# Capítulo 7

# Interrupções no STM32F407

## 7.1. Visão geral das interrupções

As interrupções são um recurso comum disponível em praticamente todos os microcontroladores. Interrupções são eventos tipicamente gerados pelo hardware (pinos de entrada ou outros periféricos, por exemplo), que causam alterações no fluxo de execução normal do programa para fornecer serviço a um evento específico. Interrupções permitem a um microcontrolador responder rapidamente a eventos sensíveis ao tempo, com baixíssima latência.

Quando um periférico precisa ser prontamente atendido pelo processador, normalmente ocorre a seguinte sequência de eventos:

1. O periférico emite um sinal de solicitação de interrupção ao processador (*IRQ – Interrupt Request*);
2. O processador suspende a tarefa que está sendo executada;
3. O processador executa uma função específica para atender o periférico (*ISR - Interrupt Service Routine*);
4. O processador retoma a tarefa suspensa anteriormente após a finalização da *ISR.*

Interrupções também permitem ao processador executar múltiplas tarefas. Em um dado instante, o microcontrolador está executando apenas uma atividade. Entretanto, as interrupções permitem ao processador executar múltiplas tarefas alternadamente de forma multiplexada no tempo.

Múltiplas tarefas podem ser mantidas de uma maneira preemptiva ou não preemptiva:

1. No cenário preemptivo, se uma nova tarefa tem maior prioridade que a tarefa corrente, esta nova tarefa pode interromper imediatamente a tarefa corrente. A nova tarefa assume o controle do processador. O processador retoma a tarefa antiga depois que a nova tarefa for concluída;
2. No cenário não preemptivo, a nova tarefa não pode parar a tarefa corrente até que essa última, voluntariamente, passe o controle para a nova tarefa.

Uma técnica alternativa de atendimento a um periférico é a "sondagem periódica" (*polling*). Nessa técnica, o processador continuamente consulta o periférico para verificar se um evento específico ocorreu e atende aquele evento, caso tenha ocorrido. A latência da detecção do evento é determinada pela periodicidade da sondagem. Nas interrupções, o processador fornece um mecanismo de hardware que permite a um dispositivo interno ou externo gerar um sinal para imediatamente informar ao processador que um evento ocorreu. Podemos assim dizer que uma interrupção é simplesmente uma chamada de função feita pelo hardware.

Usemos como exemplo uma chamada telefônica para comparar a eficiência das técnicas de *pooling* com interrupções no atendimento de eventos oriundos de periféricos. Suponha que você esteja esperando uma ligação. No *pooling*, você pega o telefone a cada 10 segundos para verificar se tem alguém chamando. Na interrupção, você continua realizando suas tarefas cotidianas enquanto espera o telefone tocar. Quando o telefone finalmente toca, você é interrompido e pode parar a tarefa que estava executando e atender a ligação. Como você pode perceber, a técnica de *polling* é muito menos eficiente que as interrupções. Com o *polling*, você gasta tempo que poderia ser empregado em outras tarefas verificando repetidamente, na maioria das vezes sem sucesso, se uma chamada está sendo recebida.

## 7.2. Interrupções no STM32F407

O STM32F407 dispõe de um hardware controlador de interrupções, vetorado e aninhado (*NVIC - Nested Vectored Interrupt Controller*) para configuração e manipulação de diversas fontes de interrupções. Além das solicitações de interrupção, existem outros eventos que precisam ser atendidos pelo processador, os quais chamamos de "exceções". Na terminologia da arquitetura ARM, uma interrupção é um tipo específico de exceção. As partes do código do programa que atendem as exceções e interrupções, isto é, as sub-rotinas de software que são executadas quando um evento de interrupção acontece, geralmente são chamadas de manipuladores (*Handler*) de exceção.

No STM32F407, o NVIC recebe solicitações de interrupção de várias fontes, como mostrado na Figura 1.

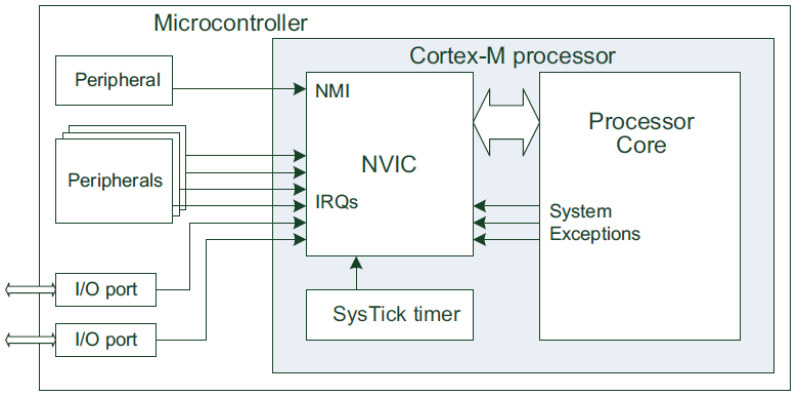


*Figura 1 – Várias fontes de interrupção do STM32F407.*

O controlador NVIC do Cortex-M4, embutido no STM32F407, pode suportar até 256 fontes de interrupções, apesar de conter apenas 92 fontes implementadas em hardware. Cada interrupção, exceto o *Reset*, que é um tipo especial de exceção, é identificada por um único número, que varia de -15 a 240. Os números identificadores de cada interrupção são definidos pela ARM e pelos fabricantes de chips, coletivamente. Estes números são fixados pelo hardware e não podem ser redefinidos.

As interrupções na arquitetura ARM Cortex-M são caracterizadas em dois grupos:

1. As primeiras 15 (numeradas de -15 a -1) são interrupções do sistema ou do núcleo do processador Cortex-M, também chamadas de “exceções” do sistema. Estes números são pré-definidos pela ARM;
2. As outras 241 interrupções são reservadas para os periféricos. A quantidade e os números identificadores das interrupções dos periféricos de um determinado microcontrolador são definidos pelo fabricante do chip específico.

Diferentes microcontroladores podem ter diferentes quantidades de fontes de interrupção implementadas fisicamente no chip e diferentes níveis de prioridade. Isso ocorre porque os fabricantes de chips podem configurar o design dos seus chips para diferentes requisitos de aplicações.

Dessa forma, na Figura 1 temos 241 fontes de solicitação de interrupção (*Interrupt Requests – IRQs)*, uma interrupção não mascarável (*Non-Maskable Interrupt - NMI*) e uma interrupção do temporizador SysTick, as duas últimas sendo exceções do sistema, além de várias outras exceções (*System Exceptions*). A maioria das *IRQs* é gerada por periféricos como portas GPIO, *timers*, conversores e interfaces de comunicação. A NMI é geralmente gerada a partir de periféricos como um temporizador *watchdog* ou um circuito do tipo *Brown-Out Detector* (BOD). As demais exceções são do próprio núcleo do processador e, também, podem ser geradas usando o software.

Para retomar um programa interrompido, a sequência de atendimento à exceção precisa armazenar o status do programa interrompido para que isso possa ser restaurado após a conclusão da sub-rotina da exceção. Em geral, isso pode ser feito por um mecanismo de hardware ou uma mistura de operações de hardware e software. Nos processadores Cortex-M, alguns dos registradores são salvos na pilha automaticamente quando uma exceção é aceita e são restaurados automaticamente em uma sequência de retorno de exceção. Esse mecanismo permite que os manipuladores de exceção sejam gravados como funções C normais, sem qualquer sobrecarga de software adicional.

As exceções também podem ser enumeradas de 1 a 15, para exceções do sistema, e de 16 a 256 para entradas de interrupção. Na maioria das exceções, incluindo todas as interrupções, é possível programar o nível de prioridade de atendimento enquanto algumas exceções do sistema têm prioridade fixa.

Os tipos de exceção de 1 a 15 são exceções do sistema, conforme descrito na Tabela 1. Exceções do tipo 16 ou superior são entradas externas de interrupção (consulte a Tabela 2).

**Tabela 1. Lista de exceções do sistema**

| Número da exceção | Tipo de exceção | Prioridade | Descrição |
|---|---|---|---|
| 1 | Reset | -3 (mais alta) | Reset |
| 2 | NMI | -2 | Interrupção não mascarável. Pode ser gerada por periféricos internos ou fontes externas |
| 3 | Hard Fault | -1 | Todas as condições de faltas internas |
| 4 | MemManage Fault | Programável | Gerenciamento de memória, violação de MPU ou acesso a endereços proibidos |
| 5 | Bus Fault | Programável | Erros de barramento |
| 6 | Usage Fault | Programável | Exceções devido a erros de programa ou tentativas de acessar coprocessadores não existentes no chip |
| 7-10 | Reservado | - | - |
| 11 | SVC | Programável | (*SuperVisor Call*) Chamada supervisória. Usualmente usada em sistemas operacionais para permitir que *tasks* de aplicativos acessem serviços do sistema |
| 12 | Debug Monitor | Programável | Exceções para eventos de *Debug* como *breakpoints e watchpoints*. |
| 13 | Reservado | - | - |
| 14 | PendSV | Programável | Chamada de serviço pendente. Usada por sistemas operacionais para realizar troca de contexto |
| 15 | SysTick | Programável | Exceção gerada pelo temporizador do sistema operacional ou como timer de uso geral em aplicações sem SO. |

**Tabela 2. Lista de interrupções**

| Número da exceção | Tipo de exceção | Prioridade | Descrição |
|---|---|---|---|
| 16 | Interrupção #0 | Programável | Gerada por periféricos internos ou outras fontes externas |
| 17 | Interrupção #1 | Programável | |
| ... | ... | ... | |
| 256 | Interrupção #240 | Programável | |

## 7.3. CMSIS e interrupções no STM32F407

O CMSIS (*Cortex Microcontroller Software Interface Standard*) é uma camada de software que propicia a abstração do hardware dos microcontroladores baseados em processadores ARM, como é o caso do STM32F407. Sua interface de software simplifica a reutilização do software e reduz a curva de aprendizado para desenvolvedores. O CMSIS fornece interfaces para processador, periféricos e sistemas operacionais em tempo real. No CMSIS, as exceções do sistema são enumeradas com valores negativos, conforme mostrado na Tabela 3.

O CMSIS também define os nomes dos manipuladores de exceção do sistema. As interrupções são específicas do dispositivo e são definidas nos arquivos de cabeçalho do dispositivo, fornecidos pelos fabricantes de microcontroladores. O número da interrupção (por exemplo, Interrupção #0) refere-se às entradas de interrupção no NVIC.

**Tabela 3. Definições de interrupções no CMSIS**

| Número da exceção | Tipo da exceção | Enumeração CMSIS (IRQn) | Enumeração CMSIS (valor) | ***Handler* da exceção** |
|---|---|---|---|---|
| 1 | Reset | - | -15 | *Reset_Handler( )* |
| 2 | NMI | NonMaskableInt_IRQn | -14 | *NMI_Handler( )* |
| 3 | Hard Fault | HardFAult_IRQn | -13 | *HardFault_Handler( )* |
| 4 | MemManage Fault | MemoryManagement_IRQn | -12 | *MemMenage_Handler( )* |
| 5 | Bus Fault | BusFault_IRQn | -11 | *BusFault_Handler( )* |
| 6 | Usage Fault | UsageFault_IRQn | -10 | *UsageFault_Handler( )* |
| 11 | SVC | SVCall_IRQn | -5 | *SVC_Handler( )* |
| 12 | Debug Monitor | DebugMonitor_IRQn | -4 | *DebugMon_Handler( )* |
| 14 | PendSV | PendSV_IRQn | -2 | *PendSV_Handler( )* |
| 15 | SysTick | SysTick_IRQn | -1 | *SysTick_Handler( )* |
| 16 | Interrupção #0 | Específico do dispositivo | 0 | Específico do dispositivo |
| 17 | Interrupção #1 | Específico do dispositivo | 1 | Específico do dispositivo |
| ... | ... | ... | ... | ... |
| 256 | Interrupção #240 | Específico do dispositivo | 240 | Específico do dispositivo |

Nas tabelas 1 e 2, a prioridade determina a ordem na qual as interrupções serão atendidas quando mais de um tipo ocorrer simultaneamente. O projeto dos processadores Cortex-M3 e Cortex-M4, por exemplo, suporta até 256 níveis de prioridade programável. O número real de níveis de prioridade programáveis disponíveis em um determinado microcontrolador é decidido pelo fabricante e depende da quantidade de periféricos disponíveis no chip específico.

A maioria dos chips Cortex-M3 ou Cortex-M4 têm uma quantidade menor de níveis de prioridade, por exemplo, 8, 16, 32 e assim por diante. Isso ocorre porque ter muitos níveis de prioridade pode aumentar a complexidade do NVIC e pode aumentar o consumo de energia. Na maioria dos casos, as aplicações requerem apenas um pequeno número de níveis de prioridade programáveis. Portanto, os projetistas de chips precisam personalizar o projeto do processador com base no número de níveis de prioridade nas aplicações alvo. Esta redução nos níveis é implementada cortando-se a parte menos significativa (LSB) dos registradores de configuração de prioridade.

O nível de prioridade de cada exceção pode ser programado por meio dos registradores SHP (*System Handlers Priority*) que ficam no bloco de controle do sistema (*SCB – System Control Block*), na última região do mapa de memória. Já o nível de prioridade de cada interrupção pode ser programado por meio dos registradores IP (*Interrupt Priority*), do mesmo bloco. Existem 12 registradores SHP e 240 registradores IP, onde cada um é utilizado para especificar a prioridade de uma determinada exceção ou interrupção, conforme a enumeração CMSIS da Tabela 3.

Especificamente, no STM32F407, é possível programar até 16 níveis (0-15) distintos de prioridade. Quanto menor o valor do nível, maior é a prioridade da interrupção. O valor zero tem a prioridade mais alta. Quando duas interrupções tem o mesmo valor no nível de prioridade nos registradores IP, aquela que tiver o menor número de exceção (coluna 1 da Tabela 3) será atendida primeiro.

Uma exceção de prioridade mais alta (número menor no nível de prioridade) pode antecipar uma exceção de prioridade mais baixa (número maior no nível de prioridade); esse é o cenário de exceção/interrupção aninhada. Algumas das exceções (*reset*, NMI e HardFault) têm níveis fixos de prioridade. Seus níveis de prioridade são representados com números negativos na enumeração CMSIS para indicar que elas são de maior prioridade. Outras exceções têm níveis de prioridade programáveis, que variam de 0 a 15. Após o reset, todas as interrupções, exceto o próprio Reset, são desativadas e recebem um valor de nível de prioridade igual a 0.

Antes de usar qualquer interrupção, é necessário:

- Configurar o nível de prioridade da interrupção (esta etapa é opcional);
- Configurar e ativar o controle de requisição de interrupção no periférico que demanda a interrupção;
- Habilitar a interrupção no NVIC ou no SCB (no caso de exceções).

Cada entrada de interrupção no NVIC tem seis bits de controle que estão alocados em registradores distintos:

- *Enable* bit, localizado no registrador *Interrupt Set Enable Register* (ISER);
- *Disable* bit, localizado no *Interrupt Clear Enable Register* (ICER);
- *Pending* bit, localizado no *Interrupt Set Pending Register* (ISPR);
- *Un-pending* bit, localizado no *Interrupt Clear Pending Register* (ICPR);
- *Active* bit, localizado no *Interrupt Active Bit Register* (IABR);
- *Software trigger* bit, localizado no *Software Trigger Interrupt Register* (STIR).

Cada um desses registradores é, na verdade, um array com 8 registradores, indexados a partir do zero. Por exemplo, o registrador ISER é formado pelos registradores ISER[0], ISER[1], ..., ISER[7], que podem habilitar qualquer uma das interrupções, já o registrador ICER é formado pelos registradores ICER[0], ICER[1], ..., ICER[7], que podem desabilitar qualquer uma das interrupções. O registrador ISER[0] possui os 32 bits de habilitação das 32 primeiras interrupções, o registrador ISER[1] possui os 32 bits de habilitação das 32 interrupções seguintes e assim, sucessivamente até o registrador ISER[7].

Para habilitar uma interrupção no NVIC, basta setar o bit correspondente no registrador ISER correspondente. Para desabilitar uma interrupção no NVIC, basta setar o bit correspondente no registrador ICER correspondente. Quando uma interrupção ocorre, o bit correspondente no registrador ISPR correspondente é setado automaticamente pelo hardware e permanece assim, indicando uma pendência, até que sua ISR seja executada. Para limpar uma pendência, basta setar o bit correspondente no registrador ICPR correspondente. Quando um bit é setado por software em algum registrador IABR ou STIR, o processador executa a ISR correspondente. Algumas exceções são ativadas apenas por hardware. Quando a interrupção é acionada, a rotina de serviço de interrupção (ISR) correspondente será executada (pode ser necessário resetar a solicitação de interrupção do periférico dentro da ISR).

Para facilitar o gerenciamento de interrupções e exceções, o CMSIS oferece várias funções de acesso aos registradores acima descritos. Para a programação geral de aplicações, a prática recomendada é usar as funções de acesso do CMSIS. As funções de controle de interrupção mais comumente usadas são mostradas na Tabela 4.

Você também pode acessar diretamente os registradores no NVIC ou no SCB, se necessário. Mas tal prática pode limitar a portabilidade do software ao portar o código de um microcontrolador Cortex-M para outro com um tipo de processador diferente.

**Tabela 4. Funções CMSIS comumente usadas no gerenciamento de interrupções no NVIC**

| **Função CMSIS** | **Descrição** |
|---|---|
| void NVIC_EnableIRQ (IRQn_Type IRQn) | Habilita uma interrupção |
| void NVIC_DisableIRQ (IRQn_Type IRQn) | Desabilita uma interrupção |
| void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority) | Especifica a prioridade de uma interrupção |

## 7.4. Tabela de vetores de interrupção

Existe uma rotina de serviço de interrupção (ISR) associada a cada tipo de interrupção ou exceção. O STM32F407 armazena o endereço inicial de cada ISR em um array especial chamado de tabela de vetores de interrupção (*Interrupt Vector Table*), que está localizada na memória de programa a partir do endereço 0x00000004. Como cada entrada na tabela representa um endereço de memória, cada entrada ocupa 4 bytes, como mostrado na Figura 2.

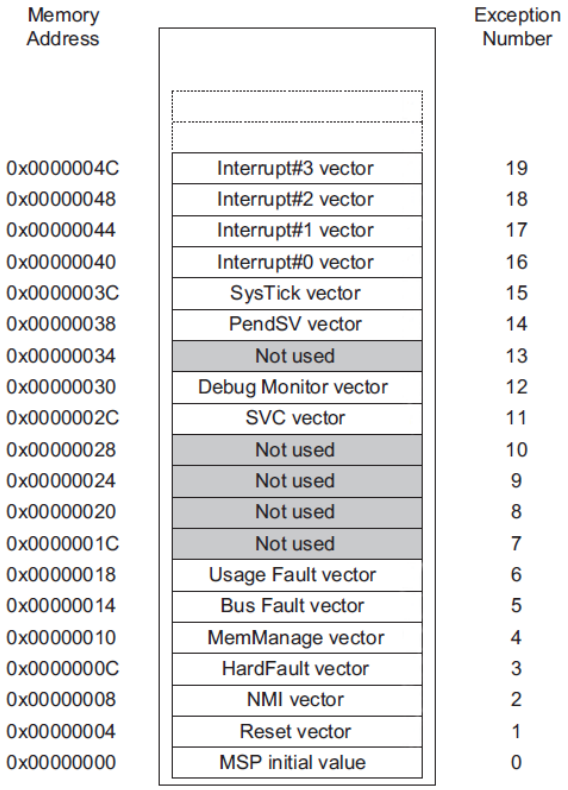


*Figura 2 – Tabela de vetores de interrupção.*

## 7.5. Controlador de interrupções/eventos externos no STM32F407

Interrupções externas são geradas por periféricos ou dispositivos externos, como botões e chaves conectados aos pinos do microcontrolador. Interrupções externas são muito úteis porque elas permitem que o microcontrolador monitore eficientemente sinais fornecidos aos pinos e prontamente responda a eventos externos.

Todos os pinos de uma porta GPIO podem ser fontes de requisição de interrupção externa. Para usarmos um pino como fonte de interrupção externa, devemos configurar o pino no modo de entrada digital, além de implementar as configurações específicas da interrupção.

O STM32F407 possui um controlador específico de interrupções/eventos externos conectado ao barramento APB2. O controlador consiste em 23 circuitos de detecção de borda usados para gerar solicitações de interrupção e enviá-las para o NVIC. Cada linha de interrupção pode ser configurada independentemente para selecionar o tipo de interrupção e o evento de acionamento correspondente (uma borda de subida, uma borda de descida ou ambas). Cada linha também pode ser habilitada/desabilitada de forma independente. Um registrador de pendências mantém o status das solicitações das linhas.

Os principais recursos do controlador de interrupções externas (EXTI) são os seguintes:

• disparo e mascaramento independentes em cada linha de interrupção
• bit de status dedicado para cada linha de interrupção
• geração de até 23 solicitações de interrupção por software

Na Figura 3 é mostrado o diagrama de blocos do controlador de interrupção/evento externo EXTI.

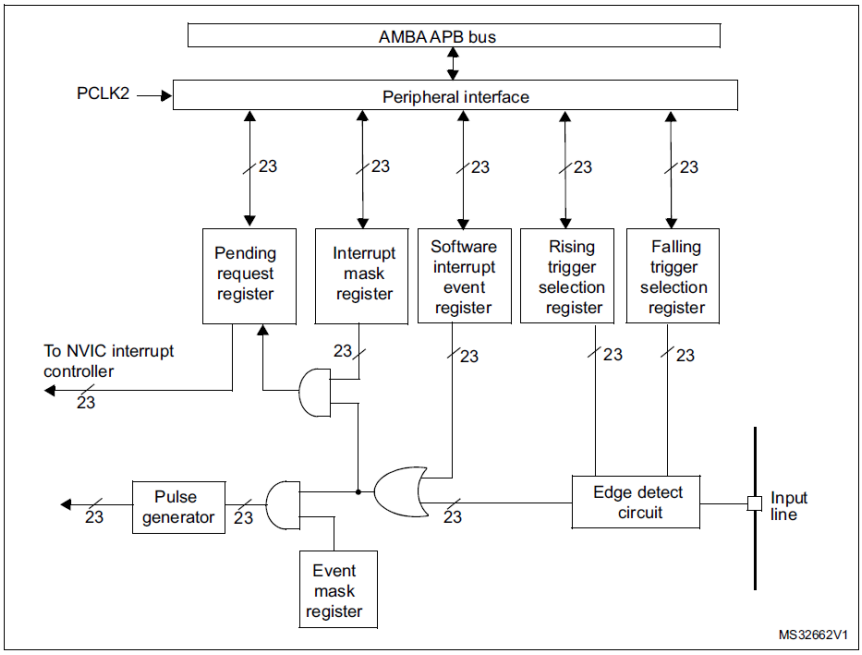


*Figura 3 – Diagrama de blocos do controlador de interrupção/evento externo (EXTI).*

Para gerar uma interrupção, a linha específica de interrupção deve ser configurada e ativada. Isso é feito programando os dois registradores de disparo com a detecção da(s) borda(s) desejada(s): borda de subida (*Rising Trigger Selection Register* – RTSR), borda de descida (*Falling Trigger Selection Register* - FTSR), e habilitando a solicitação de interrupção escrevendo um '1' no bit correspondente à linha no registrador da máscara de interrupção (*Interrupt Mask Register* – IMR). Quando as bordas selecionadas ocorrem na linha de entrada, uma solicitação de interrupção é gerada, o bit de pendência correspondente à linha de interrupção é setado no registrador de pendência (*Pending Request Register* – PR) e a solicitação é enviada ao NVIC. É necessário resetar essa solicitação de interrupção dentro da ISR escrevendo um '1' no registrador de pendência.

Além de requisições de interrupção, o controlador EXTI pode ser usado para gerar eventos. O objetivo principal da utilização de eventos, ao invés de interrupções externas, é fazer com que o processador possa ser acordado de estados de espera (*wake up*). As interrupções normalmente são usadas para executar ISRs enquanto os eventos podem sinalizar outro periférico para fazer algo sem a intervenção do processador. Por exemplo, um evento em EXTI pode inicializar uma amostragem no conversor ADC e, em seguida, gravar o valor medido na memória usando DMA. Dessa forma, as interrupções envolvem a execução de código. Os eventos não requerem necessariamente a execução de código.

Para gerar um evento, a linha de eventos deve ser configurada e ativada. Isso é feito programando os dois registradores de disparo com a detecção de borda desejada e ativando a solicitação de evento, escrevendo um '1' no bit correspondente no registrador de máscara de evento (*Event Mask Register* – EMR). Quando a borda selecionada ocorre na linha de entrada, um pulso de evento é gerado. O bit pendente correspondente à linha do evento não é resetado.

Uma solicitação de interrupção/evento também pode ser gerada pelo software, escrevendo um '1' no bit correspondente à linha específica no registrador de interrupção/evento de software (*Software Interrupt Event Register* - SWIER).

Até 80 pinos de GPIO são multiplexados e conectados a 16 das 23 linhas de interrupção externa, como mostrado na Figura 4. As outras sete linhas de interrupção externa são conectadas a outros periféricos como RTC, USB e Ethernet.

As 16 linhas de interrupções externas associadas aos pinos de I/O são nomeadas como EXTI0 (*External Interrupt 0*) até EXTI15 (*External Interrupt 15*). Cada uma dessas interrupções é associada com um pino de I/O específico. Porém, como o microcontrolador possui mais de 16 pinos de I/O, existem 16 multiplexadores que selecionam um dentre até 9 pinos de entrada. Cada multiplexador recebe os pinos de diferentes portas, mas que têm a mesma numeração, como mostrado na Figura 4. A fonte da interrupção EXTI0, por exemplo, pode ser selecionada como um dos pinos PA0, PB0, PC0, etc.

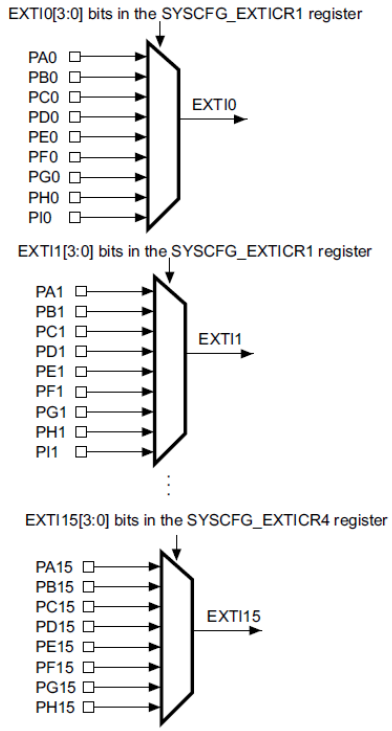


*Figura 4 – Mapeamento dos pinos de I/O nas linhas do controlador de interrupção/evento externo.*

A fonte (pino de I/O específico) da interrupção de cada linha EXTIx é selecionada por meio de quatro bits EXTIx[3:0] em um dos quatro registradores EXTICRx do módulo SYSCFG. Os 16 bits menos significativos de cada registrador EXTICRx formam os 4 campos de bits EXTIx[3:0]. Por exemplo, nos 16 primeiros bits do registrador EXTICR1 temos os campos EXTI0[3:0], EXTI1[3:0], EXTI2[3:0] e EXTI3[3:0], que selecionam, respectivamente, as fontes das interrupções das linhas EXTI0, EXTI1, EXTI2 e EXTI3.

No CMSIS, os quatro registradores EXTICR são definidos como um *array* de 4 registradores de 32 bits (EXTICR[0], EXTICR[1], EXTICR[2] e EXTICR[3]). Dessa forma, EXTICR[0] = EXTICR1, EXTICR[1] = EXTICR2 e assim sucessivamente.

Ao responder ao pedido de interrupção externa de uma linha específica, o microcontrolador executa a rotina de serviço associada àquela linha de interrupção. Por exemplo, à linha EXTI0 está associada à ISR *EXTI0_IRQHandler( )*.

Os nomes das ISRs podem ser encontrados na tabela de vetores dentro do código de inicialização, que também é fornecido pelo fornecedor do microcontrolador. O nome da ISR precisa corresponder ao nome usado na tabela de vetores para que o *linker* possa colocar corretamente o endereço inicial da ISR na tabela de vetores.

No caso do microcontrolador STM32F407, apesar de haver 16 linhas de interrupção externa, existem apenas 7 ISRs associadas a essas linhas:

- *EXTI0_IRQHandler( ),* para atendimento da interrupção em EXTI0;
- *EXTI1_IRQHandler( ),* para atendimento da interrupção em EXTI1;
- *EXTI2_IRQHandler( ),* para atendimento da interrupção em EXTI2;
- *EXTI3_IRQHandler( ),* para atendimento da interrupção em EXTI3;
- *EXTI4_IRQHandler( ),* para atendimento da interrupção em EXTI4;
- *EXTI9_5_IRQHandler ( ),* para atendimento das interrupções em EXTI5 a EXTI9;
- *EXTI15_10_IRQHandler ( ),* para atendimento das interrupções em EXTI10 a EXTI15;

Abaixo é mostrada uma sequência de configuração de software para selecionar um pino *k* de uma porta GPIO como a fonte de gatilho de uma interrupção externa EXTI*k*:

1. Habilite o clock do controlador de configuração do sistema (SYSCFG) e da porta GPIO correspondente;
2. Configure o pino *k* do GPIO como entrada digital;
3. Configure o registrador correspondente EXTICRx do módulo SYSCFG para rotear o pino *k* para a linha de entrada de interrupção *k,* EXTIk;
4. Selecione a(s) borda(s) ativa(s) para a interrupção EXTI*k* nos registradores RTSR e FTSR no controlador EXTI;
5. Habilite a interrupção da linha EXTI*k* no registrador IMR do controlador EXTI;
6. Selecione o nível de prioridade da interrupção EXTIk no registrador IP (*Interrupt Priority*) correspondente no controlador NVIC (use a função CMSIS *NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)*);
7. Habilite a interrupção da linha EXTI*k* no registrador ISER correspondente no controlador NVIC (use a função CMSIS *NVIC_EnableIRQ (IRQn_Type IRQn)*);
8. Escreva a rotina de atendimento à interrupção de EXTIk. Os nomes das rotinas de interrupção são pré-definidos no arquivo *startup_stm32.s* (cada microcontrolador tem um arquivo específico);
9. Na rotina de interrupção, o software precisa limpar o bit correspondente no registrador de pendência (PR) do controlador EXTI para indicar que o atendimento à interrupção foi concluído. Isso é feito escrevendo um '1' no bit correspondente do registrador PR.

A seguir, é mostrado um exemplo de código que configura o pino PA0 como fonte de interrupção da linha de interrupção externa EXTI0. A função *Configure_EXTI0( )* configura o pino PA0 como entrada digital com resistor de *pull-down*, seleciona a borda de subida do sinal em PA0 como gatilho, configura nível 0 de prioridade no atendimento da ISR e habilita a interrupção no controlador EXTI e no NVIC. A função de atendimento à interrupção de EXTI0, *EXTI0_IRQHandler( ),* alterna o estado da saída PA6 toda vez que ocorrer uma interrupção. O pino PA6 deve ter sido previamente configurado como saída digital.

```
void Configure_EXTI0()
{
  RCC->AHB1ENR |= 1;              //habilita o clock do GPIOA

  //Configuração do pino PA0 como entrada digital com resistor de pull-down
  GPIOA->MODER &= ~(0b11);        //seleciona modo de entrada digital no pino PA0
  GPIOA->PUPDR |= 0b10;           //habilita o resistor de pull-down no pino PA0

  //Configuração da interrupção em EXTI0
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN; //habilita o clock de SYSCFG
  SYSCFG->EXTICR[0] &= ~(0xF);          //seleciona PA0 como gatilho de EXTI0
  EXTI->RTSR |= 1;                      //seleciona borda de subida
  EXTI->IMR |= 1;                       //habilita a interrupção EXTI0 em EXTI
  NVIC_SetPriority(EXTI0_IRQn, 0);      //nível de prioridade 0 para EXTI0_IRQn
  NVIC_EnableIRQ(EXTI0_IRQn);           //habilita a interrupção EXTI0 no NVIC
}

void EXTI0_IRQHandler()
{
  GPIOA->ODR ^= 1 << 6;           //alterna o estado da saída PA6
  EXTI->PR |= 1;                  //limpa o flag de pendência em EXTI0
}
```

As outras sete linhas de interrupção externa (EXTI16 – EXTI22) são conectadas em outros periféricos conforme listado abaixo:

- EXTI16 – conectada à saída do PVD (*Programmable Voltage Detector*);
- EXTI17 – conectada a eventos de saída de alarme do relógio de tempo real (RTC - *Real Time Clock*);
- EXTI18 – conectada à saída de eventos do módulo USB;
- EXTI19 – conectada à saída de eventos do módulo Ethernet;
- EXTI20 – conectada à saída de eventos do módulo USB quando configurado em *full speed*;
- EXTI21 – conectada à saída de eventos de *Tamper Detection* e *Time Stamps* do RTC;
- EXTI22 – conectada à saída de eventos de *Wakeup* do RTC. ■