



IMAGINATION大學計劃

RVfpga實驗10

序列匯流排

1. 簡介

在本實驗中，我們首先介紹序列匯流排的工作方式，以及目前使用的其中一種最典型的序列匯流排「SPI匯流排」的主要特性（第2節）。然後，我們重點介紹Nexys A7開發板上的SPI加速計：首先分析該週邊設備的高階規格並輔以基本練習（第3和第4節），然後分析該週邊設備的低階實作並輔以一些進階練習（第5和第6節）。

2. 序列匯流排 - SPI匯流排

並行匯流排一次傳送多個位元，序列匯流排一次傳送一個位元。我們首先比較這兩種通訊方案，然後介紹目前最常用的其中一種序列匯流排，即SPI（序列週邊設備介面）通訊協定。關於這一重要的通訊協定，網上有許多相關資訊，可以幫助您加深瞭解。

根據先前實驗中的示範可知，嵌入式電子產品的主要用途在於連接處理器和電路以實現所需的功能。為了使處理器和電路共享資訊，二者必須採用相同的通訊協定。針對這類資料交換而定義的通訊協定多達數百個，但大體上主要分為以下兩種：並行介面或序列介面。

並行介面一次可以同時傳輸多個位元，因此需要多條資料匯流排（線路）。舉例來說，該通訊協定可以同時傳輸8位元、16位元甚至更多（請參閱圖1）。此外，這類介面需要依靠時鐘來確定何時準備好傳輸下一組 N 個資料位元。

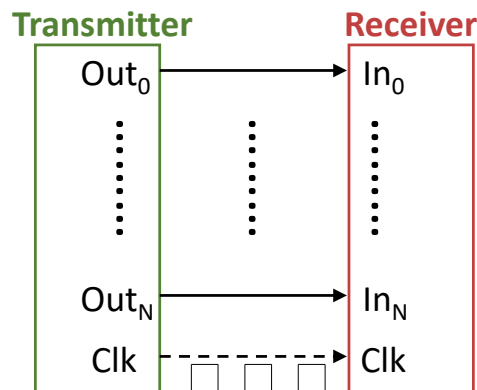


圖1. 8位元並行資料匯流排範例

與並行通訊不同的是，序列介面一次只能傳輸一個資料位元。序列介面使用一條線路即可工作，通常不會超過四條。圖2顯示了一個序列介面範例，其中包含一條資料線路和一條時鐘線路。每個新時鐘邊緣傳輸一個新資料位元。



圖2. 1位元序列資料匯流排範例

並行通訊的優勢在於快速、直接且相對易於實作，但是需要的輸入/輸出（I/O）線路也更多。因此，嵌入式系統往往會由於引腳數量有限而選擇序列通訊，放棄更高的傳輸速度。

SPI匯流排：

序列週邊設備介面（Serial Peripheral Interface，SPI）通訊協定是微控制器與週邊設備IC（如感應器、ADC、DAC、移位暫存器、SRAM等）之間使用最廣泛的通訊介面之一。SPI是一種基於控制器-週邊設備（以前稱為主-從）通訊的同步全雙工介面。

SPI匯流排通常採用4個連接埠進行通訊（請參閱圖3）：

- **SDO** – 序列資料輸出：控制器的輸出（傳送至週邊設備）
- **SDI** – 序列資料輸入：控制器的輸入（從週邊設備接收）
- **SCK** – 序列時鐘：從控制器傳送至週邊設備
- **CS** – 晶片選擇低電平有效訊號；控制器向週邊設備傳送訊號（選擇週邊設備時為0）

附註：SDO過去被稱為MOSI（主資料輸出，從資料輸入），SDI過去被稱為MISO（主資料輸入，從資料輸出）。雖然這些舊術語已經過時且遭到禁用，但某些文獻和文件中依舊在使用。

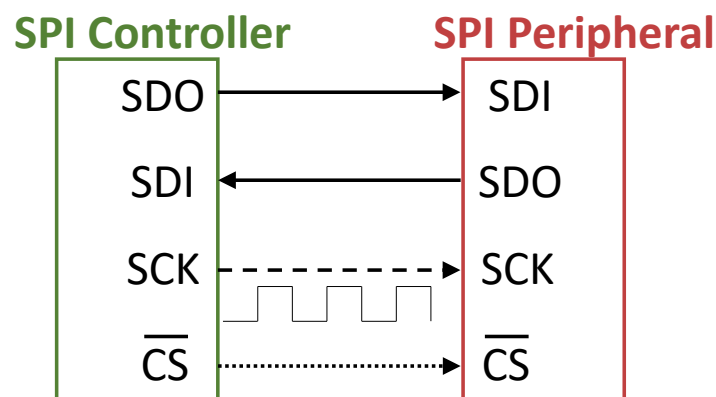


圖3. 具有一個SPI控制器和一個SPI週邊設備的系統範例

序列資料與時鐘的上升沿或下降沿同步。SPI屬於全雙工介面，控制器和週邊設備可以分別通過SDO和SDI線路同時傳送資料。SPI介面只有一個控制器，但可連接多個週邊設備。當連接多個週邊設備時，控制器可通過傳送多個低電平有效的晶片選擇訊號（CS \bar ）來選擇要存取的週邊設備。SDO和SDI是序列資料線路：SDO（序列資料輸出）是指從控制器向週邊設備輸出資料，SDI（序列資料輸入）是指從週邊設備向控制器輸入資料。

要啟動SPI通訊，控制器必須選擇週邊設備（通過將CS \bar 訊號置為有效，即CS \bar = 0），然後將時鐘訊號傳送到週邊設備。在SPI通訊期間，控制器與週邊設備分別通過SDO和SDI訊號同時相互傳輸資料。序列時鐘（SCK）邊緣用於同步資料採樣。

SPI介面還可提供CPOL和CPHA訊號，分別用於選擇時鐘的閒置狀態和採樣訊號的相位。當時鐘（SCK）的閒置狀態為0時，時鐘極性（CPOL）訊號為0；當時鐘（SCK）的閒置狀態為1時，時鐘極性（CPOL）訊號為1。時鐘相位（CPHA）訊號用於選擇在哪個時鐘相位傳送和採樣資料。當CPHA = 0時，資料在上升沿（即，SCK停止閒置之後的第一個上升沿以及隨後每個週期的上升沿）採樣（SDI或SDO上的）；因此資料（SDI和SDO）必須在下降沿變化，如圖4中上面的兩個時序圖所示。當CPHA = 1時，情況相反：資料在下降沿採樣，在上升沿變化，如圖4中下面的兩個時序圖所示。傳輸新資料的邊緣也稱為**移位邊緣**，因為這種序列通訊通常使用移位暫存器來實作。

在本實驗中，我們使用的SPI介面配置為CPHA = 0和CPOL = 0，因此SCK在低電平閒置，控制器和週邊設備在上升沿採樣資料，並在每個下降沿之後將新資料立即移到SDO或SDI線上，如圖4中上面的時序圖所示。請注意，當SCK閒置且即將上升時，SDO和SDI必須承載下一個資料位元組的最高有效位元。

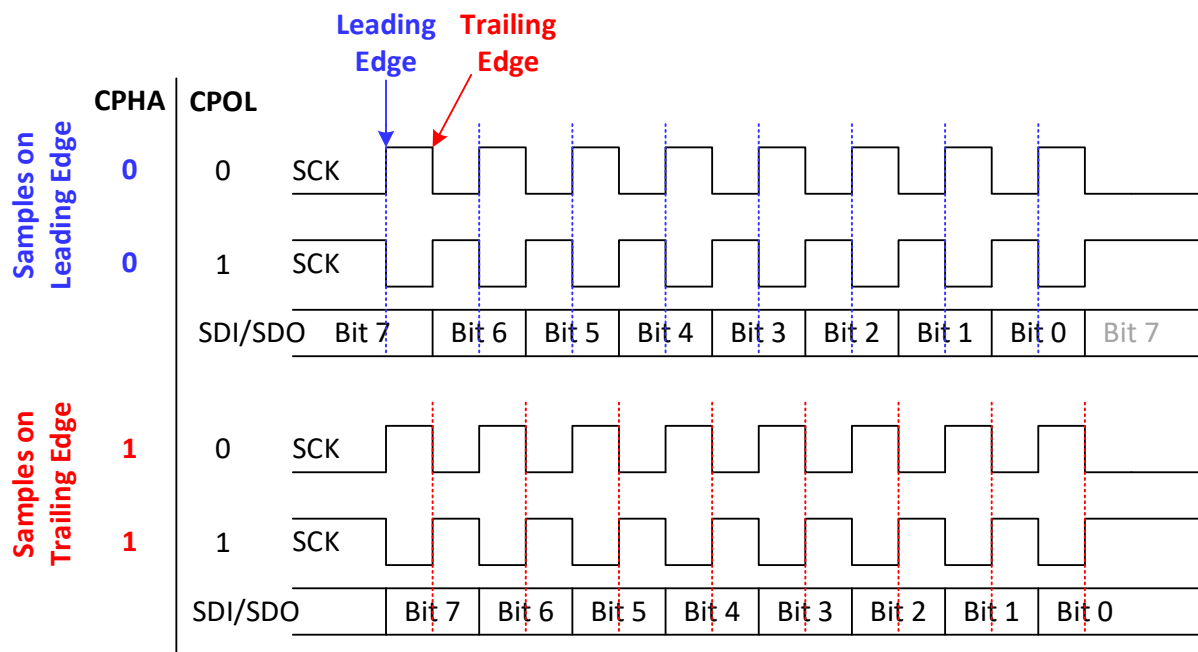


圖4. CPHA/CPOL與採樣/傳送資料的關係

3. SPI加速計：高階規格

許多週邊設備都包含SPI介面。例如，Nexys A7開發板上的加速計包含SPI介面。在本節中，我們將介紹RVfpga系統的SPI控制器的高階規格以及Nexys A7開發板上的ADXL362加速計，並輔以加速計的使用練習。

A. SPI控制器規格

RVfpga系統的SPI模組可從OpenCores（https://opencores.org/projects/simple_spi）取得。如果下載套件，其中會隨附一個文件，用於描述該模組的高階規格。該文件也可從以下位置獲取：

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

我們總結了SPI模組的主要操作和特性；如需瞭解詳細資訊，請參閱上述文件。

該模組的主要特性如下：

- 與摩托羅拉的SPI規格相容
- 使用8位元WISHBONE RevB.3經典介面
- 包含一個4項讀取FIFO緩衝區和一個4項寫入FIFO緩衝區
- 允許在傳輸1、2、3或4個位元組後產生中斷
- 可在較寬範圍的輸入時鐘頻率下工作
- 完全可綜合

SPI核心規格的第3節介紹了SPI模組內部可用的控制和狀態暫存器，每個暫存器都分配到不同的位址（請參閱表1）。SPI控制器的基本位址為**0x80001100**。下文詳細介紹了上述暫存器。

表1. SPI暫存器

名稱	位址	寬度	存取	說明
SPCR	0x80001100	8	R/W	控制暫存器
SPSR	0x80001108	8	R/W	狀態暫存器
SPDR	0x80001110	8	R/W	資料暫存器
SPER	0x80001118	8	R/W	擴展暫存器
SPCS	0x80001120	8	R/W	CS暫存器

SPI控制暫存器（SPCR）控制SPI模組；表2列出了該暫存器各個位元的功能。

表2. SPCR位元

位元	存取	名稱和說明
0:1	R/W	SPR SPI時鐘速率：這些位元選擇SPI時鐘速率。
2	R/W	CPHA 時鐘相位：確定採樣和傳送資料的相位。當CPHA = 1時，新資料在上升沿移動到線路上，隨後在下降沿進行採樣。當CPHA = 0時，新資料在下降沿移動到線路上，隨後在上升沿進行採樣。
3	R/W	CPOL 時鐘極性：確定SPI時鐘（SCK）的閒置狀態。當CPOL = 0時，SCK的閒置狀態為0；當CPOL = 1時，SCK的閒置狀態為1。
4	R/W	MSTR 模式選擇：當MSTR = 1時，SPI核心充當控制器。這是該控制器支援的唯一模式。
6	R/W	SPE SPI啟用：當SPE = 1時，啟用SPI核心。清除（SPE = 0）時，停用SPI核心。
7	R/W	SPIE SPI中斷啟用：如果SPIE = 1，並設定狀態暫存器中的SPI中斷標誌時，則將中斷主機。

SPI狀態暫存器（SPCR）提供SPI模組的狀態；表3列出了該暫存器各個位元的功能。

表3. SPSR位元

位元	存取	說明
0	R/W	RFEMPTY 讀取FIFO為空：如果RFEMPTY = 1，則讀取FIFO為空。
1	R/W	RFFULL 讀取FIFO已滿：如果RFFULL = 1，則讀取FIFO已滿。
2	R/W	WFEMPTY 寫入FIFO為空：如果WFEMPTY = 1，則寫入FIFO為空。
3	R/W	WFFULL 寫入FIFO已滿：如果WFFULL = 1，則寫入FIFO已滿。
6	R/W	WCOL 寫入衝突標誌：如果WCOL = 1，則表示寫入FIFO已滿時仍在寫入SPDATA暫存器。向WCOL寫入1可將該位元清零。
7	R/W	SPIF SPI中斷標誌：完成區塊傳輸後，SPIF = 1。如果SPIF置為有效（「1」）且SPIE設為1，則產生中斷。向SPIF寫入1可將其清零。

SPI資料暫存器（SPDR）提供要讀取或寫入的資料。SPI控制器包括4個8位元寫入緩衝區和4個8位元讀取緩衝區。

SPI擴展暫存器（SPER）提供了一些附加功能。表4列出了該暫存器包含的各個欄位。

表4. SPER位元

位元	存取	說明
0:1	R/W	ESPR 擴展SPI時鐘速率選擇：為SPR（SPI時鐘速率選擇）增加兩位元。
6:7	R/W	ICNT 中斷計數：確定傳輸區塊的大小。完成ICNT次傳輸後，SPIF位將設為1。這樣可以減少中斷服務呼叫，從而降低核心負擔。

SPI晶片選擇（SPCS）暫存器選擇要使用的週邊設備。該訊號的寬度可通過參數SS_WIDTH（SPI選擇寬度）進行配置。在RVfpga系統中，每個SPI介面只有一個對應的週邊設備，因此SS_WIDTH = 1。

任務：在SPI模組中找到暫存器SPCR、SPSR、SPDR、SPER和SPCS的宣告及其位址的定義。SPI模組位於資料夾[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/spi內。

B. ADXL362加速計規格

Nexys A7開發板包括一個類比裝置ADXL362加速計。有關該裝置的完整資訊，請造訪以下位置的資料手冊：

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

ADXL362是一款3軸MEMS加速計，在100 Hz輸出資料速率下的功耗不到2 μ A，在運動觸發喚醒模式下的功耗為270 nA。該加速計擁有12位元輸出解析度，但同時也提供8位元格式化資料，以便在低解析度也可滿足要求實現更高效的單位元組傳輸。測量範圍為 ± 2 g、 ± 4 g和 ± 8 g，其

中 ± 2 g範圍內的解析度為1 mg/LSB。當ADXL362處於測量模式時，將連續測量加速度資料並將其儲存在X資料、Y資料和Z資料暫存器中。

ADXL362加速計包括多個暫存器（表5），使用者可以使用這些暫存器對其進行配置，以及讀取加速度資料。寫入控制暫存器可以配置加速計，讀取裝置暫存器可以獲取加速計資料。與該裝置通訊時，必須指定一個暫存器位址，以及一個用於指示要進行讀取還是寫入通訊的標誌。當暫存器位址與通訊標誌傳送到裝置後，即可開始傳輸資料。

該加速計充當採用SPI通訊方案的週邊設備。圖圖5顯示了FPGA與加速計之間的連接。

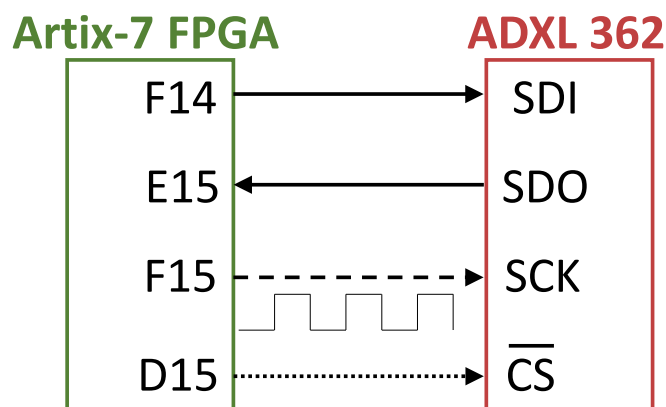


圖5. ADXL362加速計與Nexys A7開發板的連接

建議的SPI時鐘頻率範圍為1-5 MHz。SPI以SPI模式0（CPOL = 0且CPHA = 0）工作。SPI連接埠採用多位元組結構，其中第一個位元組指示通訊是執行暫存器讀取操作（0x0B）還是暫存器寫入操作（0x0A）：

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

圖6和圖7舉例說明了SPI控制器（控制器）與加速計（週邊設備）之間的通訊：圖6顯示了暫存器讀取操作，圖7顯示了暫存器寫入操作。

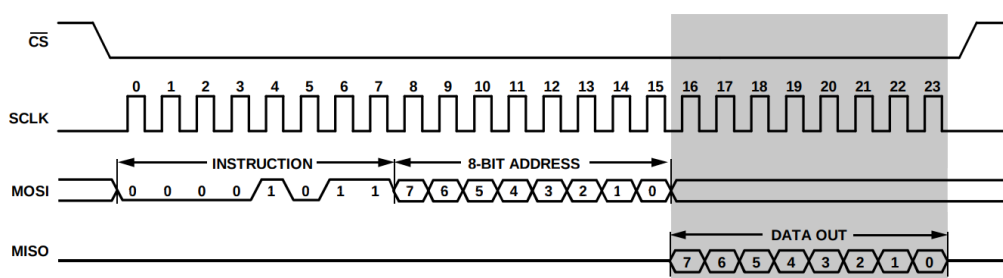


圖6. 暫存器讀取操作

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

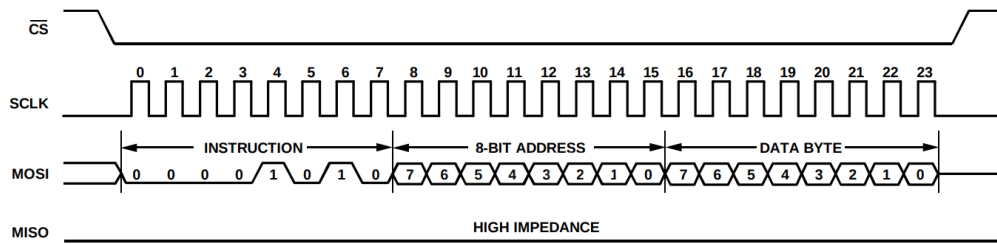


圖7. 暫存器寫入操作

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

表5列出了ADXL362加速計中提供的暫存器。有關完整的暫存器說明，請參閱ADXL362資料手冊：<https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>。

表5. ADXL362加速計暫存器

(表格來源：<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. 基本練習

練習1. 建立一個RISC-V組合語言程式，該程式讀取X軸、Y軸和Z軸加速度資料的高8位元，然後在8位7段顯示器上顯示這些值。有關配置和暫存器資訊，請參閱B節。使用以下子常式存取SPI模組。在使用這些子常式之前，請嘗試根據A節中提供的SPI模組相關資訊來加以理解。以下是每個子常式的簡要介紹：

- 函數spiInit：初始化SPI模組。
- 函數spiCS：將CS狀態傳送到SPCS暫存器。
- 函數spiCSUp：通過呼叫子常式spiCS將CS線拉為高電平。
- 函數spiCSDown：通過呼叫子常式spiCS將CS線拉為低電平。
- 函數spiSendGetData：通過SPI傳送位元組並獲取週邊設備資料。

```
# 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. 低階實作

A. SPI加速計低階實作

在本實驗的第一部分中，我們展示了如何使用RVfpga系統的SPI模組，而在本實驗的最後一部分，我們將介紹如何在RVfpga中實作SPI模組。本實驗的結構編排與之前的實驗類似，我們將分三個階段來分析SPI控制器：

1. SoC和加速計之間的物理連接（圖8中的左側陰影區域）
2. SPI控制器的整合，該控制器包含在SweRVolfX系統控制器中（圖8中的中間陰影區域）
3. SPI控制器與SweRV EH1核心之間的連接（圖8中的右側陰影區域）

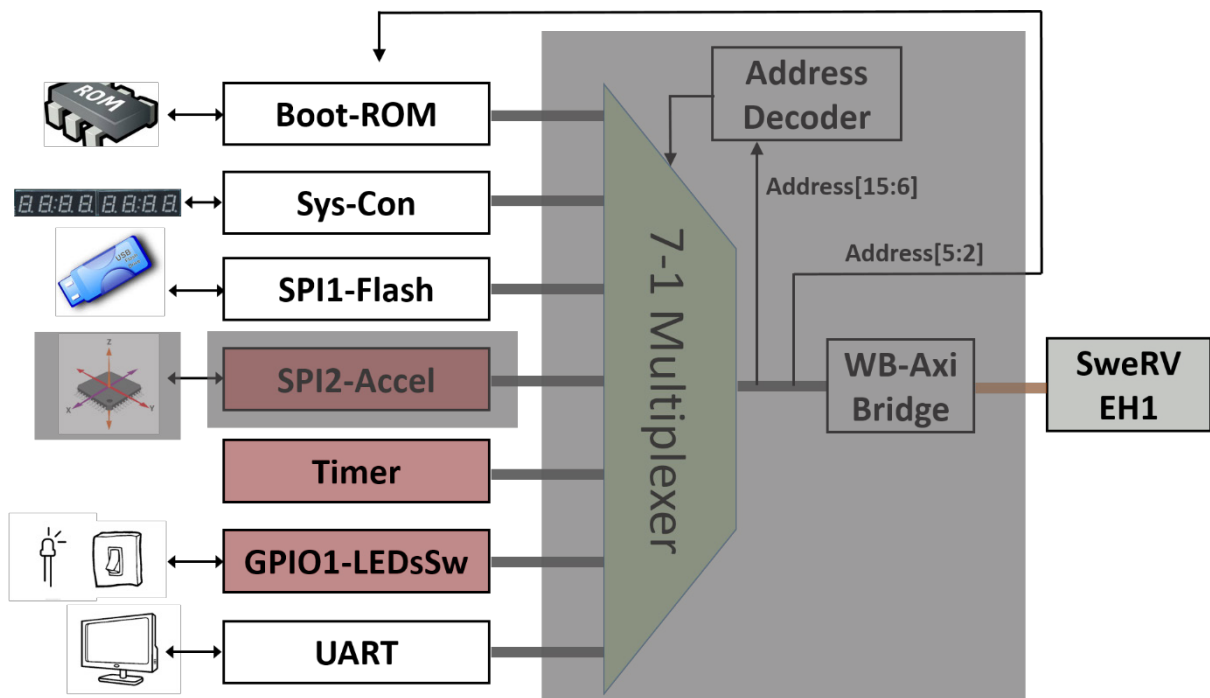


圖8. SPI控制器整合到RVfpga系統中

1. 加速計與SoC的物理連接

與其他週邊設備一樣，RVfpgaNexys限制檔必須包含與加速計的物理連接。專案的限制檔（`[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc`）定義了輸入/輸出SoC訊號與開發板裝置之間的連接。用於將加速計的四個引腳與SoC相連的訊號分別稱為：`o_accel_cs_n`、`o_accel_mosi`（相當於訊號SDO）、`i_accel_miso`（相當於訊號SDI）和`accel_sclk`。請注意，這些訊號的名稱已經過時，繼續使用的原因是為了與RVfpga系統中的OpenCores SPI模組所使用的名稱保持一致（您可以在圖11中查看該模組的實例化）。圖9顯示了定義這4個連接的Verilog程式碼片段。

```

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 }];

```

圖9. SoC與加速計的連接（檔案rvfpganexys.xdc）

在RVfpgaNexys頂層模組（即rvfpganexys模組）的第52-55行，您可以看到這四個與SoC連接的訊號（圖10的左半部分），該模組的末尾是這些訊號與swervolf_core模組的連接（圖10的右半部分）。

```

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

```

圖10. 加速計與頂層模組的連接（檔案rvfpganexys.sv）

任務：按照限制檔將這四個訊號（o_accel_cs_n、o_accel_mosi、i_accel_miso和accel_sclk）與SweRVolfX SoC模組相連。您將需要檢查以下檔案：

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

2. SPI2加速計模組到SoC的整合

在swervolf_core模組（[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v）的第387-403行，實例化加速計的SPI模組（請參閱圖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));
404

```

圖11. SPI2加速計模組的整合（檔案swervolf_core.v）

與週邊設備一樣，模組的介面可以分為兩個模組：Wishbone訊號（表6）和外部I/O訊號（表7）。Wishbone訊號允許SweRV EH1核心使用SPI通訊協定與ADC通訊。

表6. Wishbone訊號

連接埠	寬度	方向	說明
cyc_i	1	輸入	指示有效的匯流排週期（核心選擇）
adr_i	15	輸入	位址輸入
dat_i	32	輸入	資料輸入
dat_o	32	輸出	資料輸出
sel_i	4	輸入	指示資料匯流排上的有效位元組（在有效週期內，必須為0xf）
ack_o	1	輸出	認可輸出（指示正常交易終止）
err_o	1	輸出	錯誤認可輸出（指示異常交易終止）
rty_o	1	輸出	未使用
we_i	1	輸入	置為高電平時寫入交易
stb_i	1	輸入	指示有效的資料傳輸週期
inta_o	1	輸出	中斷輸出

表7. 外部I/O訊號

連接埠	寬度	方向	說明
miso_i	1	輸入	控制器資料輸入 - 週邊設備資料輸出
mosi_o	1	輸出	控制器資料輸出 - 週邊設備資料輸入
ss_o	1	輸出	晶片選擇
sck_o	1	輸出	系統時鐘

如圖11所示，Wishbone匯流排訊號中由核心提供的位址的位元[5:2]（`wb_m2s_spi_accel_adr[5:2]`）用於從5個可用的SPI暫存器中選擇1個（表1）。

3. SPI控制器與SweRV EH1核心的連接

如先前的實驗中所述，裝置控制器通過多工器和橋接器與SweRV EH1核心連接（圖8）。7:1多工器（圖12）在

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v` 檔中實作，該檔案在

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh` 檔的第104-205行實例化。後一個檔案包含在`swervolf_core`模組的第168行，該模組位於：

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

```

108 wb_mux
109 #(.num_slaves (7),
110 .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
111 .MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00}))
112 wb_mux_io
113 (.wb_clk_i (wb_clk_i),
114 .wb_rst_i (wb_rst_i),
115 .wbm_adr_i (wb_io_adr_i),
116 .wbm_dat_i (wb_io_dat_i),
117 .wbm_sel_i (wb_io_sel_i),
118 .wbm_we_i (wb_io_we_i),
119 .wbm_cyc_i (wb_io_cyc_i),
120 .wbm_stb_i (wb_io_stb_i),
121 .wbm_cti_i (wb_io_cti_i),
122 .wbm_bte_i (wb_io_bte_i),
123 .wbm_dat_o (wb_io_dat_o),
124 .wbm_ack_o (wb_io_ack_o),
125 .wbm_err_o (wb_io_err_o),
126 .wbm_rty_o (wb_io_rty_o),
127 .wbs_adr_o (wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o),
128 .wbs_dat_o (wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o),
129 .wbs_sel_o (wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o),
130 .wbs_we_o (wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o),
131 .wbs_cyc_o (wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o),
132 .wbs_stb_o (wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o),
133 .wbs_cti_o (wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o),
134 .wbs_bte_o (wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o),
135 .wbs_dat_i (wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i),
136 .wbs_ack_i (wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i),
137 .wbs_err_i (wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i),
138 .wbs_rty_i (wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i));
139
140 endmodule

```

CPU/Controller Signals

Peripheral Signals

圖12. 7-1多工器選擇與CPU連接的週邊設備 (`wb_intercon.v`)。

多工器選擇要讀取或寫入哪個週邊設備，根據位址（第110-111行）將CPU（`wb_io_*`訊號 – 圖12的第115-126行）與一個週邊設備的Wishbone匯流排（圖12的第127-138行）連接。例如，如果CPU產生的位址在0x80001100-0x8000113F範圍內，則選擇加速計模組，從而將訊號`wb_io_*`與訊號`wb_spi_accel_*`連接。

6. 進階練習

練習2. 通用異步收發器（UART）是一種異步序列通訊協定。RVfpga系統的基本設計中包含UART模組（參見圖8），該模組的相關規格位於以下位置：

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

首先，分析RVfpga中該模組的低階實作，與我們在SPI加速計的A節所做的分析類似。

然後，建立一個RISC-V組合語言程式，以透過序列連接埠向PlatformIO shell輸出一則訊息。使用以下子常式存取UART模組。在使用子常式之前，請首先嘗試理解。以下是每個子常式的簡要介紹：

- 函數uartInit：初始化UART模組。
- 函數uartSendByte：通過UART傳送位元組。
- 函數uartSendString：通過UART傳送字串。

```

# 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_RCVRLR 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_RCVRLR | 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
```

練習3. 用C語言實作以下三個函數：

- `char uart_getchar(void)`：該函數等待鍵盤透過UART向Nexys A7開發板傳送一個字元，然後將該字元作為輸出參數傳回。請記住，字元是用ASCII碼表示的（<https://www.ascii-code.com/>）。
- `int uart_putchar(char c)`：該函數接收一個字元作為輸入引數，並透過UART將其顯示在率列主控台上。必須自行實作存取UART暫存器的函數，而不是使用WD BSP（Western Digital的開發板支援套件）提供的printfNexys函數。
- `int SevSegDispl(char c)`：該函數接收一個字元作為輸入引數，並將其顯示在7段顯示器最右邊的一位數字上，而剩餘幾位數字向左移動一位（失去最左邊的一位數字）。鑒於7段顯示器只顯示字元0至9、A、B、C、D、E和F，任何其他字元均只能顯示0。您可以使用實驗7—練習3中實作的7段顯示器擴展控制器來擴展此練習以顯示更多字元。

請注意，要實作前兩個函數，必須使用UART模組規格文件，該文件位於：

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

基於上述三個函數，用C語言建立一個程式，該程式從鍵盤接收一個字元並將其顯示在序列終端機和7段顯示器上。

要初始化UART模組，可以使用WD BSP提供的uartInit函數。

練習4. 另一種常見的序列通訊協定稱為I2C（發音為「eye two see」或「eye squared see」）。Nexys A7開發板上的溫度感應器使用此通訊協定。本練習首先擴展RVfpga系統以包括I2C控制器，並將其與Nexys A7開發板上的ADT7420溫度感應器（<https://www.analog.com/media/en/technical-documentation/data-sheets/adt7420.pdf>）相連。然後編寫一個程式，與該新週邊設備通訊並在7段顯示器上顯示溫度。