interrupt handlers can create and use local variables, parameter passing between threads must be implemented using shared global memory variables.

Interrupt Service Routine (ISR)

- ISR should execute as fast(short) as possible.
- For every interrupt, there must be an ISR.
- When an interrupt occurs, the microcontroller runs the ISR.
- For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine. The table of memory locations set aside to hold the addresses of ISRs is called as the **Interrupt Vector Table**.

Interrupt Vector Table

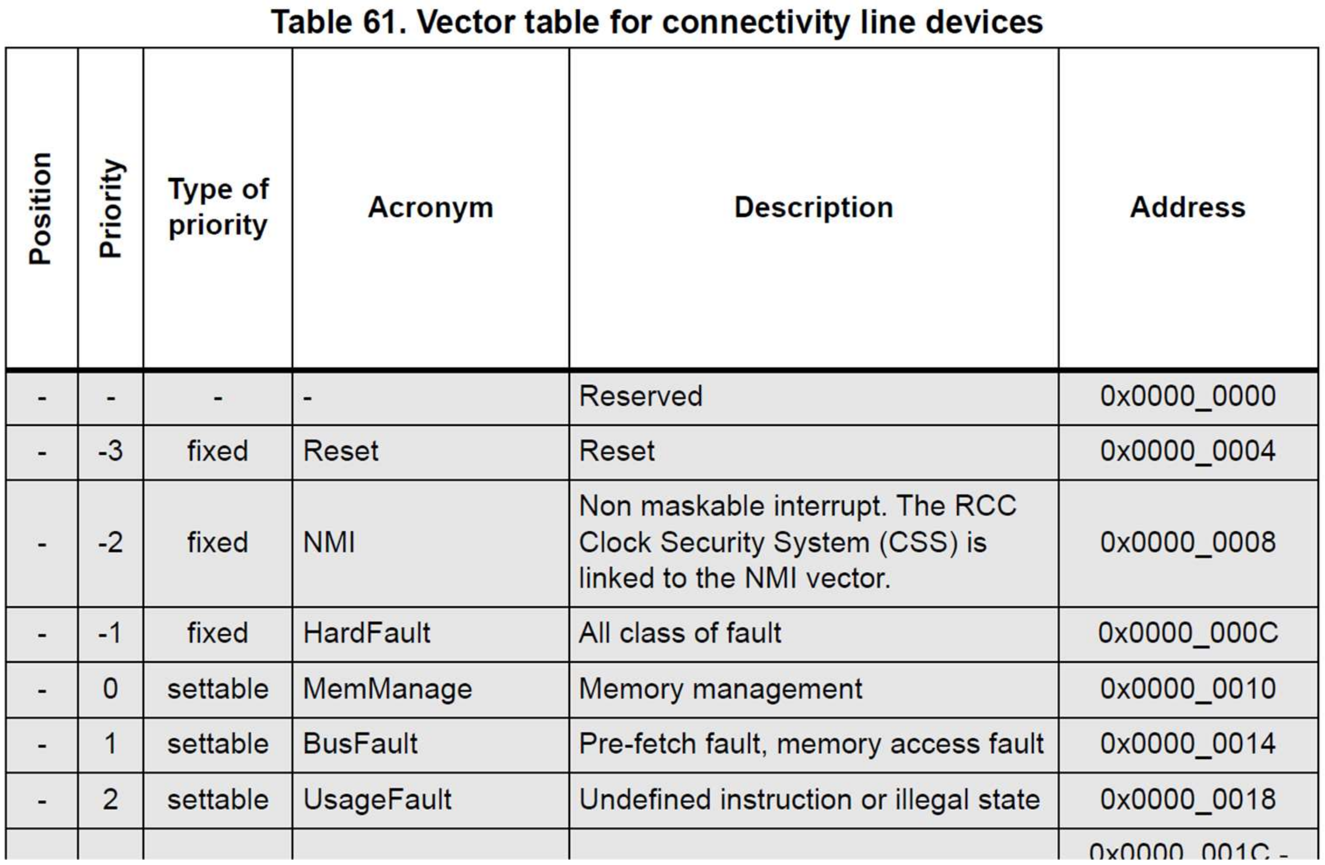


Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
					0x0000_001C -

NVIC on the ARM Cortex-M Processor

- On the ARM Cortex-M processor, **exceptions** include resets, software interrupts and hardware interrupts.
- Interrupts on the Cortex-M are controlled by the Nested Vectored Interrupt Controller (NVIC).
- Each exception has an associated 32-bit vector that points to the memory location where the ISR that handles the exception is located.
- Vectors are stored in ROM at the beginning of memory.

Nested Vectored Interrupt Controller

- The interrupt controller's job is to pass these interrupt requests to the CPU in a coordinated way.
- When multiple interrupts occur, the controller must send these to the CPU in the appropriate order, based on their relative priorities.
- The controller must also be aware of which interrupts are currently masked (disabled), so that it can ignore these interruptions completely.

Nested vectored interrupt controller (NVIC)

Features:

- 68 (not including the sixteen Cortex®-M3 interrupt lines)
- 16 programmable priority levels (4 bits of interrupt priority are used)
- Low-latency exception and interrupt handling
- Power management control
- Implementation of System Control registers

External interrupt/event controller (EXTI)

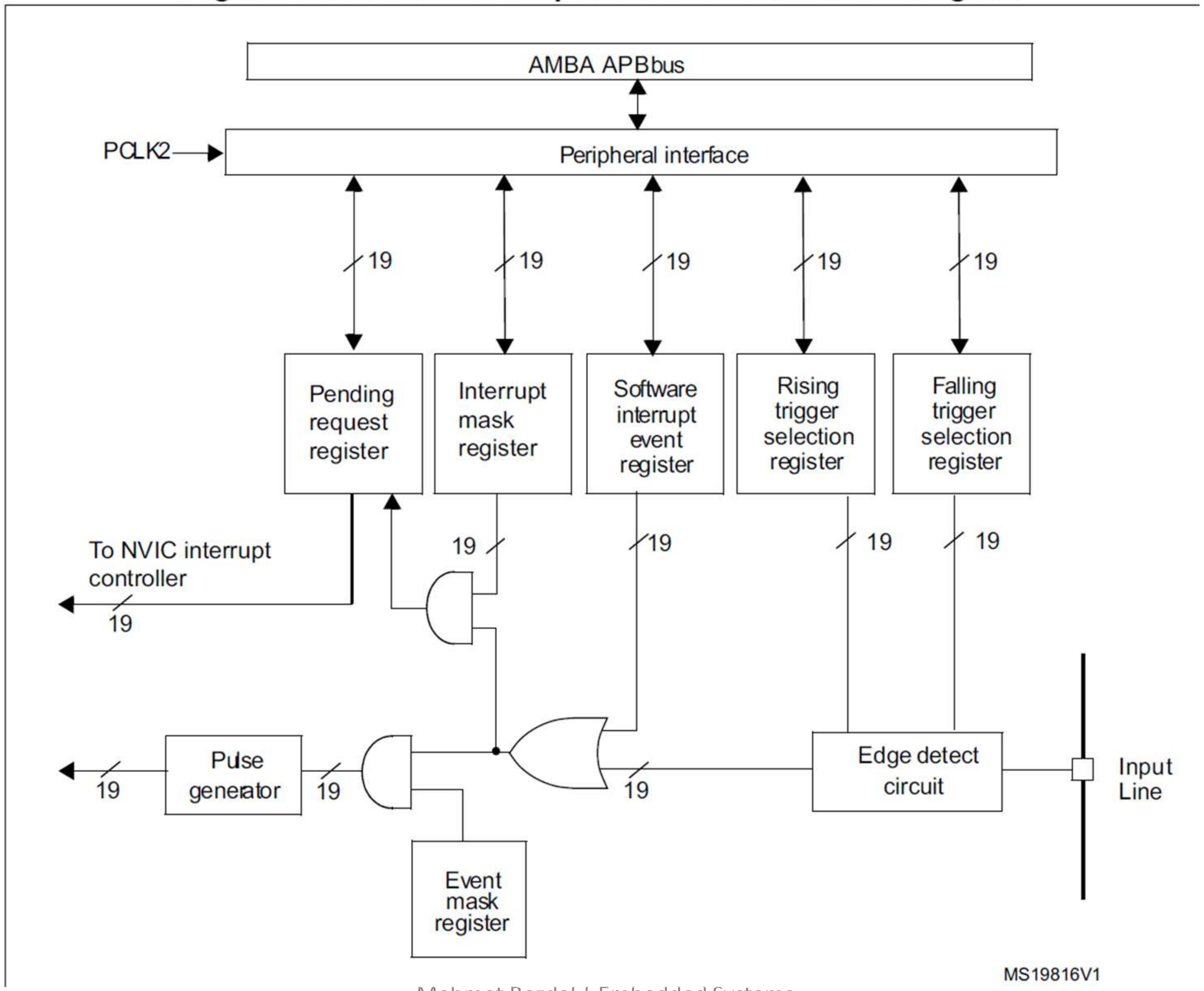
- The external interrupt/event controller consists of up to 20 edge detectors in connectivity line devices, or 19 edge detectors in other devices for generating event/interrupt requests.
- Each input line can be independently configured to select the type (event or interrupt) and the corresponding trigger event (rising or falling or both).
- Each line can also masked independently. A pending register maintains the status line of the interrupt requests.

Main features

The EXTI controller main features are the following:

- Independent trigger and mask on each interrupt/event line
- Dedicated status bit for each interrupt line
- Generation of up to 20 software event/interrupt requests
- Detection of external signal with pulse width lower than APB2 clock period.

Figure 20. External interrupt/event controller block diagram



Functional description

- To generate the interrupt, the interrupt line should be configured and enabled.
- This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a ‘1’ to the corresponding bit in the interrupt mask register.

Functional description

- When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a ‘1’ in the pending register.

```
EXTI->PR |= (1<<5); // Clear the interrupt flag by writing a 1
```

- *An interrupt/event request can also be generated by software by writing a ‘1’ in the software interrupt/event register.*

Hardware interrupt selection

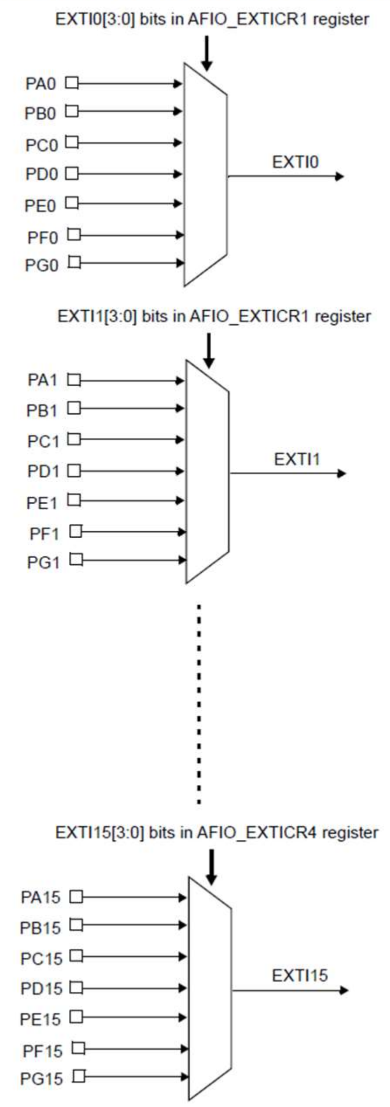
To configure the 20 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 20 Interrupt lines (EXTI_IMR)
- Configure the Trigger Selection bits of the Interrupt lines (EXTI_RTSR and EXTI_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the External Interrupt Controller (EXTI) so that an interrupt coming from one of the 20 lines can be correctly acknowledged.

External interrupt/event line mapping

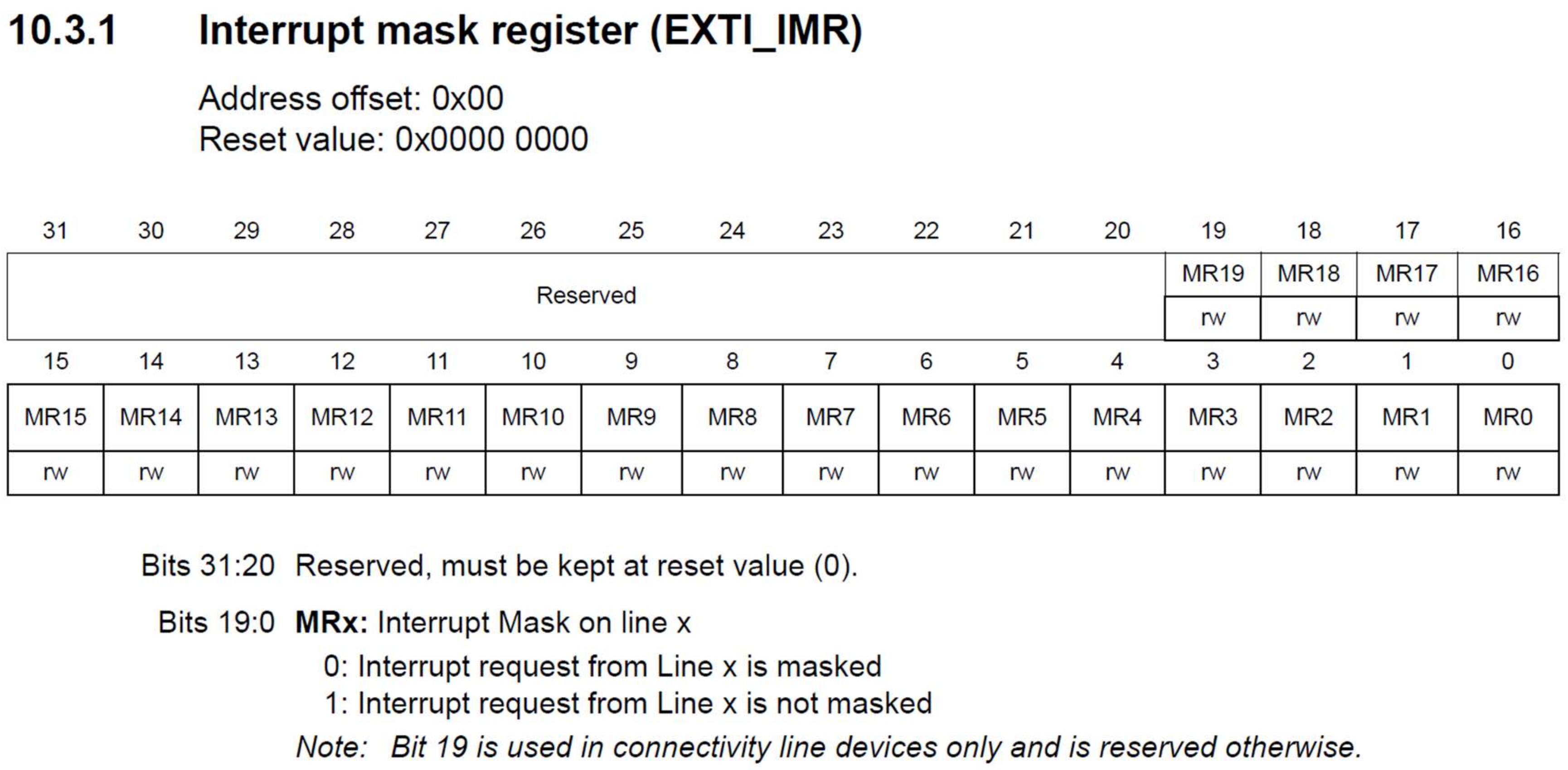
The 112 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

Figure 21. External interrupt/event GPIO mapping



EXTI0[3:0] bits in AFIO_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
PG0
EXTI0
EXTI1[3:0] bits in AFIO_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PF1
PG1
EXTI1
EXTI15[3:0] bits in AFIO_EXTICR4 register
PA15
PB15
PC15
PD15
PE15
PF15
PG15
EXTI15

EXTI registers



10.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x

0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

10.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Event mask on line x

0: Event request from Line x is masked

1: Event request from Line x is not masked

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line

1: Rising trigger enabled (for Event and Interrupt) for input line.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Falling trigger event configuration bit of line x

0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

Note: Bit 19 used in connectivity line devices and is reserved otherwise.

10.3.5 Software interrupt event register (EXTI_SWIER)

Address offset: 0x10
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												SWIER 19	SWIER 18	SWIER 17	SWIER 16
												rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

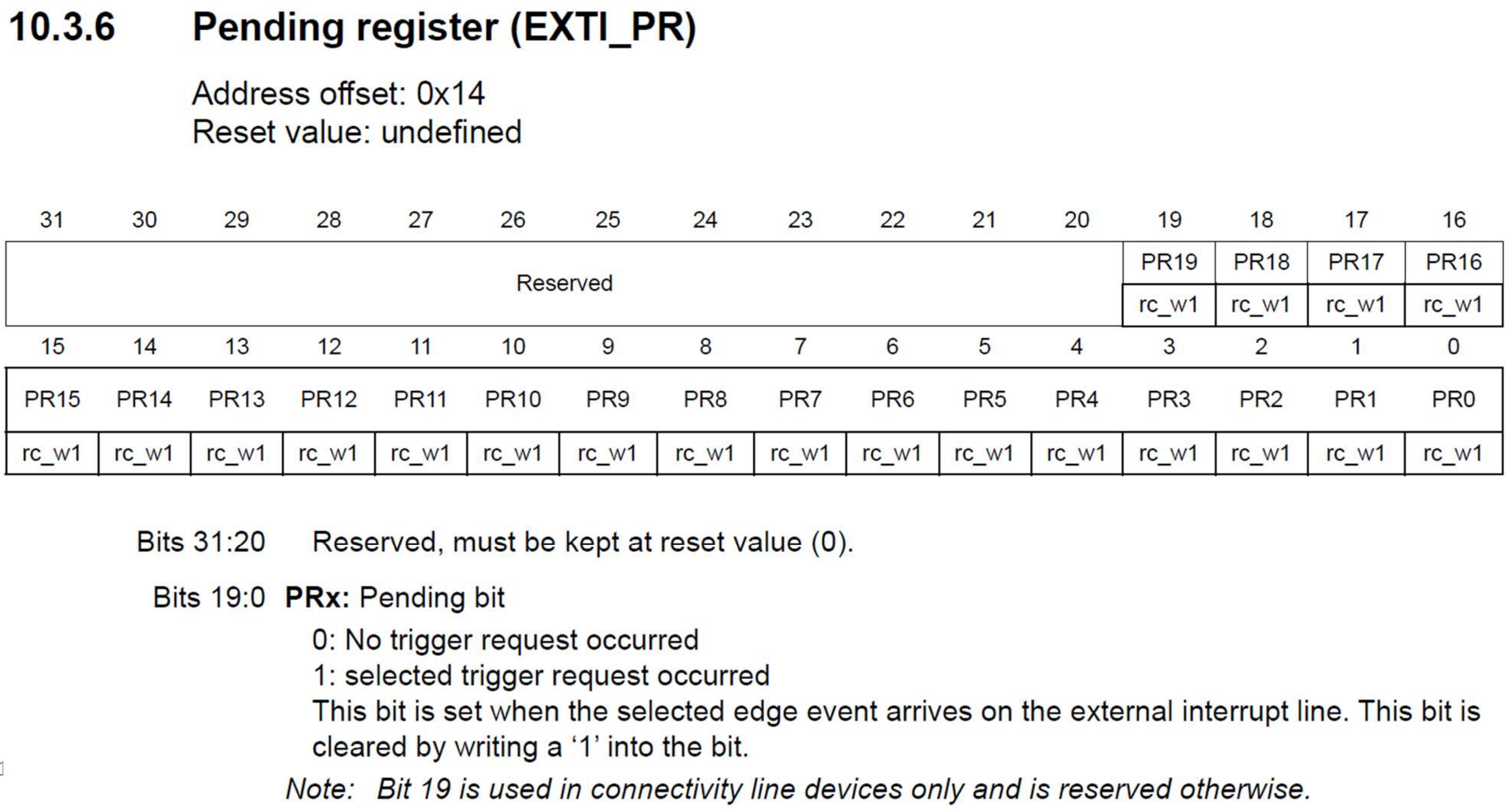
Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **SWIERx:** Software interrupt on line x

If the interrupt is enabled on this line in the EXTI_IMR, writing a '1' to this bit when it is set to '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a 1 into the bit).

Note: Bit 19 used in connectivity line devices and is reserved otherwise.



10.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit

0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a ‘1’ into the bit.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

1. Configure the External Interrupt Source

```
// Enable the GPIO port clock (e.g., for GPIOA)
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

// Configure the pin (e.g., PA0) as input with
pull-up or pull-down
GPIOA->CRL &= ~(GPIO_CRL_MODE0 | GPIO_CRL_CNF0);
GPIOA->CRL |= GPIO_CRL_CNF0_1; // Input pull-up
```

2. Configure the External Interrupt Setting

```
// Enable the AFIO clock
RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;

// Connect EXTI0 to GPIOA pin 0
AFIO->EXTICR[0] &= ~(AFIO_EXTICR1_EXTI0);
AFIO->EXTICR[0] |= AFIO_EXTICR1_EXTI0_PA;

// Configure EXTI0
EXTI->RTSR |= EXTI_RTSR_TR0;//Enable rising edge trigger
EXTI->FTSR &= EXTI_RTSR_TR0;//Disable falling edge trig.
```

3. Enable the External Interrupt

```
// Disable the Mask on EXTI0
EXTI->IMR |= EXTI_IMR_MR0;
```

4. Enable and Set the Priority for the NVIC

```
// Set the priority (adjust this value based on
your needs)
NVIC_SetPriority(EXTI0_IRQn, 2);

// Enable EXTI0 IRQ in NVIC
NVIC_EnableIRQ(EXTI0_IRQn);
```

5. Implement the Interrupt Service Routine (ISR)

```
void EXTI0_IRQHandler(void) {
    // Interrupt handling code here

    // Clear the EXTI line pending bit
    EXTI->PR |= EXTI_PR_PR0;
}
```

Q&A

Any questions?