



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga Lab 9**

## **E/S Orientadas às Interrupções**

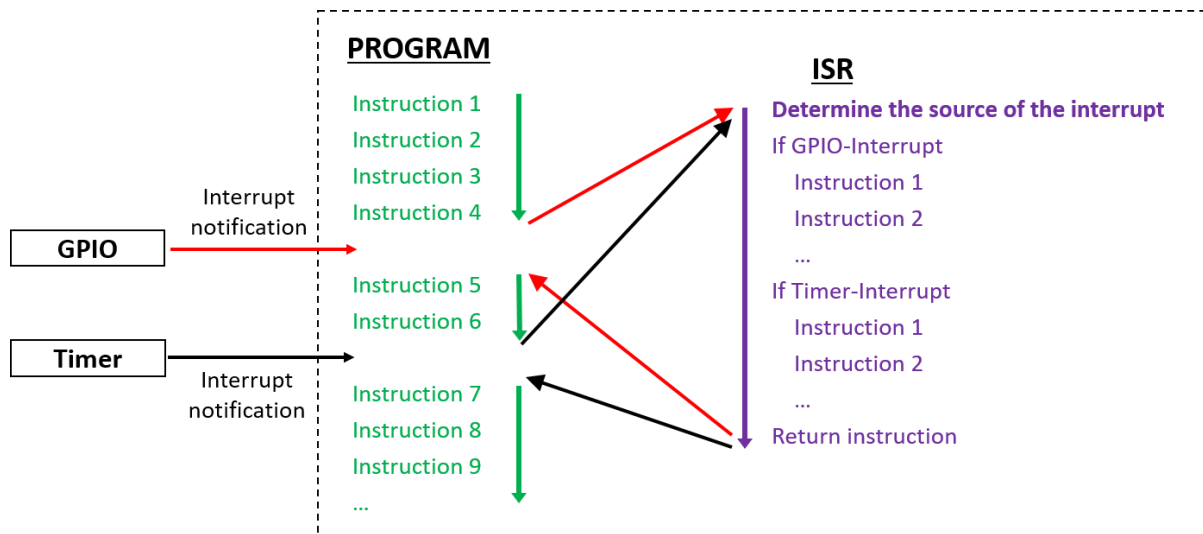
## 1. INTRODUÇÃO

Neste laboratório, introduzimos o conceito de interrupções e mostramos como utilizá-las na RVfpga. As interrupções podem ser geradas por software ou hardware. Neste laboratório, concentramo-nos nas interrupções de hardware que são desencadeadas pela mudança de valor de um pino físico. Especificamente, começamos na Secção 2 descrevendo as diferenças entre **E/S programada** e **E/S orientada à interrupção**. Em seguida, explicamos o funcionamento do Controlador de Interrupção do Sistema RVfpga, que faz parte do núcleo SweRV EH1 (Secção 3). Na Secção 4 descrevemos como configurar as interrupções externas utilizando o Pacote de Suporte de Periféricos (PSP) e o Pacote de Suporte da Placa (BSP) da Western Digital, que são software que incluem controladores para periféricos de hardware. Finalmente, introduzimos alguns programas de exemplo (Secção 5) e propomos alguns exercícios (Secção 6) para a utilização e extensão das interrupções de hardware do Sistema RVfpga.

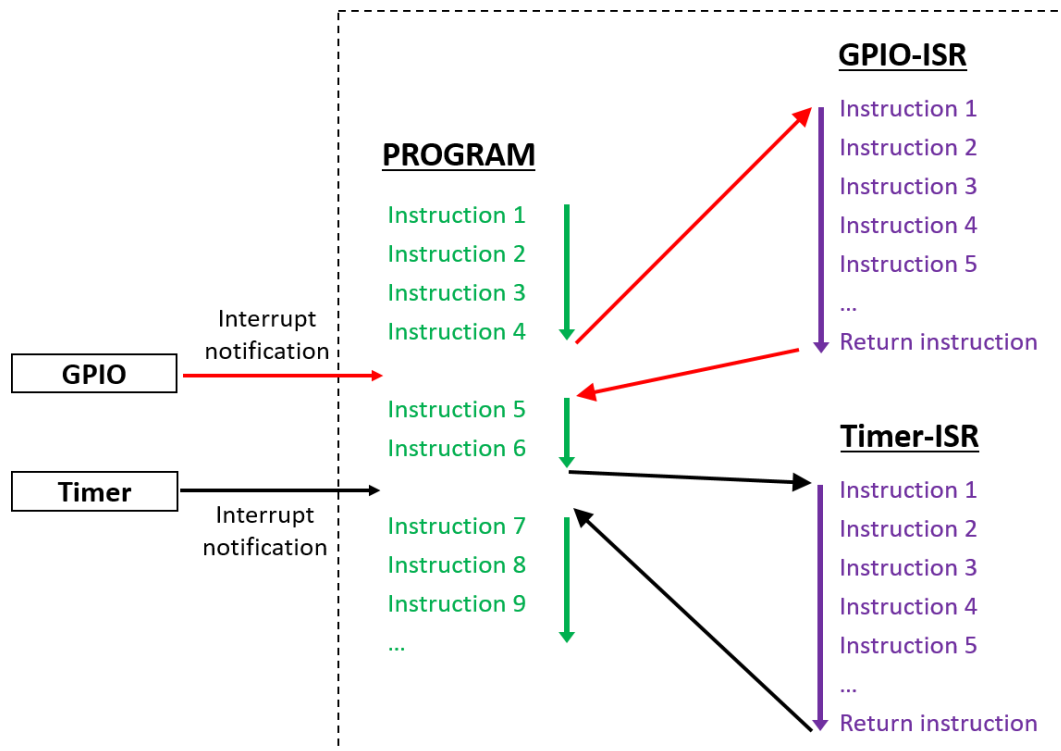
## 2. E/S PROGRAMADA VS. E/S COM INTERRUPÇÕES

Existem vários métodos para interagir com os periféricos: E/S programadas, E/S com Interrupção, e Acesso Direto à Memória (DMA). Nos laboratórios 2-8, utilizamos **E/S programadas** para interagir com periféricos. Na E/S programada, o programa do utilizador faz pedidos constantes à interface de E/S e, dependendo do seu estado, reage em conformidade. Por exemplo, o Exercício Fundamental do Laboratório 6 utilizou E/S programadas através de leitura contínua (*polling*) de interruptores 0 e 1 para controlar a velocidade e direção de um bloco de quatro LEDs acesos que se moviam repetidamente de um lado dos LEDs para o outro. A E/S programada é muito simples de implementar e requer muito pouco suporte de hardware, mas a leitura contínua da interface de E/S mantém o processador ocupado a fazer trabalho inútil.

A **E/S provocada por uma interrupção** supera este inconveniente e permite ao programa reagir apenas quando um evento ocorre no periférico. Neste esquema, o periférico é responsável por enviar um sinal (**chamado de interrupção**) para o processador quando algum evento ocorre - por exemplo, atingir o limite de contagem de um temporizador, um carácter a ser recebido numa interface UART, um botão a alternar, etc. Quando nenhum evento ocorre (ou seja, não há interrupção), o processador continua a fazer um trabalho útil. Quando o processador recebe uma interrupção, interrompe o programa que estava a executar e invoca uma rotina de atendimento de interrupção ou *Interrupt Service Routine* (ISR), também chamada de gestor de interrupção. Uma ISR é essencialmente uma função com argumentos nulos que trata da interrupção - ou seja, lê o novo valor do botão, faz alguma ação relacionada com o limite do temporizador, etc. Os processadores suportam normalmente os modos mono e multi-vetorial. No modo mono-vetorial (Figura 1), todas as interrupções invocam a mesma ISR. Assim, quando ocorre uma interrupção, o processador pára o programa principal e salta para a ISR comum, que primeiro determina a fonte de interrupção e depois executa o código ISR específico que corresponde à causa de interrupção identificada. No modo multi-vetorial (Figura 2), cada interrupção invoca uma ISR diferente. Assim, quando é gerada uma interrupção, a causa da interrupção é determinada primeiro, e depois o programa salta para o ISR que corresponde à causa identificada.



**Figura 1. Exemplo com 2 interrupções no modo de vetor único**



**Figura 2. Exemplo com 2 interrupções em modo multi-vetor**

Os processadores geralmente permitem que as interrupções sejam priorizadas. Não só as interrupções com maior prioridade serão tratadas primeiro, como uma interrupção com prioridade mais elevada evitará uma interrupção com prioridade mais baixa que estava em processo de tratamento. Por exemplo, suponha-se que uma interrupção de botão é definida para a prioridade 5, uma interrupção de temporizador é definida para a prioridade 7 e o limiar é definido para 4 (por isso ambas as prioridades estão acima do limiar). Se o programa estiver a executar o seu fluxo normal e o botão for premido, ocorrerá uma interrupção e o processador chama a ISR, que lê os dados a partir do botão e trata dos mesmos. Se um temporizador exceder o tempo, o fluxo enquanto o botão ISR estiver ativo,

a própria ISR será interrompida para que o processador possa lidar imediatamente com o excesso do temporizador. Quando estiver feito, voltará a terminar a interrupção do botão antes de voltar ao programa principal<sup>1</sup>.

### 3. CONTROLADOR DE INTERRUPTO PROGRAMÁVEL NO SWERV EH1

O core SweRV EH1 suporta interrupções como descrito nas seguintes referências e como resumido abaixo:

- **[PRM v1.7]** Revision 1.7 (June 25, 2020), Capítulo 6, “RISC-V SweRV EH1 Programmer’s Reference Manual”, disponível em [https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V\\_SweRV\\_EH1\\_PRM.pdf](https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf)
- **[ISM v1.11]** Version 1.11-draft (December 1, 2018), Capítulo 7, “The RISC-V Instruction Set Manual – Volume II: Privileged Architecture”, disponível em <https://github.com/riscv/riscv-isa-manual/releases/tag/draft-20181201-2650e2a>

As interrupções externas no núcleo SweRV EH1 (ver [PRM v1.7]) são modeladas em grande parte após a especificação RISC-V PLIC (Platform-Level Interrupt Controller) (ver [ISM v1.11]). Contudo, o controlador de interrupção está associado ao núcleo e não à plataforma. Portanto, o termo mais geral PIC (Programmable Interrupt Controller) é usado para se referir ao controlador disponível no núcleo SweRV EH1. O PIC fornece as seguintes características principais:

- Suporta até 255 fontes de interrupção externas, de 1 (prioridade máxima) a 255 (prioridade mínima); cada fonte tem a sua própria ativação.
- Para além da numeração da origem, fornece 15 níveis de prioridade adicionais; estão disponíveis dois esquemas de prioridade: 1-15 (onde 1 é a prioridade mais baixa), ou 0-14 (onde 14 é a prioridade mais baixa). A cada uma das fontes pode ser atribuída uma prioridade.
- Fornece apoio para o limiar de prioridade programável para desativar as interrupções de menor prioridade.
- Apoio a interrupções externas vectoriais, encadeamento de interrupções, e interrupções aninhadas.

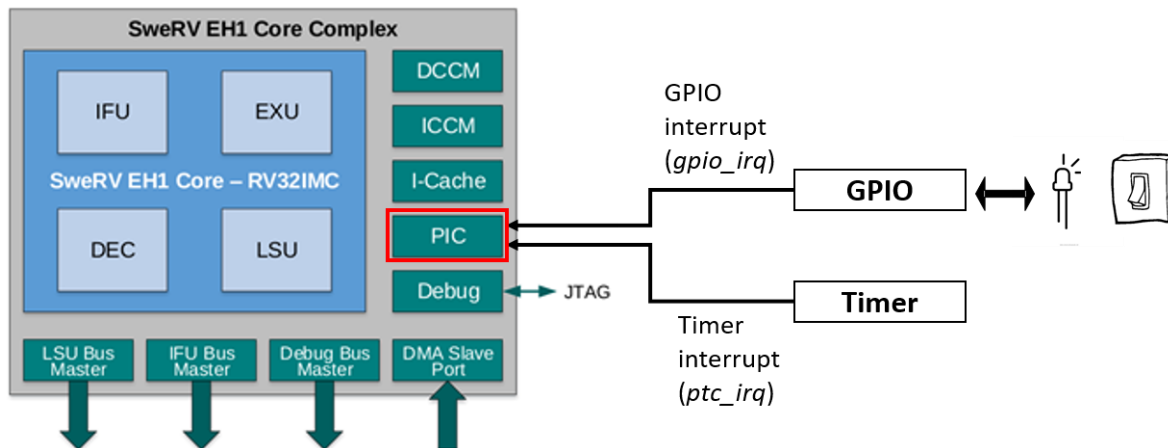
A Figura 3 ilustra uma versão simplificada do sistema de interrupção do Sistema RVfpga. Todas as unidades funcionais que geram interrupções são chamadas **fontes de interrupção externas**. As fontes de interrupção externa indicam um pedido de interrupção enviando um sinal assíncrono para o **PIC** com sinais terminando em `_irq` (uma abreviatura para pedido de interrupção). Neste laboratório, mostramos como usar interrupções do temporizador e do GPIO; estas unidades geram interrupções usando sinais `ptc_irq` e `gpio_irq`, respectivamente.

Cada fonte de interrupção externa liga-se a um gateway dedicado (localizado dentro do PIC), uma estrutura de hardware responsável por sincronizar o pedido de interrupção para o domínio do relógio do núcleo e para converter o sinal de pedido para um formato comum de pedido de interrupção (isto é, ativo à transição ascendente/descendente do sinal ou a nível

---

<sup>1</sup> D. Harris and S. Harris. “*Digital Design and Computer Architecture*”. Second Edition – 2012. Morgan Kaufmann Publishers (San Francisco, CA, United States). ISBN:978-0-12-394424-5.

alto/baixo) para o PIC. O PIC só pode lidar com um pedido de interrupção por fonte de interrupção de cada vez. O PIC avalia todos os pedidos pendentes e habilitados de interrupção e escolhe a interrupção de maior prioridade com a menor identificação de origem. Em seguida, compara esta prioridade com um limiar de prioridade programável e, para apoiar interrupções aninhadas, a prioridade do gestor de interrupções se estiver em funcionamento. Se a prioridade do pedido escolhido for superior a ambos os limiares, o PIC envia uma notificação de interrupção para o núcleo, o que suspende a execução do programa principal e salta para a ISR correspondente, como ilustrado na Figura 1 (modo mono-vector) e Figura 2 (modo multi-vector).



**Figura 3. Sistema de interrupções do sistema do RVfpga**

As principais funcionalidades do PIC são resumidas nos seguintes passos básicos:

- 1) Habilitação/desabilitação: o PIC permite habilitar/desabilitar (permitir/mascarar) as interrupções externas
- 2) Configuração: o PIC pode ser configurado para atender interrupções externas com diferentes polaridades (nível alto ou baixo) ou tipo (sensíveis à transição ou nível). O PIC também permite alocar ISRs a diferentes endereços de memória.
- 3) Filtragem e atribuições prioritárias: o PIC permite atribuir níveis prioritários a interrupções. Quando o programa principal está em execução, o PIC seleciona a interrupção ativada e desencadeada com o nível de maior prioridade.
- 4) Notificação: uma vez que o PIC selecione a interrupção com a maior prioridade, notifica o núcleo para interromper a execução do programa principal, a fim de saltar para a rotina que serve a interrupção escolhida.
- 5) Preempção: se as interrupções aninhadas estiverem ativadas, é possível suspender temporariamente a interrupção que está a ser atendida para ir atender outra com uma prioridade maior.

#### 4. CONFIGURAÇÃO DE INTERRUPTÕES EXTERNAS NO SWERV EH1

Da mesma forma que qualquer outro periférico, o PIC é configurado utilizando registos mapeados em memória que são acessíveis ao utilizador através de instruções de leitura/escrita. A utilização do sistema de interrupção a nível de registo seria possível, mas muito complexa; felizmente, o *Processor Support Package* (PSP) e o *Board Support Package* (BSP) da WD (<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>) incluem várias funções que fornecem uma abordagem muito mais simples para implementar programas usando interrupções. A Tabela 1 descreve as principais funções e macros que são necessárias para configurar as interrupções externas. Por uma questão de

abrangência, o apêndice no final deste documento fornece uma descrição dos diferentes registos disponíveis e as etapas para a configuração a nível de registo e utilização do PIC.

**Tabela 1. Funções básicas e macros usadas para configurar interrupções externas**

<b>Cabeçalho</b>	<b>Descrição</b>
<code>void pspInterruptsSetVectorTableAddress(void* pVectTable);</code>	Prepara o endereço da tabela de vectores
<code>void pspExternalInterruptSetVectorTableAddress(void* pExtIntVectTable);</code>	Prepara o endereço da tabela de vectores das interrupções externas
<code>void bspInitializeGenerationRegister(u32_t uiExtIntInterruptPolarity)</code>	Colocar o Generation Register no seu estado inicial
<code>void bspClearExtInterrupt(u32_t uiExtInterruptNumber)</code>	Libertar o mecanismo que gera a interrupção externa
<code>void pspExtInterruptSetPriorityOrder(u32_t uiPriorityOrder);</code>	Define a ordem das prioridades (Standard ou Reservado)
<code>void pspExtInterruptsSetThreshold(u32_t uiThreshold);</code>	Define o limiar de prioridade das interrupções externas no PIC
<code>void pspExtInterruptsSetNestingPriorityThreshold(u32_t uiNestingPriorityThreshold);</code>	Define o limiar de prioridade de nidificação das interrupções externas no PIC
<code>void pspExtInterruptSetPolarity(u32_t uiIntNum, u32_t uiPolarity);</code>	Define a polaridade de ativação (alta ou baixa) de um sinal de interrupção especificado
<code>void pspExtInterruptSetType(u32_t uiIntNum, u32_t uiIntType);</code>	Define o tipo (Acionado por nível ou por transição) de um sinal de interrupção especificado
<code>void pspExtInterruptClearPendingInt(u32_t uiIntNum);</code>	Elimina a indicação de interrupção pendente para o sinal de interrupção especificado
<code>void pspExtInterruptSetPriority(u32_t uiIntNum, u32_t uiPriority);</code>	Define a prioridade de um sinal de interrupção especificado
<code>void pspExternalInterruptEnableNumber(u32_t uiIntNum);</code>	Habilitar no PIC o sinal de interrupção especificado
<code>void pspInterruptsEnable(void);</code>	Habilitar interrupções (em todos os níveis de privilégio) independentemente do seu estado anterior
<code>void pspInterruptsDisable(u32_t *pOutPrevIntState);</code>	Desativa as interrupções e devolve o estado de interrupção atual em cada um dos níveis privilegiados

Exemplos de rotinas de serviço de interrupção (ISRs) são apresentadas mais tarde neste laboratório. Elas seguem os passos descritos abaixo para configurar as interrupções do Sistema RVfpga, com base nas funções da Tabela 1. De notar que, para além da configuração do PIC, os periféricos que geram a interrupção externa também devem ser configurados (isto será descrito mais tarde para cada um dos periféricos utilizados nos exemplos e exercícios).

#### **INICIALIZAÇÃO PADRÃO DO SISTEMA DE INTERRUPTÃO:**

1. No modo multi-vectorial, definir o endereço base da tabela de endereços de interrupção vectorial externa. Utilizar as funções `pspInterruptsSetVectorTableAddress` e `pspExternalInterruptSetVectorTableAddress`.
2. Colocar o Generation Register no seu estado inicial. Utilizar a função `bspInitializeGenerationRegister`.
3. Certificar-se de que os acionadores de Interrupção Externa estão limpos. Utilizar a

função `bspClearExtInterrupt`.

4. Definir valores por omissão para a ordem de prioridade (função `pspExtInterruptSetPriorityOrder`), limiar (função `pspExtInterruptsSetThreshold`) e limiar de prioridade de aninhamento (função `pspExtInterruptsSetNestingPriorityThreshold`).

### **INICIALIZAÇÃO DE CADA FONTE DE INTERRUPÇÃO:**

1. Para cada fonte de interrupção, definir a polaridade (ativa nível alto/baixo) e tipo (acionado a nível/transição) utilizando as funções `pspExtInterruptSetPolarity` e `pspExtInterruptSetType`
2. Limpar qualquer interrupção pendente usando a função `pspExtInterruptClearPendingInt`.
3. Definir o nível de prioridade para cada fonte de interrupção externa, utilizando a função `pspExtInterruptSetPriority`.
4. Ativar as interrupções para a fonte de interrupção externa apropriada, utilizando a função `pspExternalInterruptEnableNumber`.
5. No modo multi-vectorial, para cada fonte de interrupção externa, escrever o endereço do gestor correspondente na tabela de endereços de interrupção vectorial externa.

**TAREFA AVANÇADA:** A fim de obter uma compreensão mais profunda sobre estas funções básicas, veja o código PSP localizado em `.platformio/packages/framework-wd-riscv-sdk/psp` e o Código do BSP localizado em `.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp`. De especial interesse são os ficheiros listados em baixo, alguns dos quais contidos na subpasta `api_inc`.

- `bsp_external_interrupts.h`: criação de `external_interrupts` no RVfpga
- `psp_interrupts_eh1.h`: fornece APIs de informação e registo para ISRs no núcleo EH1
- `psp_ext_interrupts_eh1.h`: define as interfaces de interrupções externas PSP para SweRV EH1
- `psp_macros_eh1.h`: define as macros PSP para SweRV EH1
- `psp_csrs_eh1.h`: definições de CSRs SweRV EH1

Também se recomenda analisar pelo menos uma destas funções até ao nível do registo. Para este efeito, pode utilizar a informação fornecida no Apêndice, que descreve como o PIC do SweRV EH1 Core configura e gere as interrupções externas a nível de registo.

**TAREFA AVANÇADA:** Também recomendamos que analise e execute a demonstração de interrupções externas fornecida pela Western Digital em <https://github.com/westerndigitalcorporation/riscv-fw-infrastructure> e disponível como um projeto PlatformIO em: `[RVfpgaPath]/RVfpga/Labs/Lab9/WD_demo_external_int_Original`. Se tudo funcionar corretamente, deverá ver as seguintes mensagens na consola série:

```

Hello from SweRV core running on NexysA7
Core list:
    EH1 = 11
    EL2 = 16
Running demo on core 11...
-----
SweRVolf version 255.255255 (SHA 000000ef) (dirty 128)
-----
External Interrupts tests passed successfully

```

## 5. EXEMPLOS

Nesta secção, damos exemplos de conversão de programas de E/S programados em programas de E/S com interrupção. Mostramos três exemplos que ilustram os diferentes problemas inerentes à Programação de E/S (primeiro e segundo exemplos) e depois mostramos como estes problemas podem ser facilmente resolvidos através da utilização de um esquema de E/S com Interrupção (terceiro exemplo).

### **A. Programa LED-Switch em C**

O programa *LED-Switch\_C-Lang* (ver Figura 4) inverte o estado do LED mais à direita sempre que ocorre uma transição 0→1 no interruptor mais à direita. O programa está disponível em:

*[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch\_C-Lang.c*

Após a inicialização dos periféricos, o programa entra num ciclo infinito que compara o estado atual do interruptor com o estado anterior e, no caso de ser detectada uma transição de 0→1, inverte o estado do LED (note que, quando ocorre uma transição de 1→0, nada acontece).

Nos exemplos e exercícios anteriores escritos em C, definimos macros para aceder aos registos de E/S (`READ_GPIO`, `READ_Reg`, `WRITE_GPIO`, `WRITE_Reg`, etc.). Neste exemplo, utilizamos duas macros definidas no PSP para o mesmo fim: `M_PSP_READ_REGISTER_32`, lê um registo de 32 bits fornecido como argumento, e `M_PSP_WRITE_REGISTER_32`, escreve um registo de 32 bits com o valor fornecido no segundo argumento. Lembre-se de que, para poder utilizar estas macros, deve incluir linha `framework = wd-riscv-sdk` no ficheiro *platformio.ini* (este é o ficheiro predefinido quando um projeto é criado com a RVfpga como alvo) e linha `#include "psp_api.h"` no início do programa (Figura 4, linha 1).

```

1  #include "psp_api.h"
2
3  #define GPIO_SWs    0x80001400
4  #define GPIO_LEDs    0x80001404
5  #define GPIO_INOUT  0x80001408
6
7  int main ( void )
8  {
9      int LED_state, Sw_current_state, Sw_previous_state;
10
11      /* Configure LEDs and Switches */
12      M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF);
13
14      /* Init states */
15      LED_state = 0;
16      M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
17      Sw_previous_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
18
19      while (1) {
20          /* Invert LED-0 when SW-0 goes high */
21          Sw_current_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
22          if(Sw_current_state==1 && Sw_previous_state==0){
23              LED_state = !LED_state;
24              M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
25          }
26          Sw_previous_state = Sw_current_state;
27      }
28
29      return(0);
30 }

```

**Figura 4. Programa *LED-Switch\_C-Lang***

**TAREFA:** Analisar o programa *LED-Switch\_C-Lang* para o compreender em pormenor. Se necessário, pode utilizar o depurador para analisar o programa passo a passo.

O programa funciona corretamente, mas é muito ineficiente, já que o processador nada mais faz do que ler/escrever os interruptores/LEDs. Obviamente, queremos que o nosso processador faça mais coisas do que apenas comunicar com os dispositivos de E/S.

### **B. Programa *LED-Switch\_7SegDispl\_C-Lang***

Neste segundo exemplo, *LED-Switch\_7SegDispl\_C-Lang*, o programa estende o *LED-Switch\_C-Lang* com um segundo periférico: os mostradores de 7 segmentos. O programa executa duas tarefas:

- Como no primeiro exemplo, inverte o LED mais à direita sempre que ocorre uma transição de 0→1 no interruptor mais à direita.
- Mostra uma contagem ascendente nos mostradores de 8 dígitos de 7 segmentos, que aumenta cerca de uma vez por segundo. Note que, para simplificar, criamos o atraso de um segundo com um ciclo `for` (no Exercício 1, utilizará o temporizador do Laboratório 8 para este fim).

Pode ver este programa na Figura 5 e pode encontrá-lo em:

*[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch\_7SegDispl\_C-Lang.c*

Após algumas inicializações, o programa entra num ciclo infinito que compara o estado atual do interruptor com o anterior e, caso seja detectada uma transição de 0→1, inverte o estado do LED. Depois, o valor mostrado nos mostradores de 8 dígitos de 7 segmentos é incrementado e é gerado um atraso. Ver a caixa vermelha na Figura 5.

```

1  #include "psp_api.h"
2
3  #define SegEn_ADDR    0x80001038
4  #define SegDig_ADDR   0x8000103C
5
6  #define GPIO_SWs      0x80001400
7  #define GPIO_LEDs     0x80001404
8  #define GPIO_INOUT    0x80001408
9
10 int main ( void )
11 {
12     int i, LED_state, Sw_current_state, Sw_next_state, count=0;
13
14     /* Configure LEDs and Switches */
15     M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF);
16
17     /* Configure 7-Seg Displays */
18     M_PSP_WRITE_REGISTER_32(0x80001038, 0x0);
19
20     /* Init states */
21     LED_state = 0;
22     M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
23     Sw_current_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
24
25     while (1) {
26         /* Invert LED-0 when SW-0 goes high */
27         Sw_next_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
28         if(Sw_current_state==0 && Sw_next_state==1){
29             LED_state = !LED_state;
30             M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
31         }
32         Sw_current_state = Sw_next_state;
33
34         /* Increase 7-Seg Displays */
35         M_PSP_WRITE_REGISTER_32(SegDig_ADDR, count);
36         count++;
37
38         /* Delay */
39         for(i=0;i<1000000;i++);
40     }
41
42     return(0);
43 }

```

Figura 5. Programa **LED-Switch\_7SegDispl\_C-Lang**

**TAREFA:** Analisar o programa **LED-Switch\_7SegDispl\_C-Lang** para o compreender em pormenor. Se necessário, pode utilizar o depurador para analisar o programa passo a passo.

Note que, neste caso, o programa nem sequer funciona corretamente nalgumas situações. Por exemplo, uma transição do interruptor 0→1→0 que ocorra dentro do loop de atraso nunca será detectada. Além disso, continuamos a ter o mesmo problema do exemplo anterior: o processador está sempre ocupado apenas a ler/escrever os dispositivos ou a criar um atraso.

Como é que podemos melhorar estas situações? A resposta é **E/S orientada a interrupções**. No exemplo seguinte e nos exercícios propostos na próxima secção, mostramos como resolver todos estes problemas e implementar programas que são mais eficientes e funcionam corretamente em todas as situações.

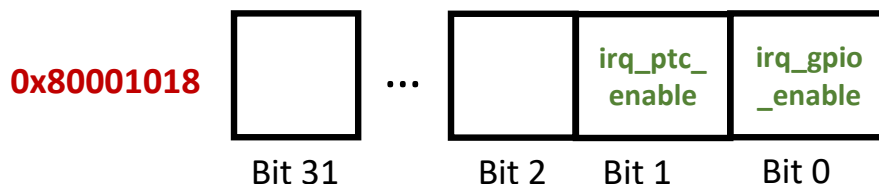
### C. Programa **LED-Switch\_7SegDispl Interrupts C-Lang**

Neste exemplo final ([RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch\_7SegDispl Interrupts\_C-Lang.c), mostramos como usar E/S controlada por interrupções para ler o estado do interruptor mais à direita. O uso dessa estratégia resolve o problema do programa perder as transições do interruptor que ocorrem durante o ciclo de atraso. Note, no entanto, que o problema de ter o processador ocupado num ciclo de atraso ainda persiste. (Você lidará com esse problema no Exercício 1.)

A nova função **main**, apresentada na Figura 7, executa as seguintes tarefas:

- Inicializar o sistema de interrupções:
  - o Inicialização por predefinição das interrupções: invocar a função

- DefaultInitialization (linha 119), que mostramos na Figura 8.
- Definir um limiar específico, invocando a função `pspExtInterruptsSetThreshold(5)` (linha 120). As interrupções externas cuja prioridade não seja superior a este limiar serão ignoradas.
  - Inicializar a linha de interrupção externa IRQ4:
    - Inicializar a linha IRQ4: invocar a função `ExternalIntLine_Initialization` (linha 123) para a linha de interrupção 4, com uma prioridade de 6 e `GPIO_ISR` como a Rotina de Serviço de Interrupção (*Interrupt Service Routine*). Analisamos esta função em Figura 9.
    - Ligar o IRQ4 à linha de interrupção GPIO (linha 124). Isto é feito definindo o bit 0 da palavra `0x80001018` (identificada como `Select_INT` no exemplo). Este registo de memória do controlador de sistema contém 2 bits (ver Figura 6): bit 0, chamado *irq\_gpio\_enable*, usado para ligar a linha de interrupção da GPIO com a IRQ4 quando está em 1; e o bit 1, chamado *irq\_ptc\_enable*, usado para ligar a linha de interrupção do timer com a IRQ3 quando está em 1. Por agora, é suficiente que conheça esta funcionalidade de alto nível; mais tarde, no Exercício 2, explicamos a implementação Verilog em pormenor, para que a possa modificar como parte desse exercício.



**Figura 6. Registo 0x80001018 do Sistema RVfpga.**

- Inicializar os periféricos (neste exemplo, o GPIO e os mostradores de 7 segmentos):
  - Invocar a função `GPIO_Initialization` na linha 127. Analisamos essa função em Figura 10.
  - Ativar os oito mostradores de 7 segmentos (linha 128).
- Ativar as interrupções:
  - Invocar a função `pspInterruptsEnable` (linha 131) e a macro `M_PSP_SET_CSR` (linha 132). As constantes `D_PSP_MIE_NUM` e `D_PSP_MIE_MEIE_MASK` são definidos pelo PSP da WD.
- Finalmente, os mostradores de 7 segmentos são escritos e um atraso é estabelecido dentro de um ciclo que se repete para sempre (linhas 134-141).

```

114 int main(void)
115 {
116     int count=0, i;
117
118     /* INITIALIZE THE INTERRUPT SYSTEM */
119     DefaultInitialization(); /* Default initialization */
120     pspExtInterruptsSetThreshold(5); /* Set interrupts threshold to 5 */
121
122     /* INITIALIZE INTERRUPT LINE IRQ4 */
123     ExternalIntLine_Initialization(4, 6, GPIO_ISR); /* Initialize line IRQ4 with a priority of 6. Set GPIO_ISR as the Interrupt Service Routine */
124     M_PSP_WRITE_REGISTER_32(Select_INT, 0x1); /* Connect the GPIO interrupt to the IRQ4 interrupt line */
125
126     /* INITIALIZE THE PERIPHERALS */
127     GPIO_Initialization(); /* Initialize the GPIO */
128     M_PSP_WRITE_REGISTER_32(SegEn_ADDR, 0x0); /* Initialize the 7-Seg Displays */
129
130     /* ENABLE INTERRUPTS */
131     pspInterruptsEnable(); /* Enable all interrupts in mstatus CSR */
132     M_PSP_SET_CSR(D_PSP_MIE_NUM, D_PSP_MIE_MEIE_MASK); /* Enable external interrupts in mie CSR */
133
134     while (1) {
135         /* Increase 7-Seg Displays */
136         M_PSP_WRITE_REGISTER_32(SegDig_ADDR, count);
137         count++;
138
139         /* Delay */
140         for(i=0; i<50000000; i++);
141     }
142 }
143

```

**Figura 7. Função *main*.**

A função **DefaultInitialization**, mostrada na Figura 8, executa as etapas explicadas na secção 4, ponto "INICIALIZAÇÃO PREDEFINIDA DO SISTEMA DE INTERRUPTAÇÃO":

- Configura a tabela de vectores (linhas 53 e 56). Note-se que, neste exemplo, o vector `G_Ext_Interrupt_Handlers` armazena a tabela de vectores.
- Inicializa o registo utilizado para ativar os IRQs (linha 59).
- Limpa todas as interrupções externas (no nosso caso IRQ3 e IRQ4) nas linhas 61-65. As constantes `D_BSP_FIRST_IRQ_NUM` e `D_BSP_LAST_IRQ_NUM` são definidos pelo BSP da WD como 3 e 4, respectivamente.
- Estabelece o limiar e as prioridades por omissão (linhas 68, 71 e 74). Mais uma vez, as constantes utilizadas por estas funções estão definidas no PSP da WD.

```

48 void DefaultInitialization(void)
49 {
50     u32_t uiSourceId;
51
52     /* Register interrupt vector */
53     pspInterruptsSetVectorTableAddress(&M_PSP_VECT_TABLE);
54
55     /* Set external-interrupts vector-table address in MEIVT CSR */
56     pspExternalInterruptSetVectorTableAddress(G_Ext_Interrupt_Handlers);
57
58     /* Put the Generation-Register in its initial state (no external interrupts are generated) */
59     bspInitializeGenerationRegister(D_PSP_EXT_INT_ACTIVE_HIGH);
60
61     for (uiSourceId = D_BSP_FIRST_IRQ_NUM; uiSourceId <= D_BSP_LAST_IRQ_NUM; uiSourceId++)
62     {
63         /* Make sure the external-interrupt triggers are cleared */
64         bspClearExtInterrupt(uiSourceId);
65     }
66
67     /* Set Standard priority order */
68     pspExtInterruptSetPriorityOrder(D_PSP_EXT_INT_STANDARD_PRIORITY);
69
70     /* Set interrupts threshold to minimal (== all interrupts should be served) */
71     pspExtInterruptsSetThreshold(M_PSP_EXT_INT_THRESHOLD_UNMASK_ALL_VALUE);
72
73     /* Set the nesting priority threshold to minimal (== all interrupts should be served) */
74     pspExtInterruptsSetNestingPriorityThreshold(M_PSP_EXT_INT_THRESHOLD_UNMASK_ALL_VALUE);
75 }

```

**Figura 8. Função *DefaultInitialization***

A função **ExternalIntLine\_Initialization**, apresentada na Figura 9, executa os passos explicados na secção 4, abaixo do ponto "INICIALIZAÇÃO DE CADA FONTE DE INTERRUPTAÇÃO":

- Configura o tipo e a polaridade da interrupção IRQ4 (as constantes utilizadas por

estas funções são definidas pelo PSP do WD) e limpa quaisquer potenciais interrupções pendentes no gateway correspondente (linhas 81, 84 e 87).

- Define a prioridade de IRQ4 (linha 90).
- Ativa as interrupções IRQ4 no PIC na linha 93.
- Regista a Rotina de Atendimento à Interrupção de GPIO (*Interrupt Service Routine*) (GPIO\_ISR) na tabela de vectores (na linha 96), que é armazenada no vector G\_Ext\_Interrupt\_Handlers.

```

78 void ExternalIntLine_Initialization(u32_t uiSourceId, u32_t priority, pspInterruptHandler_t pTestIsr)
79 {
80     /* Set Gateway Interrupt type (Level) */
81     pspExtInterruptSetType(uiSourceId, D_PSP_EXT_INT_LEVEL_TRIG_TYPE);
82
83     /* Set gateway Polarity (Active high) */
84     pspExtInterruptSetPolarity(uiSourceId, D_PSP_EXT_INT_ACTIVE_HIGH);
85
86     /* Clear the gateway */
87     pspExtInterruptClearPendingInt(uiSourceId);
88
89     /* Set IRQ4 priority */
90     pspExtInterruptSetPriority(uiSourceId, priority);
91
92     /* Enable IRQ4 interrupts in the PIC */
93     pspExternalInterruptEnableNumber(uiSourceId);
94
95     /* Register ISR */
96     G_Ext_Interrupt_Handlers[uiSourceId] = pTestIsr;
97 }

```

**Figura 9. Função *ExternalIntLine\_Initialization***

A função **GPIO\_Initialization**, apresentado na Figura 10, executa as seguintes tarefas:

- Configurar os pinos GPIO como entrada/saída e inicializar os LEDs para 0 (linhas 103 e 104).
- Configurar as interrupções GPIO. (Para compreender melhor a funcionalidade de cada registo GPIO, utilize a Especificação do núcleo (*IP CORE*) GPIO, disponível em: [\[RVfpgaPath\]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/docs/gpio\\_spec.pdf.](#))
  - o **RGPIO\_INTE**: determina quais os pinos de uso geral que geram uma interrupção (linha 107).
  - o **RGPIO\_PTRIG**: determina a transição que gera uma interrupção (linha 108).
  - o **RGPIO\_INTS**: limpa as interrupções de todos os pinos (linha 109).
  - o **RGPIO\_CTRL**: o bit menos significativo deste registo permite a geração de interrupções (linha 110).

```

100 void GPIO_Initialization(void)
101 {
102     /* Configure LEDs and Switches */
103     M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF); /* GPIO_INOUT */
104     M_PSP_WRITE_REGISTER_32(GPIO_LEDS, 0x0); /* GPIO_LEDS */
105
106     /* Configure GPIO interrupts */
107     M_PSP_WRITE_REGISTER_32(RGPIO_INTE, 0x10000); /* RGPIO_INTE */
108     M_PSP_WRITE_REGISTER_32(RGPIO_PTRIG, 0x10000); /* RGPIO_PTRIG */
109     M_PSP_WRITE_REGISTER_32(RGPIO_INTS, 0x0); /* RGPIO_INTS */
110     M_PSP_WRITE_REGISTER_32(RGPIO_CTRL, 0x1); /* RGPIO_CTRL */
111 }

```

**Figura 10. Função *GPIO\_Initialization*.**

Finalmente, a ISR (i.e., a função **GPIO\_ISR** mostrada na Figura 11) é invocada quando uma interrupção é acionada no GPIO. Esta ISR (Interrupt Service Routine - Rotina de Atendimento de Interrupção) executa as seguintes tarefas:

- É lido o estado atual dos LEDs (linha 35).
- Os LEDs são invertidos e mascarados (linhas 36-37).
- Os LEDs são escritos com o novo valor (linha 38).
- A interrupção do GPIO é desativada (linha 41).
- A interrupção externa IRQ4 é limpa (linha 44).

```

30 void GPIO_ISR(void)
31 {
32     unsigned int i;
33
34     /* Write the LED */
35     i = M_PSP_READ_REGISTER_32(GPIO_LEDS);          /* RGPIO_OUT */
36     i = !i;                                          /* Invert the LEDs */
37     i = i & 0x1;                                    /* Keep only the right-most LED */
38     M_PSP_WRITE_REGISTER_32(GPIO_LEDS, i);         /* RGPIO_OUT */
39
40     /* Clear GPIO interrupt */
41     M_PSP_WRITE_REGISTER_32(RGPIO_INTS, 0x0);      /* RGPIO_INTS */
42
43     /* Stop the generation of the specific external interrupt */
44     bspClearExtInterrupt(4);
45 }

```

**Figura 11. Função *GPIO\_ISR*.**

**TAREFA:** Analisar o programa *LED-Switch\_7SegDispl\_Interrupts\_C-Lang* para o compreender em pormenor. Pode comparar a implementação com as explicações da Secção 4 e, se necessário, utilizar o depurador para analisar o programa passo a passo.

## 6. EXERCÍCIOS

**Exercício 1.** Modifique o programa *LED-Switch\_7SegDispl\_Interrupts\_C-Lang* para incluir uma segunda fonte de interrupção, neste caso gerada pelo temporizador. Recorde-se que um temporizador pode atuar como gerador de PWM, temporizador ou contador, pelo que é geralmente referido como uma unidade PTC.

- No sistema RVfpga, a interrupção do temporizador está ligada à IRQ3 através da definição do bit 1 (*irq\_ptc\_enable*) da palavra 0x80001018 (ver Figura 6).
- Crie uma função que inicialize as interrupções PTC, semelhante a *GPIO\_Initialization* no exemplo anterior.
- Crie uma segunda ISR chamada *PTC\_ISR*. Deve ser semelhante a *GPIO\_ISR* no programa *LED-Switch\_7SegDispl\_Interrupts\_C-Lang*, mas deve ser invocada utilizando a IRQ3. *PTC\_ISR* deve tratar e limpar a interrupção do temporizador.

Quando o programa estiver implementado e depurado, utilizar as funções PSP

`pspExtInterruptsSetThreshold(threshold)` e

`pspExtInterruptSetPriority(interrupt_source, priority)` para analisar

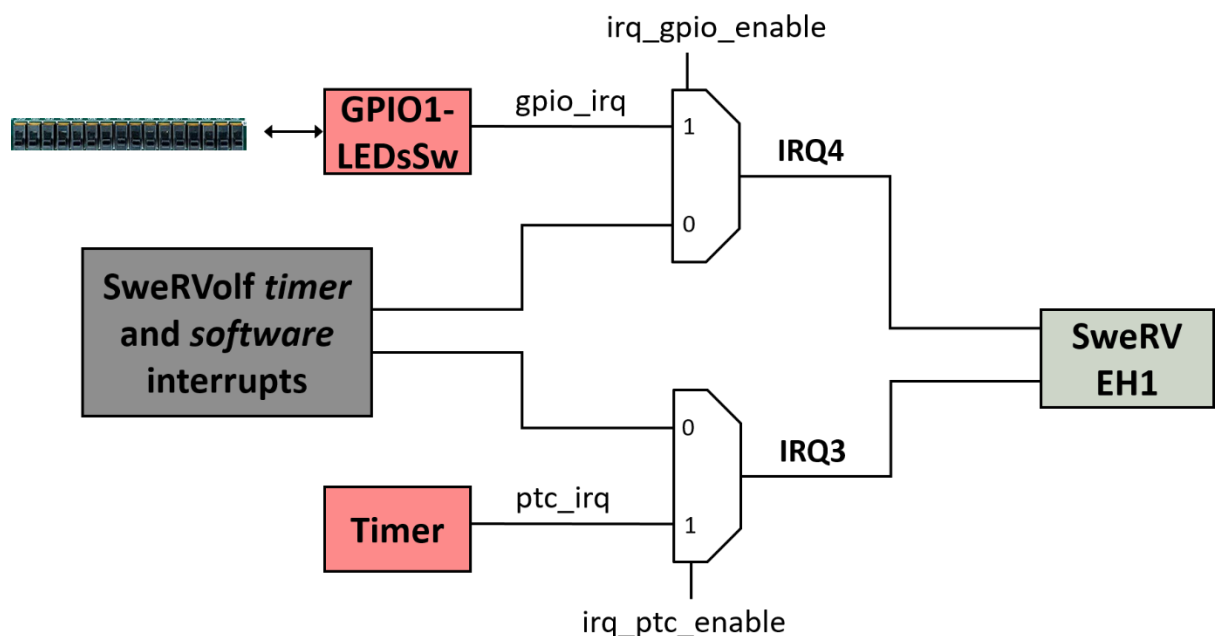
diferentes combinações de prioridades e limiares. Note-se que pode mesmo alterar as prioridades em tempo de execução; por exemplo, pode fazer com que os mostradores de 7 segmentos contêm até 10 e depois parem de contar, modificando a prioridade da fonte de interrupção externa adequada.

**Exercício 2.** Modifique o RVfpgaNexys para incluir uma terceira fonte de interrupção proveniente do segundo GPIO que concebeu no Laboratório 6 para controlar os botões de pressão integrados (GPIO2). São possíveis duas abordagens para completar este exercício:

- Pode ligar a interrupção GPIO2 a uma fonte de interrupção externa livre. O SweRV EH1 fornece até 255 linhas de interrupção diferentes e até agora só utilizámos 2 delas. A desvantagem desta abordagem é que as bibliotecas do WD precisam de ser modificadas.
- Pode ligar a interrupção GPIO2 à IRQ4, de modo a que o módulo GPIO (que se liga aos LEDs e interruptores) e o GPIO2 (que se liga aos botões de pressão) utilizem um modo de interrupção de vector único. Embora o modo multi-vector seja preferível nalgumas situações, a vantagem desta abordagem é que pode reutilizar o BSP.

Damos alguma orientação para a segunda abordagem fornecendo alguns pormenores sobre a implementação de baixo nível das interrupções no sistema RVfpga.

Figura 12 mostra o circuito que liga as várias fontes de interrupção (interrupção GPIO, interrupção do temporizador - e as fontes de interrupção originalmente disponíveis no núcleo SweRVolf, que não analisamos nem usamos aqui) com IRQ4 e IRQ3. Especificamente, a IRQ4 é ligada à GPIO quando *irq\_gpio\_enable* = 1 (Figura 6), enquanto o IRQ3 está ligado ao temporizador quando *irq\_ptc\_enable* = 1 (Figura 6). Quando *irq\_gpio\_enable* = *irq\_ptc\_enable* = 0, IRQ4 e IRQ3 estão ligados às fontes de interrupção originais do SweRVolf, que não utilizamos neste laboratório (se estiver interessado em utilizar estas fontes de interrupção, pode ver mais informações em <https://github.com/chipsalliance/Cores-SweRVolf>).



**Figura 12. Implementação lógica: ligação das interrupções do GPIO e do temporizador com IRQ4 e IRQ3, respectivamente**

Figura 13 mostra a região Verilog do módulo **swervolf\_core** que implementa a ligação entre as fontes de interrupção e IRQ4 e IRQ3. A interrupção GPIO está ligada ao IRQ4 quando o sinal *irq\_gpio\_enable* é 1 (parte superior da caixa vermelha). A interrupção do temporizador

está ligada ao IRQ3 quando o sinal *irq\_ptc\_enable* é 1 (parte inferior da caixa vermelha). Quando ambos os sinais são 0 (código não realçado na Figura), as fontes de interrupção implementadas no SweRVolfX estão ligadas ao IRQ3 e ao IRQ4.

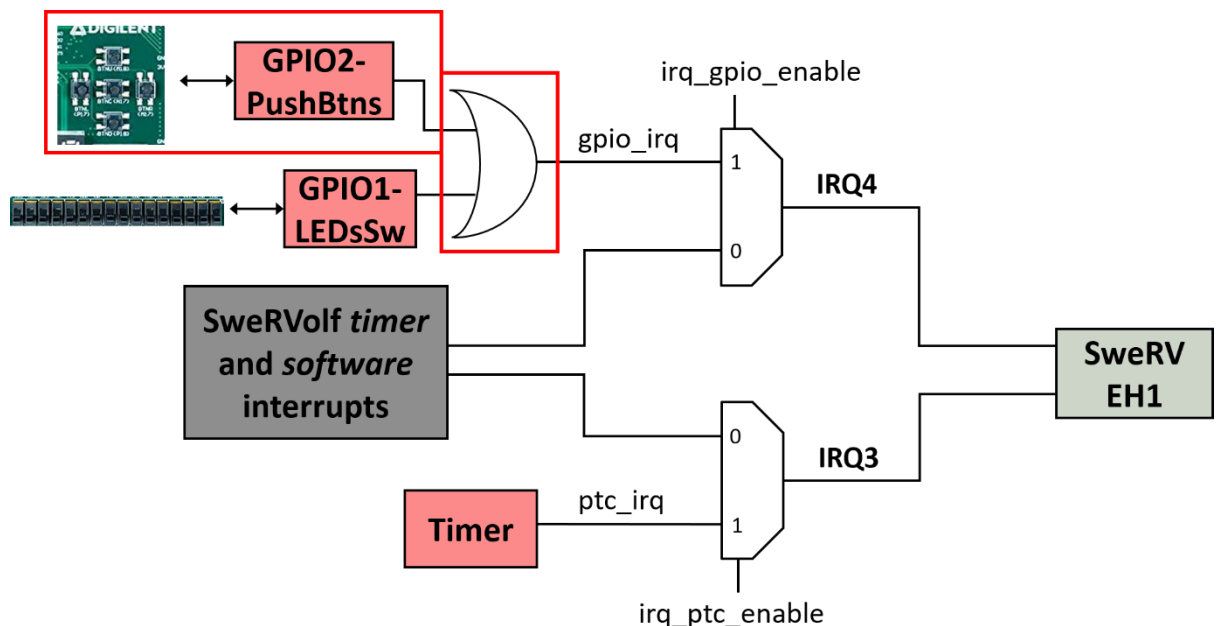
```

123 always @(posedge i_clk) begin
124     o_wb_ack <= i_wb_cyc & !o_wb_ack;
125
126     nmi_int    <= 1'b0;
127     nmi_int_r <= nmi_int;
128
129     // GPIO Interrupt through IRQ4. Enable by setting bit 0 of word 0x80001018
130     if (irq_gpio_enable & gpio_irq) begin
131         sw_irq4 <= 1'b1;
132     end
133
134     // Timer (PTC) Interrupt through IRQ3. Enable by setting bit 1 of word 0x80001018
135     if (irq_ptc_enable & ptc_irq) begin
136         sw_irq3 <= 1'b1;
137     end
138
139     // SweRVolf simple timer and software interrupts. Enable by resetting bits 0 and 1 of word 0x80001018
140     if (!irq_gpio_enable & !irq_ptc_enable) begin
141
142         if (sw_irq3_edge)
143             sw_irq3 <= 1'b0;
144         if (sw_irq4_edge)
145             sw_irq4 <= 1'b0;
146
147         if (irq_timer_en)
148             irq_timer_cnt <= irq_timer_cnt - 1;
149
150         if (irq_timer_cnt == 32'd1) begin
151             irq_timer_en <= 1'b0;
152             if (sw_irq3_timer)
153                 sw_irq3 <= 1'b1;
154             if (sw_irq4_timer)
155                 sw_irq4 <= 1'b1;
156             if (!(sw_irq3_timer | sw_irq4_timer))
157                 nmi_int <= 1'b1;
158         end
159     end
160 end

```

**Figura 13. Implementação Verilog: destacada a vermelho, ligação das interrupções do GPIO e do temporizador com IRQ4 e IRQ3, respectivamente.**

Neste exercício, é necessário estender a implementação anterior (Figura 12) para incluir uma nova fonte de interrupção ligada ao IRQ4, como mostrado na Figura 14.



**Figura 14. Implementação lógica: ligação de uma segunda fonte de interrupção (fornecida pelo GPIO que lê os botões de pressão) com IRQ4**

Destacamos algumas outras regiões Verilog que também devem ser compreendidas, embora não seja necessário modificá-las neste exemplo.

- As fontes de interrupção são inseridas no processador SweRV na linha 600 do módulo **swervolf\_core** (Figura 15). Embora estejam disponíveis quatro fontes de interrupção, neste laboratório só nos interessam as fontes *sw\_irq4* e *sw\_irq3*.

```
600      .extintsrc_req ({4'd0, sw_irq4, sw_irq3, spi0_irq, uart_irq}),
```

**Figura 15. Fontes de interrupção enviadas para o SweRV**

- Os sinais de ativação, *irq\_gpio\_enable* e *irq\_ptc\_enable* (acessível no endereço 0x80001018, ver Figura 6), são escritos pelo núcleo nas linhas 192-196 do módulo **swervolf\_syscon** (Figura 16).

```
192      6: begin //0x18-0x1B
193          if (i_wb_sel[0])
194              irq_gpio_enable <= i_wb_dat[0];
195              irq_ptc_enable <= i_wb_dat[1];
196      end
```

**Figura 16. Escrita do registo 0x80001018 do núcleo SweRV**

Estes sinais de ativação, *irq\_gpio\_enable* e *irq\_ptc\_enable*, são lidos nas linhas 248-249 pelo módulo **swervolf\_syscon** do núcleo (ver Figura 17).

```
248      //0x18-0x1B
249      6 : o_wb_rdt <= {30'd0, irq_ptc_enable, irq_gpio_enable};
```

**Figura 17. Leitura do registo 0x80001018 no núcleo SweRV**

**Exercício 3.** Use a versão estendida do RVfpgaNexys que projetou no exercício anterior para implementar um programa em C que exibe uma contagem binária crescente nos LEDs, começando em 1. Crie um atraso com o timer, usando interrupções, para esperar entre a exibição de cada valor incrementado, de modo que os valores sejam visíveis pelo olho humano. Leia o BTNC e use-o para alterar a velocidade da contagem, leia o Switch[0] e use-o para reiniciar a contagem sempre que for premido.

Com o RVfpgaNexys estendido do Exercício 2, temos agora três fontes de interrupção possíveis:

- GPIO** (interrupções dos interruptores)
- GPIO2** (interrompe os botões, que criou no exercício anterior, Exercício 2)
- PTC** (o temporizador)

Dado que a implementação estendida do RVfpgaNexys do Exercício 2 tem duas fontes de interrupção que compartilham a mesma linha (IRQ4), a Rotina de Atendimento da Interrupção (*Interrupt Service Routine*) correspondente (**GPIO\_ISR**) tem de identificar o dispositivo que gerou a interrupção. Pode obter essa informação a partir dos registos GPIO.

## APÊNDICE

Este apêndice descreve como o Controlador de Interrupção Programável (PIC) do Núcleo SweRV EH1 gere as interrupções externas a um nível de registo. O PIC utiliza os registos memorizados mostrados na Tabela 2. Deve-se notar que o espaço de memória do PIC começa no endereço 0xF00C0000. Este endereço é referido como *RV\_PIC\_BASE*. São dados endereços relativos a este endereço base.

**Tabela 2. Mapa de Registos do PIC em Endereços Mapeados em Memória**

Nome	Endereços (relativos a <i>RV_PIC_BASE</i> )	Descrição	Localização no manual
meipIS	$0x0004 - 0x0004 + S_{max} * 4 - 1$	Registo do nível de prioridade da interrupção externa	Tabela 6-2 de [PRM v1.7]
meipX	$0x1000 - 0x1000 + (X_{max} + 1) * 4 - 1$	Registo de interrupção externa pendente	Tabela 6-3 de [PRM v1.7]
meieS	$0x2000 - 0x2000 + S_{max} * 4 - 1$	Registo de interrupção externa habilitada	Tabela 6-4 de [PRM v1.7]
mpiccfg	0x3000 – 0x3003	Registo de configuração de interrupção externa do PIC	Tabela 6-1 de [PRM v1.7]
meigwctrlS	$0x4004 - 0x4004 + S_{max} * 4 - 1$	Registo de configuração de gateways de interrupção externa (apenas para gateways configuráveis)	Tabela 6-11 de [PRM v1.7]
meigwclrS	$0x5004 - 0x5004 + S_{max} * 4 - 1$	Registo de cancelamento de gateways de interrupção externa (apenas para gateways configuráveis)	Tabela 6-12 de [PRM v1.7]

Todos os registos têm 32 bits de largura e são acessíveis através de instruções leitura e escrita (*load/store*), como é habitual para E/S mapeadas na memória. O tipo de acesso depende dos bits específicos a que queremos aceder (isto pode ser visto em [PRM v1.7]).

Alguns dos registos têm nomes parametrizados, que terminam em S ou X. Podem existir várias instâncias destes registos. O parâmetro S refere-se ao número de fontes de interrupção externas, que no SweRV EH1 é equivalente ao número de gateways. Assim, os registos terminados em 'S' têm entre 1 e 255 instâncias de registo disponíveis. Neste laboratório, utilizamos apenas 2 fontes de interrupção externas: IRQ3 (associada ao temporizador), e IRQ4 (associada à GPIO). O parâmetro X refere-se a um grupo de 32 gateways. Isto não significa que as gateways estejam agrupadas, mas o seu agrupamento reduz o tamanho da memória necessária para certos registos de 32 bits onde 1 bit é suficiente para executar uma ação num grupo de fontes de interrupção externas. É o caso do registo de interrupção externa pendente, onde um bit é suficiente para distinguir se a interrupção foi ou não atendida. Para obter mais informações sobre estes registos, a coluna mais à direita da Tabela 1 aponta para o local em [PRM v1.7] onde está contida a descrição do nível de bits (interrupção específica).

Para além dos registos apresentados na Tabela 2, o PIC contém Registos de Controlo e Estado (CSRs). O ISA RISC-V standard estabelece um espaço de codificação de 12 bits (*csr[11:0]*) para um máximo de 4.096 CSRs. Por convenção, os 4 bits superiores do endereço do CSR (*csr[11:8]*) são usados para codificar a acessibilidade de leitura e escrita dos CSRs de acordo com o nível de privilégio. Os dois bits superiores (*csr[11:10]*) indicam se o registo é de leitura/escrita (00, 01 ou 10) ou apenas de leitura (11). Os dois bits seguintes (*csr[9:8]*) codificam o nível de privilégio mais baixo que pode aceder ao CSR. Estão disponíveis mais informações sobre os CSRs em [PRM v1.7] e [ISM v1.11]. A Tabela

3 lista os CSRs que são úteis para gerir as interrupções externas no núcleo EH1 do SwerRV. São acessíveis através de instruções dedicadas de leitura e escrita, como *csrrw* ou *csrrs* (CSR read/write e CSR read/set).

**Tabela 3. Mapa de endereços RISC-V CSR não normalizado do PIC.**

Nome	Número	Descrição	Localização
meivt	0xBC8	Registo da tabela de vectores de interrupção externa	Tabela 6-6 de [PRM v1.7]
meipt	0xBC9	Registo de limiar de prioridades de interrupção externa	Tabela 6-5 de [PRM v1.7]
meicpct	0xBCA	Registo de disparo de captura de nível de prioridade / ID de pedido de interrupção externa	Tabela 6-8 de [PRM v1.7]
meicidpl	0xBCB	Registo do nível de prioridade da ID de pedido de interrupção externa	Tabela 6-9 de [PRM v1.7]
meicurpl	0xBCC	Registo do nível de prioridade actual da interrupção externa	Tabela 6-10 de [PRM v1.7]
meihap	0xFC8	Registo do ponteiro de endereço do gestor de interrupções externas	Tabela 6-7 de [PRM v1.7]
mie	0x304	Registo de ativação da interrupção da máquina	Tabela 11-1 de [PRM v1.7]
mstatus	0x300	Registo de estado da máquina	Figura 3.7 de [ISM v1.11]

A coluna mais à direita na Tabela 3 aponta para o local em [PRM v1.7] ou [ISM v1.11] onde a informação ao nível dos bits é descrita para o CSR em causa (note-se que a descrição dos bits *mstatus* não é fornecida em [PRM v1.7] mas em [ISM v1.11]).

## A. Configuração de Interrupções Externas

Nesta subsecção, resumimos os passos básicos necessários para configurar uma interrupção externa utilizando os registos acima mencionados:

1. Desativar todas as interrupções externas limpando os bits *mie* e *mip* no CSR.
2. Configurar a ordem de prioridade escrevendo o bit *prprio* do registo *mpiccfg*.
3. No modo multi-vector, se não estiver configurado, definir o endereço de base da tabela de endereços de interrupção com vector externo escrevendo o campo base do registo *meivt*.
4. Definir o limite de prioridade escrevendo o campo *prthrsh* do registo *meipt*.
5. Inicializar os limiares de prioridade de agrupamento escrevendo "0" (ou "15" para ordem de prioridade invertida) no campo *clidpri* do registo *meicidpl* e no campo *currpri* do registo *meicurpl*.
6. Para cada gateway configurável S, defina a polaridade (ativa-alta/ativa-baixa) e o tipo (disparada por nível/por transição) no registo *meigwctrlS* e apague o bit IP escrevendo no registo *meigwclrS* da gateway.
7. No modo multi-vectorial, para cada fonte de interrupção externa S, escrever o endereço do gestor (*handler*) correspondente na tabela de endereços de interrupção externa vectorial.
8. Definir o nível de prioridade para cada fonte de interrupção externa S, escrevendo o campo de prioridade correspondente dos registos *meipIS*.
9. Ativar as interrupções para as fontes de interrupção externas adequadas, definindo o bit *inten* dos registos *meieS* para cada fonte de interrupção S.

10. Ativar o bit *mei* no CSR *mstatus*.
11. Ativar todas as interrupções externas definindo o bit *miep* no CSR *mie*.

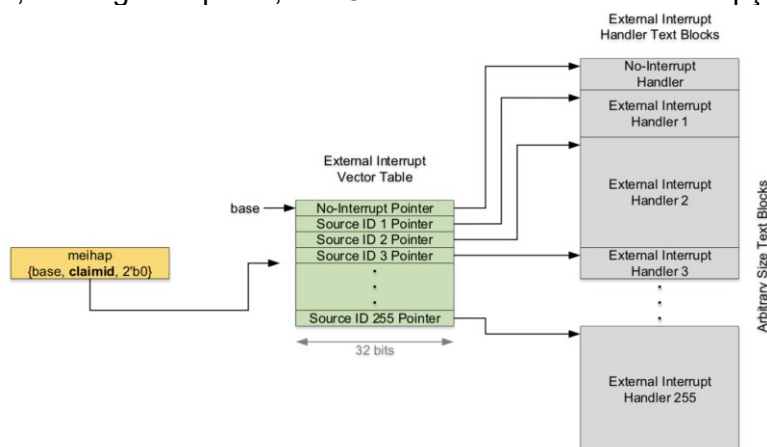
Estes são os passos gerais para as gateways S. No entanto, no sistema RVfpga usamos apenas 2 fontes de interrupção (IRQ3 e IRQ4), cada uma das quais tem a sua própria gateway. Além disso, é preciso notar que a ordem não é totalmente rígida, pois algumas ações são permutáveis (por exemplo, o passo 4 pode ser completado antes do passo 2). Além disso, como cada função chama *pspInterruptsDisable* na entrada, o passo 1 não é estritamente necessário.

## B. Modo de funcionamento da interrupção externa

Nesta subsecção descrevemos como funciona o PIC uma vez que uma interrupção externa é desencadeada. Uma vez que o evento desejado ocorra na linha de interrupção externa (fio), realizam-se as seguintes ações:

1. O PIC decide qual a interrupção pendente que tem a maior prioridade.
  2. Quando o *hart* (hardware thread) visado trata a interrupção externa, desativa todas as interrupções (ou seja, limpa o bit *mie* no registo *mstatus* do *hart* do RISC-V) e salta para o gestor da interrupção externa.
  3. O gestor da interrupção externa escreve para o registo *meicpct* para desencadear a captura da identificação da fonte de interrupção da interrupção externa de maior prioridade que se encontra pendente (no registo *meihap*) e a sua prioridade correspondente (no registo *meicidpl*).
  4. O gestor lê então o registo *meihap* para obter a identificação da fonte de interrupção fornecida no campo *claimid*. Com base no conteúdo do registo *meihap*, o manipulador de interrupção externa salta para o manipulador específico para esta fonte de interrupção externa. Isto pode ser observado na Figura 18.
  5. O gestor de interrupções específicas da fonte (ISR) atende a interrupção externa, e depois:
    - a. Para fontes de interrupção a nível, o gestor da interrupção limpa o estado no IP do SoC que iniciou o pedido de interrupção.
    - b. Para as fontes de interrupção acionadas por transição de flanco, o gestor de interrupção limpa o bit IP no gateway da fonte, escrevendo para o registo *meigwclrS*.
- Isto desativa o pedido de interrupção da fonte.

6. Entretanto, em segundo plano, o PIC continua a avaliar as interrupções pendentes.



**Figura 18. Interrupções externas vectorizadas (tirada de [PRM v1.7])**

Deve notar-se que se trata de um modo de funcionamento regular. As interrupções aninhadas (um máximo de 15) também são suportadas no núcleo SweRV EH1. Para mais informações, por favor consulte [PRM v1.7].