



IMAGINATION大學計劃

SweRV EH1參考

階層、模組、訊號和類型

本文件提供有關以下主題的額外說明：

- 第1部分：**Sigasi Studio**
- 第2部分：**SweRV EH1處理器的組態**
- 第3部分：**模組的RVfpga系統階層及其最相關的訊號**
- 第4部分：**分組控制位元的主要結構/類型**
- 第5部分：**RISC-V壓縮指令**
- 第6部分：**實際基準**

1. SIGASI STUDIO

Sigasi Studio可幫助設計人員以最直覺的方式編寫、檢查和修改數位電路設計，進而提高工作效率。該工具能夠瞭解設計環境，並藉助智慧自動完成和程式碼重構等進階功能實現更為簡單高效的VHDL、Verilog和SystemVerilog設計。

使用者需要付費購買授權，才能使用專業版本的Sigasi Studio。幸運的是，您可以造訪以下連結輕鬆取得供教育用的免費授權：<https://www.sigasi.com/try-form-edu/>。填寫個人資料並獲得授權後，您將收到一封包含Sigasi Studio下載連結以及安裝和使用說明的電子郵件（<https://www.sigasi.com/download/>，參見圖1）。該軟體適用於Windows、Linux和MacOS作業系統。



圖1. Sigasi Studio的下載連結以及安裝和使用說明

一旦在系統中安裝了Sigasi Studio，就可以使用該軟體來檢查RVfpga。以下連結為Hendrik Eeckhaut兩年前發表的文章，其中介紹了如何建立和配置SweRV EH1專案：https://insights.sigasi.com/tech/swerv_riscv/。接下來，我們將以這些資訊為切入點，提供建立和配置RVfpga專案的完整說明。

1. 建立一個目錄[RVfpgaPath]/RVfpga/src的副本，將其命名為[RVfpgaPath]/RVfpga/src_SigasiStudio
2. 開啟Sigasi Studio，進入下載目錄中，按兩下檔案sigasi_internal（參見圖2）。

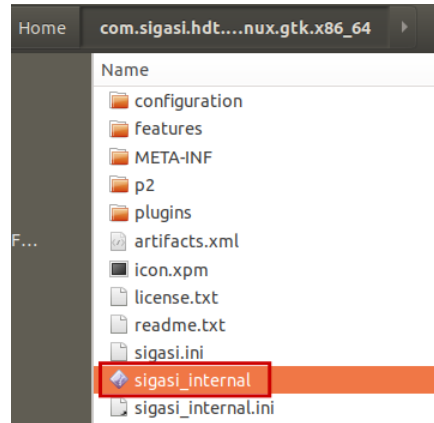


圖2. 開啟Sigasi Studio

3. 在Sigasi Studio視窗中，按一下「File」（檔案）→「Import」（匯入），將開啟一個新視窗，要求您選擇要新增到系統中的專案類型。選擇「Import a (System) Verilog project」（匯入（System）Verilog專案），然後按一下「Next」（下一步）（圖3）。

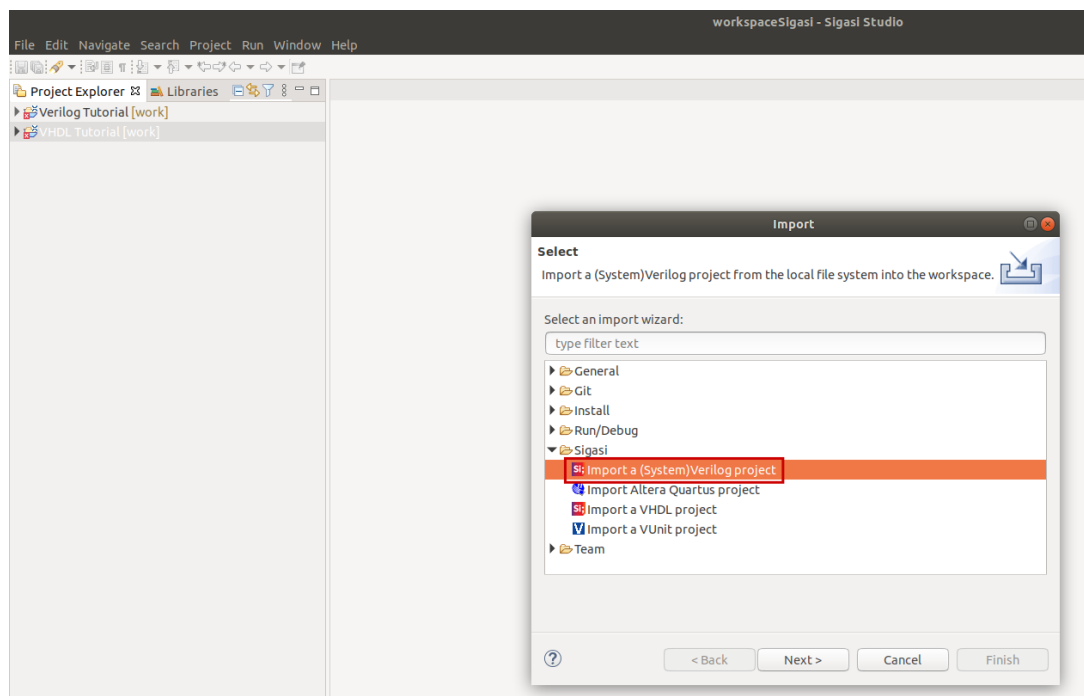


圖3. 匯入RVfpga專案

4. 現在按一下「Browse...」（瀏覽...），導覽到src_SigasiStudio目錄並選擇該目錄，按一下「Open」（開啟）（參見圖4），然後按一下「Finish」（完成）。

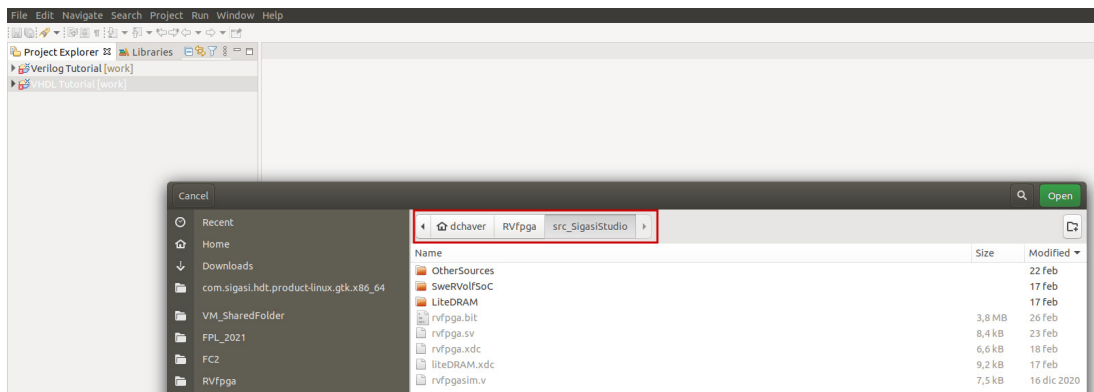


圖4. 開啟RVfpga來源目錄

5. 開啟專案時會出現大量錯誤（參見圖5），其中大多數錯誤是由於專案組態中缺少多個包含檔案而引起的。

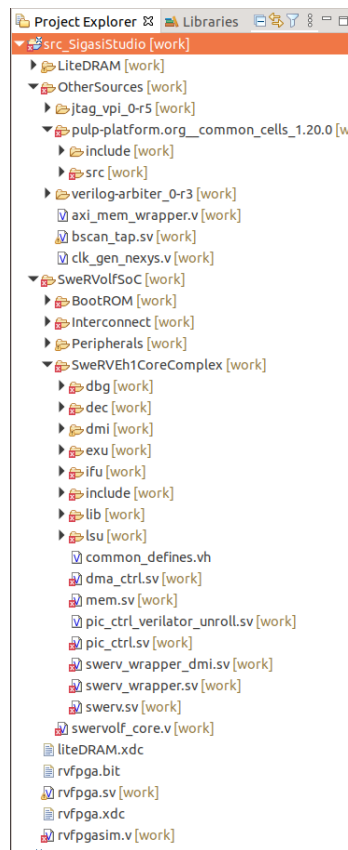


圖5. RVfpga Sigasi Studio專案的初始錯誤

6. 在「Project Explorer」（專案資源管理器）中，以右鍵按一下 *src_SigasiStudio* 專案，開啟「Properties」（屬性）視窗（參見圖6）。

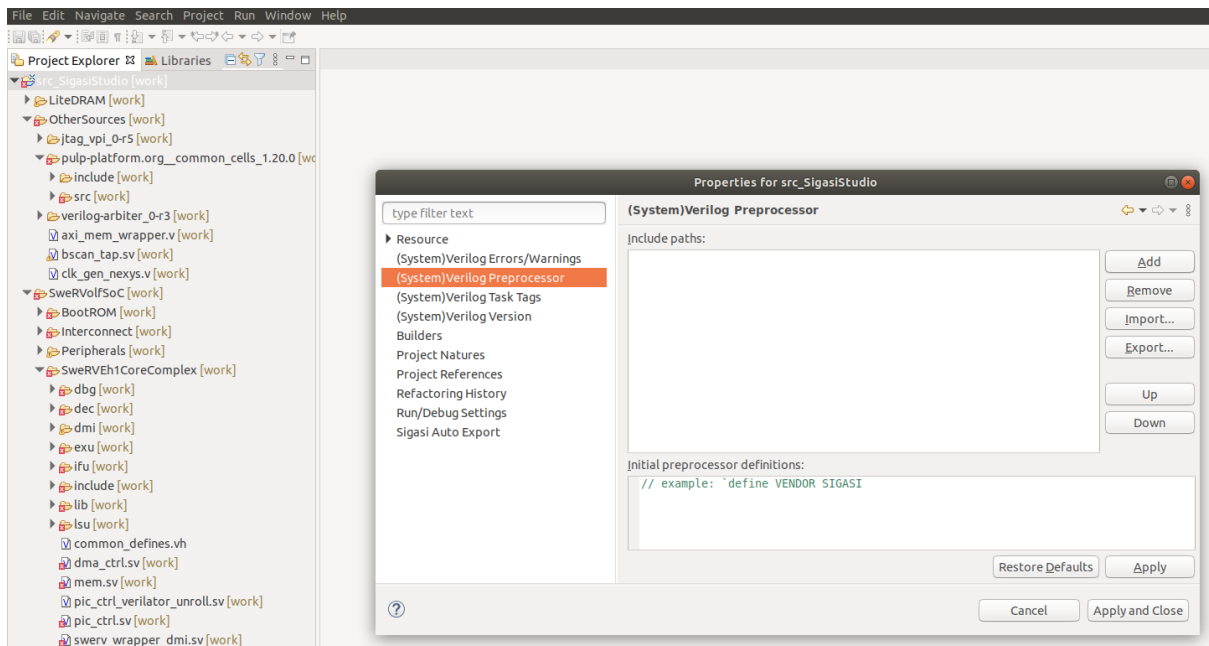


圖6. 專案屬性。

7. 在「Properties」（屬性）視窗（圖6）中選擇「(System)Verilog Preprocessor」（（System）Verilog預先處理器），並新增以下包含路徑（方法為按一下右側的「Add」（新增）按鈕）：

- *[RVfpgaPath]/RVfpga/src_SigasiStudio/SweRVolfSoC/SweRVEh1CoreComplex/include*
- *[RVfpgaPath]/RVfpga/src_SigasiStudio/OtherSources/pulp-platform.org__common_cells_1.20.0/include*
- *[RVfpgaPath]/RVfpga/src_SigasiStudio/SweRVolfSoC/Interconnect/AxiInterconnect/pulp-platform.org__axi_0.25.0/include*
- *[RVfpgaPath]/RVfpga/src_SigasiStudio/SweRVolfSoC/Interconnect/AxiInterconnect*
- *[RVfpgaPath]/RVfpga/src_SigasiStudio/SweRVolfSoC/Interconnect/WishboneInterconnect*

新增上述五個目錄後，按一下「Apply」（套用）按鈕。

然後，在同一視窗的底部方塊（初始預先處理器定義）中，輸入以下行：``include "common_defines.vh"`。按一下「Apply」（套用）和「Close」（關閉）按鈕。

圖7所示為最終的狀態。

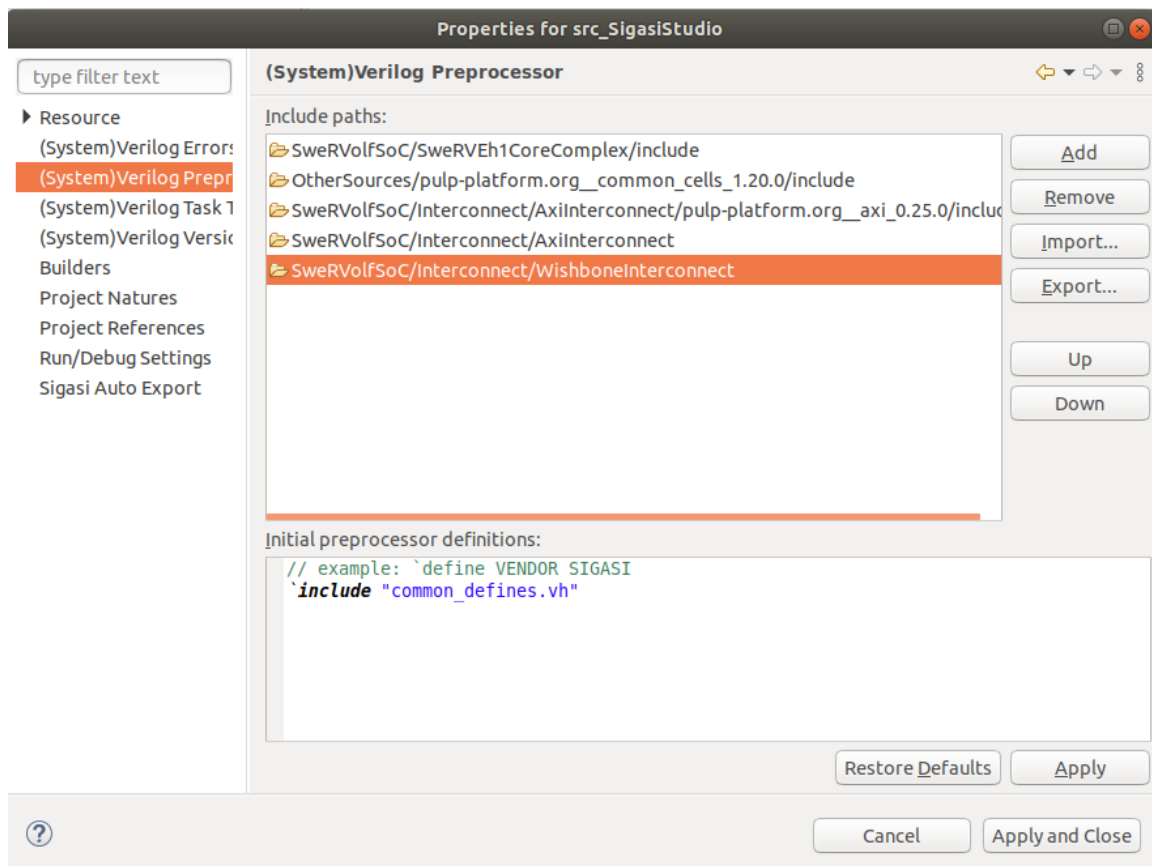


圖7. 包含目錄和檔案

8. 最後，刪除檔案

[RVfpgaPath]/RVfpga/src_SigasiStudio/SweRVolfSoC/BootROM/sw/boot_main.vh，我們的專案無需使用該檔案，並且保留該檔案會導致一些錯誤。可以在檔案資源管理器中或Sigasi Studio內部刪除該檔案。

完成上述步驟後，錯誤應會全部消失，只剩下一些警告，您可以忽略這些警告。

此時便可開始使用Sigasi Studio檢查RVfpga SoC。我們接下來將對該工具的一些功能進行測試：

1. 在頂部功能表中，開啟「Window」（視窗）→「Show View」（顯示檢視）→「Block Diagram」（區塊圖），工具右側將開啟一個新視窗，以便您以圖形方式在模組中導覽。
2. 在本實驗中，我們將分析算術指令和邏輯指令。這些指令在ALU中執行，ALU則在模組exu_alu_ctl內部實作。在「Project Explorer」（專案資源管理器）視窗中按兩下該模組將其開啟。將顯示圖8所示的介面。

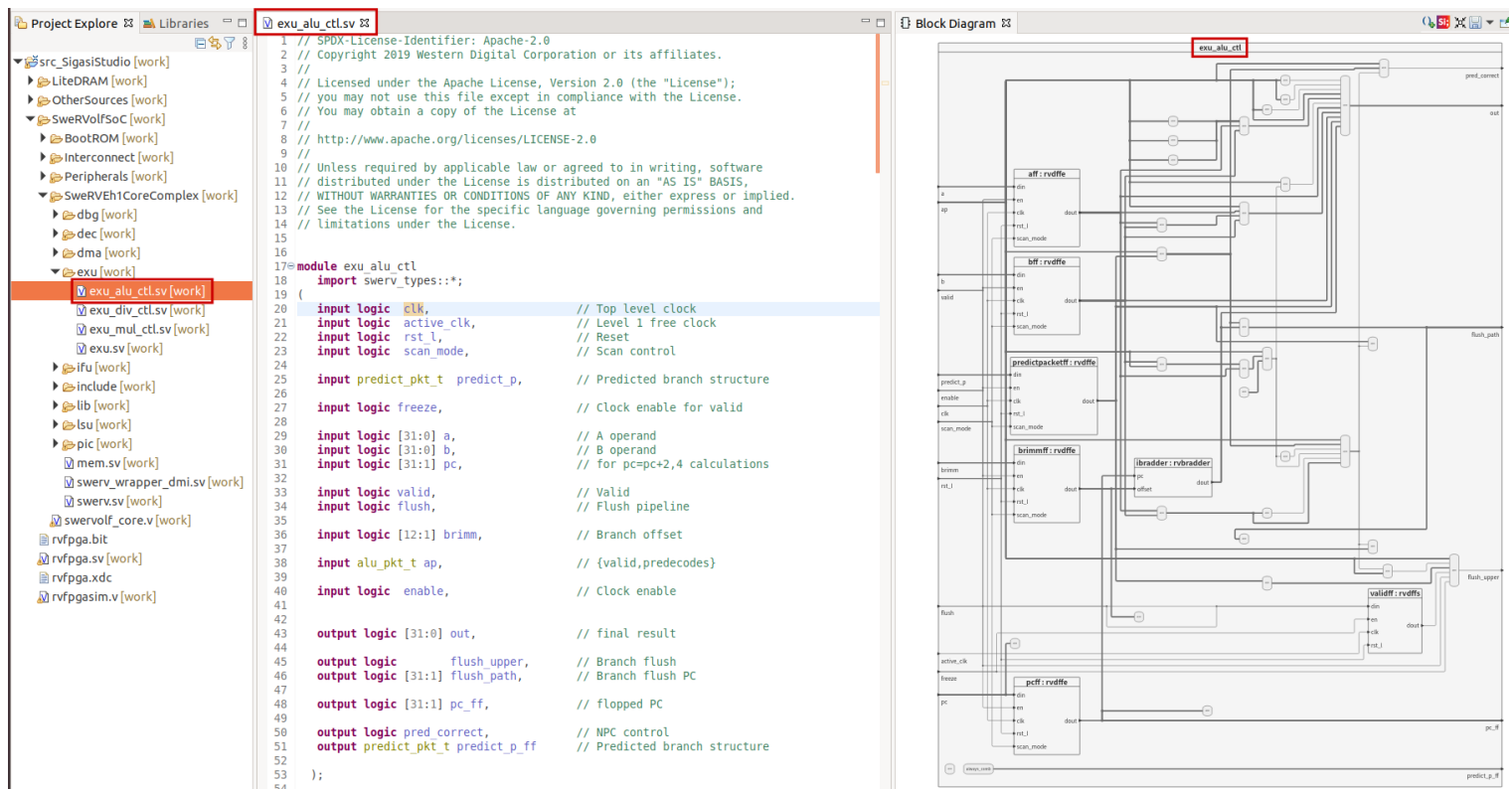


圖8. 檔案exu_alu_ctl.v : Verilog程式碼和區塊圖

- 透過在Verilog程式碼中以右鍵按一下某個訊號並選擇「Show In」（顯示方式）→「Block Diagram」（區塊圖），即可在區塊圖中強調顯示該訊號。與強調顯示的訊號相關的線路也將在區塊圖視窗中強調顯示，如圖9所示，其中強調顯示的封包為ap。

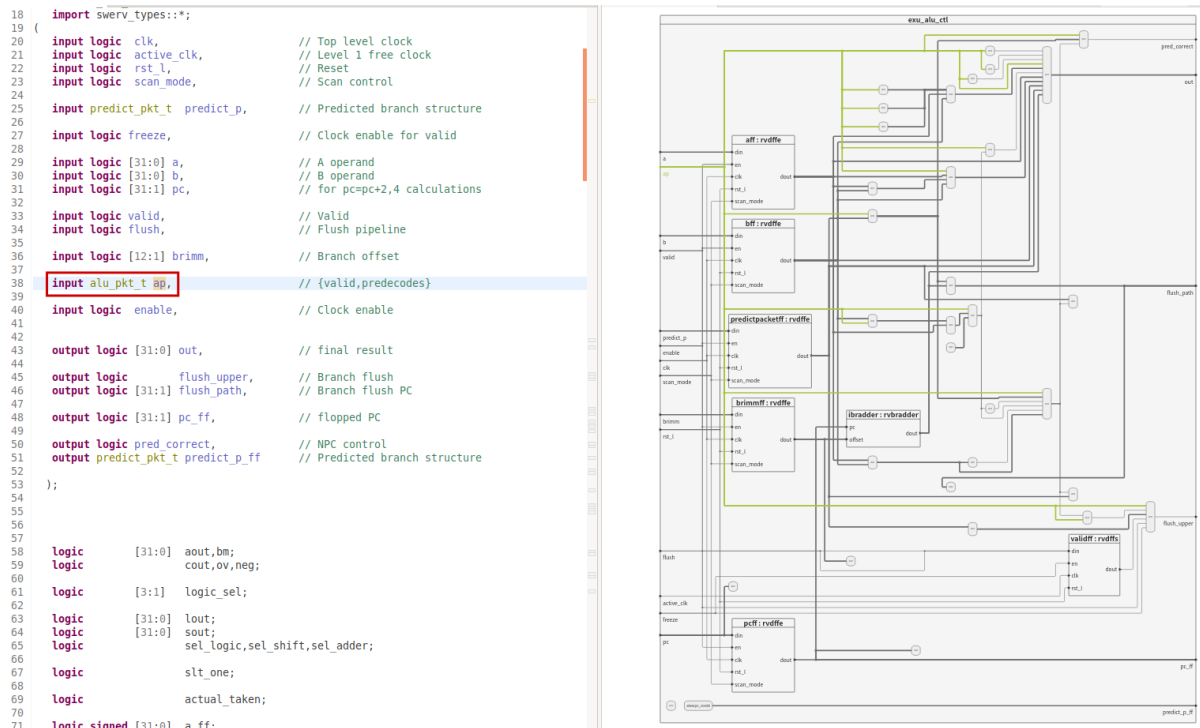


圖9. 強調顯示訊號ap

4. 還可以透過按兩下區塊圖中的模組，在Verilog程式碼中實作組合模組。例如，圖10中顯示了產生訊號out的模組。

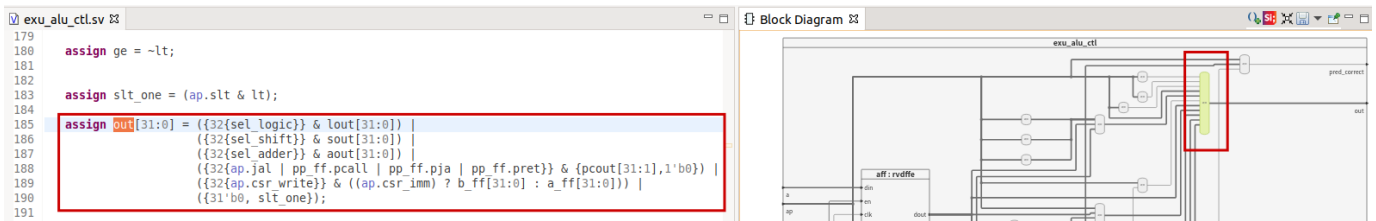


圖10. 強調顯示產生訊號out的組合模組的Verilog程式碼

5. 最後，我們透過以右鍵按一下Verilog程式碼中的模組實例並選擇「Open Declaration」(開啟宣告)，在區塊圖上開啟一個模組宣告。圖11所示為在檔案beh_lib.sv中實作的模組rvdfffe。

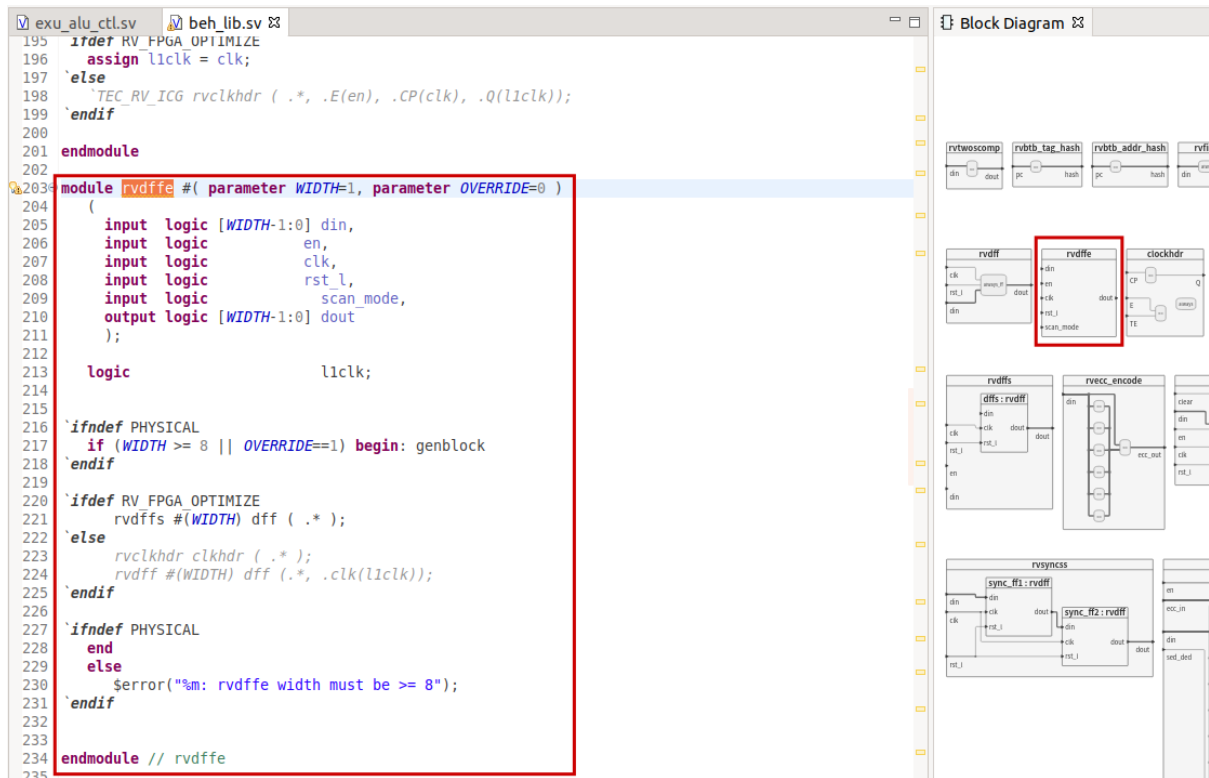


圖11. 模組rvdfffe

2. SWERV EH1處理器的組態

A. 組態核心結構

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common_defines.vh

允許使用者配置核心的眾多結構，如指令快取、ICCM/DCCM、分支預測器等。RVfpga系統提供預設組態，您可以透過兩種不同的方式變更組態：

- 可以在檔案 `common_defines.vh` 中手動編輯參數。
- 可以使用Western Digital隨SweRV EH1套件一起提供的 `swerv.config` 指令碼。有關該指令碼的使用方法，請參見 <https://github.com/chipsalliance/Cores-SweRV/tree/branch1.8> 在RVfpga中，可造訪以下位置取得 `swerv.config` 指令碼：
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/

如實驗室1中所述，產生新的組態檔案後，可以在Vivado中重新合成SoC，並取得新的RVfpga系統位元流。

B. 停用壓縮指令的使用

在某些情況下，我們可能希望停用壓縮指令的使用。為此，必須對我們的PlatformIO進行兩項變更：

- 在檔案 `platformio.ini` 中包含以下新行：

```
build_unflags = -Wa,-march=rv32imac -march=rv32imac
build_flags = -Wa,-march=rv32ima -march=rv32ima
extra_scripts = extra_script.py
```
- 將檔案 `extra_script.py` 新增到專案的原始程式碼中。該檔案包含以下程式碼行：

```
Import ("env")
env.Append(
    LINKFLAGS=[
        "-Wa,-march=rv32ima",
        "-march=rv32ima"
    ]
)
```

在實驗11-20使用的大多數範例中，為了簡單起見，我們將停用使用壓縮指令。

C. 啟用/停用核心功能

《SweRV EH1程式設計師參考手冊》（[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)）中的表10-1列出了 `mfdc` 暫存器（CSR 0x7F9）位元。該暫存器包含底層核心控制位元，可停用特定功能，如管線或雙指令執行、分支預測器等。表1所示為該暫存器可控制的九個核心功能。要啟用或停用各項核心功能，只需將暫存器的相應位清零或置1。例如，您可以在組合語言程式中加入以下兩條組合語言指令，以停用雙指令執行、輔助ALU和管線執行：

```
li t2, 0x481
csrrs t1, 0x7F9, t2
```

表1. 功能停用控制暫存器 (*mfdc* : CSR 0x7F9)

| | | | | | |
|-------|--|---|---|---|--|
| 31-11 | 保留 | 7 | 0 : 啟用輔助ALU 1 : 停用輔助ALU | 3 | 0 : 啟用分支預測和傳回位址堆疊 1 : 停用分支預測和傳回位址堆疊 |
| 10 | 0 : 雙指令執行 1 : 單指令執行 | 6 | 0 : 副作用儲存採用管線形式 1 : 副作用儲存阻止所有後續的匯流排交易， 直至儲存發出收到預設值的回應 | 2 | 0 : 啟用寫入緩衝區合併 1 : 停用寫入緩衝區合併 |
| 9 | 保留 | 5 | 0 : 啟用非阻塞載入/除法 1 : 停用非阻塞載入/除法 | 1 | 保留 |
| 8 | 0 : 啟用ICCM/DCCM ECC檢查 1 : 停用ICCM/DCCM ECC檢查 | 4 | 0 : 啟用快速除法 1 : 停用快速除法 | 0 | 0 : 管線執行 1 : 單指令執行 |

我們將在實驗11-20中使用不同的組態，以便比較SweRV EH1在啟用/停用不同核心功能時的效能、I\$命中/未命中數和分支預測器命中/未命中數等。

3. SweRV EH1核心的主要模組和訊號

RVfpga系統在Nexys A7開發板的Artix-7 FPGA上執行，如圖12所示。該圖詳細介紹了系統的階層，包括Verilog模組和子模組的名稱。RVfpga系統由SweRVolf核心（**swervolf_core**）、DRAM控制器（**litedram_top**）、時脈產生模組（**clk_gen_nexys**）和一些介面模組組成。SweRVolf核心則由SweRV EH1處理器（**swerv_wrapper_dmi**）和其他介面模組（**wb_intercon**、**axi_intercon**和**uart_top**等）組成。SweRV EH1處理器的頂層模組**swerv_wrapper_dmi**會對核心的兩個主要模組**mem**和**swerv**進行實例化。在本文件的後續部分，我們將列出這兩個模組的子模組和主要訊號。請注意，您可以在模組介面處找到每個模組的其餘訊號。在實驗11-20中，我們將專門研究這些訊號，以分析處理器不同部分的操作。

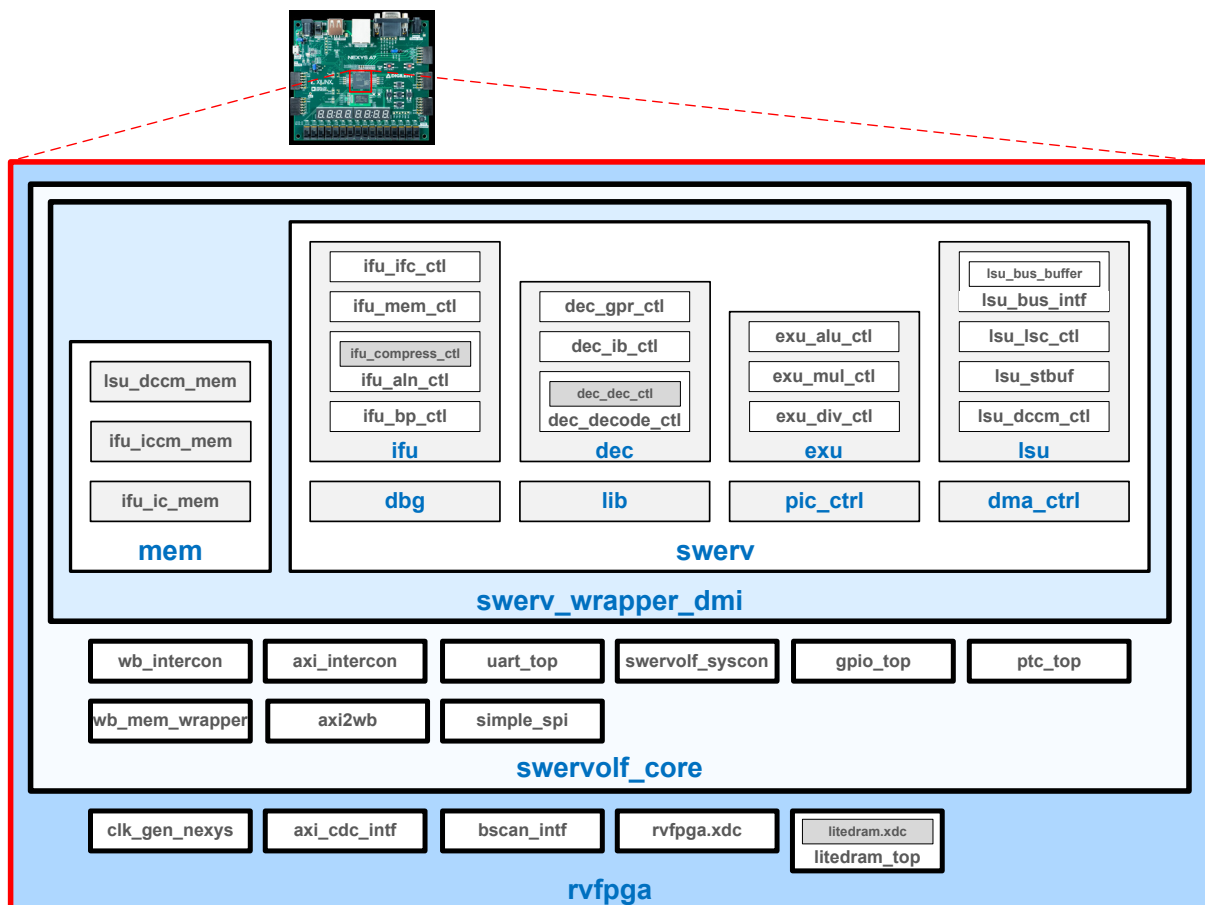


圖12. RVfpga系統的階層

模組：**mem**

功能：此模組會對SweRV中提供的三個內部記憶體進行實例化：ICCM、DCCM和I\$. 表2列出了**mem**的子模組及其介面訊號。

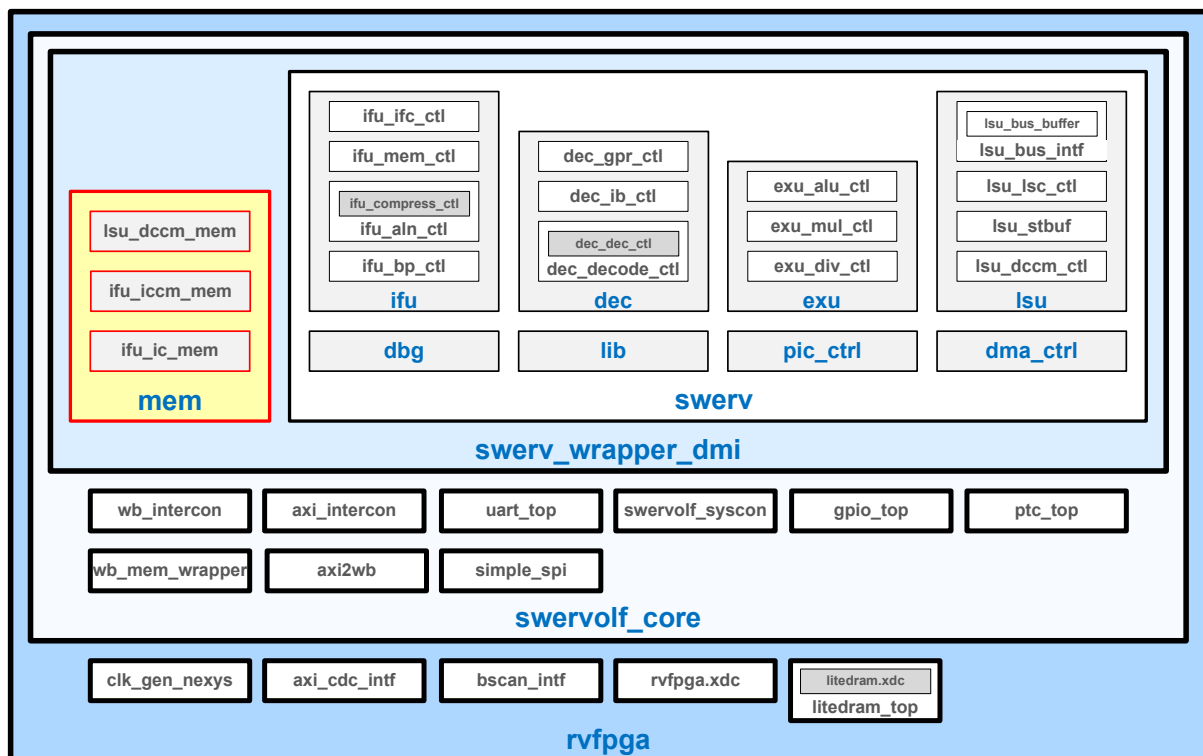


圖13. 模組mem及其子模組

表2. mem子模組和I/O

| 單元 | I/O | 名稱 | 說明 |
|---|-----|--|----------------------------|
| ICCM : ifu_iccm_mem (其中包含ICCM 模組包裝函式) | 輸入 | iccm_wren | 寫入啟用 |
| | | iccm_rden | 讀取啟用 |
| | | [`RV_ICCM_BITS-1:2] iccm_rw_addr | 讀/寫位址 |
| | | [77:0] iccm_wr_data | 寫入資料 |
| | 輸出 | [155:0] iccm_rd_data | 讀取資料 |
| IS : ifu_ic_mem (其中包含指令快 取資料和標籤模組 包裝函式) | 輸入 | [3:0] ic_wr_en | 寫入啟用 |
| | | ic_rd_en | 讀取啟用 |
| | | [31:2] ic_rw_addr | 讀/寫位址 |
| | | [67:0] ic_wr_data | 用於填充指令快取的資 料。具有同位檢查。 |
| | 輸出 | [135:0] ic_rd_data | 從指令快取讀取的資料。 F2階段具有同位檢查。 |
| | | [3:0] ic_rd_hit | 每條通路中的命中/ 未命中 |
| DCCM : lsu_dccm_mem (其中包含DCCM 模組包裝函式) | 輸入 | dccm_wren | 寫入啟用 |
| | | dccm_rden | 讀取啟用 |
| | | [`RV_DCCM_BITS-1:0] dccm_wr_addr | 寫入位址 |
| | | [`RV_DCCM_BITS-1:0] dccm_rd_addr_lo | 讀取位址 |
| | | [`RV_DCCM_BITS-1:0] dccm_rd_addr_hi | 進行未對齊存取時高位 儲存區的讀取位址 |
| | | | |

| | | | |
|--|----|---|-----------|
| | | [`RV_DCCM_FDATA_WIDTH-1:0] dccm wr data | 寫入資料 |
| | 輸出 | [`RV_DCCM_FDATA_WIDTH-1:0] dccm rd data lo | 讀取資料低位儲存區 |
| | | [`RV_DCCM_FDATA_WIDTH-1:0] dccm rd data hi | 讀取資料高位儲存區 |

模組：**swerv**

功能：如圖14所示，**swerv**是SweRV EH1核心的頂層模組。該模組會對核心的主要模組進行實例化，其中最重要的模組包括：**ifu**、**dec**、**exu**和**lsu**。表3至表6列出了這些單元的子模組和介面訊號。 **swerv**模組透過SweRV包裝函式（**swerv_wrapper_dmi**）與**mem**模組通訊。

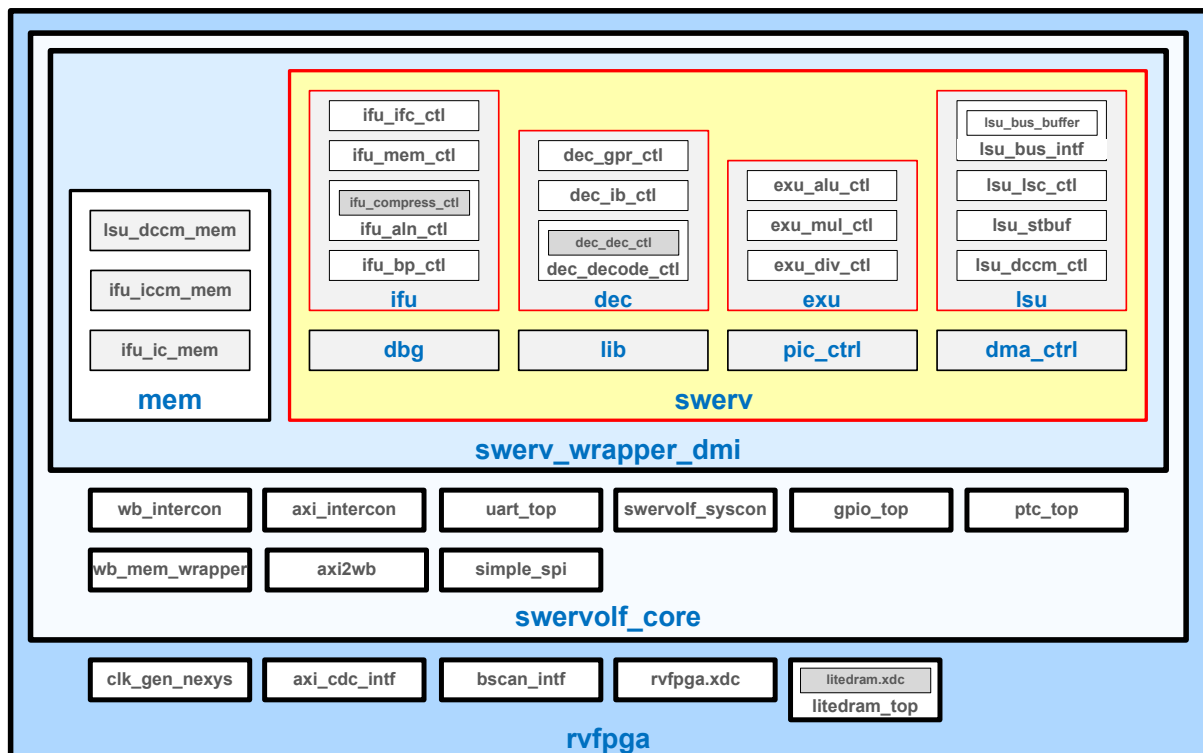


圖14. **swerv**及其子模組

表3. **ifu**（指令擷取單元）的I/O和子模組（包括子模組的I/O）

| 單元 | I/O | 名稱 | 說明 |
|--|-------|--------------------------------|-------------------------------|
| 指令擷取單元： ifu （此模組是用於指令擷取、分支預測器和對齊器的頂層模組） | 輸入/輸出 | 多個訊號 | 用於 mem 模組輸入/輸出的ICCM連接埠 |
| | | 多個訊號 | 用於 mem 模組輸入/輸出的IS\$連接埠 |
| | | 多個訊號 | IFU AXI連接埠 |
| | 輸入 | exu_flush_final | 排清管線 |
| | | [31:1] exu_flush_path_final | 排清擷取位址 |

| | | | |
|---|----|--|---|
| | 輸出 | [31:0] ifu_i0_instr | 指令0。從對齊階段到解碼階段 |
| | | [31:0] ifu_i1_instr | 指令1。從對齊階段到解碼階段 |
| | | [31:1] ifu_i0_pc | 指令0 PC（程式計數器）。從對齊階段到解碼階段 |
| | | [31:1] ifu_i1_pc | 指令1 PC。從對齊階段到解碼階段 |
| 擷取控制： ifu_ifc_ctl （此模組可實作擷取管道控制，產生從指令記憶體中擷取的下一個位址。） | 輸入 | exu_flush_final | 排清管線 |
| | | [31:1] ifu_bp_btb_target_f2 | 預測的目標PC |
| | | [31:1] exu_flush_path_final | 排清路徑 |
| | 輸出 | output logic [31:1] ifc_fetch_addr_f1 | FC1階段的擷取位址 |
| | 內部 | logic [31:1] fetch_addr_next | 順序位址 |
| 指令記憶體 （I\$和ICCM） 控制： ifu_mem_ctl （指令記憶體控制 – lcache和ICCM –） | 輸入 | [31:1] fetch_addr_f1 | FC1階段的擷取位址 （重新命名 ifc_fetch_addr_f1） |
| | 輸出 | [127:0] ic_data_f2 | I\$或ICCM在FC2階段讀取的資料，將傳送至對齊階段 |
| 對齊控制： ifu_aln_ctl （指令對齊器） | 輸入 | [127:0] ifu_fetch_data | 來自擷取階段的128位元擷取資料 |
| | 內部 | logic [127:0] q2,q1,q0 | 3個緩衝區 |
| | 輸出 | [31:0] ifu_i0_instr | 指令通路0 |
| | | [31:0] ifu_i1_instr | 指令通路1 |
| | | [31:1] ifu_i0_pc | 指令通路0 PC |
| | | [31:1] ifu_i1_pc | 指令通路1 PC |
| 分支預測器： ifu_bp_ctl | 輸入 | [31:1] ifc_fetch_addr_f1 | FC1階段的擷取位址 |
| | 輸出 | [31:1] ifu_bp_btb_target_f2 | 預測的目標PC |
| | | ifu_bp_kill_next_f2 | 發生/未發生的分支 |

表4. dec (解碼單元) 的I/O和子模組 (包括子模組的I/O)

| 單元 | I/O | 名稱 | 說明 |
|---|-----|--|-------------------------|
| 解碼單元： dec (此模組是用於指令解碼、相依性記分牌和暫存器檔案存取的頂層模組) | 輸入 | exu_flush_final | 當訊號值為1時，排清管線 |
| | | [31:0] ifu_i0_instr, [31:1] ifu_i1_instr | 來自對齊階段的指令 |
| | | [31:1] ifu_i0_pc [31:1] ifu_i1_pc | 來自對齊階段的PC |
| | | | |
| | 輸出 | alu_pkt_t i0_ap | ALU控制訊號 |
| | | alu_pkt_t i1_ap | |
| | | lsu_pkt_t lsu_p | LSU控制訊號 |
| | | mul_pkt_t mul_p | MUL控制訊號 |
| | | div_pkt_t div_p | DIV控制訊號 |
| | | predict_pkt_t i0_predict_p_d i1_predict_p_d | 傳送至ALU的預測訊號 |
| | | [31:1] dec_i0_pc_d [31:1] dec_i1_pc_d | 解碼階段的指令位址 |
| | | [31:0] gpr_i0_rs1_d [31:0] gpr_i0_rs2_d [31:0] gpr_i1_rs1_d [31:0] gpr_i1_rs2_d | 來自暫存器檔案的I0/I1 rs1/rs2資料 |
| | | [31:0] dec_i0_immed_d [31:0] dec_i1_immed_d | 立即數值 |
| | | [12:1] dec_i0_br_immed_d [12:1] dec_i1_br_immed_d | 分支偏移 |
| | | [31:0] i0_rs1_bypass_data_d [31:0] i0_rs2_bypass_data_d [31:0] i0_rs1_bypass_data_e2 [31:0] i0_rs2_bypass_data_e2 [31:0] i0_rs1_bypass_data_e3 [31:0] i0_rs2_bypass_data_e3 | I0 rs1/rs2旁路資料 |
| | | [31:0] i1_rs1_bypass_data_d [31:0] i1_rs2_bypass_data_d [31:0] i1_rs1_bypass_data_e2 [31:0] i1_rs2_bypass_data_e2 [31:0] i1_rs1_bypass_data_e3 [31:0] i1_rs2_bypass_data_e3 | I1 rs1/rs2旁路資料 |
| | | | |
| | 內部 | [31:0] dec_i0_instr_d [31:0] dec_i1_instr_d | 解碼階段的指令 |
| | | | |

| | | | |
|--|----|--|-----------------|
| | | [31:0] dec_i0_rs1_d [31:0] dec_i0_rs2_d [31:0] dec_i1_rs1_d [31:0] dec_i1_rs2_d | rs1/rs2資料 |
| 從對齊階段傳送至 解碼階段的指令/ PC： dec_ib_ctl (用於將指令和 PC從對齊階段傳 播到解碼階段的緩 衝區) | 輸入 | [31:0] ifu_i0_instr [31:0] ifu_i1_instr | 對齊階段的I0/I1指令 |
| | | [31:1] ifu_i0_pc [31:1] ifu_i1_pc | 對齊階段的I0/I1 PC |
| | 輸出 | [31:0] dec_i0_instr_d [31:0] dec_i1_instr_d | 解碼階段的I0/I1指令 |
| | | [31:1] dec_i0_pc_d [31:1] dec_i1_pc_d | 解碼階段的I0/I1 PC |
| 對指令進行解碼並 計算旁路值： dec_decode_ctl (對2條指令進行 解碼並計算旁路 值) | 輸入 | [31:1] dec_i0_pc_d [31:1] dec_i1_pc_d [31:0] exu_i0_result_e1 | I0/I1 PC |
| | | [31:0] dec_i0_instr_d, [31:0] dec_i1_instr_d | 解碼階段的指令 |
| | 輸出 | alu_pkt_t i0_a alu_pkt_t i1_ap | ALU控制訊號 |
| | | lsu_pkt_t lsu_p | LSU控制訊號 |
| | | mul_pkt_t mul_p | MUL控制訊號 |
| | | div_pkt_t div_p | DIV控制訊號 |
| | | predict_pkt_t i0_predict_p_d i1_predict_p_d | 傳送至ALU的預測 訊號 |
| | | [4:0] dec_i0_rs1_d [4:0] dec_i0_rs2_d [4:0] dec_i1_rs1_d [4:0] dec_i1_rs2_d | I0/I1 rs1/rs2索引 |
| | | [31:0] dec_i0_immed_d [31:0] dec_i1_immed_d | 立即數值 |
| | | [12:1] dec_i0_br_immed_d [12:1] dec_i1_br_immed_d | 分支偏移 |
| | | [31:0] i0_rs1_bypass_data_d [31:0] i0_rs2_bypass_data_d [31:0] i0_rs1_bypass_data_e2 [31:0] i0_rs2_bypass_data_e2 [31:0] i0_rs1_bypass_data_e3 [31:0] i0_rs2_bypass_data_e3 | I0 rs1/rs2旁路資料 |
| | | [31:0] i1_rs1_bypass_data_d [31:0] i1_rs2_bypass_data_d [31:0] i1_rs1_bypass_data_e2 | I1 rs1/rs2旁路資料 |

| | | | |
|---|----|---|------|
| | | [31:0] i1_rs2_bypass_data_e2 [31:0] i1_rs1_bypass_data_e3 [31:0] i1_rs2_bypass_data_e3 | |
| 暫存器檔案： dec_gpr_ctl (暫存器檔案) | 輸入 | [4:0] raddr0, raddr1 | 讀取位址 |
| | | [4:0] raddr2, raddr3 | |
| | | [4:0] waddr0, waddr1 | 寫入位址 |
| | | [4:0] waddr2 | |
| | | [31:0] wd0, wd1, wd2 | 寫入資料 |
| | | rden0, rden1, rden2, rden3 | 讀取啟用 |
| | | wen0, wen1, wen2 | 寫入啟用 |
| | 輸出 | [31:0] rd0, rd1, rd2, rd3 | 讀取資料 |

表5. exu (執行單元) 的I/O和子模組 (包括子模組的I/O)

| 單元 | I/O | 名稱 | 說明 |
|---|-----|----------------------------------|----------------|
| 執行單元： exu (該模組為用於執行A-L指令的頂層模組) | 輸入 | alu_pkt_t i0_ap, alu_pkt_t i1_ap | ALU控制 |
| | | mul_pkt_t mul_p | MUL控制 |
| | | div_pkt_t div_p | DIV控制 |
| | | [31:1] dec_i0_pc_d, dec_i1_pc_d | 解碼階段的PC |
| | | [31:0] gpr_i0_rs1_d | I0/I1 rs1/rs2 |
| | | [31:0] gpr_i0_rs2_d | |
| | | [31:0] gpr_i1_rs1_d | |
| | | [31:0] gpr_i1_rs2_d | |
| | | [31:0] dec_i0_immed_d | 立即數值 |
| | | [31:0] dec_i1_immed_d | |
| | | [12:1] dec_i0_br_immed_d | 分支偏移 |
| | | [12:1] dec_i1_br_immed_d | |
| | | [31:0] i0_rs1_bypass_data_d | I0 rs1/rs2旁路資料 |
| | | [31:0] i0_rs2_bypass_data_d | |
| | | [31:0] i0_rs1_bypass_data_e2 | |
| | | [31:0] i0_rs2_bypass_data_e2 | |
| | | [31:0] i0_rs1_bypass_data_e3 | |
| | | [31:0] i0_rs2_bypass_data_e3 | |
| | | [31:0] i1_rs1_bypass_data_d | I1 rs1/rs2旁路資料 |
| | | [31:0] i1_rs2_bypass_data_d | |
| | | [31:0] i1_rs1_bypass_data_e2 | |
| | | [31:0] i1_rs2_bypass_data_e2 | |
| | | [31:0] i1_rs1_bypass_data_e3 | |
| | | [31:0] i1_rs2_bypass_data_e3 | |
| | 輸出 | exu_flush_final | 當訊號值為1時排清管線 |
| | | [31:0] exu_i0_result_e1 | 主ALU結果 |
| | | [31:0] exu_i1_result_e1 | |
| | | [31:0] exu_i0_result_e4 | 輔助ALU結果 |
| | | [31:0] exu_i1_result_e4 | |
| | | [31:0] exu_mul_result_e3 | MUL結果 |
| | | [31:0] exu_div_result | DIV結果 |
| | | [31:0] exu_lsu_rs1_d | 載入/儲存位址 |

| | | | |
|--|----|----------------------|--------------------------|
| | | [31:0] exu_lsu_rs2_d | 儲存資料 |
| ALU : exu_alu_ctl (算術邏輯單元) | 輸入 | [31:0] a | A運算元 |
| | | [31:0] b | B運算元 |
| | | [31:1] pc | 用於計算下一PC (即PC+2或PC+4) |
| | | [12:1] brimm | 分支偏移 |
| | | alu_pkt_t ap | ALU控制 |
| | 輸出 | [31:0] out | ALU結果 |
| | | flush_upper | 分支排清 |
| | | [31:1] flush_path | 目標PC |
| | | [31:1] pc_ff | |
| 乘法器 : exu_mul_ctl | 輸入 | [31:0] a | A運算元 |
| | | [31:0] b | B運算元 |
| | | mul_pkt_t mp | MUL控制 |
| | 輸出 | [31:0] out | MUL結果 |
| 除法器 : exu_div_ctl | 輸入 | [31:0] dividend | 分子 |
| | | [31:0] divisor | 分母 |
| | | div_pkt_t dp | DIV控制 |
| | 輸出 | [31:0] out | DIV結果 |

表6. lsu (載入/儲存單元) 的I/O和子模組 (包括子模組的I/O)

| 單元 | I/O | 名稱 | 說明 |
|--|-------|--|----------------------|
| 載入/儲存單元 : lsu (該模組為用於指令載入/儲存單元的頂層模組) | 輸入/輸出 | 多個訊號 | 用於mem模組輸入/輸出的DCCM連接埠 |
| | | 多個訊號 | DMA從屬連接埠 |
| | | 多個訊號 | LSU AXI連接埠 |
| | 輸入 | [31:0] exu_lsu_rs1_d | 載入/儲存位址 |
| | | [31:0] exu_lsu_rs2_d | 儲存資料 |
| | | [11:0] dec_lsu_offset_d | 位址偏移 |
| | | lsu_pkt_t lsu_p | LSU控制 |
| | 輸出 | [31:0] lsu_result_dc3 | LSU讀取資料 |
| 位址計算 : lsu_lsc_ctl (LSU控制並計算載入/儲存位址) | 輸入 | [31:0] exu_lsu_rs1_d | 載入/儲存位址 |
| | | [31:0] exu_lsu_rs2_d | 儲存資料 |
| | | [11:0] dec_lsu_offset_d | 位址偏移 |
| | | lsu_pkt_t lsu_p | LSU控制 |
| | 輸出 | [31:0] lsu_addr_dc1 [31:0] end_addr_dc1 | 初始/最終位址 |
| DCCM控制 : lsu_dccm_ctl (DCCM控制) | 輸入 | [`RV_DCCM_FDATA_WIDTH-1:0] dccm_rd_data_lo | 讀取資料 (低位儲存區) |
| | | [`RV_DCCM_FDATA_WIDTH-1:0] dccm_rd_data_hi | 讀取資料 (高位儲存區) |
| | 輸出 | dccm_wren | 寫入啟用 |
| | | dccm_rden | 讀取啟用 |
| | | [`RV_DCCM_BITS-1:0] dccm_wr_addr | 寫入位址 |

| | | | |
|---------------------------------------|----|--|-------------------------|
| | | [`RV_DCCM_BITS-1:0] dccm_rd_addr_lo | 讀取位址（低位儲存區） |
| | | [`RV_DCCM_BITS-1:0] dccm_rd_addr_hi | 讀取位址（高位儲存區）：未對齊載入需使用此位址 |
| | | [`RV_DCCM_FDATA_WIDTH-1:0] dccm_wr_data | 寫入資料 |
| 儲存緩衝區： lsu_stbuf （儲存緩衝區） | 輸入 | lsu_addr_dc3 | 位址 |
| | | [`RV_DCCM_DATA_WIDTH-1:0] store_ecc_datafn_hi_dc3 | 寫入資料（高位儲存區） |
| | | [`RV_DCCM_DATA_WIDTH-1:0] store_ecc_datafn_lo_dc3 | 寫入資料（低位儲存區） |
| | 輸出 | [`RV_LSU_SB_BITS-1:0] stbuf_addr_any | 儲存緩衝區位址 |
| | | [`RV_DCCM_DATA_WIDTH-1:0] stbuf_data_any | 儲存緩衝區資料 |

4. 用於對控制位元進行分組的結構/類型

下文摘要了檔案

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/swerv_types.sv` 中定義的、用於在SweRV EH1處理器中對控制訊號進行分組的主要結構/類型。

- **dec_pkt_t**：此類型為主要控制結構類型，其中包含alu（如果執行算術邏輯指令，則訊號值為1，否則為0）、load（如果執行load指令，則訊號值為1，否則為0）、legal（如果指令合法，則訊號值為1，如果不合法，則為0）、rs1（如果指令從暫存器檔案取得第一個輸入運算元，則訊號值為1，否則為0）、imm12（如果指令使用12位元立即數作為輸入運算元，則訊號值為1，否則為0）等處理器主要控制訊號。

此結構類型用於模組**dec_decode_ctl**內部，可產生許多其他控制訊號。宣告此類型的四個訊號（通路0：i0_dp_raw, i0_dp。通路1：i1_dp_raw, i1_dp）並使用這些訊號產生檔案**swerv_types.sv**定義的其他結構的控制位元。

這些位元在模組**dec_dec_ctl**內部指定，該模組可透過檔案

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec_decode_ctl.sv` 末尾提供的開放原始碼工具（**coredecode**和**espresso**）自動產生。

- **alu_pkt_t**：此結構類型包含與ALU操作相關的控制訊號，例如valid（如果執行算術邏輯指令，則訊號值為1，否則為0）、add（如果執行add指令，則訊號值為1，否則為0）、beq（如果執行beq指令，則訊號值為1，否則為0）等。此類型的兩個訊號稱為i0_ap和i1_ap，在模組**dec_decode_ctl**中定義。

這些位元在模組**dec_decode_ctl**（實作路徑：

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec_decode_ctl.sv`）中基於結構**dec_pkt_t**的位元進行指定（參見**dec_decode_ctl**的第711-770行）。

- **reg_pkt_t**：此結構類型包含兩個來源暫存器（欄位rs1和rs2）和一個目標暫存器（欄位rd）的識別碼。此類型的兩個訊號稱為i0r和i1r，在模組**dec_decode_ctl**內部定義。這些訊號透過模組**dec_decode_ctl**內部指令暫存器的適當欄位指定（參見本模組第1121-1127行）。
- **dest_pkt_t**：此結構類型包含寫回階段使用的控制位元，我們將在後續部分進行分析。此類型的一個訊號稱為dd，在模組**dec_decode_ctl**內部定義。
- **rets_pkt_t**、**br_pkt_t**、**br_tlu_pkt_t**和**predict_pkt_t**：這些結構類型與分支指令和分支預測器相關。
- **lsu_pkt_t**：此結構類型包含與載入/儲存單元相關的控制訊號，例如half（如果讀/寫半字，則訊號值為1，否則為0）、load（如果執行load指令，則訊號值為1，否則為0）、valid（如果指令有效，則訊號值為1，否則為0）等。此類型的一個訊號稱為lsu_p，在模組**dec_decode_ctl**中定義。

- **mul_pkt_t**：此結構類型包含與乘法單元相關的控制訊號，例如rs1_sign和rs2_sign（用於確定輸入運算元被視為有符號還是無符號）以及valid（如果指令有效，則訊號值為1，否則為0）等。此類型的一個訊號稱為mul_p，在模組**dec_decode_ctl**內部定義。
- **div_pkt_t**：此結構類型包含與除法單元相關的控制訊號，例如unsign（如果運算為無符號運算，則訊號值為1，否則為0）和valid（如果指令有效，則訊號值為1，否則為0）等。此類型的一個訊號稱為div_p，在模組**dec_decode_ctl**內部定義。

任務：開啟檔案

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/swerv_types.sv，並在後續介紹用於對控制位元進行分組的結構類型時分析該檔案。

任務：快速查看模組**dec_decode_ctl**和**dec_dec_ctl**，瞭解如何根據指令的32位元來指定控制訊號欄位。這兩個模組用途廣泛且結構複雜，因此我們不打算對其進行詳細分析。此外，查看模組**dec_dec_ctl**的自動建立過程，如**dec_decode_ctl.sv**的第2482-2495行所述。

5. 壓縮指令

在我們的大多數實驗中，為了簡單起見，我們停用了壓縮指令的使用，但在本部分中，我們描述並分析了RISC-V的壓縮指令延伸（RVC）以及在SweRV EH1中執行壓縮指令的情況。顯然，您可以隨意在實驗中啟用壓縮指令，自行延伸分析。

附註：開始本實驗前，建議您先閱讀S.Harris和D.Harris所著的《數位設計和電腦體系結構》（RISC-V版本，Morgan Kaufmann出版，簡稱[DDCARV]）的第7.7.4節。本部分的一些內容受這本書的啟發。

RVC延伸透過減少控制、立即數和暫存器欄位的大小並利用冗餘或隱含暫存器將常見整數和浮點指令的大小減小到16位元。指令大小減小後可降低成本、功耗和所需的儲存空間。對於手持和行動應用來說，這些改進至關重要。由於SweRV EH1包含RVC，我們的組合語言程式可以使用壓縮指令和32位元指令的組合。

在SweRV EH1中，有一個專用於解壓縮指令的模組：**ifu_compress_ctl**。該模組會接收16位元壓縮指令，並輸出相應的32位元未壓縮指令。在圖15中，我們比實驗11更詳細地顯示了對齊階段的情況（我們仍然留下了一些黑色方塊，供您自行分析）。三個**ifu_compress_ctl**模組在模組**ifu_aln_ctl**中實例化，從訊號aligndata[63:0]處接收壓縮指令，並在訊號uncompress0[31:0]、uncompress1[31:0]和uncompress2[31:0]中傳回相應的未壓縮指令。如果指令已經為未壓縮格式，則直接由aligndata[63:0]訊號提供指令。

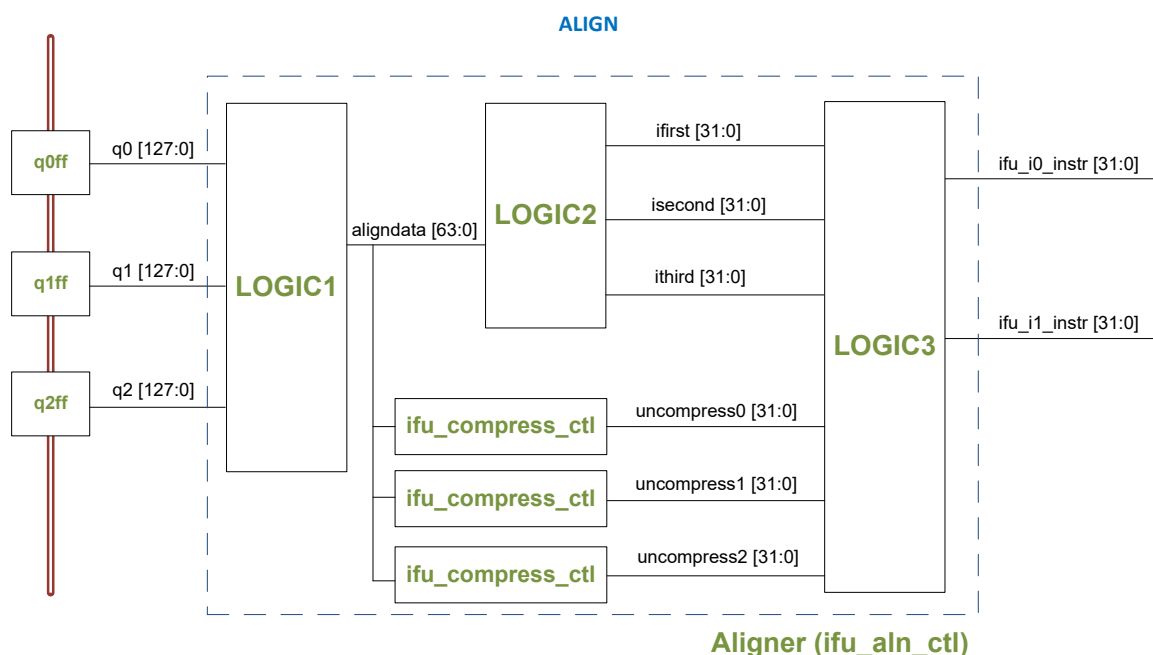


圖15. 對齊階段

圖16上半部分所示的程式碼是DDCARV第6章範例31中的簡單C程式。圖16下半部分所示的程式碼是在啟用RVC延伸功能的情況下，在PlatformIO中編譯該C程式時產生的組合語言程式碼（請注意，此組合語言程式碼與[DDCARV]中所示的略有不同）。我們用紅色強調顯示構成迴圈主體的指令，其中同時包含16位元和32位元指令。

| | | |
|--|------|---------------------------|
| <pre> int scores[200]; int main(void) { int i; for (i = 0; i < 200; i = i + 1){ scores[i] = scores[i] + 10; } return(0); } </pre> | | |
| 00000088 <main>: | | |
| 88: 6789 | lui | a5,0x2 |
| 8a: 12078793 | addi | a5,a5,288 # 2120 <scores> |
| 8e: 32078693 | addi | a3,a5,800 |
| 92: 4398 | lw | a4,0(a5) |
| 94: 0791 | addi | a5,a5,4 |
| 96: 0729 | addi | a4,a4,10 |
| 98: fee7ae23 | sw | a4,-4(a5) |
| 9c: fed79be3 | bne | a5,a3,92 <main+0xa> |
| a0: 4501 | li | a0,0 |
| a2: 8082 | ret | |

圖16. 壓縮指令範例

圖17所示為圖16中迴圈的一次完整迭代的Verilator模擬結果。請注意，當指令addi a5,a5,4處於對齊階段（在圖中的第一個週期中以紅色強調顯示）時，將從64位元束（aligndata[63:0]）中擷取該指令，並將其從16位元指令（0x0791）解壓縮為32位元指令（0x00478793）。（可造訪[\[RVfpgaPath\]/RVfpga/Labs/Lab11/Compressed_C-Example](#)取得程式碼，以便自行執行Verilator模擬。）

- 在RISC-V中，16位元c.addi指令的操作碼如下（參見[DDCARV]的附錄B）：
000 | imm(1-bit) | rd/rs1 | imm(5-bits) | 01

因此，可以輕鬆驗證0x0791（0000011110010001）是否對應於c.addi a5,4（請記住，a5 = x15）。

- Imm = 000100
- rd = rs1 = 01111 (x15)

- 在RISC-V中，32位元addi指令的操作碼如下（參見[DDCARV]的附錄B）：
imm(12-bits) | rs1 | 000 | rd | 0010011

因此，可以輕鬆驗證0x00478793（00000000010001111000011110010011）是否對應於addi a5,a5,4（請記住，a5 = x15）。

- Imm = 000000000100
- rs1 = 01111 (x15)
- rd = 01111 (x15)

在圖17所示的第二個週期中，將對齊sw指令。由於該指令在RISC-V架構中缺乏相應的壓縮版本，因此不需要解壓縮，只需直接從aligndata[63:0]訊號中選擇指令並傳播到解碼階段。

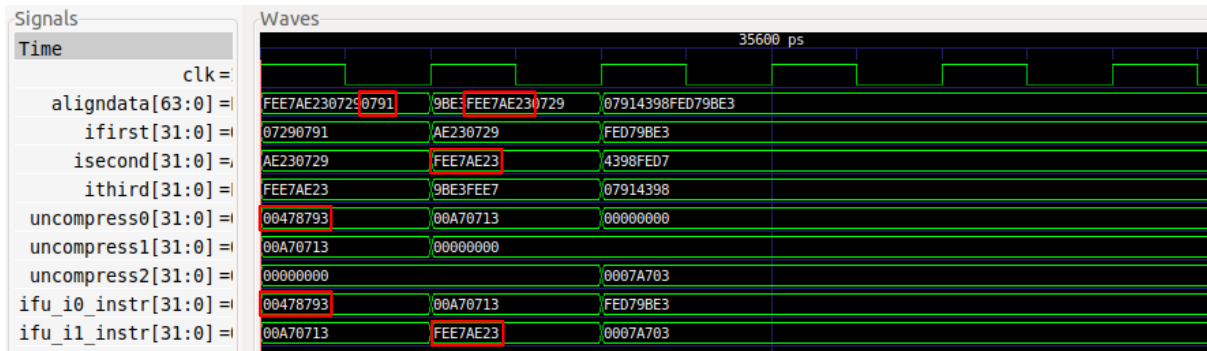


圖17. 圖16所示程式碼的模擬結果

任務：以壓縮/未壓縮指令為分類依據，分析迴圈主體中的其餘指令。

任務：觀察模組ifu_compress_ctl的內部結構，分析其工作原理。

6. 實際基準

在資料夾[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks中，我們提供了三個實際應用，實驗20中將使用它們來測試SweRV EH1處理器的不同功能。該實驗將進一步介紹這三種基準，以及我們為每種基準提供的不同應用版本。

- **CoreMark**：資料夾
[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/CoreMark_HwCounters中提供了一個PlatformIO專案，其中包含在RVfpgaNexys上執行程式時的CoreMark基準。我們使用<https://github.com/chipsalliance/Cores-SweRV>提供的原始程式碼對該基準進行了修改，使其能夠適用於RVfpga系統。
- **Dhrystone**：資料夾
[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Dhrystone_HwCounters中提供了一個PlatformIO專案，其中包含在RVfpgaNexys上執行程式時的Dhrystone基準。我們使用<https://github.com/chipsalliance/Cores-SweRV>提供的原始程式碼對該基準進行了修改，使其能夠適用於RVfpga系統。
- **影像處理**：資料夾
[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/ImageProcessing_HwCounters中提供了一個PlatformIO專案，其中包含我們在實驗5中將RGB影像轉換為灰階影像時所使用的應用程式。