**Verilog Laboratory Exercise**

**Design Preparation**

unix% cd ~

unix% \rm -rf \* ( Do it when disk space is not enough )

unix% tar xvf /\*\*\*\*\*\*/vloglab\_15s.tar

Ask the instructor !

**Environment Setup**

unix% cp /\*\*\*\*\*\*/tcshrc .tcshrc

Ask the instructor !

unix% source .tcshrc

**Start**

unix% cd vloglab

『Download lab files』

*Mail : hlchen@narlabs.org.tw*

**Lab1 : 2-1 MUX**

* 請設計一個『**2對1的多工器**』
* Specifications
  + Module name : mux
  + Input pins : a, b, sel
  + Output pins : out
  + Symbol View :



1. 切換工作目錄到 **Lab1** 資料夾. 此資料夾內包含 **mux.v** 及 **mux\_text.v** 兩個檔案。在**terminal**內執行以下指令：

**unix%** cd Lab1

1. 使用以上的**DUT**電路圖撰寫 **mux.v** 的RTL Code，其RTL Code可參考以下範例（本範例含Gate-Delay）：

`define dly\_and 1

`define dly\_or 2

module MUX (out,a,b,sel);

**// Port declarations**

output out;

input a,b,sel;

**// The netlist**

not not1(sel\_, sel);

and #`dly\_and and1(a1, a, sel\_);

and #`dly\_and and2(b1, b, sel);

or #`dly\_or or1(out, a1, b1);

endmodule

1. 修改測試檔 ( **mux\_test.v** )。

測試檔的內容已經撰寫完成The signal declarations, model instantiation, and response generation 等三個部份。但測試檔對**DUT**的Apply Stimulus部分尚未完成，請同學完成其測試檔對**DUT**輸入訊號之產生，本Apply Stimulus之時間點與動作請按照下表撰寫；並請於時間點100個時間單位（time 100）時讓本次simulation結束。

|  |  |  |  |
| --- | --- | --- | --- |
| **time** | **a** | **b** | **sel** |
| *10* | *0* | *0* | *0* |
| *20* | *1* | *0* | *1* |
| *30* | *1* | *1* | *0* |
| *40* | *0* | *1* | *1* |
| *50* | *0* | *1* | *0* |
| *60* | *1* | *0* | *0* |
| *70* | *0* | *0* | *1* |
| *80* | *1* | *1* | *1* |

1. 您可以在測試檔中加入waveform display的相關code來做simulation。如此可讓simulation時產生waveform。（以下內容可視您需要的情況加入）。

**// waveform display**

initial begin

$dumpfile(“lab1.vcd”); // The VCD Database

$dumpvars; // The VCD Database

$fsdbDumpfile(“lab1.fsdb”); // The FSDB Database

$fsdbDumpvars; // The FSDB Database

$shm\_open(“lab1.shm”); // The SHM Database

$shm\_probe(“AC”); // The SHM Database

end

1. 執行Simulation。

若使用**Verilog-XL**執行simulation的話，則使用以下指令：

**unix%** verilog mux\_test.v mux.v

若使用**NC-Verilog**執行simulation的話, 則使用以下指令（建議使用）：

**unix%** ncverilog mux\_test.v mux.v +access+r

「Note : 在NC-Verilog執行的指令內時加入 +access+r 這個option是為了可以在Waveform Tool內看到訊號的波形變化。如果不打算看Waveform的話，則可以不用加此option。」

1. 執行silumation完成後的結果將如下所示。

ncsim> run

0 a = x, b = x, sel = x, out = x

10 a = 0, b = 0, sel = 0, out = x

13 a = 0, b = 0, sel = 0, out = 0

20 a = 1, b = 0, sel = 1, out = 0

30 a = 1, b = 1, sel = 0, out = 0

33 a = 1, b = 1, sel = 0, out = 1

40 a = 0, b = 1, sel = 1, out = 1

50 a = 0, b = 1, sel = 0, out = 1

53 a = 0, b = 1, sel = 0, out = 0

60 a = 1, b = 0, sel = 0, out = 0

63 a = 1, b = 0, sel = 0, out = 1

70 a = 0, b = 0, sel = 1, out = 1

73 a = 0, b = 0, sel = 1, out = 0

80 a = 1, b = 1, sel = 1, out = 0

83 a = 1, b = 1, sel = 1, out = 1

Simulation complete via $finish(1) at time 180 NS + 0

./mux\_test.v:22 #100 $finish;

**Wave Tool and Waveform Database**

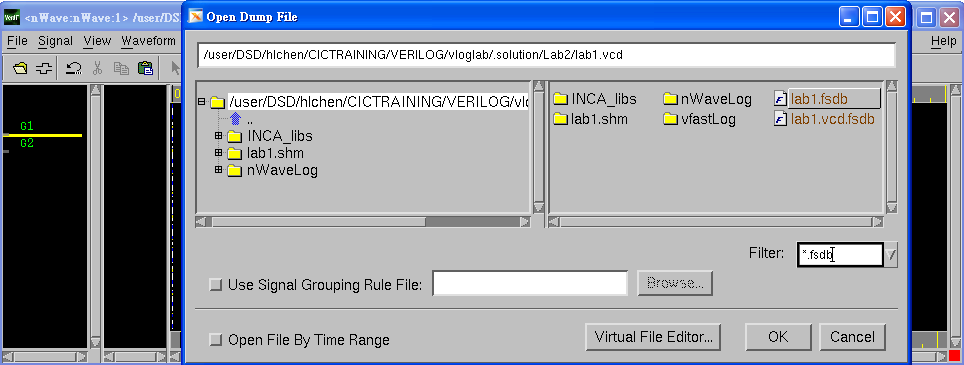
* **Verdi – nWave**

1. In the command shell, open the waveform tool from the same directory where you started the simulation.

To invoke **nWave**, enter :

**unix%** nWave &

1. In the waveform window, select **File – Open** . The file browser appears.



* 1. double click the **lab1.fsdb** database
  2. click **OK**

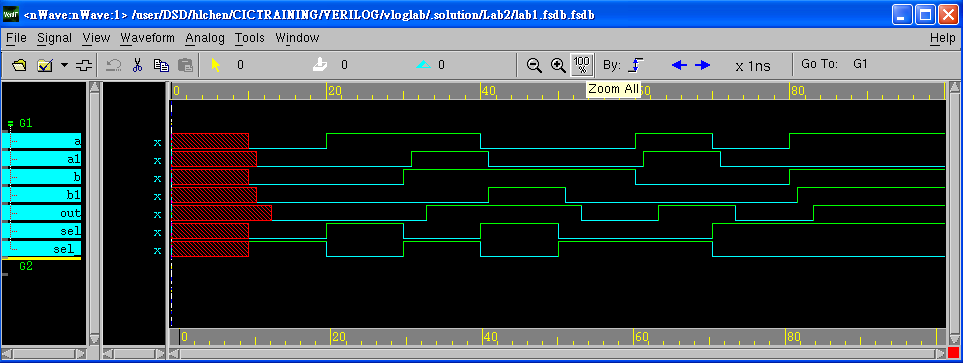
1. From the **nWave** window, select **signal – get signal**.

The Get signal window appears.

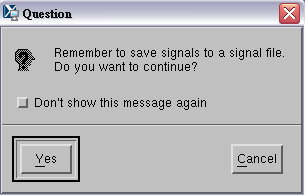
1. You can select **mux\_test** or **mux** to find the I/O signal of the module.
2. Select the signal to scope.
3. Click **OK**



1. Then the waveform will be shown in the **nWave** browser.



1. From the **nWave** menu, select **File – Exit.**



1. Click **Yes**.

The **nWave** window closes.

* **cadence simvision waveform viewer**

1. In the command shell, open the waveform tool from the same directory where you started the simulation.

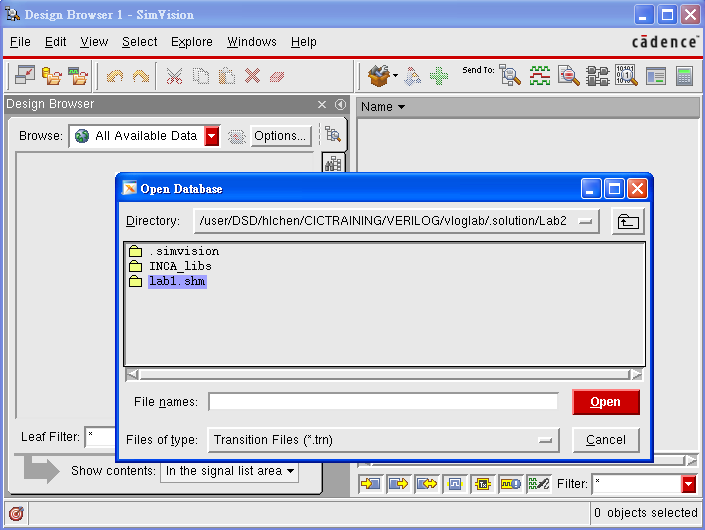
To invoke **simvision** , enter :

**unix%** simvision &

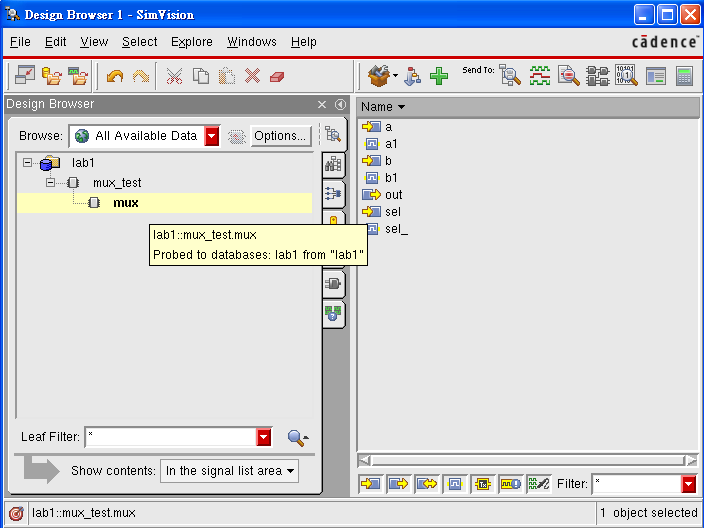
or supply the database name as an argument.

**unix%** simvision file.shm &

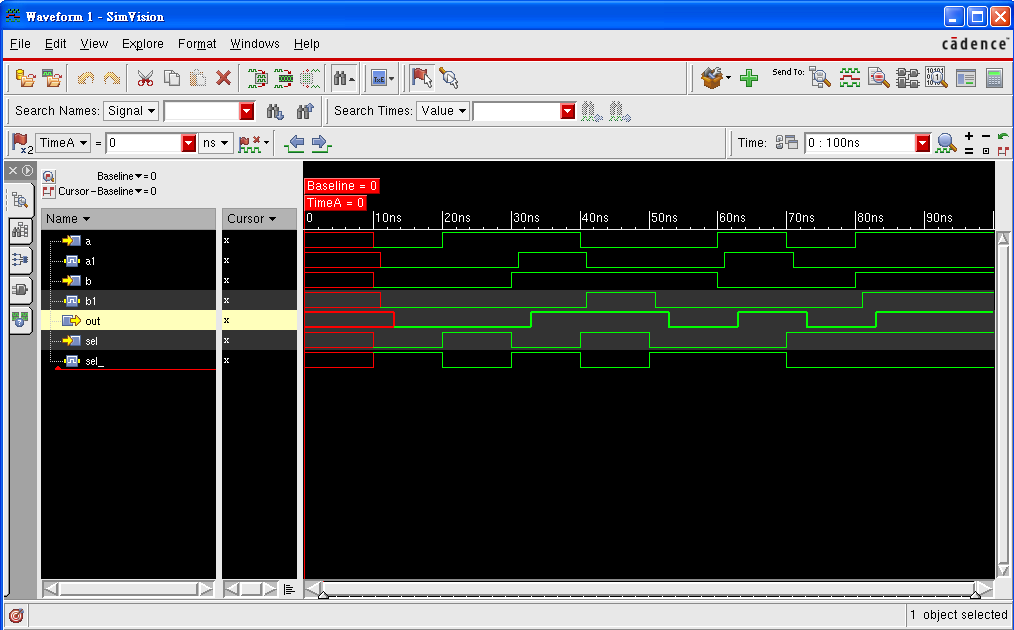
1. In the waveform window, select **File – Open Database** file. The file browser appears.



1. Select the **lab1.shm** database, click **Open**
2. Then select the **lab1.trn,** click **Open**
3. From the **simvision** Design Browser, click the launch of the **mux\_text** module to find the **mux** module signal view.

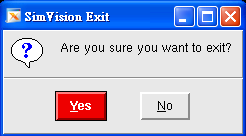


1. select the **mux** module instance, you can find all the pins of the module in the right browser. Shown as over.
2. click the **Waveform** icon, the waveform window appears .



1. From the **simvision** menu, select **File - Exit simvision**.

A pop-up window appears to verify your intentions.



1. Click **Yes**.

The **simvision** window closes.

**Lab2a : Full Adder Module Design**

* 請設計一個Full-Adder模組
* Specifications
  + Module name : fa
  + Input pins : a, b, ci
  + Output pins : sum, cout
  + Function : { cout, sum } = a + b + ci;
  + Truth Table :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a** | **b** | **ci** | **cout** | **sum** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

* **Solution 1 :**

1. 切換工作目錄到 **Lab2a** 下。此目錄下包含 **fa.v** 及 **fa\_test.v** 兩個檔案。

**unix%** cd Lab2a

1. 開啟 **fa.v** 檔案後。參考以上真值表之內容，並使用Structural Modeling章節所闡述之方法來撰寫這**Full Adder**之RTL Code。您可以使用**Verilog**語言所內建的built-in primitives來描述其電路的連結與架構。

module fa(a, b, ci, sum. cout);

input a, b, ci;

output sum, cout;

and g1(……);

not g2(……);

…………

endmodule

1. 您可以使用**NC-Verilog***來執行*simulation

**unix%** ncverilog fa\_test.v fa.v +access+r

1. 本lab的測試程式有自動比對的功能，假如您所設計的全加器功能正常的話，將可在終端機上看見類似以下的訊息！！

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function test pass !!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Solution 2: Half-Adder and Full-Adder**



1. 改變工作目錄到 **Lab2a**下. 此目錄下包含 **fa.v, ha.v**及**fa\_test.v**等三個檔案。

**unix%** cd Lab2a

1. 在**fa.v**檔案中撰寫您的full-adder module程式. 在full-adder module中使用下列的module header及port interface：

module fa(a, b, ci, sum, cout);

output sum, cout;

input a, b, ci;

......

endmodule

這個Full-adder module可以由兩個Half-adder與一些邏輯閘組成。請撰寫Half-adder module( **ha.v** )程式, 並使用下列的module header及port interface：

module ha(a, b, sum, cout);

output sum, cout;

input a, b;

……

Endmodule

1. 請自行創造一個verilog control file 並命名為**vlog.f** 。

**unix% touch vlog.f**

1. 在此vlog.f中指名design file及testbench file檔案名稱，以及您所需要的command-line options。接下來請使用**vlog.f** 檔案來進行模擬，請執行以下的命令 ：

**unix%** ncverilog -f vlog.f

1. 如果您的design功能正確的話，其模擬結果將如下所示 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function test pass !!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Lab2b : A 16-bit ripple-carry adder**

* Please modeling an Arithmetic Logic Unit (ALU)
* Specifications
  + Module name : rca16
  + Input pins : a[15:0], b[15:0], c\_in
  + Output pins : c\_out, sum[7:0]
  + Symbol view :





1. 切換工作目錄到**Lab2b**資料夾**。**這資料夾下包含測試檔 **rca16\_test.v** 及設計檔**rca16.v 。**

**unix%** cd Lab2b

1. 依模組需求撰寫**rca16.v**檔案。本Ripple Carry Adder（Adder\_rca16）架構為16個bits，其由4個4 bits的ripple carry adder（Add\_rca4）組成，而每個Add\_rca4則是由4個Add\_full組成，每個Add\_full又可由2個Add\_half及1個OR gate組成。如上圖所示。
2. 當您撰寫完成這alu模組之後，請使用**rca16\_test.v**檔案作為測試檔來測試此電路。

您可使用**NC-Verilog**或**Verilog-XL**來跑simulation。

例如：**unix%** ncverilog rca16\_test.v rca16.v +access+r

1. 本測試檔已完成自動比對的功能，請檢查simulation的輸出結果。

**Lab3a : ALU**

* Please modeling an Arithmetic Logic Unit (ALU)
* Specifications
  + Module name : alu
  + Input pins : accum[7:0], data[7:0], opcode, clk, reset
  + Output pins : zero, alu\_out[7:0]
  + Symbol view :



1. 切換工作目錄到**Lab3a**資料夾**。**這資料夾下包含測試檔 **alu\_test.v** 及設計檔**alu.v 。**

**unix%** cd Lab3a

1. 依模組需求撰寫**alu.v**檔案 . 您必須撰寫依照上述的symbol view 撰寫port interface.並依照下列的規範撰寫此alu的功能描述。
   1. 所有輸入及輸出（除了「zero」訊號以外）均需同步於clock的正緣（rising edge）。
   2. 同步reset架構。當reset為1時表示reset啟動，此時alu\_out訊號輸出為0。
   3. accum、data及alu\_out訊號的數值使用2補數表示。
   4. 當accum輸入為0時，zero訊號輸出為1；反之當accum輸入不為0時，zero訊號輸出為0。並且zero訊號不需理會reset訊號的動作。
   5. 當opcode輸入為X( unknow )時，其alu\_out訊號輸出為0。
2. 以以下3-bits的opcode訊號值來定義其alu的操作模式

|  |  |
| --- | --- |
| opcode | ALU operation |
| 000 | Pass accum |
| 001 | accum + data (add) |
| 010 | accum - data (subtraction) |
| 011 | accum AND data (bit-wise AND) |
| 100 | accum XOR data (bit-wise XOR) |
| 101 | accum取絕對值 (absolute value) |
| 110 | accum\*5 + accum/8 |
| 111 | 假如( accum >= 128)，則alu\_out=data取2補數  否則alu\_out=data取1補數 |

1. 當opcode為101時opcode將輸出絕對值（absolute value），請使用accum的第7個bit（MSB, accum[7]）做為signed bit。
2. 當opcode為110及111時，請將accum及data之值皆視為正整數執行運算即可。
3. 當您撰寫完成這alu模組之後，請使用**alu\_test.v**檔案作為測試檔來測試此alu。

您可使用**NC-Verilog**或**Verilog-XL**來跑simulation。

例如：**unix%** ncverilog alu\_test.v alu.v +access+r

1. 本測試檔已完成自動比對的功能，請檢查simulation的輸出結果。

**Lab3b : FIR**

* Please modeling an 8-tap Gaussian lowpass FIR filter
* Specifications
  + Module name : FIR
  + Input pins : Din[7:0], clk, reset
  + Output pins : Dout[17:0]
  + Symbol view :



1. 切換工作目錄到**Lab3b**資料夾**。**這資料夾下包含測試檔 **FIR\_test.v** 及設計檔**FIR.v 。**

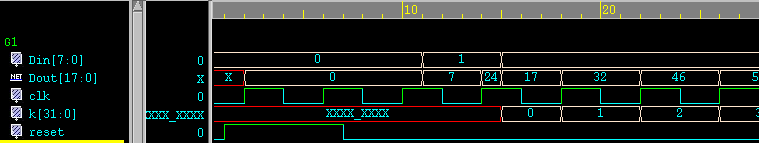
**unix%** cd Lab3b

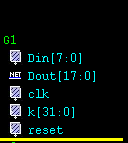
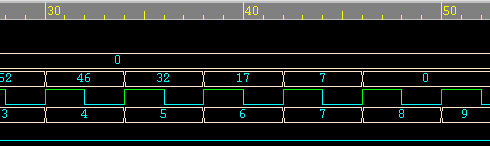
1. 依模組需求撰寫**FIR.v**檔案 . 您必須撰寫依照上述的symbol view 撰寫port interface.並依照下列的規範撰寫此FIR的功能描述。
   1. 輸入訊號需同步於clock的正緣（rising edge）。
   2. 同步reset架構。當reset為1時表示reset啟動，此時Shift Register的輸入皆為0。因此此時的輸出值為Din[7:0]乘上7的結果。
   3. 依本題symbol view所示可知其輸出訊號為Shift Register、各階系數及輸入訊號分別進行相乘及累加之後的結果，因此不限制必須與colck同步。
   4. 提示1：您可使用for迴圈來撰寫Shift Register的動作。
   5. 提示2：您可宣告一Register Array(Memory Array)來代表Shift Register架構。
2. 當您撰寫完成這FIR模組之後，請使用**FIR\_test.v**檔案作為測試檔來測試此FIR。

您可使用**NC-Verilog**或**Verilog-XL**來跑simulation。

例如：**unix%** ncverilog FIR\_test.v FIR.v +access+r

1. 您可參考以下輸出確認輸出結果是否正確。



**Lab4 : Modeling Delay**

* Use path delays and timing checks to improve the accuracy of a model.
* Specifications
  + Module name : dffr\_b
  + Input pins : clr\_, clk, d
  + Output pins : q, q\_
  + Symbol and Schematic :



1. 切換工作目錄到**Lab4**資料夾下，此資料夾下包含測試檔**test.v**及設計檔**dffr\_b.v**。
2. 查看**test.v**檔案內容。在**test.v**內容裡設置有flag訊號，當系統中有timing violation發生時，此flag訊號將會有相當的反應產生，請在dffr\_b模組中設置flag訊號以便讓test模組偵測dffr\_b模組是否有timing violation發生。
3. 撰寫**dffr\_b.v**檔案內容，其dffr\_b模組須包含以下timing information。
   1. 在模組內加入一個Specify block，並將以下所列的timing information描述其中。
   2. 在Specify block內加入setup time及hold time check的動作，並且須包含notifier旗標使用，其notifier須命名為flag（目的在跟test模組搭配使用），除此之外flag亦須宣告為reg的data type。

|  |  |  |  |
| --- | --- | --- | --- |
| Timing Constraints | min | typ | max |
| Tsu (setup) | 3 | 5 | 6 |
| Th (hold) | 2 | 3 | 6 |

* 1. 請以以下表格撰寫Specify block內有關timing information的程式。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pin-to-Pin Path Delays | | min | typ | max | min | typ | max |
| inputs | outputs | rise | | | fall | | |
| clr\_ | q, q\_ | 3 | 3 | 3 | 3 | 3 | 3 |
| clk | q | 2 | 3 | 5 | 4 | 5 | 6 |
| clk | q\_ | 2 | 4 | 5 | 3 | 5 | 6 |

1. Simulation看看。在不同的delay mode下，min、typ或max delay mode分別會有多少的violation產生?

**Lab5a : Testbench of MATRIX\_MUL**

* Please modeling an Testbench of Matrix\_multiplieri (MATRIX\_MUL)
* Specifications
  + Module name : matrix\_mul
  + Interface with MATRIX\_MUL : clk, di0[3:0], di1[3:0], di2[3:0],

mo0[9:0], mo1[9:0], mo2[9:0]

* + Symbol view :



其中，a = 7, b = 13, c = 5

* + 為簡化題目難度，本Lab不需考慮正負號及溢位問題。若乘加結果大於mo\*訊號可表示的10bits範圍，則只取較低位元的10 bits即可。

1. 切換工作目錄到**Lab5a**資料夾，這資料夾內包含測試檔**matrix\_mul\_test.v**及已完成的設計檔**matrix\_mul.v**。在這個lab中，您必須撰寫測試檔**matrix\_mul\_test.v**的內容，並使用**matrix\_mul\_test**模組去測試**matrix\_mul**模組的功能是否正常。

**unix%** cd Lab5a

1. 為了測試matrix\_mul模組功能，在**matrix\_mul\_test.v**測試檔中，我們必須撰寫對於測試matrix\_mul模組功能所需之各種測試向量，。請依照下列指示撰寫您的matrix\_mul測試模組功能：
   1. 首先您必須先instance matrix\_mul模組。
   2. 接下來您必須對matrix\_mul模組中有使用到的訊號宣告其data type，例如：reg、wire或其他data type。
   3. 在功能的部分，由於matrix\_mul模組必須有個clock訊號，因此我們必須在matrix\_mul模組內建立clock產生器。
      1. 定義clock週期為20ns。其中10ns為High，另外10ns為Low。
      2. Clock的初始狀態為0(Low)。
   4. 在功能部分，需要分別輸入di0、di1、di2的訊號，並觀察mo0、mo1、mo2的輸出結果。
      1. 任意由輸入di0、di1、di2一些值。本DUT module使用clock的正緣觸發，因此為求讓di0~2資料輸入可以正確地被Flip-Flop觸發，因此請在clock負緣時將di0~2資料輸入。
      2. 由$monitor觀察mo0、mo1、mo2的輸出結果。
      3. 您可以設計mo0、mo1、mo2的expected result，並且讓程式自動比對real result及expected result。
      4. 在驗證的部分，或許可以加上waveform display的宣告，以方便觀察波形變化。
2. 撰寫完成**matrix\_mul\_test.v**檔案後，您可使用**NC-Verilog**或**Verilog-XL**進行simulation。
3. 是否有其他狀況尚未考慮到的，請補充！

**Lab5b : Testbench of ALU**

* Please modeling an Testbench of Arithmetic Logic Unit (ALU)
* Specifications
  + Module name : alu\_test
  + Interface with ALU : accum[7:0], data[7:0], opcode, clk, reset, zero, alu\_out
  + Symbol view :



1. 切換工作目錄到**Lab5b**資料夾，這資料夾內包含測試檔**alu\_test.v**及已完成的設計檔**alu.v**。在這個lab中，您必須撰寫測試檔**alu\_test.v**的內容，並使用alu\_test模組去測試alu模組的功能是否正常。

**unix%** cd Lab5b

1. 為了測試alu模組功能，在**alu\_test.v**測試檔中，我們必須撰寫對於測試alu模組功能所需之各種測試向量，例如:測試reset功能、測試opcode功能及測試zero訊號功能等測試向量。請依照下列指示撰寫您的alu\_test測試模組功能：
   1. 在alu\_test模組內，首先您必須先instance alu模組，您可使用call by orders或call by name的方法來宣告。

alu alu(.clk(clk), .opcode(opcode), ….......);

* 1. 接下來您必須對alu\_test模組中有使用到的訊號宣告其data type，例如：reg、wire或其他data type。
  2. 在功能的部分，由於alu模組必須有個clock訊號，因此我們必須在alu\_test模組內建立clock產生器，並將其產生的clock（clk）訊號連接到alu模組的clock（clk）訊號。您可參考以下structural clock做法來設計：
     1. 定義clock週期為20ns。其中10ns為High，另外10ns為Low。
     2. Clock的初始狀態為unknown。



* 1. 在功能部分，我們需要測試各種opcode的輸入情況及reset的功能是否正常。
     1. 將reset啟動或停止動作來測試alu模組的reset是否有正常動作，當reset訊號為1時代表reset動作。
     2. 您可以使用「for loop」來自動產生8個opcodes訊號的值，並將之送入alu模組以便觀察功能。
     3. 為測試alu模組收到opcode訊號為unknown時的反應，您必須將opcode訊號設定為unknown。
     4. 測試zero訊號的功能（accum訊號為0時，zero訊號才會為1）。
     5. 為方便完成以上規範，您可在testbench內使用initial block。
  2. 在驗證的部分，或許可以加上waveform display的宣告，以方便觀察波形變化。

initial begin

$shm\_open("alu.shm"); // SHM Database

$shm\_probe(“AC”);

$fsdbDumpfile("alu.fsdb"); // FSDB Database

$fsdbDumpvars();

end

* 1. 我們可以加入monitor或display等輸出宣告，以方便simulation時就可觀察訊號輸入與相對應的訊號輸出關係。

1. 撰寫完成**alu\_test.v**檔案後，您可使用**NC-Verilog**或**Verilog-XL**進行simulation。
2. 檢查您給予alu\_test模組對alu模組的輸入的訊號及其相對應的輸出是否正常？是否有其他狀況尚未考慮到的，請補充！

**Lab6 : Memory**

* Modeling a Memory with a Bidirectional Data Bus.
* Specifications
  + Module name : mem
  + Input pins : addr[4:0], read, write
  + Output pins : data[7:0]
  + Symbol view：



1. 切換工作目錄到**Lab6**資料夾，此資料夾下包含測試檔**mem\_test.v**及設計檔**mem.v**。
2. 此lab的設計檔**mem.v**內容只完成module header的部分，特別注意其中data訊號是bidirectional的。請依照以下規範撰寫memory模組的內容：
   1. 使用memory register array，並命名為**memory**。此memory register array須符合以下規範：
      1. The MSB of each word is bit **7**
      2. The LSB of each word is bit **0**
      3. The first address is address **0**
      4. The last address is address **31** (hex 1F)
   2. 使用非同步存取方式此memory model，並使用兩個控制訊號做為memory model讀取（read）或寫入（write）的控制。



* 1. 在write控制訊號的positive edge發生時，此時在data訊號線出現的資料將會被寫入memory中，其寫入位址由當下addr訊號線之資料所定義。您可以使用procedural assignment來撰寫這部分的code。
  2. 當read控制訊號值為High時，memory將以目前addr訊號線之資料做為位址，將memory內該位址之內容藉由data訊號線讀出。
  3. 此memory須支援block read功能。當read控制訊號線維持在High的狀態下，若addr訊號線之資料改變，則被讀出資料之位址亦會隨之改變，因此可連續讀到不同位址之資料。
  4. 當read控制訊號值為Low時，其read控制將呈現不致能的狀態，在此情況下read控制訊號對data訊號線將呈現high-impedance的狀態。亦即在此情形下若write控制訊號值為high的話，則寫入的動作將可正常執行。

1. 使用**mem\_test.v**檔來測試**mem.v**設計檔。執行完simulation後，您會發現有error information出現。請查看**mem\_test.v**檔案內容，並思考以下問題：
   1. data訊號線為一bidirectional port，其data type宣告為wire
   2. 當在procedural block中，我們將資料值設給data訊號線時，此data訊號線是否應該宣告為reg的data type。
2. 為修正上述error information的問題，請修改**mem\_test.v**檔案內容後再跑一次simulation看看，直到error information消失為止。

提示：您可在procedural assignment中使用shadow register來修正。

1. procedural assignment內容主要用在write的情況下，您可對procedural block內的data訊號線，另外宣告使用一個register。
2. 使用continuous assignment及conditional operator來決定當read不啟動時，可將procedural block內的register內容寫入到memory。若read啟動時，表示memory要做讀取的動作，因此continuous assignment的結果須為High-impednace以避免互相衝突。
3. 請使用**mem\_test.v**檔案及**mem.v**檔案執行simulation，並確定其功能正常。



**Lab7a : bit-stream pattern detector**

* Please design a serial input bit-stream pattern detector module.
* Specifications
  + Module name : fsm\_bspd
  + Input pins : clk, reset, bit\_in
  + Output pins : det\_out
  + Function : serial input bit-stream pattern detector

Using finite state Mealy-machine. “*det\_out*” is to be low(logic 0), unless the input bit-stream is “0010” sequentially.

Example :

bit\_in : …. 0 0 1 0 0 1 0 0 0 1 0 0…

det\_out : …. 0 0 0 1 0 0 1 0 0 0 1 0…

1. 切換工作目錄到到**Lab7a**資料夾，此資料夾內包含測試檔**fsm\_test.v**及設計檔**fsm\_bspd.v**。其中**fsm\_bspd.v**只完成了module header的部分。
2. 撰寫**fsm\_bspd.v**檔案內容並完成上述所提之serial input bit-stream pattern detector功能。請使用Mealy-Machine狀態機實現。
   1. 所有輸入訊號須同步於clock（clk）的rising edge。
   2. 您可使用下列任何模式來設計此模組。
      1. Separates CS、NS and OL
      2. Combine CS and NS，Separate OL
      3. Combine NS and OL，Separate CS
   3. 您可自訂數個狀態來完成此設計。
      1. Less than 4 states：S0（00），S1（01），S2（10），S3（11）
      2. Between 5 and 8 states：S0（000），S1（001），S2（010），S3（011），

S4（100），S5（101），S6（110），S7（111）。

1. 當完成您的fsm\_bspd模組後，請用**NC-Verilog**或**Verilog-XL**跑simulation看看。如果有錯誤產生的話，請修正，直到正確無誤為止。
2. 您可以參考以下狀態圖與狀態表來完成您的fsm\_bspd設計。

State Diagram



State Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present state | Next state | | Present output | |
| X = 0 | X = 1 | X = 0 | X = 1 |
| S0 | S1 | S0 | 0 | 0 |
| S1 | S2 | S0 | 0 | 0 |
| S2 | S2 | S3 | 0 | 0 |
| S3 | S1 | S0 | 1 | 0 |

**Lab7b : FSM**

* Please design a finite state mealy-machine module.
* Specifications
  + Module name : fsm\_mealy
  + Input pins : clk, reset, enable, xin (X)
  + Output pins : zout (Z)
  + Function :



1. 切換工作目錄到到**Lab7b**資料夾，此資料夾內包含測試檔**fsm\_mealy\_test.v**及設計檔**fsm\_mealy.v**。其中**fsm\_mealy.v**只完成了module header的部分。
2. 撰寫**fsm\_mealy.v**檔案內容並完成上述所提之功能。請使用Mealy-Machine狀態機實現。
   1. 上述功能描述之A’為A訊號的下一狀態，而B’為B訊號的下一狀態。故{A,B}為目前狀態；而{A’,B’}為下一狀態。
   2. 所有輸入訊號須同步於clock（clk）的rising edge。
   3. 當en訊號如果為HIGH時，此狀態機才會動作。因此當en為LOW時，狀態機之狀態將不會推進，且輸出為0。
   4. 您可使用下列任何模式來設計此模組。
      1. Separates CS、NS and OL
      2. Combine CS and NS，Separate OL
      3. Combine NS and OL，Separate CS
   5. 您可自訂數個狀態來完成此設計。
      1. Less than 4 states：S0（00），S1（01），S2（10），S3（11）
      2. Between 5 and 8 states：S0（000），S1（001），S2（010），S3（011），

S4（100），S5（101），S6（110），S7（111）。

1. 當完成您的fsm\_mealy模組後，請用**NC-Verilog**或**Verilog-XL**跑simulation看看。如果有錯誤產生的話，請修正，直到正確無誤為止。
2. 提示1：您可先將A’及B’的卡諾圖畫出，依這兩個卡諾圖可訂出狀態表的Present state及Next State。
3. 提示2：將Z的卡諾圖畫出，可訂出狀態表的Present output。
4. 參考解答：





**Lab8 : Serial Transmitter**

**1. 問題描述**

請完成一個系統，內容為一序列傳輸介面處理電路(STI.v)。STI電路動作為從並列埠進行資料輸入處理後由序列埠將處理完成之資料以序列輸出。

本Lab電路中，共有六只信號輸入(clk、rst、load、pi\_data、pi\_msb、pi\_low)、二只信號輸出（so\_data, so\_valid）。請將STI處理電路設計於一設計檔(STI)內。相關信號說明，請參考表一。

**2. 設計規格**

**2.1 系統方塊圖**



圖一、系統架構圖

**2.2 輸出入訊號和記憶體描述**

表一、STI輸入/輸出信號

|  |  |  |  |
| --- | --- | --- | --- |
| **信號名稱** | **輸入/輸出** | **位元寬度** | **說明** |
| *clk* | input | 1 | 系統提供的時脈信號。 |
| *reset* | input | 1 | 高位準非同步(active high asynchronous)之系統重置信號。  說明：此信號於系統啟動時送出。 |
| *load* | input | 1 | 系統提供的讀取控制訊號。  說明：訊號寬度持續一個時脈週期。當load為High時且經時脈訊號正緣觸發時，表示並列資料輸入埠及序列控制訊號為有效。 |
| *pi\_data* | input | 16 | 十六位元並列資料輸入埠。 |
| *pi\_msb* | input | 1 | 序列輸出順序控制訊號。 |
| *pi\_low* | input | 1 | 序列低位元輸出致能訊號。 |
| *so\_data* | output | 1 | 序列資料輸出埠。 |
| *so\_valid* | output | 1 | 序列資料輸出致能訊號。  說明 : 當此信號為High時，表示so\_data傳輸的資料被認為是有效的。 |

**2.3系統功能描述**

本序列傳輸介面處理電路功能如下：

當reset結束後。每當load輸入為High且經時脈訊號正緣觸發時，表示STI輸入訊號為有效，STI將依據控制訊號(pi\_msb、pi\_low)之設定將pi\_data輸入訊號進行相對應之並列轉序列資料處理，處理完成後將so\_valid拉成High表示有效資料輸出，並將處理完成之資料由so\_data依序送出。當load輸入為Low時，表示輸入資料無效，STI將不進行任何動作。

**以下文中MSB代表Most Significant Bit；LSB代表Least Significant Bit**。

並列轉序列資料處理規範如下:

1. 當pi\_msb輸入為High時，表示序列輸出由序列輸出緩衝資料的MSB開始；當pi\_msb設定為Low時，表示序列輸出由序列輸出緩衝資料的LSB開始。(詳細處理關係描述在2.3.1)
2. 當pi\_low設定為High時，表示序列輸出緩衝資料為pi\_data並列資料共16bits；當pi\_low設定為Low時，表示序列輸出緩衝資料為pi\_data並列資料的LSB的8bits。(詳細處理關係描述在2.3.2)

當序列輸出緩衝資料處理完畢後，將so\_valid拉為High並將序列資料由so\_data依序送出。

每筆pi\_data會輸入16bits的並列資料，而每筆so\_data輸出為8或16筆序列輸出，每筆pi\_data處理完成後，須將so\_valid設定為Low，之後testfixture將會自動把下一筆待處理資料輸入。

**2.3.1 最高位元優先功能描述（pi\_msb）**

當pi\_msb設定為High時，表示so\_data序列輸出由序列輸出緩衝資料的MSB開始，如圖二.（範例使用16 bits說明）所示，若pi\_msb為High時，其16 bits序列輸出緩衝資料由so\_data依序輸出0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0。



圖二、最高位元優先傳輸資料格式（pi\_msb=1）

當pi\_msb設定為Low時，表示so\_data序列輸出由序列輸出緩衝資料的LSB開始，如圖三.（範例使用16 bits說明）所示，若pi\_msb為Low時，其16 bits序列輸出緩衝資料由so\_data依序輸出0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0。



圖三、最高位元優先傳輸資料格式（pi\_msb=0）

**2.3.2 低位元組資料致能功能描述（pi\_low）**

當pi\_low設定為High時，表示序列輸出緩衝資料為pi\_data並列資料共16bits。如圖四.範例所示，pi\_data=16’h0402，pi\_low=1，則輸出pi\_data的16bits（8’h0402）



圖四、低位元組資料模式資料格式（pi\_low =1）

當pi\_low設定為Low時，表示序列輸出緩衝資料為pi\_data並列資料的低位元組共8bits。如圖五.範例所示，pi\_data=16’h0402，pi\_low=0，則輸出pi\_data的低位元組8bits（8’h02）。



圖八、低位元組資料模式資料格式（pi\_low =0）

**2.3.7時序規格圖**

圖十二.是系統初始時序圖，其中reset將維持至少一個tCYCLE的High，待reset設定為Low之後，testfixture將開始輸入並列資料，此時若load訊號為High且經clk的rising edge觸發則表示當下的控制指令為有效指令，STI電路須針對此有效指令進行並列轉序列處理，load訊號將只會維持一個tCYCLE為High。當testfixture偵測到so\_valid由High轉Low時，將會隨後送出下一筆並列輸入訊號；待最後一筆pi\_data輸入後，testfixture 隨即會將pi\_end設定為High表示不會再有資料輸入。



圖十二、主控端訊號傳輸時序圖

STI進行並列轉序列輸出時序可參考下列圖十三.所示。每筆pi\_data資料處理完成後利用so\_data序列輸出，並且把so\_valid設定為High，表示目前輸出為有效的。而testfixture將會在偵測到so\_valid為High且clk的falling edge時進行序列資料輸出比對。每一筆並列資料輸入處理後的序列輸出必須為連續完整輸出。



圖十三、序列訊號資料輸出時序圖

**3. 模擬**

Lab共提供一組測試樣本，可依下面範例來進行模擬:

 ncverilog 指令範例如下：

ncverilog testfixture.v STI.v +access+r



**4. 結果及驗證**

若模擬結果正確的話，則會出現笑臉圖示及文字說明。若有錯誤的話，則會出現其他圖示及說明，請依圖示說明修正您的code。