



IMAGINATION大學計劃

RVfpga實驗6

I/O簡介

1. 簡介

在實驗6-10中，您將學習如何使用和擴展RVfpga的輸入/輸出（Input/Output，I/O）系統，以便實現RISC-V處理器與週邊設備的互動。下文概述了這些實驗涵蓋的主題：

- **實驗6**：瞭解如何使用連接到Nexys A7開發板上的LED、開關和按鈕的通用輸入/輸出（General-Purpose Input/Output，GPIO）引腳
- **實驗7**：瞭解如何使用板上7段顯示器
- **實驗8**：瞭解如何使用計時器
- **實驗9**：瞭解如何使用中斷與外部裝置連接
- **實驗10**：瞭解如何將RVfpga系統與板上SPI加速計連接

在本實驗中，我們首先將介紹通用I/O系統的主要特性以及RVfpga系統中用到的特性（第2節），然後介紹通用GPIO控制器的簡化理論版本（第3節）。最後，我們將重點關注SweRVolfX SoC中使用的GPIO控制器：首先分析其高階規格並介紹基本練習（第4部分和第5部分）。我們將通過分析其低階實作、在Verilator中模擬RVfpgaSim並介紹進階練習來總結實驗（第6節和第7節）。

我們在實驗7-10中使用相同的通用結構。在開始各節中，我們將先介紹I/O控制器的高階規格（其主要特性、暫存器及其操作以及記憶體映射），然後介紹一些基本練習來實踐如何使用週邊設備。在進階各節中，我們將介紹控制器的低階實作，並通過練習幫助您瞭解如何修改低階實作以及編寫測試修改的程式。

講師注意事項：您可以根據課程等級選擇練習的複雜度。例如，在第一年/第二年的課程（如電腦基礎或電腦組成原理）中，基本練習（本實驗的第5節）十分合適。但是，在更進階的課程（如電腦體系結構或嵌入式系統設計）中，可以使用基礎練習和進階練習（本實驗的第5節到第7節）。

2. 輸入/輸出架構

圖1闡述了馮·諾依曼架構的結構，這種架構由CPU、記憶體和I/O系統三個主要模組組成。在實驗6-10中，我們將重點介紹CPU與輸入/輸出（I/O）裝置的互動。I/O裝置也稱為週邊設備或簡稱為裝置。下面我們將概述每個主要單元的作用：

- **CPU**：CPU是所有I/O操作的發起者，也是所有I/O交易的**控制器**（過去稱為「主裝置」，但該術語已被棄用）。直接記憶體存取（Direct-Memory-Access，DMA）控制器（DMAC）也可以充當控制器，但本實驗中不涉及。
- **裝置控制器**：裝置控制器等待來自**控制器的讀/寫請求**來執行操作。裝置控制器在I/O系統中充當**週邊設備**（以前稱為「從屬裝置」，但該術語已被棄用）。從概念上講，裝置控制器由一系列可從**控制器**存取的**暫存器**組成。這些暫存器的值指示**週邊設備**應當執行哪些操作。
- **互連**（匯流排和交錯器等）在**控制器**和**週邊設備**之間建立了一條路徑。互連通常通過**橋接器**連接的多個層實作，作用是防止某些裝置降低整個系統的速度。

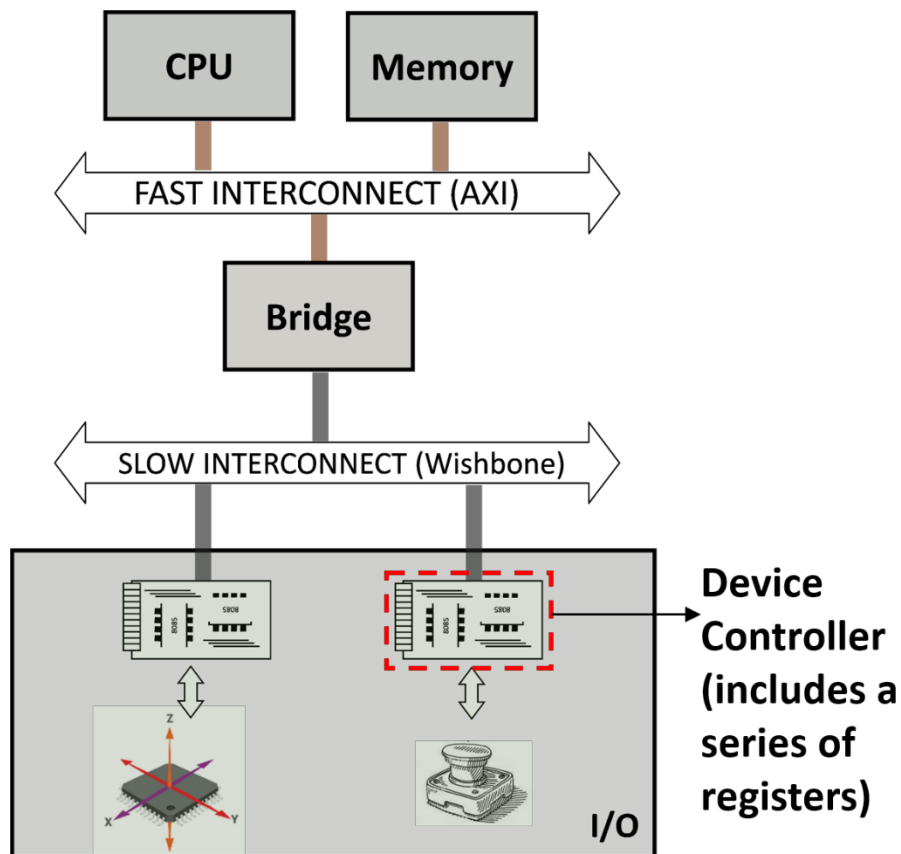


圖1通用計算系統

圖2顯示了RVfpga的I/O系統。系統包括以下七個週邊設備：

- 連接到GPIO1模組的LED和開關（被視為單一週邊設備）
- 連接到系統控制器模組的7段顯示器
- 連接到SPI1模組的快閃記憶體
- 連接到SPI2模組的加速計
- 計時器
- UART
- 開機ROM

多工器在七種可能性中選擇一個週邊設備並將其與CPU連接。請注意，必須採用Wishbone-AXI橋接器，因為週邊設備使用Wishbone匯流排（灰色），而SweRV EH1核心使用AXI橋接器（橙色）。

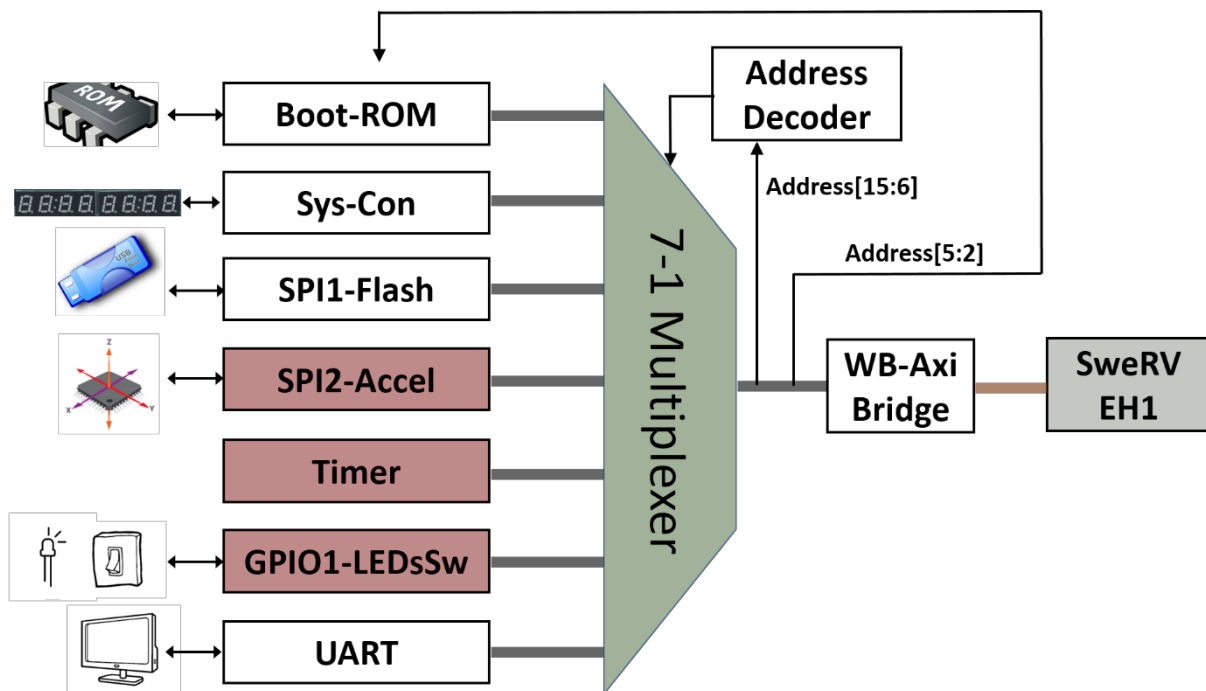


圖2. RVfpga系統中的I/O系統

任務：在SoC中找到圖2的每個元素。您將需要檢查以下檔案和目錄：

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v (主檔案，其中圖2的元素已實例化)。
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v
 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh

如《RVfpga入門指南》所述，原始SweRVolf (<https://github.com/chipsalliance/Cores-SweRVolf>) 僅包含圖2所示的一些週邊設備：具體包括開機ROM、系統控制器（無7段顯示器）、SPI快閃記憶體和UART（在圖2中顯示為白色）。根據本GSG，SweRVolfX SoC使用全新週邊設備擴展原始SweRVolf SoC：SPI加速計、計時器、GPIO模組（在圖2中顯示為紅色）和7段顯示控制器（擴展SweRVolf的現有系統控制器）。

每個週邊設備從處理器接收值和/或將值傳送回處理器。記憶體位址保留用於I/O值，分別稱為暫存器、記憶體映射I/O暫存器和裝置控制器暫存器。要向週邊設備傳送值，CPU會將值儲存在指定的記憶體位址（即，記憶體映射暫存器）。要從週邊設備讀取值，CPU從指定的記憶體位址載入值。這樣一來，CPU只需通過簡單的載入/儲存操作即可配置裝置、檢查裝置狀態或對裝置執行資料讀/寫操作。

圖2中的多工器使用位址[15:6]選擇請求的裝置控制器。裝置控制器使用位址[5:2]從幾個暫存器中選擇用來控制裝置的暫存器。

3. 通用輸入/輸出 (GPIO)

通用I/O (General-Purpose I/O, GPIO) 控制器將外部數位引腳提供給程式設計師。在程式中的任何給時間，這些引腳都可以配置為輸入或輸出。該名稱是按引腳分配的，如有需要，可以在整個程式中變更。GPIO引腳可以連接到外部裝置，例如LED、開關和按鈕。

圖3圖示了將一個外部引腳連接到CPU的通用GPIO模組的簡化圖。該引腳可以連接到任何輸入/輸出裝置，例如LED和開關等。在這張圖中，該引腳連接到三態緩衝區（在圖中以綠色突出顯示）。該緩衝區允許程式設計師將引腳配置為輸入或輸出。如果啟用三態緩衝區，則該引腳充當輸出（例如，用於驅動LED）。如果停用三態緩衝區，則該引腳充當輸入（例如，用於讀取開關值）。

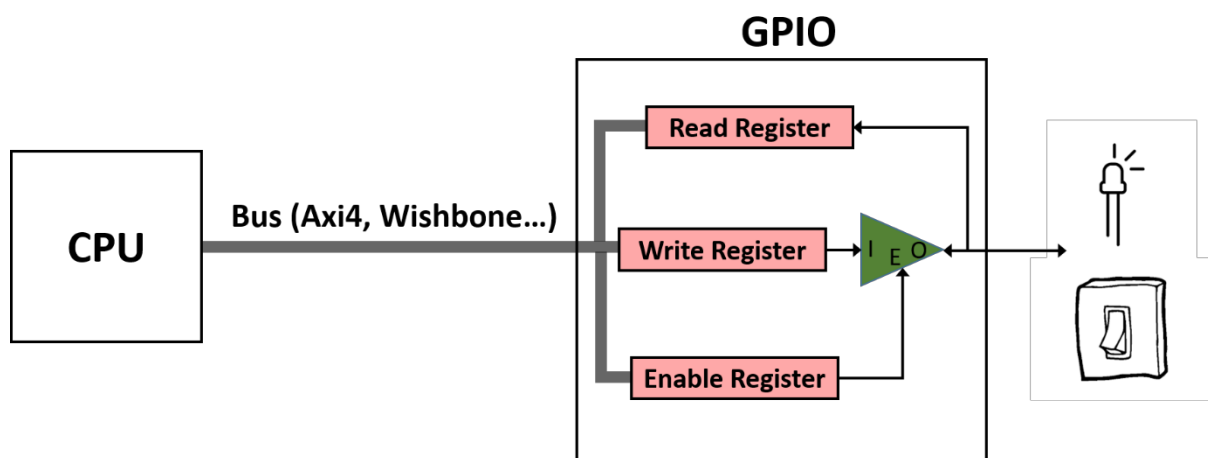


圖3. GPIO簡化電路

三態緩衝區既可充當常規緩衝區（啟用時），也可以具有懸空的輸出（停用時）。三態緩衝區具有兩個輸入（E（啟用）和I（輸入））和一個輸出（O），其真值表如表1所示。當E為1時，三態緩衝區充當常規緩衝區，輸出（O）和輸入（I）相同。當E為0時，輸入和輸出之間不存在連接且輸出（O）未被驅動；O處於懸空狀態。在圖3中，要將引腳配置為輸出，E應為1，這樣CPU便可驅動引腳。當引腳配置為輸入時，E為0，這樣CPU便無法驅動該引腳，此時可由週邊設備驅動引腳。

表1. 三態真值表

E	I	O
0	0	高阻態
0	1	高阻態
1	0	0
1	1	1

RVfpga系統使用記憶體對映I/O讀寫儲存在這些暫存器中的值。例如，假設圖3中的引腳連接到開關，且GPIO中的三個暫存器的映射如下：

- 讀取暫存器 = 位址0x80001400
- 寫入暫存器 = 位址0x80001404
- 啟用暫存器 = 位址0x80001408

要讀取開關的狀態，請執行以下操作：

1. 通過向啟用暫存器寫入0（即，向位址0x80001408執行儲存0的指令）來將引腳配置為輸入。
2. 通過向位址0x80001400執行載入指令讀取讀取暫存器。

4. GPIO高階規格

在本節中，我們首先分析SweRVolfX GPIO的高階規格，然後提供一個使用該週邊設備的練習。

A. GPIO高階規格

可從OpenCores（<https://opencores.org/projects/gpio>）取得SweRVolfX中使用的GPIO模組。下載的OpenCore GPIO模組所隨附的gpio_spec.pdf文件介紹了模組的高階規格，這篇文件的下載位置為：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/docs/gpio_spec.pdf*。在本實驗中，我們總結了GPIO模組的主要操作和特性。不過，您也可以在*gpio_spec.pdf*中獲取完整的規格。

GPIO模組的主要特性如下：

- 使用Wishbone互連。
- 僅用作週邊設備。
- 使用者可以使用1-32個GPIO引腳。
- 可以並行使用多個GPIO模組（也稱為GPIO核心）來存取超過32個GPIO引腳。
- 所有GPIO引腳都可以：
 - 用作雙向引腳（這種情況下需要外部雙向I/O單元）。
 - 啟用三態或漏極開路（這種情況下需要外部三態或漏極開路I/O單元）。
- 程式設計為輸入的GPIO引腳：
 - 可通過暫存器進行操作。
 - 可以向CPU發出中斷請求。

GPIO核心規格的第4節介紹了GPIO模組內部可用的控制和狀態暫存器。其中每個暫存器都分配給一個不同的位址，如表2所示。GPIO暫存器的基本位址為**0x80001400**。

表2. GPIO暫存器

名稱	位址	寬度	存取	說明
RGPIO_IN	0x80001400	1-32	R	GPIO輸入資料
RGPIO_OUT	0x80001404	1-32	R/W	GPIO輸出資料
RGPIO_OE	0x80001408	1-32	R/W	GPIO輸出驅動器啟用
RGPIO_INTE	0x8000140C	1-32	R/W	中斷啟用
RGPIO_PTRIG	0x80001410	1-32	R/W	觸發中斷的事件類型
RGPIO_AUX	0x80001414	1-32	R/W	多工處理輔助輸入與GPIO輸出
RGPIO_CTRL	0x80001418	2	R/W	控制暫存器

RGPIO_INTS	0x8000141C	1-32	R/W	中斷狀態
RGPIO_ECLK	0x80001420	1-32	R/W	啟用gpio_eclk以鎖存RGPIO_IN
RGPIO_NEC	0x80001424	1-32	R/W	選擇gpio_eclk的有效邊緣

盡管OpenCore的GPIO模組比圖3中所示的簡化版本更為複雜，但我們仍可從圖3識別出以下三個暫存器：讀取（輸入）、寫入（輸出）和啟用。在OpenCore的GPIO模組中，這些暫存器分別稱為：RGPIO_IN、RGPIO_OUT和RGPIO_OE，分別映射到0x80001400、0x80001404和0x80001408。

任務：在GPIO模組中找到暫存器RGPIO_IN、RGPIO_OUT和RGPIO_OE的宣告及其位址的定義。GPIO模組位於：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/gpio_top.v*。

RGPIO_IN暫存器鎖存通用輸入。RGPIO_OUT暫存器驅動通用輸出。RGPIO_OE將每個I/O引腳配置為輸入或輸出。設定啟用位元（位於RGPIO_OE內部）時，將啟用相應的通用輸出驅動器，因此可將引腳連接到輸出週邊設備，例如LED。清除啟用位元時，輸出驅動器將在漏極開路（也稱為三態或高阻態）模式下工作，因此可將引腳連接到輸入週邊設備，例如開關或按鈕。

在RVfpgaNexys中，GPIO模組的前16個GPIO引腳（引腳15:0）連接到Nexys A7電路開發板上的16個LED。GPIO控制器的後16個GPIO引腳（引腳31:16）連接到16個開發板上開關。

5. 基本練習

練習1. 編寫一個RISC-V組合語言程式和一個C程式，此程式會顯示一個由四個點亮的LED組成的模組重複地從開發板上16個LED的一側移動到另一側。另外，還包括兩個控制速度和方向的開關。Switch[0]用於變更速度，而Switch[1]用於變更方向，如下所示：

- 如果Switch[0]為ON（高電平），則點亮的LED應快速移動。否則，點亮的LED應緩慢移動。可以定義「快」和「慢」的含義，但任何一種速度都必須明顯可見，並且必須能夠僅通過觀察便察覺到速度的不同。
- 如果Switch[1]為ON（高電平），則點亮的LED應重複從右向左移動（當它們到達最左側的LED時，將從右側重新開始）。否則，點亮的LED應重複地從左向右移動。

下面的圖4顯示了Nexys A7開發板，其中突出顯示了LED和開關。

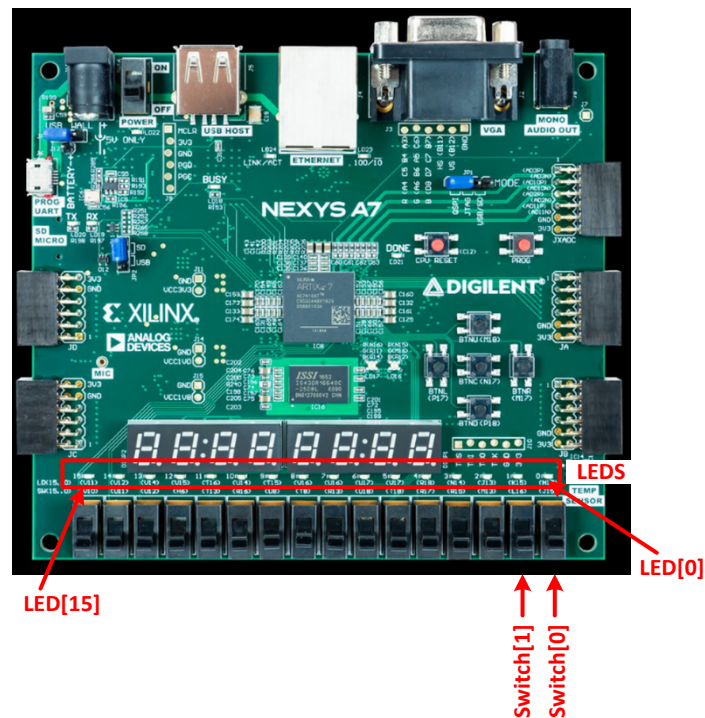


圖4. Nexys A7 FPGA開發板：LED和開關

提示：回顧一下可知，開關已連接到暫存器映射I/O暫存器的引腳31:16。因此，要讀取開關[0]，您需要向RGPIO_OE[16]寫入0，然後讀取RGPIO_IN[16]的值。您將需要適當地配置RGPIO_OE來存取其他LED和開關。

6. GPIO低階實作和模擬

在本節中，我們將介紹SweRVolfX中使用的GPIO的低階細節。隨後，我們會修改RVfpgaSim並在Verilator中執行一個模擬範例，以獲得簡單的組合語言程式碼範例。最後，我們提供了一些練習，您需要先模擬RVfpgaSim，然後對其進行修改以新增GPIO週邊設備，最後編寫一個使用該新週邊設備的程式。

A. GPIO低階實作

現在，您已經熟悉如何使用記憶體映射I/O存取GPIO引腳，我們接下來深入瞭解GPIO的低階細節。GPIO可以分為三個主要部分，如圖5所示：(1) RVfpgaNexys與板上LED/開關的外部連接（圖5中的左側陰影區域）；(2) GPIO模組到SweRVolfX SoC的整合（圖5中的中間陰影區域）；(3) GPIO與SweRV EH1核心之間的連接（圖5中的右側陰影部分）。

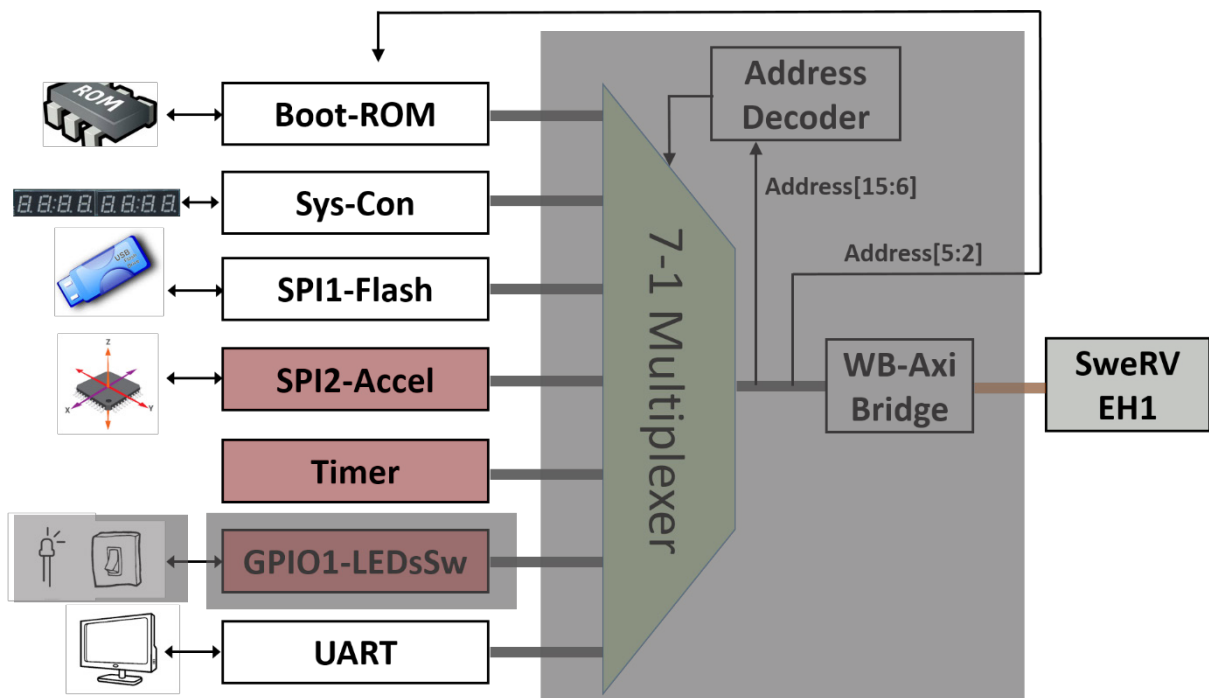


圖5. GPIO分析（3個階段）

i. LED/開關與SoC的連接

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。每個開發板裝置都與給定的FPGA引腳關聯。例如，開發板上最右邊的開關Switch[0]通過印刷開發板（Printed Circuit Board，PCB）走線連接到FPGA引腳J15。

Nexys A7開發板包括16個LED和16個開關。將16個LED與SoC（稱為rvfpganexys，在內部檔案`[RVfpgaPath]/RVfpga/src/rvfpganexys.sv`中提供）的頂層模組相連的訊號名稱為`o_led[15:0]`，將16個開關與頂層模組相連的訊號名稱為`i_sw[15:0]`。圖6顯示了Xilinx設計限制（xdc）檔rvfpganexys.xdc部分（位於`[RVfpgaPath]/RVfpga/src`），其中定義了訊號與FPGA引腳之間的這32個連接。

```

26 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]
27 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]
28 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[2] }]
29 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[3] }]
30 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { i_sw[4] }]
31 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[5] }]
32 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[6] }]
33 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[7] }]
34 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[8] }]
35 set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[9] }]
36 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[10] }]
37 set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[11] }]
38 set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { i_sw[12] }]
39 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { i_sw[13] }]
40 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { i_sw[14] }]
41 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { i_sw[15] }]
42
43 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
44 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { o_led[1] }]
45 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { o_led[2] }]
46 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { o_led[3] }]
47 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { o_led[4] }]
48 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { o_led[5] }]
49 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { o_led[6] }]
50 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { o_led[7] }]
51 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { o_led[8] }]
52 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { o_led[9] }]
53 set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { o_led[10] }]
54 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { o_led[11] }]
55 set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { o_led[12] }]
56 set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { o_led[13] }]
57 set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { o_led[14] }]
58 set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { o_led[15] }]

```

圖6. `i_sw[15:0]`與開發板上開關的連接，以及`o_led[15:0]`與開發板上LED的連接
(`rvfpganexys.xdc`檔)

頂端模組（`rvfpganexys`）的第48-49行顯示了連接到SoC的這兩個訊號（圖7的左側部分），該模組的末尾顯示了其與`swervolf_core`模組的連接（圖7的右側部分）。請注意，`i_sw`和`o_led`訊號合併到訊號`io_data`中（第257行），這是與`swervolf_core`模組中的GPIO連接的32位元輸入/輸出訊號（如後面的圖8所示）。此外，請注意`o_led`訊號通過中間訊號`gpio_out`（第266行）鎖存。

```

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,

```

```

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

```

圖7. LED與開關和頂端模組的連接（`rvfpganexys.sv`）

任務：追蹤從限制檔到SweRVolf SoC模組的這兩個訊號（`i_sw`和`o_led`），它們在SweRVolf SoC模組中合併為`io_data`。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

在上節中我們提到過，在RVfpgaNexys中，GPIO模組的前16個GPIO引腳（15到0）連接到16個開發板上LED，而GPIO控制器的後16個GPIO引腳（31到16）連接到16個開發板上開關。這是否符合本節和圖8中描述的實作？

ii. GPIO模組到SoC的整合

在swervolf_core模組（[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v）的第299-354行中，GPIO模組經過實例化並整合到SoC中（請參閱圖8）。

```

299 // GPIO - Leds and Switches
300 wire [31:0] en_gpio;
301 wire      gpio_irq;
302 wire [31:0] i_gpio;
303 wire [31:0] o_gpio;
304
305 bidirec gpio0 (.oe(en_gpio[0]), .inp(o_gpio[0]), .outp(i_gpio[0]), .bidir(io_data[0]));
306 bidirec gpio1 (.oe(en_gpio[1]), .inp(o_gpio[1]), .outp(i_gpio[1]), .bidir(io_data[1]));
307 bidirec gpio2 (.oe(en_gpio[2]), .inp(o_gpio[2]), .outp(i_gpio[2]), .bidir(io_data[2]));
308 bidirec gpio3 (.oe(en_gpio[3]), .inp(o_gpio[3]), .outp(i_gpio[3]), .bidir(io_data[3]));
309 bidirec gpio4 (.oe(en_gpio[4]), .inp(o_gpio[4]), .outp(i_gpio[4]), .bidir(io_data[4]));
310 bidirec gpio5 (.oe(en_gpio[5]), .inp(o_gpio[5]), .outp(i_gpio[5]), .bidir(io_data[5]));
311 bidirec gpio6 (.oe(en_gpio[6]), .inp(o_gpio[6]), .outp(i_gpio[6]), .bidir(io_data[6]));
312 bidirec gpio7 (.oe(en_gpio[7]), .inp(o_gpio[7]), .outp(i_gpio[7]), .bidir(io_data[7]));
313 bidirec gpio8 (.oe(en_gpio[8]), .inp(o_gpio[8]), .outp(i_gpio[8]), .bidir(io_data[8]));
314 bidirec gpio9 (.oe(en_gpio[9]), .inp(o_gpio[9]), .outp(i_gpio[9]), .bidir(io_data[9]));
315 bidirec gpio10 (.oe(en_gpio[10]), .inp(o_gpio[10]), .outp(i_gpio[10]), .bidir(io_data[10]));
316 bidirec gpio11 (.oe(en_gpio[11]), .inp(o_gpio[11]), .outp(i_gpio[11]), .bidir(io_data[11]));
317 bidirec gpio12 (.oe(en_gpio[12]), .inp(o_gpio[12]), .outp(i_gpio[12]), .bidir(io_data[12]));
318 bidirec gpio13 (.oe(en_gpio[13]), .inp(o_gpio[13]), .outp(i_gpio[13]), .bidir(io_data[13]));
319 bidirec gpio14 (.oe(en_gpio[14]), .inp(o_gpio[14]), .outp(i_gpio[14]), .bidir(io_data[14]));
320 bidirec gpio15 (.oe(en_gpio[15]), .inp(o_gpio[15]), .outp(i_gpio[15]), .bidir(io_data[15]));
321 bidirec gpio16 (.oe(en_gpio[16]), .inp(o_gpio[16]), .outp(i_gpio[16]), .bidir(io_data[16]));
322 bidirec gpio17 (.oe(en_gpio[17]), .inp(o_gpio[17]), .outp(i_gpio[17]), .bidir(io_data[17]));
323 bidirec gpio18 (.oe(en_gpio[18]), .inp(o_gpio[18]), .outp(i_gpio[18]), .bidir(io_data[18]));
324 bidirec gpio19 (.oe(en_gpio[19]), .inp(o_gpio[19]), .outp(i_gpio[19]), .bidir(io_data[19]));
325 bidirec gpio20 (.oe(en_gpio[20]), .inp(o_gpio[20]), .outp(i_gpio[20]), .bidir(io_data[20]));
326 bidirec gpio21 (.oe(en_gpio[21]), .inp(o_gpio[21]), .outp(i_gpio[21]), .bidir(io_data[21]));
327 bidirec gpio22 (.oe(en_gpio[22]), .inp(o_gpio[22]), .outp(i_gpio[22]), .bidir(io_data[22]));
328 bidirec gpio23 (.oe(en_gpio[23]), .inp(o_gpio[23]), .outp(i_gpio[23]), .bidir(io_data[23]));
329 bidirec gpio24 (.oe(en_gpio[24]), .inp(o_gpio[24]), .outp(i_gpio[24]), .bidir(io_data[24]));
330 bidirec gpio25 (.oe(en_gpio[25]), .inp(o_gpio[25]), .outp(i_gpio[25]), .bidir(io_data[25]));
331 bidirec gpio26 (.oe(en_gpio[26]), .inp(o_gpio[26]), .outp(i_gpio[26]), .bidir(io_data[26]));
332 bidirec gpio27 (.oe(en_gpio[27]), .inp(o_gpio[27]), .outp(i_gpio[27]), .bidir(io_data[27]));
333 bidirec gpio28 (.oe(en_gpio[28]), .inp(o_gpio[28]), .outp(i_gpio[28]), .bidir(io_data[28]));
334 bidirec gpio29 (.oe(en_gpio[29]), .inp(o_gpio[29]), .outp(i_gpio[29]), .bidir(io_data[29]));
335 bidirec gpio30 (.oe(en_gpio[30]), .inp(o_gpio[30]), .outp(i_gpio[30]), .bidir(io_data[30]));
336 bidirec gpio31 (.oe(en_gpio[31]), .inp(o_gpio[31]), .outp(i_gpio[31]), .bidir(io_data[31]));
337
338 gpio_top gpio_module(
339     .wb_clk_i      (clk),
340     .wb_rst_i      (wb_rst),
341     .wb_cyc_i      (wb_m2s_gpio_cyc),
342     .wb_adr_i      ({2'b0,wb_m2s_gpio_adr[5:2],2'b0}),
343     .wb_dat_i      (wb_m2s_gpio_dat),
344     .wb_sel_i      (4'b1111),
345     .wb_we_i      (wb_m2s_gpio_we),
346     .wb_stb_i      (wb_m2s_gpio_stb),
347     .wb_dat_o      (wb_s2m_gpio_dat),
348     .wb_ack_o      (wb_s2m_gpio_ack),
349     .wb_err_o      (wb_s2m_gpio_err),
350     .wb_inta_o     (gpio_irq),
351     // External GPIO Interface
352     .ext_pad_i     (i_gpio[31:0]),
353     .ext_pad_o     (o_gpio[31:0]),
354     .ext_padoe_o   (en_gpio));
355

```

圖8. GPIO模組的整合（swervolf_core.v檔）

模組的介面可以分為兩個部分：允許SweRV EH1核心使用控制器/週邊設備模型與GPIO通訊的Wishbone訊號（表3），以及外部I/O訊號（表4）。

表3. Wishbone訊號

連接埠	寬度	方向	說明
wb_cyc_i	1	輸入	指示有效的匯流排週期（核心選擇）
wb_adr_i	15	輸入	位址輸入
wb_dat_i	32	輸入	資料輸入
wb_dat_o	32	輸出	資料輸出
wb_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）
wb_ack_o	1	輸出	認可輸出（指示正常交易終止）
wb_err_o	1	輸出	錯誤認可輸出（指示異常交易終止）
wb_rty_o	1	輸出	未使用
wb_we_i	1	輸入	置為高電平時寫入交易
wb_stb_i	1	輸入	指示有效的資料傳輸週期
wb_inta_o	1	輸出	中斷輸出

表4. 外部I/O訊號

連接埠	寬度	方向	說明
in_pad_i	1-32	輸入	GPIO輸入
out_pad_o	1-32	輸出	GPIO輸出
oen_padoen_o	1-32	輸出	GPIO輸出驅動器啟用（用於三態或漏極開路驅動器）

如圖8的第342行所示，您可從Wishbone匯流排訊號中由核心提供的位址的位元5:2（`wb_m2s_gpio_adr[5:2]`）中，於從10個可用的記憶體映射暫存器中選擇1個。這四位元通過`wb_adr_i`訊號提供給GPIO核心（如圖8所示）。

輸入`ext_pad_i`直接與GPIO讀取暫存器（`RGPIO_IN`）連接。同理，輸出`ext_pad_o`直接與GPIO寫入暫存器（`RGPIO_OUT`）連接。這兩個訊號通過32個三態緩衝區模組連接到LED和開關（`i_gpio`、`o_gpio`和`io_data`）（圖8的第305-336行）。這樣，全部32個引腳都可以配置為輸入或輸出。本例中的低16個引腳（引腳15:0）連接到LED（圖7），因此必須配置為輸出；高16個引腳（31:16）連接到開關（圖7），因此必須配置為輸入。我們透過在`swervolf_core`模組的末尾（第634-640行）包含以下模組來實作這32個三態緩衝器：

```
module bidirec (input wire oe, input wire inp, output wire outp, inout wire bidir);
    assign bidir = oe ? inp : 1'bZ ;
    assign outp  = bidir;
endmodule
```

任務：GPIO引腳（`io_data`）通過三態緩衝區連接到GPIO模組（請參閱圖8）。分析三態緩衝區的啟用訊號的兩種可能狀態（`oe = 0`和`oe = 1`）。

考慮到GPIO模組和開發板上LED/開關之間的連接，程式設計師應將哪些值分配給`en_gpio`？

iii. GPIO與SweRV EH1核心的連接

如圖2所示，裝置控制器通過多工器和橋接器連接到SweRV EH1核心。多工器根據CPU產生的位址，在N個可能的週邊設備中選擇一個（本例中N = 7）。橋接器將裝置控制器使用的Wishbone訊號轉換為SweRV核心使用的AXI4訊號，反之亦然（在 `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/AxiToWb/axi2wb.v` 中檔中實作）。

7:1多工器（圖9）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`。

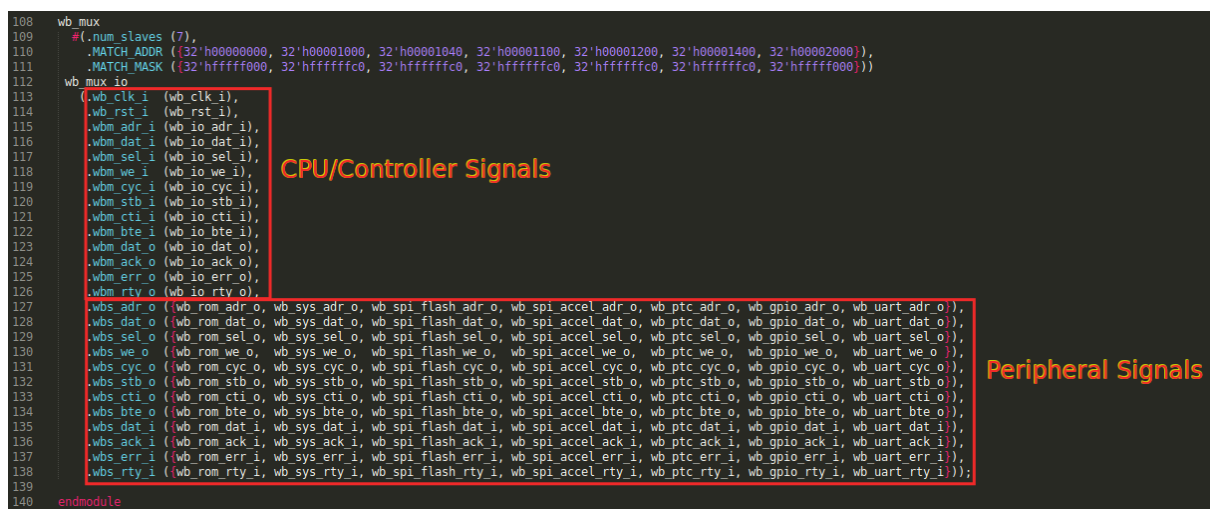


圖9. 7-1多工器選擇與CPU連接的週邊設備（`wb_intercon.v`）。

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9的第115-126行）與一個週邊設備的Wishbone匯流排（圖9的第127-138行）連接。例如，如果CPU產生的位址在0x80001400-0x8000143F範圍內，則選擇GPIO週邊設備，從而將訊號 `wb_io_*` 與訊號 `wb_gpio_*` 連接。

圖10顯示了多工器的Verilog實作（位於

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2-r1/wb_mux.v`）。

任務：詳細分析多工器的實作。您可以重點關注與GPIO相關的訊號（`wb_gpio_*`）。您將需要檢查以下檔案：

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v`
`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v`
`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`
`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2-r1/wb_mux.v`

理解SoC的這一部分對本實驗以及後面的實驗都十分重要。如果您通過新增與多工器相關的新訊號來擴展模擬，則在下一節執行的模擬可以幫助您加深理解。

```

82 //////////////////////////////////////////////////
83 // Master/slave connection
84 //////////////////////////////////////////////////
85
86 //Use parameter instead of localparam to work around a bug in Xilinx ISE
87 parameter slave_sel_bits = num_slaves > 1 ? $clog2(num_slaves) : 1;
88
89 reg wbm_err;
90 wire [slave_sel_bits-1:0] slave_sel;
91 wire [num_slaves-1:0] match;
92
93 genvar idx;
94
95 generate
96   for(idx=0; idx<num_slaves ; idx=idx+1) begin : addr_match
97     assign match[idx] = (wbm_adr_i & MATCH_MASK[idx*aw+:aw]) == MATCH_ADDR[idx*aw+:aw];
98   end
99 endgenerate
100
101 //
102 // Find First 1 - Start from MSB and count downwards, returns 0 when no bit set
103 //
104 function [slave_sel_bits-1:0] ffl;
105   input [num_slaves-1:0] in;
106   integer i;
107
108   begin
109     ffl = 0;
110     for (i = num_slaves-1; i >= 0; i=i-1) begin
111       if (in[i])
112         ffl = i;
113     end
114   end
115 endfunction
116
117 assign slave_sel = ffl(match);
118
119 always @(posedge wb_clk_i)
120   wbm_err <= wbm_cyc_i & !(match);
121
122 assign wbs_adr_o = {num_slaves{wbm_adr_i}};
123 assign wbs_dat_o = {num_slaves{wbm_dat_i}};
124 assign wbs_sel_o = {num_slaves{wbm_sel_i}};
125 assign wbs_we_o = {num_slaves{wbm_we_i}};
126
127 /* verilator lint_off WIDTH */
128 assign wbs_cyc_o = match & (wbm_cyc_i << slave_sel);
129 /* verilator lint_on WIDTH */
130 assign wbs_stb_o = {num_slaves{wbm_stb_i}};
131
132 assign wbs_cti_o = {num_slaves{wbm_cti_i}};
133 assign wbs_bte_o = {num_slaves{wbm_bte_i}};
134
135 assign wbm_dat_o = wbs_dat_o[slave_sel*dw+:dw];
136 assign wbm_ack_o = wbs_ack_o[slave_sel];
137 assign wbm_err_o = wbs_err_o[slave_sel] | wbm_err;
138 assign wbm_rty_o = wbs_rty_o[slave_sel];
139
140 endmodule

```

圖10. Wishbone多工器 (wb_mux.v檔)

B. Verilator模擬

在本節中，我們首先通過新增輸入訊號來修改RVfpgaSim模擬器。然後，我們使用Verilator重新編譯RVfpgaSim，並在模擬器執行簡單程式時分析該新訊號。

i. 修改並重新編譯RVfpgaSim

模擬程序中沒有真正的LED或開關。因此，在測試平台 ([RVfpgaPath]/RVfpga/src/rvfpgasim.v) 中，我們將通過為該訊號 (i_sw) 分配常數值0xFE34 (圖11的左側部分) 來模擬開關驅動。開關隨後將作為輸入提供給SweRVofX SoC (圖11的右側部分)。


```
80    wire [15:0] i_sw;
81    assign i_sw = 16'hFE34; 248    .io_data      ({i_sw,16'bz}));
```

圖11. 訊號*i_sw*在*rvfpgasim.v*中分配並傳遞到SweRVolfX SoC。

根據《入門指南》，測試平台（*rvfpgasim.v*）接收RVfpgaSim的輸入訊號（*clk*和*rst*等）（圖12的左側部分）並實例化*swervolf_core*模組（圖12的右側部分）。

```
28    (input wire clk,
29    input wire rst,
30    input wire i_jtag_tck,
31    input wire i_jtag_tms,
32    input wire i_jtag_tdi,
33    input wire i_jtag_trst_n,
34    output wire o_jtag_tdo,
35    output wire o_uart_tx,
36    output wire o_gpio)

190    swervolf_core
191    #(.bootrom_file (bootrom_file))
192    swervolf
193    (.clk (clk),
194    .rstn (!rst),
195    .dmi_reg_rdata (dmi_reg_rdata),
```

圖12. RVfpgaSim和SweRVolfX實例化的輸入訊號（*rvfpgasim.v*檔）。

在某些情況下，您可能需要向模擬器新增輸入/輸出訊號。在下面的範例中，我們將說明如何包含RVfpgaSim的輸入訊號（名稱為*i_sw0*），此訊號提供了最右側開關的值。

請按下面的步驟操作：

1. 修改檔案[RVfpgaPath]/RVfpga/src/rvfpgasim.v：

a. 包含新的1位元輸入訊號（名稱為*i_sw0*）。請參閱圖13。

```
28    (input wire clk,
29    input wire rst,
30    input wire i_jtag_tck,
31    input wire i_jtag_tms,
32    input wire i_jtag_tdi,
33    input wire i_jtag_trst_n,
34    output wire o_jtag_tdo,
35    output wire o_uart_tx,
36    output wire o_gpio,
37    input wire i_sw0)
```

圖13. 新的*i_sw0*輸入訊號。

b. 提供該訊號作為最右側的開關訊號。如前文所述，為其餘開關值分配0xFE34（位元0除外）。請參閱圖14。

```
80
81    wire [15:0] i_sw;
82    // assign i_sw = 16'hFE34;
83    assign i_sw = {15'b111111100011010,i_sw0};
84
```

圖14. 提供*i_sw0*作為最右側的開關訊號。

- 修改檔案[RVfpgaPath]/RVfpga/verilatorSIM/tb.cpp：這是Verilator的C++主檔案。該檔案的末尾有一個while迴圈（如圖15所示），其中每次迭代都構成一個時鐘脈衝。請注意，SoC的時鐘訊號是在該迴圈（第175行）內產生的，具體方法是在每次迭代中反轉其二進位值（1→0或0→1）。此外，模擬時間在變數main_time（第180行）中計算，測量單位為ns（時鐘週期為20 ns，因此時鐘脈衝為10 ns）。最後請注意，當模擬時間達到值timeout（第171-174行）時，模擬結束。

```

136 top->clk = 1;
137 top->rst = 1;
138 while (!(done || Verilated::gotFinish())) {
139     if (main_time == 100) {
140         printf("Releasing reset\n");
141         top->rst = 0;
142     }
143     if (main_time == 200)
144         top->i_jtag_trst_n = true;
145
146     top->eval();
147     if (tftp)
148         tftp->dump(main_time);
149     if (baud_rate) do_uart(&uart_context, top->o_uart_tx);
150     if (jtag && (main_time > 300)) {
151         int ret = jtag->doJTAG(main_time/20, //doJtag requires t to only increment by one
152             &top->i_jtag_tms,
153             &top->i_jtag_tdi,
154             &top->i_jtag_tck,
155             top->o_jtag_tdo);
156         if (ret != VerilatorJtagServer::SUCCESS) {
157             if (ret == VerilatorJtagServer::CLIENT_DISCONNECTED) {
158                 printf("Ending simulation. Reason: jtag_vpi client disconnected.\n");
159                 done = true;
160             }
161             else {
162                 printf("Ending simulation. Reason: jtag_vpi error encountered.\n");
163                 done = true;
164             }
165         }
166     }
167     if (gpio0 != top->o_gpio) {
168         printf("%lu: gpio0 is %s\n", main_time, top->o_gpio ? "on" : "off");
169         gpio0 = top->o_gpio;
170     }
171     if (timeout && (main_time >= timeout)) {
172         printf("Timeout: Exiting at time %lu\n", main_time);
173         done = true;
174     }
175     top->clk = !top->clk;
176     main_time+=10;
177 }

```

圖15. 用於模擬的While迴圈

在進入迴圈之前為新的i_sw0訊號分配二進位值0（圖16的左側部分），然後在進入迴圈30 μs時將其變更為1（請參閱圖16的右側部分）。

<pre> 136 top->clk = 1; 137 top->rst = 1; 138 139 top->i_sw0 = 0; 140 141 while (!(done Verilated::gotFinish())) { </pre>	<pre> 178 top->clk = !top->clk; 179 main_time+=10; 180 181 if (main_time == 30000) { 182 top->i_sw0 = 1; 183 } 184 185 } </pre>
---	--

圖16. 為訊號i_sw0分配值。

- 完成所有這些變更後，通過執行以下命令重新編譯RVfpgaSim（GSG中對此進行了說明）：

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
```

```
make clean
make
```

新檔案 *Vrvfpgasim* (RRVfpgaSim 模擬二進位檔) 應在目錄 *[RVfpgaPath]/RVfpga/verilatorSIM* 內產生。

WINDOWS : 必須在Cygwin終端機內執行最後一步 (步驟4) (有關詳細說明, 請參閱《入門指南》中的第6節和附錄C)。請注意, C: Windows 資料夾位於Cygwin中的以下位置: */cygdrive/c*。

MacOS : 有關詳細說明, 請參閱《入門指南》的附錄D。

ii. 分析程式 *LedsSwitches.S* 的模擬

在本節中, 我們將模擬《RVfpga入門指南》中的 *LedsSwitches.S* 程式範例 (圖17)。該程式讀取開關上的值, 並將讀取到的值寫入Nexys A7開發板上的LED。請注意, 我們需要配置啟用暫存器, 以便根據其連接將32個輸入/輸出引腳配置為輸入或輸出。具體來說, GPIO的低16個引腳連接到LED, 這樣它們便是相對於CPU的輸出引腳 (啟用 = 1)。GPIO的高16個引腳連接到開關, 它們是相對於CPU的輸入引腳 (啟用 = 0)。由於開關占用讀取暫存器的高16位元, 因此在將它們的值寫入LED之前, 必須先將其右移。

```
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

.globl main
main:

li x28, 0xFFFF
li x29, GPIO_INOUT
sw x28, 0(x29)                # Write the Enable Register

next:
    li a1, GPIO_SWs          # Read the Switches
    lw t0, 0(a1)

    li a0, GPIO_LEDs
    srl t0, t0, 16
    sw t0, 0(a0)              # Write the LEDs

    beq zero, zero, next
.end
```

圖17. 用於在SweRVolfX SoC中執行的 *LedsSwitches.s*

請按照以下步驟執行模擬。

1. 在電腦上開啟VSCode/PlatformIO。
2. 在頂端列上, 按一下「*File*」(檔案) → 「*Open Folder...*」(開啟資料夾...) (圖18), 然後導覽至目錄 *[RVfpgaPath]/RVfpga/examples/*

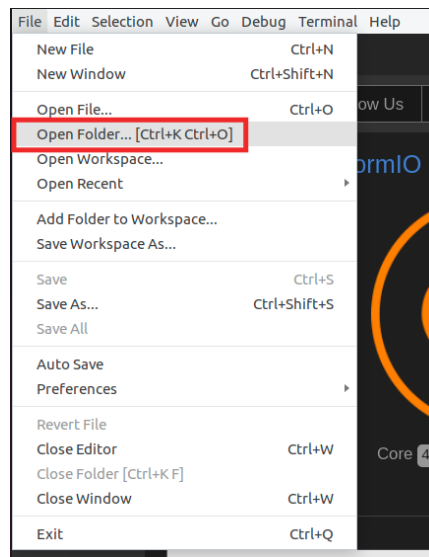


圖18. 開啟LedsSwitches.S範例

- 選擇目錄**LedsSwitches**（不要開啟，只需選擇它），然後按一下「OK」（確定）。該範例將在PlatformIO中開啟。
- 開啟**platformio.ini**檔，檢查上面產生的RVfpgaSim模擬二進位檔圖19的路徑（上一節的步驟3）是否正確。根據本GSG，其形式應如下所示：

```
board_debug.verilator.binary                                     =
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```

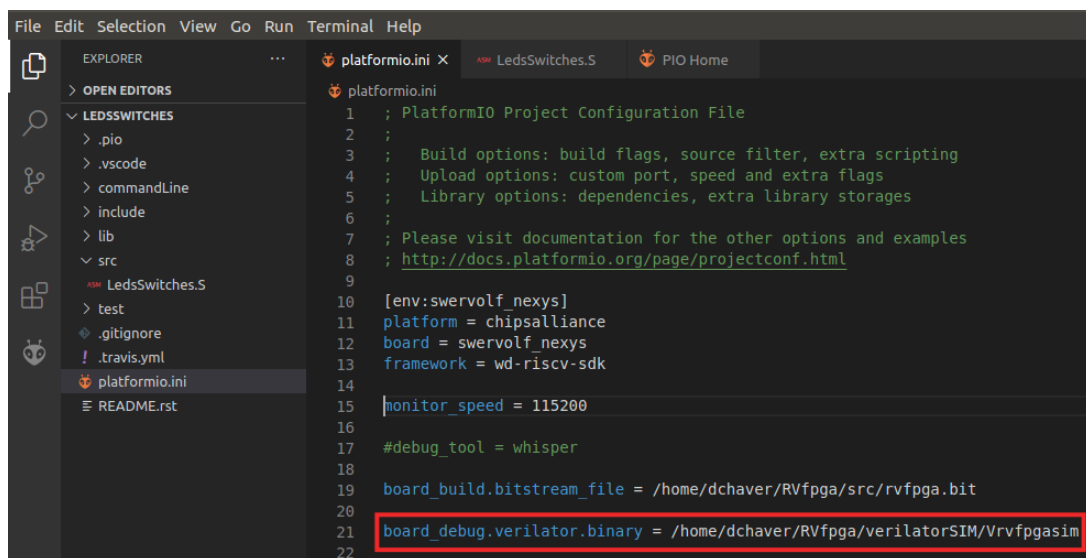


圖19. Platformio初始化檔：platformio.ini

Windows：RVfpgaSim模擬可執行檔案的名稱為Vrvfpgasim.exe。因此：

```
board_debug.verilator.binary = [RVfpgaPath]\RVfpga\verilatorSIM\Vrvfpgasim.exe
```

5. 按一下左側功能表功能區中的PlatformIO圖示執行模擬，然後展開「Project Tasks」（專案任務）→ `env:swervolf_nexys` → 「Platform」（平台），並按一下「Generate Trace」（產生軌跡）。

`trace.vcd`檔應當已在`[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys`內部產生，隨後通過在PlatformIO終端機中輸入以下命令，可使用GTKWave將其開啟。

```
gtkwave [RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/trace.vcd
```

WINDOWS：已下載的資料夾`gtkwave64`包括一個稱作`gtkwave.exe`的應用程式，該應用程式位於`bin`資料夾內。按兩下該應用程式啟動GTKWave。在應用程式的頂端，按一下「File」（檔案）–「Open New Tab」（開啟新索引標籤），然後開啟在資料夾`[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys`中產生的`trace.vcd`檔。

6. 在軌跡中包含以下訊號（移至模組`rvfpgasim-swervolf`，找到下面各個訊號）：
 - 新增時鐘訊號：`clk`
 - 新增GPIO輸入訊號：`i_gpio`
 - 新增GPIO輸出訊號：`o_gpio`

在圖（圖20）中，您將看到16個開關的值（訊號`i_gpio`的高16位元）已複製到16個LED（訊號`o_gpio`的低16位元）中，但存在一些延遲。此外，最右側的開關將在30 μ s時發生變更（0→1），這會使最右側的LED在一段時間後也發生變更。

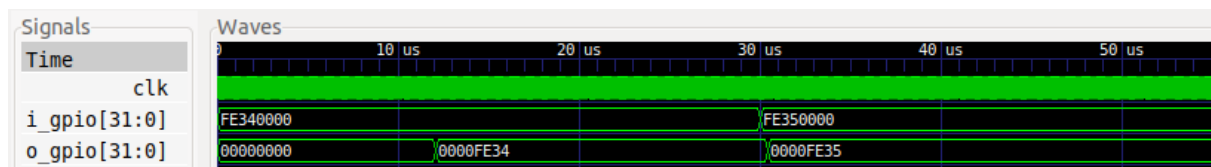


圖20. LedsSwitches程式的模擬

7. 進階練習

練習2. 更詳細地分析上一節中的模擬。圖21 顯示了`.elf LedsSwitches`程式（圖17）的反組合語言程式碼版本，其中突出顯示了用於存取GPIO暫存器的三行指令（啟用、讀取和寫入）。根據本《入門指南》，您可以通過在PlatformIO中開啟`firmware.dis`檔輕鬆檢視`.elf`程式的反組合語言程式碼版本，該檔案於編譯時在以下資料夾內產生：

`[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf-nexys/`
（請參閱圖21）。

```

> PSP
> src
  ≡ .sconsign36.dblite
  ≡ firmware.bin
  ≡ firmware.dis
  ≡ firmware.elf
  ≡ LedsSwitches.map
  ≡ libBoardBSP.a
  ≡ libPSP.a
  ≡ project.checksum
> libdeps
> .vscode
> commandLine
> include
> lib
  ≡ src

```

```

65 Disassembly of section .text:
66
67 00000090 <main>:
68   90: 00010e37      lui t3,0x10
69   94: fffe0e13      addi t3,t3,-1 # ffff <_sp+0xcebf>
70   98: 80001eb7      lui t4,0x80001
71   9c: 408e8e93      addi t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
72  a0: 01cea023      sw  t3,0(t4)
73
74 000000a4 <next>:
75  a4: 800015b7      lui a1,0x80001
76  a8: 40058593      addi a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
77  ac: 0005a283      lw  t0,0(a1)
78  b0: 80001537      lui a0,0x80001
79  b4: 40450513      addi a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
80  b8: 0102d293      srli t0,t0,0x10
81  bc: 00552023      sw  t0,0(a0)
82  c0: fe0002e3      beqz zero,a4 <next>

```

圖21. LedsSwitches.S程式的反組合語言程式碼版本

在RVfpgaSim中模擬該程式，並在執行圖21中以紅色強調顯示的三條記憶體指令（sw、lw和sw）期間分析GPIO訊號。這將幫助您瞭解A節說明的GPIO低階實作。

您可以從B節的模擬開始，新增並分析以下訊號的值（移至參考模組找到每個訊號）：

- rvfpgasim → swervolf → swerv_eh1 → swerv → ifu
 - 時鐘：**clk**。
 - 已擷取指令：**ifu_i0_instr**和**ifu_i1_instr**。
- rvfpgasim – swervolf
 - 32位元輸入/輸出針腳：**i_gpio**和**o_gpio**。
 - CPU提供的位址：**wb_m2s_io_adr**。
- rvfpgasim – swervolf – gpio_module
 - GPIO外部介面：**ext_pad_i**、**ext_pad_o**和**ext_padoe_o**。
- rvfpgasim – swervolf – wb_intercon0
 - 圖2的多工器的輸出位址和資料訊號：**wb_io_adr_i**、**wb_io_dat_i**和**wb_io_dat_o**。
 - 圖2的多工器的輸入GPIO資料訊號：**wb_gpio_adr_i**、**wb_gpio_dat_i**和**wb_gpio_dat_o**。
 - 圖2的多工器的選擇訊號：**wb_*_cyc_o**。
- rvfpgasim – swervolf – wb_intercon0 – wb_mux_io
 - 圖2的多工器的匹配訊號：**match**。
- rvfpgasim – swervolf – swerv_eh1 – swerv – dec – arf – gpr_banks(0) – gpr(5) – gprff
 - t0的暫存器值：**dout**。

練習3. 擴展RVfpgaNexys以支援五個板上按鈕。按鈕如圖22所示。這五個按鈕根據其位置命名：上、下、左、右和中 – BTNU、BTND、BTNL、BTNR、BTNC。

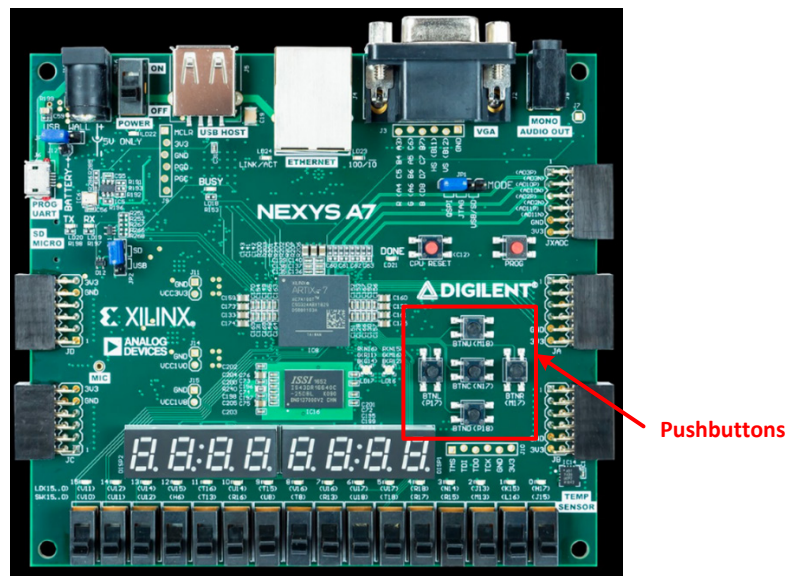


圖22. Nexys A7 FPGA開發板上的按鈕

- 假設我們正在使用的GPIO模組（`gpio_top`）的最大大小為32，即I/O引腳數為32（16個LED + 16個開關），則需要在SweRVolfX中包含GPIO模組的另一個實例，以及5個新的三態緩衝區和所有必要的訊號。
- 使用從0x80001800開始的位址（可用）來映射新GPIO控制器對應的暫存器。請注意，必須修改多工器（圖9）以包含新週邊設備。
- 在修改限制檔時，還必須考慮到5個按鈕連接到以下FPGA引腳：
 - BTNC連接到引腳N17
 - BTNU連接到引腳M18
 - BTNL連接到引腳P17
 - BTNR連接到引腳M17
 - BTND連接到引腳P18

練習4. 在RVfpgaNexys中為5個板上按鈕設計另一個控制器。

- 與練習3相比，在此範例中，必須根據圖3中說明的方案在Verilog或SystemVerilog中實作自己的GPIO控制器。實際上，甚至可以簡化相應電路，只包含一個讀取暫存器（即無需包含三態緩衝區或寫入暫存器）。
- 無需從上一個練習中刪除控制器，因為按鈕可以映射到該GPIO控制器未使用的位址。
- 將新控制器包含在系統控制器週邊設備中。可以使用未使用的位址範圍0x8000101C-0x8000101F。請注意，系統控制器中包含的暫存器基於CPU產生的位址（`i_wb_adr`）直接連接到Wishbone匯流排（`o_wb_rdt`）的資料訊號，以這種方式讀入到CPU中。檢查模組`swervolf_syscon`（`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v`）的第234-266行，以幫助您瞭解如何繼續操作。

練習5. 編寫一個RISC-V組合語言程式和一個C程式，以便從1開始在LED上顯示遞增的二進位計數。程式包含一個用於在各個遞增值的顯示之間留出等待時間的空迴圈，以便人眼可以看到這些值。通過練習3中實作的OpenCores週邊設備讀取BTNC，並使用它來變更計數速度，然後通過練習4中實作的專用週邊設備讀取BTNU，並在每次按下時使用它重新啟動計數。