111-2 計算機組織 Midterm Project: ALU Design 報告

111 學年度第 2 學期

老師：朱守禮　教授
學生/組員：
資訊二甲 10727120 洪錦彤
電資二 11020107 蘇伯勳
電資二 11020137 關翔謙
電資二 11020140 葉柏榆

一、背景

此次 Midterm Project 之目的為實作算數邏輯單元、一階無號數乘法器、桶狀邏輯右移移位器、高低位暫存器及 MIPS 指令集中的”set on less than”指令。由於本組有三人都是電資系學生，以往的課程尚未提及 VHDL 此描述語言的語法、ModelSim 模擬工具及波型等，因此在剛開始接觸如此硬體思維的內容時，撰寫的思維上一時半刻無法轉換。在繳交日期將至前夕，才總算是將成果呈現了出來。

小組的分工為：洪錦彤負責使用 PPT 繪製架構設計圖；蘇伯勳負責將程式碼排版、加上註解、增加可讀性，手繪架構圖初稿及報告撰寫；關翔謙負責產生多樣測資，葉柏榆負責所有程式碼的撰寫、除錯及觀測總結波形。

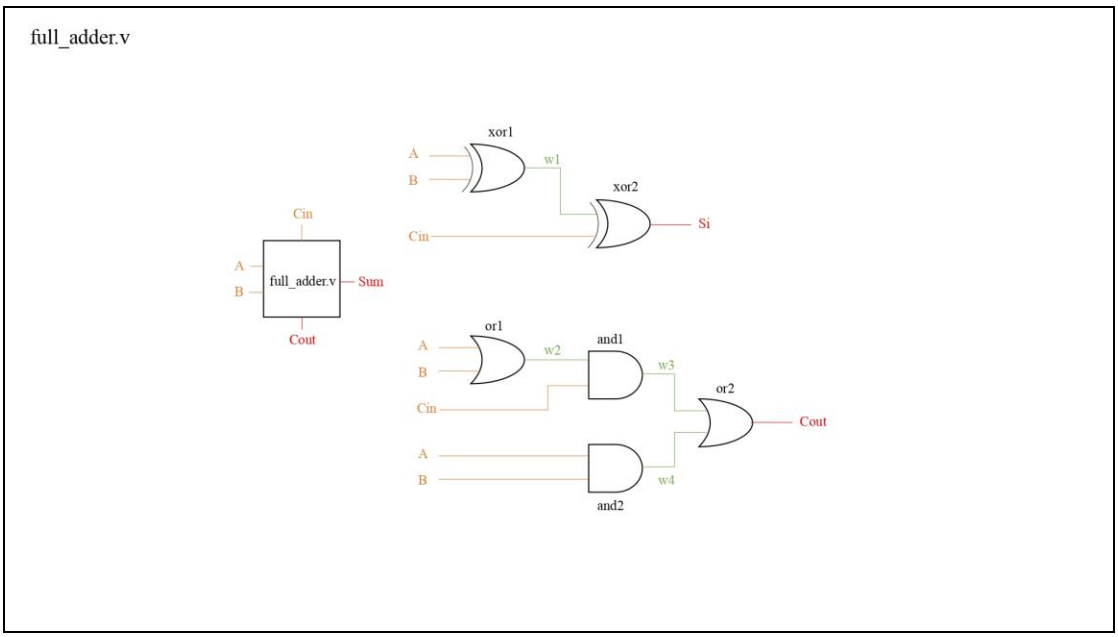
二、方法

本設計將各 module 分為六大類：ALU、乘法器、移位器、輸入輸出、總架構與測試，共計 14 個檔案，分為 12 個 module 與 2 個純文字檔。ALU 包含了”full_adder.v”、”ALU_1bit.v”、”ALU_1bit32th.v”、”ALU.v”，乘法器包含了”Multiplier.v”、”HiLo.v”，移位器包含了”mux_2to1.v”、”Shifter.v”，輸入輸出包含了”ALUControl.v”、”MUX.v”、總架構包含了”TotalALU.v”，測試檔包含了”tb_ALU.v”、”input.txt”、”ans.txt”。以下為分段說明：

Part 1. ALU

“full_adder.v”：

此一位元全加器採用 Ripple-Carry 模式，輸入進 A 與 B 兩個待加訊號與 Cin 紀錄進位，Sum 作為加總結果，Cout 為下一位元的進位輸入。在設計上按照”si = ai XOR bi XOR ci”與”ci+1 = ai * bi + ai * ci + bi * ci”兩公式，實例化了兩個 XOR 閘、2 個 AND 閘與 2 個 OR 閘，按照直式加法之計算邏輯撰寫。



“ALU_1bit.v”：

此一位元 ALU 在此 Project 中應用於 0~30 位元的 ALU。輸入進 A 與 B 兩個一位元訊號、Cin（一位元）代表進位、Less（一位元）用於 SLT 指令、Signal 為六位元，代表控制五種指令的訊號；輸出有 Sum（一位元）代表運算結果，與 Cout 為下一位元的進位輸入。

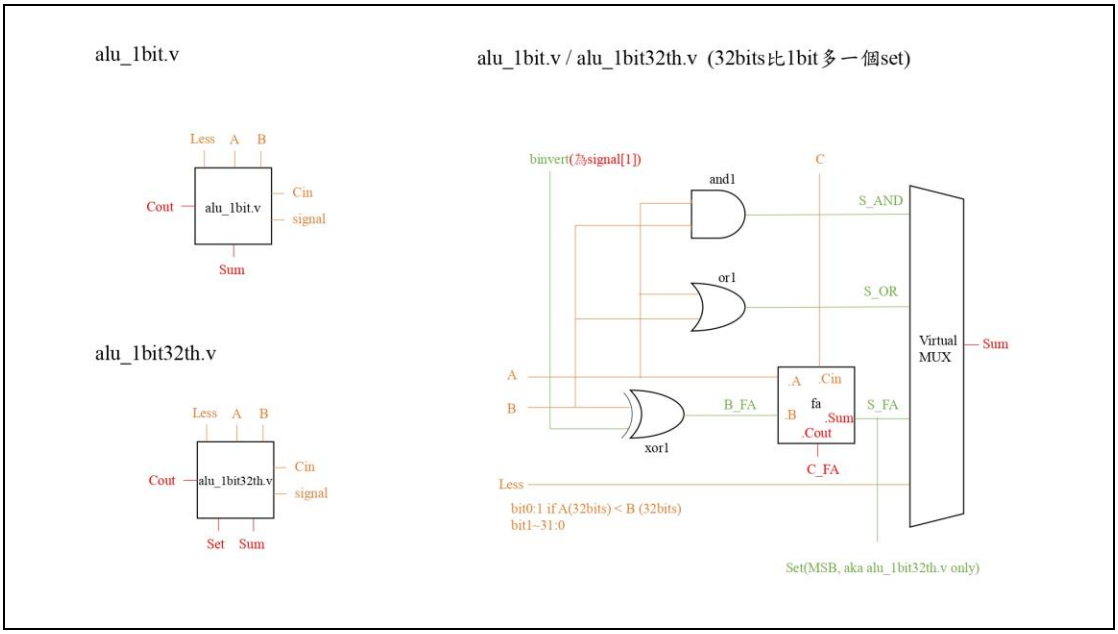
在此 module 中，有一條名為”binvert”的 wire，assign 為輸入控制訊號的第一位元。由於在 AND、OR、ADD、SUB 與 SLT 五種訊號中，SUB 與 SLT 實際上都做了減法運算，指令的第一位元都為 1。故 binvert 為 1 時表示減法，為 0 時表示非減法的其餘三種運算。由於使用二補數系統，在進入全加器之前，輸入訊號 B 與 binvert 透過一個 XOR 閘判斷要做加法還是減法，XOR 運算後的結果用 B_FA 這條 wire 接上。

依照所學，我們實例化了一個 AND 閘、一個 OR 閘、一個 XOR 閘，與一個先前撰寫的漣波進位全加器。AND 閘用於”AND”指令，”OR”閘用於”OR”指令，XOR 用於判斷加減運算，全加器用於進行加減運算。

最後，我們選擇使用兩個三元運算處理下一位元的進位輸入(Cout)與運算結果輸出(Sum)。由於兩種邏輯運算的 Cout 皆為 0，因此若為邏輯運算則將 Cout 賦值為 0，若非則為全加器之輸出(C_FA)；而在最後的運算結果，由三元運算達到「虛擬」的多工器之效果。首先判斷控制訊號的第三位，因為只有”SLT" 此指令的第三位元為 1；再來看第二位元，因為若為邏輯運算則控制訊號的第二位元為 1，並且若第零位元為 1 則是 OR，為 0 則是 AND。最後就剩下全加器的結果。由於先前使用 XOR 判斷做加法還是減法，因此這邊無須再次判斷，直接 assign 全加器的輸出結果即可。

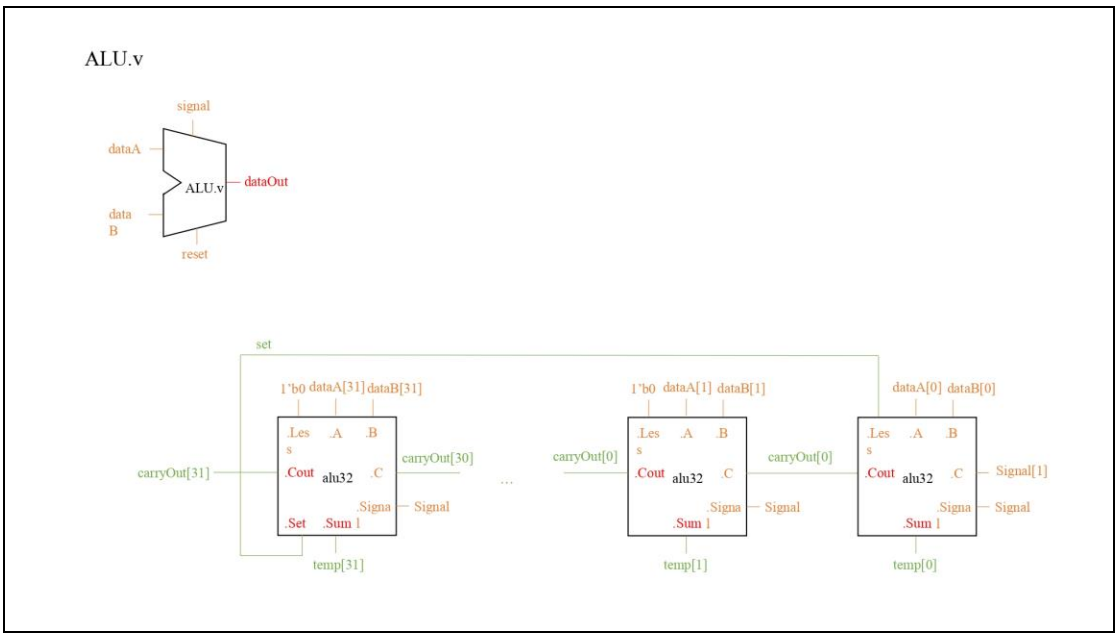
“ALU_1bit32th.v”：

此 module 僅能用於最高位元（第三十二位元，MSB），與上述”ALU_1bit.v”之差異僅在於多了一條名為”Set”的 wire 從全加器的輸出結果(S_FA)牽出。其目的是為了進行 SLT 運算。由於 SLT 的行為是「若第一個訊號較第二個訊號小，則暫存器的值設為 1，反之設為 0」，因此將最高位元的全加器運算結果（下稱 S_FA）牽至最低位元的 Less，在透過所謂的「虛擬多工器」（即三元運算）判斷。由於判斷第一個訊號（下稱 sigA）小於第二個訊號（下稱 sigB）的方法最簡單的方式就是相減，因此若 sigA<sigB，則 Set 為 1，牽到最低位(LSB)的 Less，否則 Set 為 0，一樣牽到 LSB 的 Less。



“ALU.v”：

此 module 為 32 顆 1-bit ALU 串接的總合。由於限制了必須以組合邏輯撰寫，因此實例化並串接了 32 顆 1-bit ALU。在前 31 位元使用”ALU_1bit.v”實例化，僅最後一位元使用”ALU_1bit32th.v”，因為只有 MSB 才能有 Set 這個 output port。如同上述，MSB 的 Set 接到 LSB 的 Less，而第一位元到第 31 位元的 Less 都為 0，因為 SLT 只會用到 0 或 1。特別提到的是，bit invert 我們沒有再寫一個 input port，因為只要看控制訊號的第一位元就能判斷了，所以 bit invert 是做在內部的，不用多寫一個輸入埠。

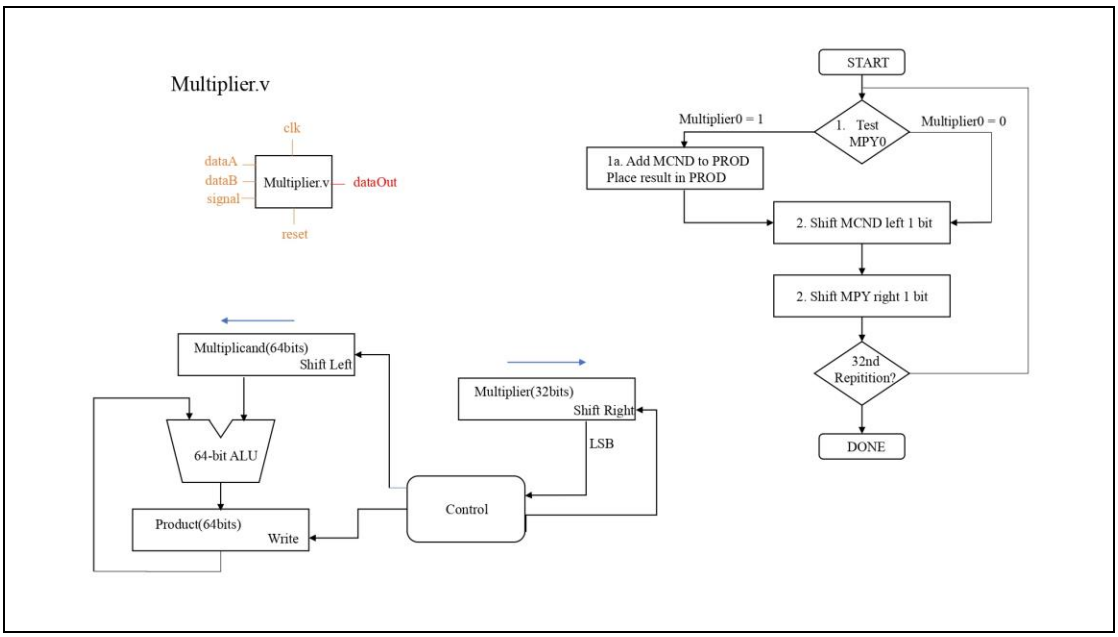


Part 2. 乘法器

“Multiplier.v”：

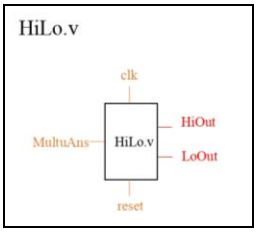
本次乘法器為一階無號數循序乘法器。Timescale 設為 1ns/1ns，輸入為被乘數 dataA（32 位元）、乘數 dataB（32 位元）、MULTU 指令 Signal（6 位元）、同步訊號 clk（一位元）與重置訊號 reset（一位元），輸出為乘積 dataOut（64 位元）。採用循序邏輯，因此需要與訊號同步，使用三個 always block 實現。

第一個 always block 用於重置訊號，只要當 reset 有變動時就會觸發，將暫存器與計數器歸零；第二個 always block 用於初始化，當 Signal 有變動時就會觸發，且若為 MULTU 指令時做初始化動作，幫傳入 A 訊號的暫存器高 32 位元補 0，B 訊號的暫存器歸零，其餘暫存器與計數器也歸零；第三個 always block 是核心的判斷所在，使用同步訊號 clk 的正緣觸發。首先判斷計數器是否 <33，若是則判斷乘數的最低位，若為 1 則將被乘數加到乘積暫存器中，0 則繼續做下一步。接著，左移被乘數一位元，右移乘數一位元。最後，計數器加一。在整個 module 的末端，將乘積暫存器 temp 這條 wire 連到 dataOut 作為最終 64 位元乘積輸出。整體架構按照直式乘法的運算邏輯撰寫。



“HiLo.v”：

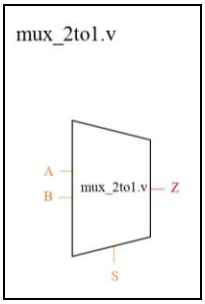
此 HiLo 暫存器將輸入的 64 位元訊號分為高 32 位元與低 32 位元輸出，輸入有乘積 MultuAns（64 位元）、同步訊號 clk 與重置訊號 reset，輸出有 HiOut 與 LoOut 各 32 位元。並且由於是基於循序邏輯撰寫，因此使用一個 always block，觸發條件為 clk 的正緣觸發或 reset 訊號。若是 reset 則將乘積暫存器歸 0，否則將乘積 MultuAns 暫存到暫存器中。在 module 的最後，將暫存器的高 32 位元 assign 給 HiOut，低 32 位元 assign 給 LoOut。



Part 3. 移位器

“mux_2to1.v”：

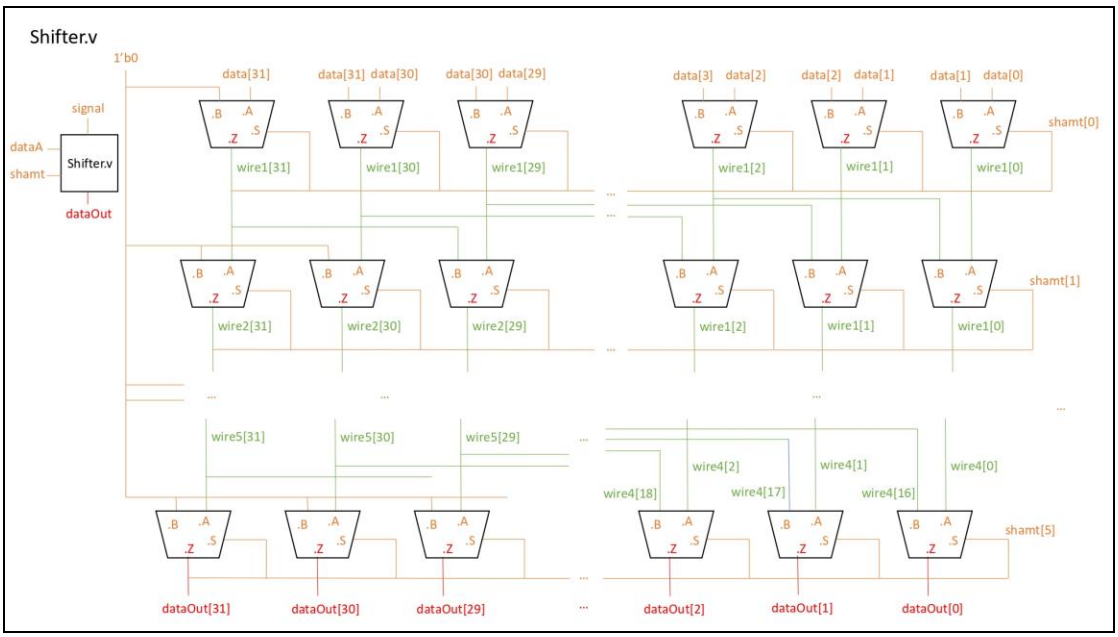
在整個桶式移位器中，使用了大量的二對一多工器組成整體，此 module 即為二對一多工器。輸入為訊號 A、訊號 B 與控制訊號 S（各為一位元），輸出為 Z（一位元）。根據真值表，我們可以總結出：當 S=0 時，Z=A，輸出 A 訊號；當 S=1 時，Z=B，輸出 B 訊號。因此這邊使用三元運算進行判斷。



“Shifter.v”：

此為邏輯右移的 32 位元桶式移位器。輸入為要移位的訊號 dataA（32 位元）、移位量 shamt（32 位元）、控制訊號 Signal（六位元）、重置訊號 reset（一位元），輸出為移位結果 dataOut（32 位元）。在 R-type 架構中，移位量為五位元，因此只會用到 shamt 的低 5 位元。Signal 則是完全用不到，傳入是為了在除錯時方便。

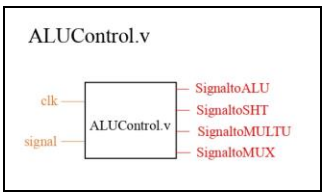
由於要移位 32 位元的訊號，並且移位量為五位元，因此總共會有 32*5=160 個二對一多工器排成五列，一列有 32 個二對一多工器，每列最右側各為移位量的 0～4 位元，五列 32 位元多工器之間由 4 個 32 位元的 wire 連接（wire1～wire4），以達到 128 條 wire 的效果。第一列使用的移位量為 shamt 的第 0 位元，輸出至 wire1 的 0～31 位元（虛擬的 32 條 wire，實際上是一條 32 位元的 wire）。第一個多工器的訊號 A 為 dataA 的第 0 位元，訊號 B 為 dataA 的第一位元，S 為 shamt 的第 0 位元，Z 為 wire1 的第 0 位元。第二個多工器使用 dataA 的第一、二位元，以此類推。直到最後一個多工器，訊號 A 為 dataA 的第 31 位元(MSB)，訊號 B 由於使用移位量的第 0 位元，因此傳入 0。第一列會有 2**0=1 個 0 被使用，從最高位元往低位元塞。第二列會有 2**1=2 個 0 從第 32 與第 31 個多工器的 B port 傳入，第三列會有 2**2=4 個 0 從 32～29 個多工器的 B port 傳入，第四列會有 2**3=8 個 0 從 32～25 個多工器的 B port 傳入，第五列會有 2**5=16 個 0 從 32～17 個多工器的 B port 傳入，共計 1+2+4+8+16=128 個 0。第二列的第一個多工器的 A port 為 wire1 的第 0 位元，B port 為 wire1 的第 1 位元；第二個多工器的 A port 為 wire1 的第一位元，B port 為 wire1 的第二位元，以此類推。到了第 31 與 32 個多工器，A port 分別為 wire1 的第 30、31 位元，B port 皆為 0，以此類推後面三列。最後一列的的輸出為 dataOut 的 0～31 位元做為移位的最終結果。



Part 4. 輸入輸出

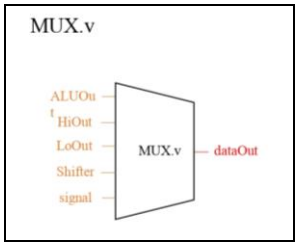
“ALUControl.v”：

此 module 的目的為將一開始傳入的控制訊號判斷要做何種功能，並將訊號分配給 ALU、移位器、乘法器與多工器。輸入為控制訊號 Signal（32 位元）、同步訊號 clk（一位元），輸入為給 ALU 的 SignaltoALU、給移位器的 SignaltoSHT、給乘法器的 SignaltoMULTU、給多工器的 SignaltoMUX（皆為 32 位元）。在第一個 always block 中，當有訊號且為 MULTU 指令時，計數器歸零；第二個 always block 中，條件為同步訊號 clk 正緣觸發。先將控制訊號（下稱 Signal）放置暫存器（下稱 temp），再判斷 Signal 是否為 MULTU，是則將計數器加一。接著判斷計數器，若其為 33 則將 temp 設為 111111 以在後續不進入”Signal == MULTU”這個 if-else block，並且將計數器歸零。在整個 module 的最後，將 temp 賦值給 SignaltoALU、SignaltoSHT、SignaltoMULTU、SignaltoMUX。



“MUX.v”：

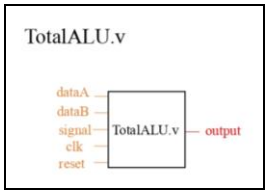
此 module 為四對一多工器。輸入為 ALUOut、HiOut、LoOut、Shifter 與控制訊號 Signal，輸出為 dataOut。除了控制訊號為六位元，其餘全為 32 位元。我們使用三元運算判斷，控制訊號（下稱 Signal）的第五位為 1 代表 ALU 的運算，接著看 Signal 的第四位元，為一則判斷第一位元（1 為 Lo，0 為 Hi）；最後 default 設為 Shifter。

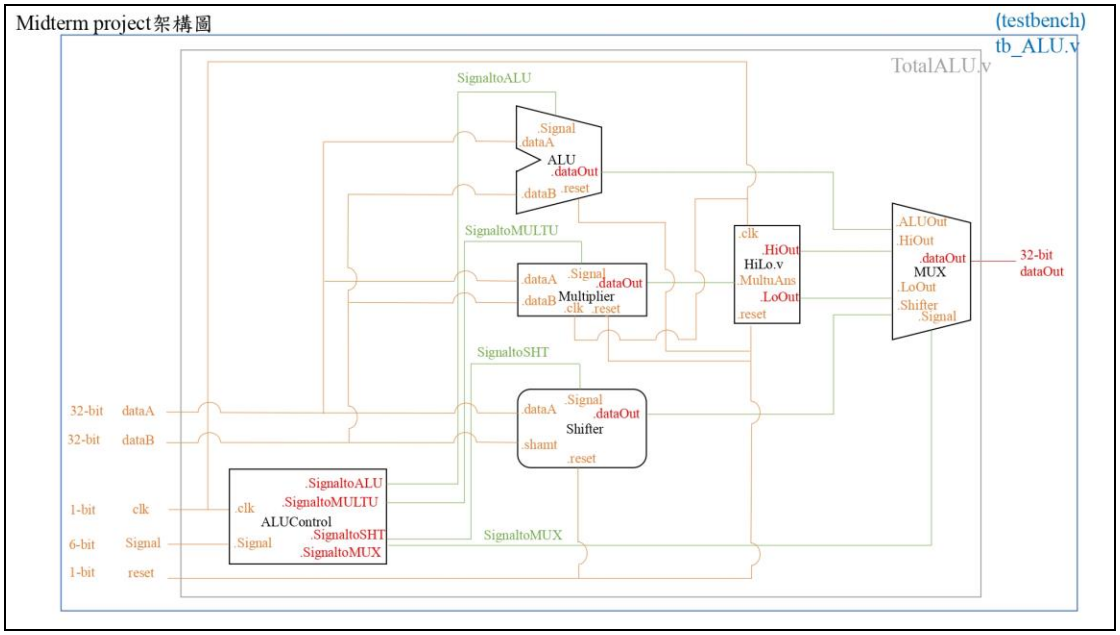


Part 5. 總架構

“TotalALU.v”：

這個 module 為上述功能之集大成。輸入為 dataA（32 位元）、dataB（32 位元）、控制訊號 Signal（6 位元）、同步訊號 clk（一位元）、重置訊號 reset（一位元），輸出為 Output（32 位元）。內部的接線有：五位元的 SignaltoALU、SignaltoSHT、SignaltoMULTU、SignaltoMUX，32 位元的 ALUOut、HiOut、LoOut、ShifterOut、dataOut，與 64 位元的 MultuAns。先實例化 ALUControl 分配訊號，輸入 Signal 與 clk，輸出到 SignaltoALU、SignaltoSHT、SignaltoMULTU、SignaltoMUX。再來實例化 ALU，輸入 dataA、dataB、SignaltoALU、reset，輸出到 ALUOut。接著實例化 Multiplier，輸入 dataA、dataB、SignaltoMULTU、clk、reset，輸出到 MultuAns。再實例化 HiLo，輸入 MultuAns、clk、reset，輸出到 Hiout、LoOut。下一個實例化 Shifter，輸入到 dataA、dataB、SignaltoSHT、reset，輸出到 ShifterOut。最後實例化 MUX，輸入 ALUOut、HiOut、LoOut、ShifterOut、SignaltoMUX，輸出到 dataOut。最後將 dataOut 接到 Output，做為結果。





Part 6. 測試

“tb_ALU.v”：

此 module 為 testbench，在一開始使用 fopen，預設讀入路徑為 "C:/ CO_Midterm_二甲_第十一組_11020107_蘇伯勳/alu/input.txt" 與 "C:/ CO_Midterm_二甲_第十一組_11020107_蘇伯勳/alu/ans.txt"。

請自行手動更改，路徑中不能有中文，因此無法產生 work 資料夾。請在改正路徑後重新產生。

“input.txt”為輸入，格式為：<指令> <資料 A> <資料 B>。

“ans.txt”為正確答案，一行一個 10 進制數字。若為乘法(MULTU)，則先輸出 Hi 暫存器之值，再輸出 Lo 暫存器之值；其它指令則都只有一個答案。

如：

input　　ans

→2 3 15　→0

→25 2 3　→0

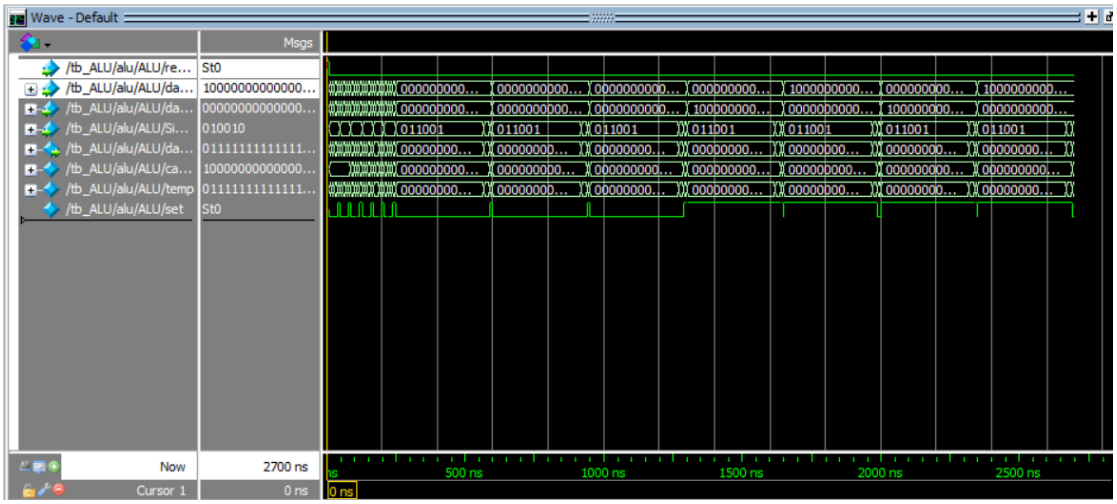
//…　　→6

　　//…

三、結果（此處僅顯示與訊號同步之 module 波形。）

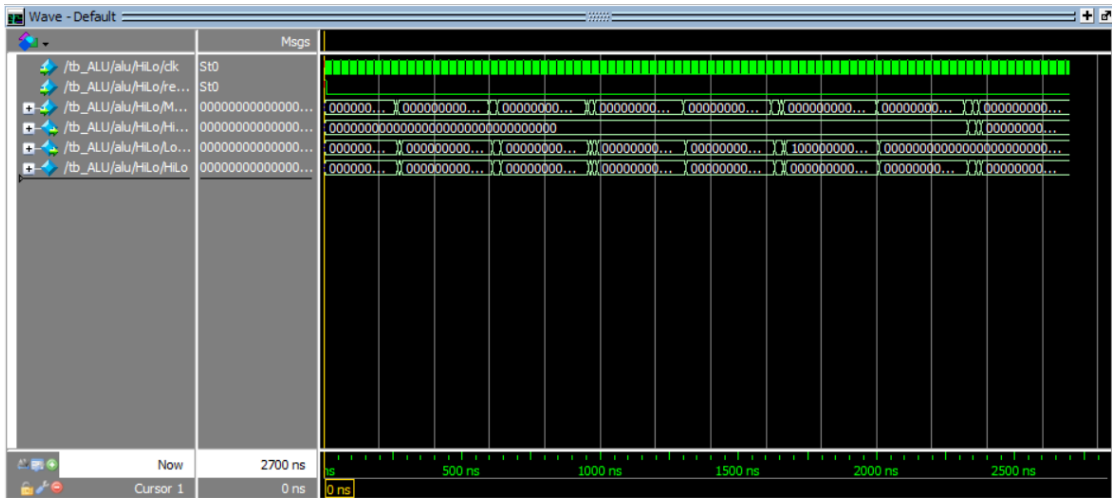
ALUControl：

ALU



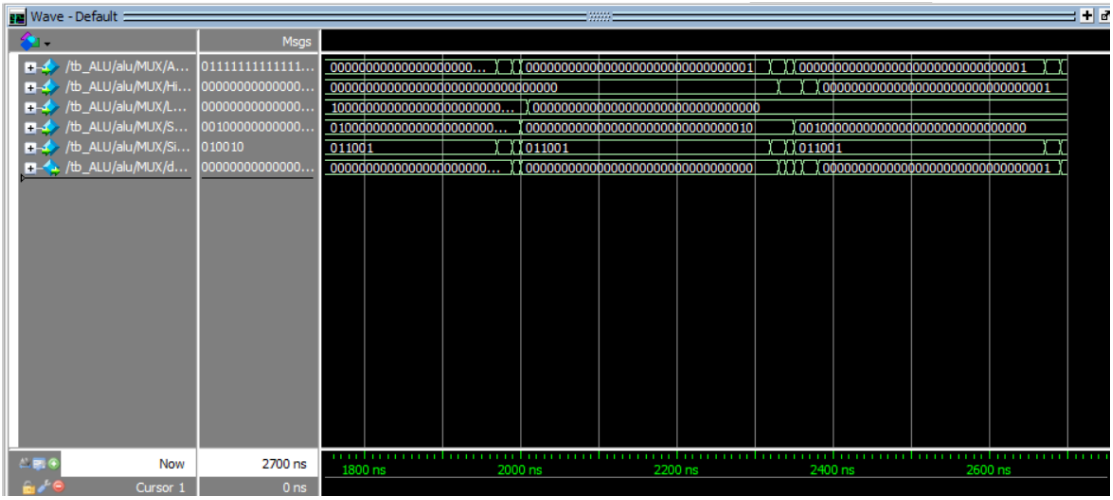
Multiplier

HiLo



Shifter

MUX



四、討論

為了符合講義與教學所教授的流程與演算法，雖然有很多種寫法能簡化或是更加直觀，但最終經過多次修改，才呈現了目前的最終版本。例如，ALU 與 Barrel Shifter 可以用 generate 實例化，MUX 可用 always block…等等。但根據要求，最終還是一個一個刻出來，因為這樣最能了解各種走線與暫存器。並且，當初在乘法器的除錯時，對著波形與程式碼交互除錯，既新鮮又困難。要算好計數器的時間，對著波形圖，才能找出條件要怎麼設定。尤其是在 Barrel Shifter，對著架構圖寫就不會出錯，但前提時不能眼花。

五、結論

對於硬體設計的初學者，要在兩週之內無中生有確實有些困難。因為不能使用寫軟體的思維，例如 ALU 中 MSB 的 Set 接到 LSB 的 Less，在硬體中直接在

LSB 接上接線就完事了，但用軟體思維撰寫，則會想說程式碼應該是循序的，到底要怎麼從 MSB 往回到 LSB 去？並且，在軟體開發時習慣使用指令與終端機，但在硬體必須要觀察波形除錯。原本以為一個 module 就像一個 C++的 library，直接 include 進來就能用了，但硬體還有「實例化」這一步。對於這些新的事物，在要不要向硬體設計發展的未來職涯有很大的幫助。

六、未來展望

透過此 Midterm Project 習慣了硬體設計的思維，希望在撰寫 Final Projec 時能夠加快小 module 的撰寫速度，並且釐清接線，試著不要以撰寫軟體演算法與變數宣告等的思維來撰寫執行流程與接線及暫存器。此外，若是有時間，也希望能夠試著撰寫第二階與第三階的乘法器及第一至三階除法器，訓練對硬體及底層運算的熟悉度。最後，希望能夠以 Final Project 所撰寫的 CPU 為基礎，增加功能，了解常用的各種指令架構。