



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 10

Barramentos Série

1. INTRODUÇÃO

Neste laboratório, descrevemos primeiro como funcionam os barramentos em série e as principais características de um dos barramentos em série mais típicos atualmente utilizados, o barramento SPI (Secção 2). Depois concentramo-nos no acelerómetro SPI disponível na placa Nexys A7: analisamos a especificação de alto-nível para este periférico e propomos exercícios elementares (Secções 3 e 4), e depois analisamos a sua implementação de baixo-nível e propomos alguns exercícios avançados (Secções 5 e 6).

2. BARRAMENTO SÉRIE - O BARRAMENTO SPI

Os barramentos paralelos enviam vários bits ao mesmo tempo, enquanto os barramentos em série enviam um bit de cada vez. Primeiro comparamos estes dois esquemas de comunicação e depois descrevemos o protocolo SPI (Serial Peripheral Interface), que é um dos barramentos seriais mais comuns atualmente utilizados. Pode encontrar muita informação na Internet para alargar os seus conhecimentos sobre este protocolo de comunicação importante.

Como já demonstrado em laboratórios anteriores, o principal objetivo da electrónica embebida é ligar processadores e circuitos para criar as funções desejadas. Para que os processadores e circuitos possam partilhar informação, devem partilhar um protocolo de comunicação comum. Centenas de protocolos de comunicação foram definidos para alcançar esta troca de dados, e, em geral, podem ser separados em duas categorias principais: interfaces paralelas ou em série.

As interfaces paralelas transferem vários bits em paralelo, ou seja, ao mesmo tempo. Requerem barramentos (múltiplos fios) de dados. Por exemplo, o protocolo pode transmitir oito, dezasseis, ou mais bits ao mesmo tempo (ver Figura 1). Requerem também um sinal de relógio para sincronizar quando novos grupos de N bits de dados estão prontos para transferência.

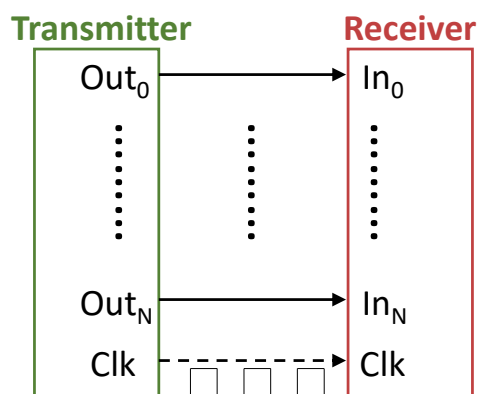


Figura 1. Exemplo de um barramento paralelo de dados de 8 bits.

Em contraste com a comunicação paralela, as interfaces em série transmitem os seus dados um bit de cada vez. Estas interfaces podem funcionar usando tão pouco como um fio e normalmente nunca mais do que quatro. A Figura 2 mostra um exemplo de interface de série com um fio para sinais de dados e outro para o sinal de relógio. A cada nova transição do sinal de relógio, um novo bit de dados é transferido.

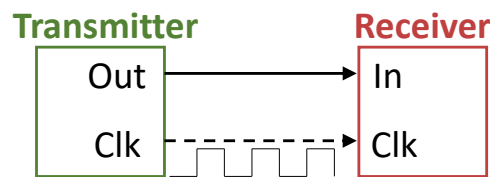


Figura 2. Exemplo de um barramento de dados em série de 1 bit.

A comunicação paralela tem as vantagens de ser rápida, simples e relativamente fácil de implementar. No entanto, requer muito mais linhas de entrada/saída (I/O). Assim, como os pinos são limitados, os sistemas embebidos optam frequentemente pela comunicação em série, sacrificando a velocidade potencial pelos pinos disponíveis.

Barramento SPI:

O protocolo *Serial Peripheral Interface* (SPI), ou Interface de Periférico Série, é uma das interfaces mais utilizadas entre microcontroladores e CI periféricos tais como sensores, ADCs, DACs, SRAM, e outros. O SPI é uma interface síncrona, *full-duplex* baseada na comunicação controlador-periférico (anteriormente chamado mestre-escravo).

O barramento SPI comunica geralmente através de 4 sinais (ver Figura 3):

- **SDO** – *Serial Data Out* / Saída de dados em série: Saída do controlador para o dispositivo periférico
- **SDI** – *Serial Data Input* / Entrada de dados em série: Entrada do controlador a partir de dispositivo periférico
- **SCK** – *Serial Clock* / Relógio Série: Enviado do controlador para o dispositivo periférico
- **CS** – *Chip Select* / Seleção do chip: Sinal ativo a zero; o controlador envia o sinal (0 quando o periférico é selecionado) para o periférico

Nota: historicamente, a SDO também tem sido chamada MOSI (*master data out, slave data in*) e a SDI tem sido chamada MISO (*master data in, slave data out*). Estes termos ainda podem ser encontrados nalguma literatura e documentação mais antigos.

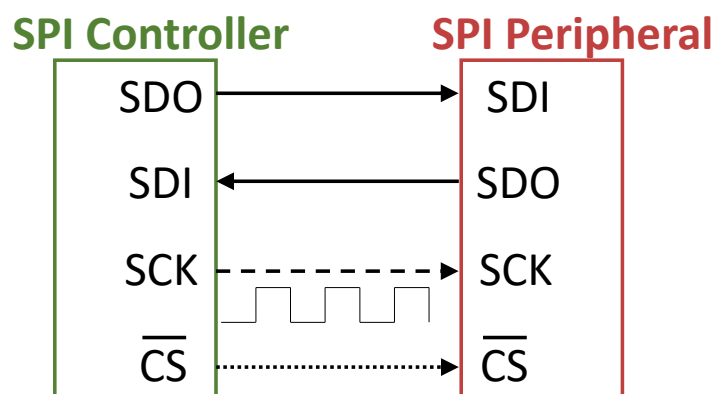


Figura 3. Exemplo de um sistema com um controlador SPI e um periférico SPI.

Os dados em série são sincronizados com o flanco ou transição crescente ou decrescente do relógio. O SPI é uma interface *full-duplex*; o controlador e o periférico podem enviar dados ao mesmo tempo através das linhas SDO e SDI, respectivamente. As interfaces SPI apenas têm um controlador, mas podem ter múltiplos periféricos. Quando mais do que um

periférico está ligado, múltiplos sinais de seleção de chip ativos a zero (/CS) do controlador são utilizados para selecionar qual o periférico que está a ser usado. O SDO e o SDI são os sinais de dados em série: SDO (*serial data out*) são os dados de saída do controlador para o periférico e SDI (*serial data in*) são os dados de entrada do periférico para o controlador.

Para iniciar a comunicação SPI, o controlador deve selecionar o dispositivo periférico (ativando o sinal /CS, isto é, /CS = 0) e depois enviar o sinal do relógio para o periférico. Durante a comunicação SPI, os dados são transmitidos simultaneamente de e para o controlador através dos sinais SDO e SDI, respectivamente. A transição do relógio de série (SCK) sincroniza a amostragem dos dados.

A interface SPI fornece sinais adicionais, CPOL e CPHA, para selecionar o estado de repouso do relógio e a fase de amostragem do sinal de dados. A polaridade do sinal de relógio (CPOL) é 0 quando o relógio (SCK) fica em repouso a 0, e 1 quando em repouso a 1. O sinal da fase do relógio (CPHA) seleciona a fase do relógio para enviar e amostrar dados. Quando CPHA = 0, os dados (em SDI ou SDO) são amostrados no flanco ascendente (i.e., a primeira transição de SCK depois de abandonar o repouso - e em todos os ciclos seguintes); portanto os dados (SDI e SDO) mudam no flanco final, como mostrado nos dois principais diagramas temporais da Figura 4. CPHA = 1 faz o oposto: os dados são amostrados no flanco final e os dados são alterados na transição inicial, como se mostra nas duas figuras inferiores da Figura 4. A transição em que os novos dados são transmitidos chama-se *shifting edge* (transição de deslocamento), porque esta comunicação em série é tipicamente implementada utilizando um registo de deslocamento (*shift register*).

A interface SPI que utilizamos neste laboratório usa CPHA = 0 e CPOL = 0, por isso, o SCK fica em repouso no nível 0 e os dados do controlador e periférico são amostrados no flanco ascendente. Os novos dados são deslocados no sinal (SDO ou SDI) imediatamente após cada flanco descendente do sinal de relógio, tal como mostrado no diagrama de tempo superior da Figura 4. Note que quando o SCK está inativo, e imediatamente antes de subir, SDO e SDI deve conter o bit mais significativo do próximo byte de dados.

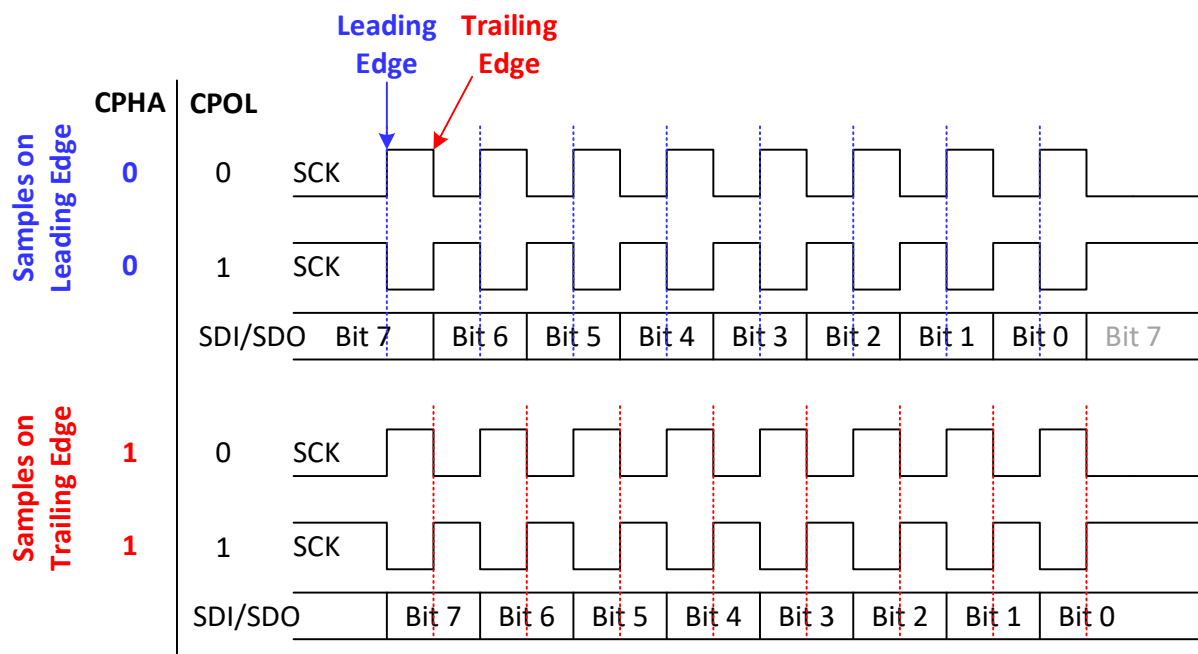


Figura 4. Relação de CPHA/CPOL com dados de amostragem/envio.

3. ACELERÓMETRO DE SPI: ESPECIFICAÇÃO DE ALTO-NÍVEL

Muitos periféricos incluem uma interface SPI. Por exemplo, o acelerómetro na placa Nexys A7 tem uma interface SPI. Nesta secção descrevemos a especificação de alto nível do controlador SPI do sistema RVfpga e introduzimos o acelerómetro ADXL362 incluído na placa Nexys A7. Introduzimos também um exercício que utiliza o acelerómetro.

A. Especificação do Controlador SPI

O módulo SPI do Sistema RVfpga é da OpenCores (https://opencores.org/projects/simple_spi). Se descarregar o módulo, é fornecido um documento que descreve a especificação de alto nível do módulo. Este documento também é fornecido aqui:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/spi/docs/simple_spi.pdf

Resumimos as principais operações e características do módulo SPI; contudo, consulte o documento acima para informações adicionais.

Este módulo tem como principais características:

- É compatível com as especificações SPI da Motorola;
- Utiliza a interface clássica de 8 bits WISHBONE RevB.3;
- Contém um buffer FIFO de leitura de 4 entradas e um buffer FIFO de escrita de 4 entradas;
- Permite a geração de interrupções após 1, 2, 3, ou 4 bytes transferidos;
- Pode funcionar com uma vasta gama de frequências de relógio de entrada
- É plenamente sintetizável

A secção 3 da especificação principal do SPI descreve os registos de controlo e de estado disponíveis dentro do módulo SPI, cada um dos quais é atribuído a um endereço diferente (ver Tabela 1). O endereço base do controlador do SPI é **0x80001100**. Estes registos são descritos em pormenor a seguir.

Tabela 1. Registos SPI

Nome	Endereço	Largura	Acesso	Descrição
SPCR	0x80001100	8	R/W	Registo de controlo
SPSR	0x80001108	8	R/W	Registo de estado
SPDR	0x80001110	8	R/W	Registo de dados
SPER	0x80001118	8	R/W	Registo de extensões
SPCS	0x80001120	8	R/W	Registo CS

O registo de controlo SPI - *Control Register* (SPCR) controla o módulo SPI. A Tabela 2 mostra a função de cada um dos seus bits.

Tabela 2. Bits do SPCR

Bit	Acesso	Nome e Descrição
0:1	R/W	SPR Velocidade do relógio SPI: Estes bits selecionam a velocidade do relógio SPI.
2	R/W	CPHA Fase do Relógio: Determina a fase de amostragem e envio de dados. Quando CPHA = 1, os novos dados são transferidos para o fio na transição

		inicial e os dados são amostrados na transição final. Quando CPHA = 0, os novos dados são deslocados para o fio da transição final e amostrados na transição inicial.
3	R/W	CPOL Polaridade do relógio: Determina o estado inativo do relógio SPI (SCK). Quando CPOL = 0, SCK fica em 0, quando CPOL = 1, SCK fica em 1.
4	R/W	MSTR Seleção de modo: Quando MSTR = 1, o núcleo SPI é um dispositivo controlador. Este é o único modo suportado para este controlador.
6	R/W	SPE Habilitação do SPI: Quando SPE = 1, o núcleo SPI está ativado. Quando estiver limpo (SPE = 0), o núcleo SPI está desativado.
7	R/W	SPIE Habilitação de Interrupção SPI: Quando SPIE = 1, quando a Flag de Interrupção do SPI é ativada no registo de estado, o sistema anfitrião é interrompido.

O Registo do Estado do SPI *Status Register* (SPSR) fornece o estado do módulo SPI. A Tabela 3 mostra a função de cada um dos seus bits.

Tabela 3. Bits SPSR

Bit	Acesso	Descrição
0	R/W	RFEMPTY FIFO de leitura vazio: se RFEMPTY = 1, o FIFO de leitura está vazio.
1	R/W	RFFULL FIFO de leitura cheio: se RFFULL = 1, o FIFO de leitura está cheio.
2	R/W	WFEMPTY FIFO de escrita vazio: se WFEMPTY = 1, o FIFO de escrita está vazio.
3	R/W	WFFULL FIFO de escrita cheio: se WFFULL = 1, o FIFO de escrita está cheio.
6	R/W	WCOL Flag Colisão de escrita: Quando WCOL = 1, o registo SPDATA foi escrito enquanto o FIFO de escrita estava cheio. Escrever um 1 para WCOL limpa esta bit.
7	R/W	SPIF Flag de Interrupção SPI: SPIF = 1 após a conclusão de um bloco de transferência. Se o SPIF for definido ('1') e SPIE estiver ativo, uma interrupção é gerada. Escrevendo um 1 para SPIF limpa-o.

O Registo de Dados do SPI - *Data Register* (SPDR) fornece os dados para ler ou escrever. O controlador SPI inclui um Buffer de escrita 4x 8-bit, e um Buffer de leitura 4x 8-bit.

O Registo Estendido do SPI - *Extended Register* (SPER) fornece alguma funcionalidade adicional. A Tabela 4 descreve os vários campos que ele contém.

Tabela 4. Bits SPER.

Bit	Acesso	Descrição
0:1	R/W	ESPR Seleção da Taxa de Relógio do SPI Estendido: Adicione dois bits ao SPR (SPI Clock Rate Select).
6:7	R/W	ICNT Contador de interrupções: Determinar o tamanho do bloco de transferência. O bit SPIF é ativado após a transferência da ICNT. Assim, é possível reduzir a sobrecarga do núcleo devido à redução das chamadas de serviço de interrupção.

Finalmente, o registo SPI Chip Select - *Chip Select* (SPCS) seleciona o periférico a utilizar. A largura deste sinal é configurável através do parâmetro `SS_WIDTH` (SPI Select Width). No Sistema RVfpga, só existe um periférico para cada interface SPI, por isso `SS_WIDTH = 1`.

TAREFA: Localizar a declaração dos registos `SPCR`, `SPSR`, `SPDR`, `SPER` e `SPCS` no módulo SPI, bem como a definição dos seus endereços. O módulo SPI está disponível dentro da pasta `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/spi`.

B. ADXL362 Especificação do Acelerómetro

A placa Nexys A7 inclui um acelerómetro analógico ADXL362. Pode encontrar a informação completa para o dispositivo na sua ficha técnica (*datasheet*), localizada aqui:

<https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>

O ADXL362 é um acelerómetro MEMS de 3 eixos que consome menos de 2µA a uma taxa de dados de saída de 100Hz e 270 nA quando no modo de despertar provocado pelo movimento. Fornece uma resolução de saída de 12 bits, embora sejam também fornecidos dados formatados de 8 bits para transferências de bytes únicos. Isto é mais eficiente quando uma resolução mais baixa é suficiente. Gammas de medição de ± 2 g, ± 4 g, e ± 8 g estão disponíveis com uma resolução de 1 mg/LSB na gama de ± 2 g. Enquanto o ADXL362 está em Modo de Medição, ele mede e armazena continuamente os dados de aceleração nos registos dos dados X, Y e Z.

O acelerómetro ADXL362 inclui vários registos (Tabela 5) que permitem ao utilizador configurá-lo e ler os dados de aceleração. O dispositivo é configurado escrevendo nos registos de controlo. Os dados do acelerómetro são obtidos através da leitura dos registos do dispositivo. Toda a comunicação com o dispositivo deve especificar um endereço de registo e uma *flag* que indique se a comunicação é uma leitura ou uma escrita. A transferência de dados ocorre após o endereço do registo e a *flag* de comunicação serem enviados para o dispositivo.

Este acelerómetro funciona como um dispositivo periférico utilizando um esquema de comunicação SPI. A interface entre a FPGA e o acelerómetro é mostrada na Figura 5.

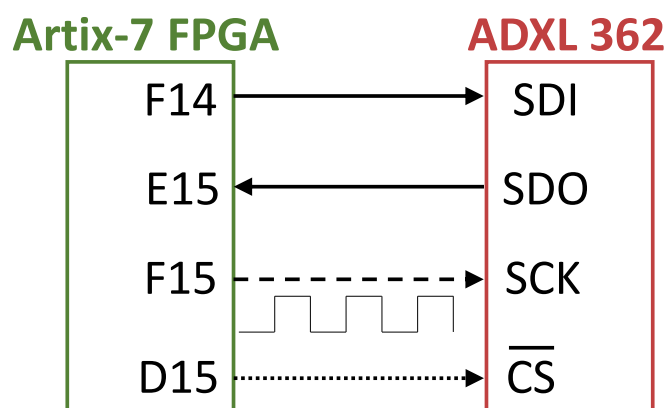


Figura 5. Interface do acelerómetro ADXL362 com a placa Nexys A7

A frequência de relógio SPI recomendada varia entre 1-5 MHz. O SPI funciona no modo SPI 0 (CPOL = 0 e CPHA = 0). O porto SPI utiliza uma estrutura de múltiplos bytes em que o

primeiro byte indica se a comunicação realiza uma leitura de registo (0x0B) ou uma escrita de registo (0x0A):

<CS down> <Write/Read (0x0A/0x0B)> <address byte> <data byte> <CS up>

A Figura 6 e a Figura 7 ilustram dois exemplos da comunicação entre o controlador SPI (controlador) e o acelerómetro (periférico): a Figura 6 mostra a leitura de um registo e a Figura 7 mostra a escrita de um registo.

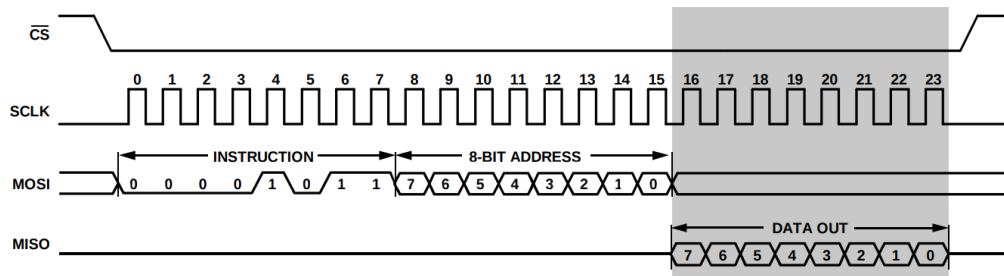


Figura 6. Leitura de registo

(Figura de <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)

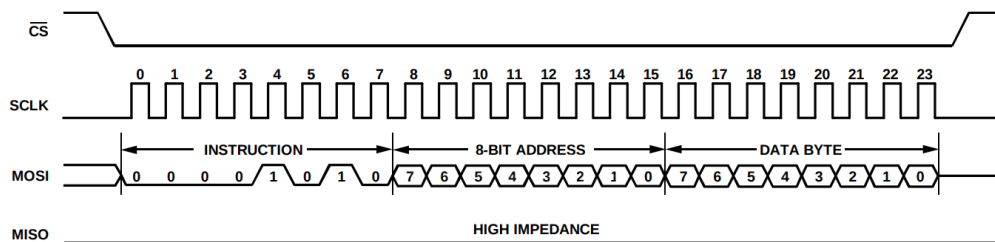


Figura 7. Escrita de registo

(Figura de <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)

A Tabela 5 mostra os registos disponíveis no acelerómetro ADXL362. Para a descrição completa dos registos, consulte o manual técnico ou *datasheet* do ADXL362: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>.

Tabela 5. Registos do acelerómetro ADXL362
(Tabela de <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)

Reg	Name	Bits	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset	RW	
0x00	DEVID_AD	[7:0]	DEVID_AD[7:0]								0xAD	R	
0x01	DEVID_MST	[7:0]	DEVID_MST[7:0]								0x1D	R	
0x02	PARTID	[7:0]	PARTID[7:0]								0xF2	R	
0x03	REVID	[7:0]	REVID[7:0]								0x01	R	
0x08	XDATA	[7:0]	XDATA[7:0]								0x00	R	
0x09	YDATA	[7:0]	YDATA[7:0]								0x00	R	
0x0A	ZDATA	[7:0]	ZDATA[7:0]								0x00	R	
0x0B	STATUS	[7:0]	ERR_USER_REGS	AWAKE	INACT	ACT	FIFO_OVER-RUN	FIFO_WATER-MARK	FIFO_READY	DATA_READY	0x40	R	
0x0C	FIFO_ENTRIES_L	[7:0]	FIFO_ENTRIES_L[7:0]								0x00	R	
0x0D	FIFO_ENTRIES_H	[7:0]	UNUSED								FIFO_ENTRIES_H[1:0]	0x00	R
0x0E	XDATA_L	[7:0]	XDATA_L[7:0]								0x00	R	
0x0F	XDATA_H	[7:0]	SX				XDATA_H[3:0]				0x00	R	
0x10	YDATA_L	[7:0]	YDATA_L[7:0]								0x00	R	
0x11	YDATA_H	[7:0]	SX				YDATA_H[3:0]				0x00	R	
0x12	ZDATA_L	[7:0]	ZDATA_L[7:0]								0x00	R	
0x13	ZDATA_H	[7:0]	SX				ZDATA_H[3:0]				0x00	R	
0x14	TEMP_L	[7:0]	TEMP_L[7:0]								0x00	R	
0x15	TEMP_H	[7:0]	SX				TEMP_H[3:0]				0x00	R	
0x16	Reserved	[7:0]	Reserved[7:0]								0x00	R	
0x17	Reserved	[7:0]	Reserved[7:0]								0x00	R	
0x1F	SOFT_RESET	[7:0]	SOFT_RESET[7:0]								0x00	W	
0x20	THRESH_ACT_L	[7:0]	THRESH_ACT_L[7:0]								0x00	RW	
0x21	THRESH_ACT_H	[7:0]	UNUSED				THRESH_ACT_H[2:0]				0x00	RW	
0x22	TIME_ACT	[7:0]	TIME_ACT[7:0]								0x00	RW	
0x23	THRESH_INACT_L	[7:0]	THRESH_INACT_L[7:0]								0x00	RW	
0x24	THRESH_INACT_H	[7:0]	UNUSED				THRESH_INACT_H[2:0]				0x00	RW	
0x25	TIME_INACT_L	[7:0]	TIME_INACT_L[7:0]								0x00	RW	
0x26	TIME_INACT_H	[7:0]	TIME_INACT_H[7:0]								0x00	RW	
0x27	ACT_INACT_CTL	[7:0]	RES		LINKLOOP		INACT_REF	INACT_EN	ACT_REF	ACT_EN	0x00	RW	
0x28	FIFO_CONTROL	[7:0]	UNUSED				AH	FIFO_TEMP	FIFO_MODE		0x00	RW	
0x29	FIFO_SAMPLES	[7:0]	FIFO_SAMPLES[7:0]								0x80	RW	
0x2A	INTMAP1	[7:0]	INT_LOW	AWAKE	INACT	ACT	FIFO_OVER-RUN	FIFO_WATER-MARK	FIFO_READY	DATA_READY	0x00	RW	
0x2B	INTMAP2	[7:0]	INT_LOW	AWAKE	INACT	ACT	FIFO_OVER-RUN	FIFO_WATER-MARK	FIFO_READY	DATA_READY	0x00	RW	
0x2C	FILTER_CTL	[7:0]	RANGE		RES	HALF_BW	EXT_SAMPLE	ODR			0x13	RW	
0x2D	POWER_CTL	[7:0]	RES	EXT_CLK	LOW_NOISE		WAKEUP	AUTOSLEEP	MEASURE		0x00	RW	
0x2E	SELF_TEST	[7:0]	UNUSED								ST	0x00	RW

4. EXERCÍCIOS ELEMENTARES

Exercício 1. Crie um programa Assembly RISC-V que lê os oito bits mais significativos dos dados de aceleração dos eixos X, Y e Z e depois mostra esses valores nos mostradores de 8 dígitos de 7-segmentos. Consultar a secção B para informação sobre configuração e registo. Utilize as seguintes sub-rotinas para aceder ao módulo SPI. Antes de utilizar as sub-rotinas, tente compreendê-las com base nas informações fornecidas na Secção A sobre o módulo SPI. Aqui está um breve resumo de cada sub-rotina:

- Função `spiInit`: Inicializa o módulo SPI.
- Função `spiCS`: Envia o estado de CS para o registo SPCS.
- Função `spiCSUp`: Coloca a Linha CS ativa, invocando a sub-rotina `spiCS`.
- Função `spiCSDown`: Coloca a Linha CS inativa, invocando a sub-rotina `spiCS`.
- Função `spiSendGetData`: Envia dados através do SPI e recebe os dados periféricos de volta.

```
# Register addresses for SPI Peripheral
#define SPCR          0x80001100
```

```
#define SPSR      0x80001108
#define SPDR      0x80001110
#define SPER      0x80001118
#define SPCS      0x80001120
```

```
# Function: Initialize SPI peripheral
# call: by call ra, spiInit
# inputs: None
# outputs: None
# destroys: t0, t1

spiInit:
    li t1, SPCR # control register
    li t0, 0x53 # 01010011 no ints, core enabled, reserved, controller, cpol=0,
                cha=0, clock divisor 11 for 4096
    sb t0, 0(t1)
    li t1, SPER # extension register
    li t0, 0x02 # int count 00 (7:6), clock divisor 10 (1:0) for 4096
    sb t0, 0(t1)
ret
```

```
# Function: Pull CS Line to either high or low - Provides quick calls spiCSUp
                and spiCSDown
# call: by call ra, spiCS
# inputs: CS status in a0 (0 is low, 1 is high)
# outputs: None
# destroys: t0

spiCS:
    li t0, SPCS # CS register
    sb a0, 0(t0) # Send CS status
ret

spiCSUp:
    li a0, 0x00
    j spiCS

spiCSDown:
    li a0, 0xFF
    j spiCS
```

```
# Function: Send byte through SPI and get the peripheral data back
# call: by call ra, spiSendGetData
# inputs: data byte to send in a0
# outputs: received data byte in a1
# destroys: t0, t1

spiSendGetData:
internalSpiClearIF: # internal clear interrupt flag
    li t1, SPSR # status register
    lb t0, 0(t1) # clear SPIF by writing a 1 to bit 7
    ori t0,t0,0x80
    sb t0, 0(t1)
internalSpiActualSend:
    li t0, SPDR # data register
    sb a0, 0(t0) # send the byte contained in a0 to spi
internalSpiTestIF:
    li t1, SPSR # status register
    lb t0, 0(t1)
    andi t0, t0, 0x80
    li t1, 0x80
    bne t0,t1,internalSpiTestIF # loop while SPSR.bit7 == 0. (transmission in
                                progress)
internalSpiReadData:
    li t0, SPDR # data register
```

```
lb a1, 0(t0) # read the message from SPI
ret
```

5. IMPLEMENTAÇÃO DE BAIXO-NÍVEL

A. Implementação de baixo nível do acelerómetro SPI

Na primeira parte deste laboratório, mostrámos como utilizar os módulos SPI do Sistema RVfpga, e nesta última parte do laboratório descrevemos como o módulo SPI é implementado no RVfpga. Semelhante ao formato dos laboratórios anteriores, dividimos a análise do controlador SPI em três fases:

1. Ligação física entre o SoC e o acelerómetro (região sombreada à esquerda na Figura 8)
2. Integração do controlador SPI, que está incluído dentro do controlador do sistema SweRVolfX (região sombreada do meio na Figura 8)
3. Ligação entre o controlador SPI e o Núcleo SweRV EH1 (região sombreada à direita na Figura 8)

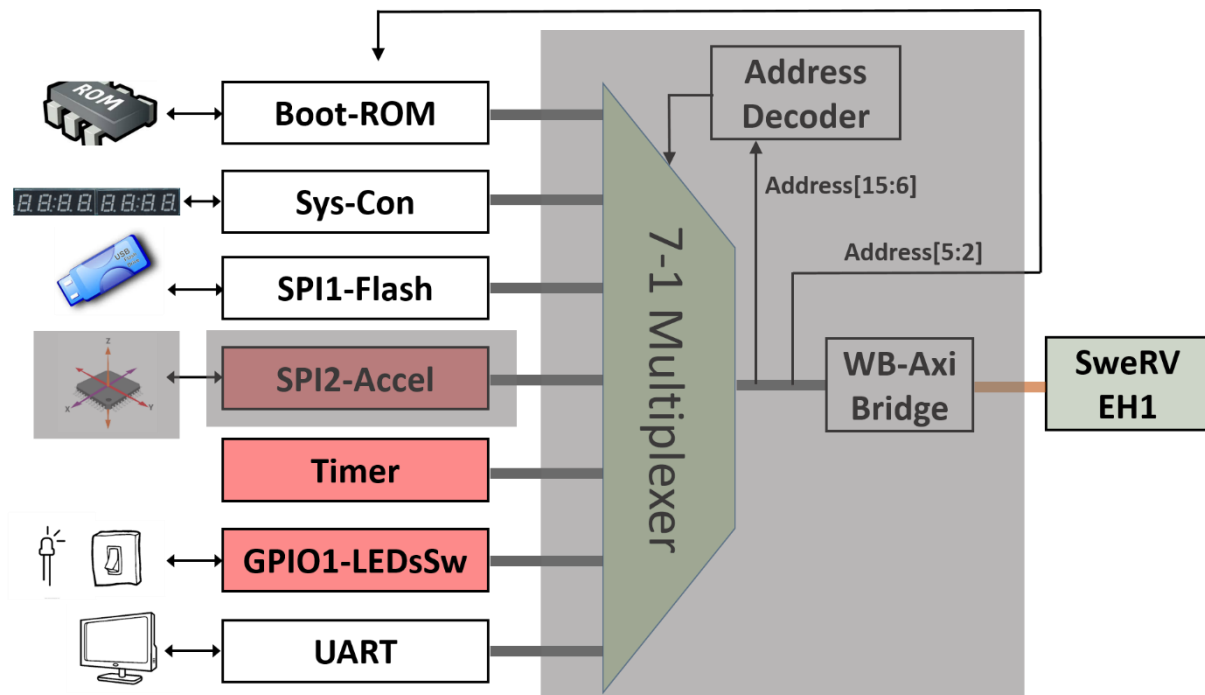


Figura 8. Controlador SPI integrado no sistema RVfpga

1. Ligação física do acelerómetro ao SoC

Tal como com outros periféricos, o ficheiro de restrições do RVfpgaNexys deve incluir as ligações físicas ao acelerómetro. O ficheiro de restrições do projeto ([RVfpgaPath]/RVfpga/src/rvfpganexys.xdc) define a ligação entre os sinais de entrada/saída do SoC e os dispositivos da placa. Os sinais que ligam os quatro pinos do acelerómetro com o SoC são chamados: *o_accel_cs_n*, *o_accel_mosi* (equivalente ao sinal SDO), *i_accel_miso* (equivalente ao sinal SDI) e *accel_sclk*. Note que estes sinais se referem a nomes desatualizados, mas mantemo-los de forma a serem coerentes com os nomes usados pelo módulo SPI do OpenCores que usamos no Sistema RVfpga (pode ver a instanciação deste módulo em Figura 11). A Figura 9 mostra o troço de código Verilog onde estas 4 ligações estão definidas.

```

78 ##Accelerometer
79 set_property -dict { PACKAGE_PIN E15      IOSTANDARD LVCMOS33 } [get_ports { i_accel_miso }]; #IO L11P T1 SRCC 15 Sch=acl_miso
80 set_property -dict { PACKAGE_PIN F14      IOSTANDARD LVCMOS33 } [get_ports { o_accel_mosi }]; #IO L5N T0 AD9N 15 Sch=acl_mosi
81 set_property -dict { PACKAGE_PIN F15      IOSTANDARD LVCMOS33 } [get_ports { accel_sclk }]; #IO L14P T2 SRCC 15 Sch=acl_sclk
82 set_property -dict { PACKAGE_PIN D15      IOSTANDARD LVCMOS33 } [get_ports { o_accel_cs_n }];

```

Figura 9. Ligação do acelerómetro ao SoC (ficheiro *rvfpganexys.xdc*).

Nas linhas 52-55 do módulo principal, ou de topo, do RVfpgaNexys (ou seja, o módulo **rvfpganexys**) pode ver estes quatro sinais ligados ao SoC (parte esquerda da Figura 10), e o fim desse módulo está a sua ligação com o módulo **swervolf_core** (parte direita da Figura 10).

```

25 module rvfpganexys
26     #(parameter bootrom_file = "boot_main.mem")
27     (input wire      clk,
28      input wire      rstn,
29      output wire [12:0] ddram_a,
30      output wire [2:0] ddram_ba,
31      output wire      ddram_ras_n,
32      output wire      ddram_cas_n,
33      output wire      ddram_we_n,
34      output wire      ddram_cs_n,
35      output wire [1:0] ddram_dm,
36      inout wire [15:0] ddram_dq,
37      inout wire [1:0] ddram_dqs_p,
38      inout wire [1:0] ddram_dqs_n,
39      output wire      ddram_clk_p,
40      output wire      ddram_clk_n,
41      output wire      ddram_cke,
42      output wire      ddram_odt,
43      output wire      o_flash_cs_n,
44      output wire      o_flash_mosi,
45      input wire      i_flash_miso,
46      input wire      i_uart_rx,
47      output wire      o_uart_tx,
48      inout wire [15:0] i_sw,
49      output reg [15:0] o_led,
50      output reg [7:0]  AN,
51      output reg        CA, CB, CC, CD, CE, CF, CG,
52      output wire      o_accel_cs_n,
53      output wire      o_accel_mosi,
54      input wire      i_accel_miso,
55      output wire      accel_sclk);
56
248     .o_ram_bready (cpu.b_ready),
249     .i_ram_rid    (cpu.r_id),
250     .i_ram_rdata  (cpu.r_data),
251     .i_ram_rresp  (cpu.r_resp),
252     .i_ram_rlast  (cpu.r_last),
253     .i_ram_rvalid (cpu.r_valid),
254     .o_ram_rready (cpu.r_ready),
255     .i_ram_init_done (litedram_init_done),
256     .i_ram_init_error (litedram_init_error),
257     .io_data         ((i_sw[15:0],gpio_out[15:0])),
258     .AN (AN),
259     .Digits Bits ({CA,CB,CC,CD,CE,CF,CG}),
260     .o_accel_sclk   (accel_sclk),
261     .o_accel_cs_n   (o_accel_cs_n),
262     .o_accel_mosi   (o_accel_mosi),
263     .i_accel_miso   (i_accel_miso));
264
265     always @(posedge clk core) begin
266         o_led[15:0] <= gpio_out[15:0];
267     end
268
269     assign o_uart_tx = 1'b0 ? litedram_tx : cpu_tx;
270
271 endmodule

```

Figura 10. Ligação do acelerómetro com o módulo de topo (ficheiro *rvfpganexys.sv*).

TAREFAS: Siga estes quatro sinais (*o_accel_cs_n*, *o_accel_mosi*, *i_accel_miso* e *accel_sclk*) do ficheiro de restrições para o módulo SweRVolfX SoC. Terá de inspecionar os seguintes ficheiros:

```

[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc
[RVfpgaPath]/RVfpga/src/rvfpganexys.sv
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v

```

2. Integração do módulo SPI2-Acelerómetro no SoC

Nas linhas 387-403 do módulo **swervolf_core** (`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`) é instanciado o módulo SPI para o acelerómetro (ver Figura 11).

```

382 // SPI for the Accelerometer
383 wire [7:0] spi2_rdt;
384 assign wb_s2m_spi_accel_dat = {24'd0, spi2_rdt};
385 wire spi2_irq;
386
387 simple_spi spi2
388 (// Wishbone slave interface
389 .clk_i (clk),
390 .rst_i (wb_rst),
391 .adr_i (wb_m2s_spi_accel_adr[2] ? 3'd0 : wb_m2s_spi_accel_adr[5:3]),
392 .dat_i (wb_m2s_spi_accel_dat[7:0]),
393 .we_i (wb_m2s_spi_accel_we),
394 .cyc_i (wb_m2s_spi_accel_cyc),
395 .stb_i (wb_m2s_spi_accel_stb),
396 .dat_o (spi2_rdt),
397 .ack_o (wb_s2m_spi_accel_ack),
398 .inta_o (spi2_irq),
399 // SPI interface
400 .sck_o (o_accel_sclk),
401 .ss_o (o_accel_cs_n),
402 .mosi_o (o_accel_mosi),
403 .miso_i (i_accel_miso));

```

Figura 11. Integração do módulo SPI2-Acelerómetro (ficheiro `swervolf_core.v`).

Como é hábito, com os periféricos, a interface do módulo pode ser dividida em dois blocos: sinais Wishbone (Tabela 6) e sinais de E/S externos (Tabela 7). Os sinais Wishbone permitem que o SweRV EH1 Core comunique com o ADC usando o protocolo SPI.

Tabela 6. Sinais Wishbone

Porto	Largura	Direção	Descrição
cyc_i	1	Entradas	Indica um ciclo de barramento válido (seleção do núcleo)
adr_i	15	Entradas	Entradas de endereços
dat_i	32	Entradas	Entradas de dados
dat_o	32	Saídas	Saídas de dados
sel_i	4	Entradas	Indica bytes válidos no barramento de dados (durante o ciclo válido deve ser 0xf)
ack_o	1	Saída	Saída de confirmação (indica a conclusão normal da transação)
err_o	1	Saída	Saída de indicação de erro (indica um término anormal da transação)
rty_o	1	Saída	Não utilizado
we_i	1	Entrada	Transação de escrita quando for nível alto
stb_i	1	Entrada	Indica um ciclo de transferência de dados válido
inta_o	1	Saída	Saída de interrupção

Tabela 7. Sinais externos de E/S

Porto	Largura	Direção	Descrição
miso_i	1	Entrada	Entrada de dados do controlador - Saída de dados do periférico

mosi_o	1	Saída	Saída de dados do controlador - Entrada de dados do periférico
ss_o	1	Saída	Seleção do chip
sck_o	1	Saída	Relógio do sistema

Como mostrado na Figura 11, os bits [5:2] do endereço fornecido pelo núcleo no sinal do barramento Wishbone (*wb_m2s_spi_accel_adr[5:2]*) são usados para selecionar um entre os 5 registos SPI disponíveis (Tabela 1).

3. Ligação entre o Controlador SPI e o núcleo SweRV EH1

Como explicado em laboratórios anteriores, os controladores de dispositivos são ligados ao núcleo SweRV EH1 através de um multiplexador e uma ponte (Figura 8). O multiplexador 7:1 (Figura 12) está implementado no ficheiro

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v, que é instanciada nas linhas 104-205 do ficheiro

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh

. Este último ficheiro está incluído na linha 168 do módulo **swervolf_core** localizado em:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v.

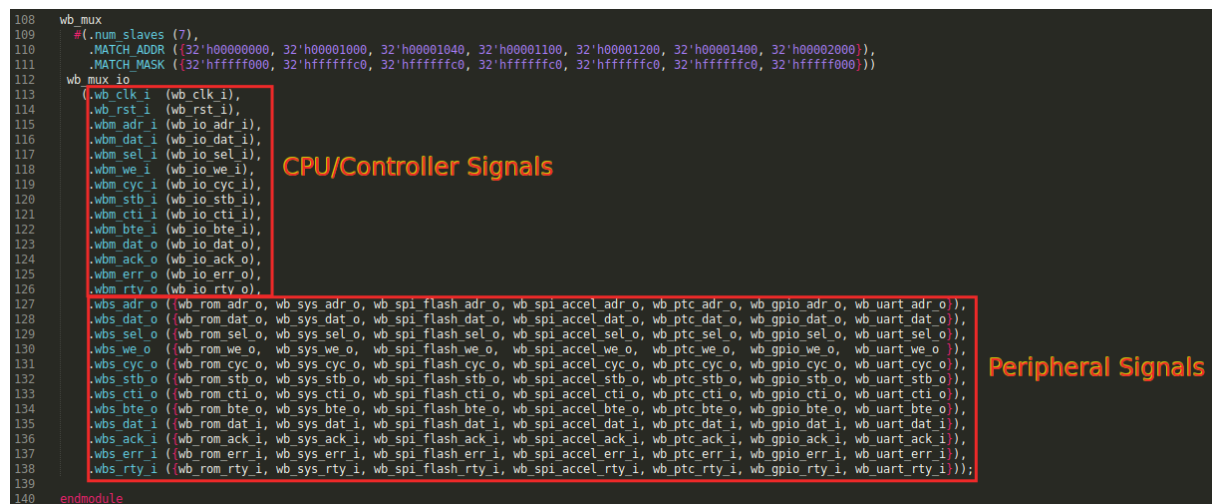


Figura 12. O multiplexador 7-1 seleciona o periférico a ligar ao CPU (*wb_intercon.v*).

O multiplexador seleciona o periférico a ler ou a escrever, ligando o CPU (sinais *wb_io_** – linhas 115-126 da Figura 12) com os barramento Wishbone de um periférico (linhas 127-138 da Figura 12), dependendo do endereço (linhas 110-111). Por exemplo, se o endereço gerado pelo CPU estiver na gama 0x80001100-0x8000113F, o módulo acelerómetro é selecionado, e assim os sinais *wb_io_** estão ligados aos sinais *wb_spi_accel_**.

6. EXERCÍCIOS AVANÇADOS

Exercício 2. O Recetor-Transmissor Assíncrono Universal / *Universal Asynchronous Receiver-Transmitter* (UART) é um protocolo de comunicação em série assíncrono. O Sistema RVfpga inclui um módulo UART na sua conceção básica (ver Figura 8), para o qual pode encontrar a especificação em:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/uart/docs/UART_spec.pdf

Em primeiro lugar, analise a implementação de baixo nível deste módulo no RVfpga, à

semelhança do que fizemos na Secção A para o Acelerómetro SPI.

Depois, criar um programa de Assembly RISC-V que imprime uma mensagem para a consola do PlatformIO através da porta de série. Utilize as seguintes sub-rotinas para aceder ao módulo UART. Antes de utilizar as sub-rotinas, tente compreendê-las. Aqui está um breve resumo de cada sub-rotina:

- A função `uartInit`: Inicializa o módulo UART.
- A função `uartSendByte`: Envia um byte através da UART.
- A função `uartSendString`: Enviar cadeia de caracteres através da UART.

```
# Register addresses for UART Peripheral
# -----

#define CONSOLE_ADDR 0x80001008
#define HALT_ADDR 0x80001009
#define UART_BASE 0x80002000

#define REG_BRDL (4*0x00) /* Baud rate divisor (LSB) */
#define REG_IER (4*0x01) /* Interrupt enable reg. */
#define REG_FCR (4*0x02) /* FIFO control reg. */
#define REG_LCR (4*0x03) /* Line control reg. */
#define REG_LSR (4*0x05) /* Line status reg. */
#define LCR_CS8 0x03 /* 8 bits data size */
#define LCR_1_STB 0x00 /* 1 stop bit */
#define LCR_PDIS 0x00 /* parity disable */

#define LSR_THRE 0x20
#define FCR_FIFO 0x01 /* enable XMIT and RCVR FIFO */
#define FCR_RCVRCLR 0x02 /* clear RCVR FIFO */
#define FCR_XMITCLR 0x04 /* clear XMIT FIFO */
#define FCR_MODE0 0x00 /* set receiver in mode 0 */
#define FCR_MODE1 0x08 /* set receiver in mode 1 */
#define FCR_FIFO_8 0x80 /* 8 bytes in RCVR FIFO */
```

```
.section .data

welcome:
.string "\nHELLO WORLD !!!\n"
```

```
# Function: Initialize UART peripheral
# call: by call ra, uartInit
# inputs: None
# outputs: None
# overwrites: t0, t1
# -----

uartInit:
    li    t0, UART_BASE

    /* Set DLAB bit in LCR */
    li    t1, 0x80
    sb    t1, REG_LCR(t0)

    /* Set divisor regs */
    li    t1, 27
    sb    t1, REG_BRDL(t0)

    /* 8 data bits, 1 stop bit, no parity, clear DLAB */
    li    t1, LCR_CS8 | LCR_1_STB | LCR_PDIS
    sb    t1, REG_LCR(t0)

    li    t1, FCR_FIFO | FCR_MODE0 | FCR_FIFO_8 | FCR_RCVRCLR | FCR_XMITCLR
    sb    t1, REG_FCR(t0)
```

```

/* disable interrupts */
sb      zero, REG_IER(t0)

ret

```

```

# Function: Send byte through UART
# call: by call ra, uartSendByte
# inputs: a0, byte to be sent
# outputs: None
# destroys: t0, t1
# -----

uartSendByte:
    li t1, UART_BASE

    /* Check for space in UART FIFO */
    lb t0, REG_LSR(t1)
    andi t0, t0, LSR_THRE
    beqz t0, uartSendByte
    sb a0, 0(t1)

ret

```

```

# Function: Send string through UART (terminated by \0)
# call: by call ra, uartSendString
# uses: uartSendByte
# inputs: a0, address of first character of string to be sent
# outputs: None
# destroys: t0, t1, t2
# -----

uartSendString:
    li t1, UART_BASE
    add t2,zero,ra # save caller address
    add a1,zero,a0 # use a1 as index
    /* Load first byte */
    lb a0, 0(a1)

internalNextChar:
    call ra, uartSendByte
    addi a1, a1, 1
    lb a0, 0(a1)
    bne a0, zero, internalNextChar

    add ra,zero,t2 # restore caller address
ret

```

Exercício 3. Implemente as seguintes três funções em linguagem C:

- `char uart_getchar(void)`: Esta função espera que o teclado envie um carácter através da UART para a placa Nexys A7 e depois devolve este carácter como parâmetro de saída. Lembre-se de que os caracteres são representados em código ASCII (<https://www.ascii-code.com/>).
- `int uart_putchar(char c)`: Esta função recebe um carácter como argumento de entrada e apresenta-o na consola série através da UART. Tem de implementar a sua própria função que acede aos registos UART em vez de utilizar a função `printfNexys` fornecido pelo BSP da WD (*Western Digital's Board Support Package*).

- `int SevSegDispl(char c)`: Esta função recebe um carácter como argumento de entrada e mostra-o no dígito mais à direita dos mostradores de 7 segmentos, deslocando os dígitos restantes uma posição para a esquerda (o dígito mais à esquerda é perdido). Dado que nos 7 segmentos só são exibidos os caracteres 0 a 9, A, B, C, D, E e F, para qualquer outro carácter pode simplesmente exibir um 0. Poderá expandir este exercício para mostrar mais caracteres utilizando o controlador estendido de exibição de 7 segmentos implementado no Lab 7 – Exercício 3.

Note que para implementar as duas primeiras funções deve utilizar o documento de especificação do módulo UART, disponível em:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/uart/docs/UART_spec.pdf

Com base nas três funções acima, crie um programa em C que recebe um carácter do teclado e o exibe tanto no terminal de série como nos mostradores de 7-Segmentos.

Para inicializar o módulo UART, pode usar a função `uartInit` fornecida pelo BSP da WD.

Exercício 4. Outro protocolo comum de comunicação em série é chamado Inter-Integrated Circuit (I2C) (é pronunciado “i dois cê” em português ou “eye two see” ou “eye squared see” em Inglês). O sensor de temperatura na placa Nexys A7 utiliza este protocolo. Expanda o Sistema RVfpga para incluir um controlador I2C, e conectá-lo com o sensor de temperatura ADT7420 presente na placa Nexys A7 (<https://www.analog.com/media/en/technical-documentation/data-sheets/adt7420.pdf>). Depois escreva um programa que comunique com este novo periférico e exiba a temperatura nos mostradores de 7 segmentos.