

## 一、 背景

大二下學期接觸到了”計算機組織”這門課，課堂中了解了硬體和晶片的運作及各元件的運作原理，期中考後進而開始接觸這支 Project。

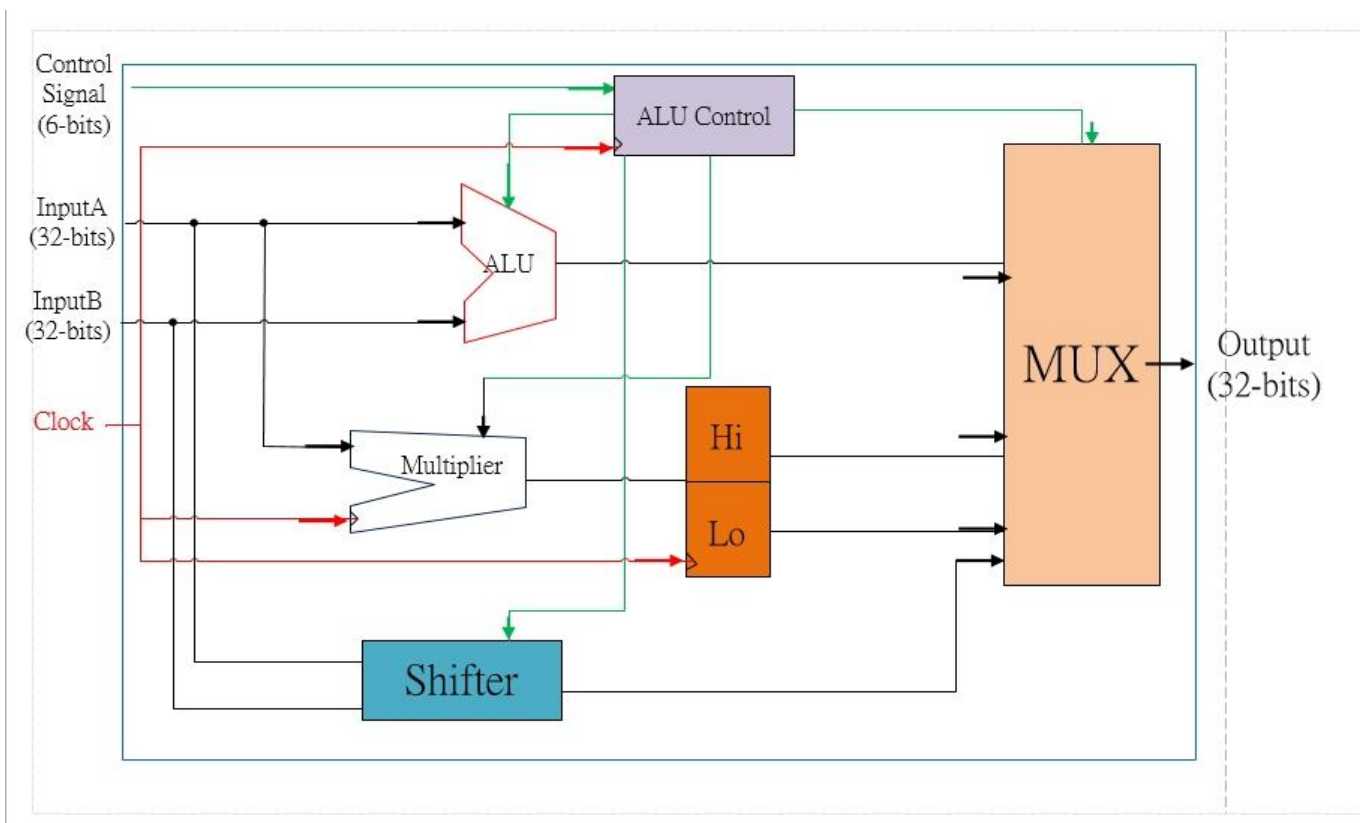
參考資料:

講義和 verilog 書本

以這些為基礎來進行 ALU、乘法器、Shifters、HiLo Reg、Mux、ALU Control 之開發。

## 二、 方法

ALU total



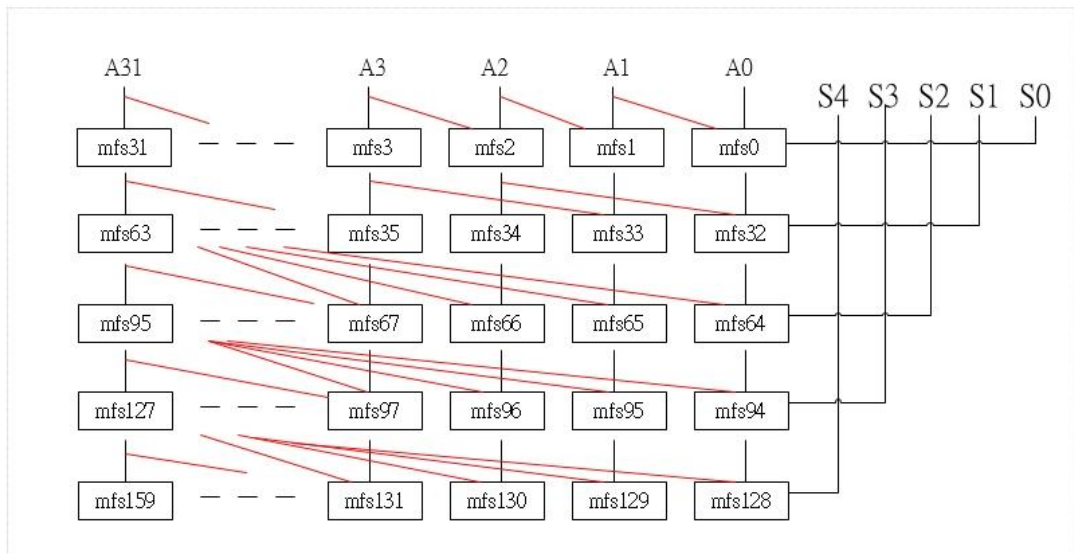
The diagram illustrates a 1-bit ALU circuit. It features several inputs:  $A_i$ ,  $B_i$ ,  $C_i$ , and a control signal  $BINVERTB$ . The circuit includes an XOR gate that takes  $A_i$  and  $B_i$  as inputs, producing the next carry-out  $C_{i+1}$ . An AND gate takes  $C_i$  and the output of the XOR gate as inputs. An OR gate takes  $C_i$  and the output of the XOR gate as inputs. The outputs of the AND and OR gates are connected to a multiplexer (MUX) labeled "MUX FOR ALU 1 BIT". The MUX also receives a "Signal" input and a "Less" control signal. The output of the MUX is the ALU result. A "Set" control signal is also shown, which is connected to the MUX.

The diagram illustrates a 32-bit ALU architecture composed of three ALU units: ALU31, ALU1, and ALU0. Each unit is represented by a blue box with various inputs and outputs.

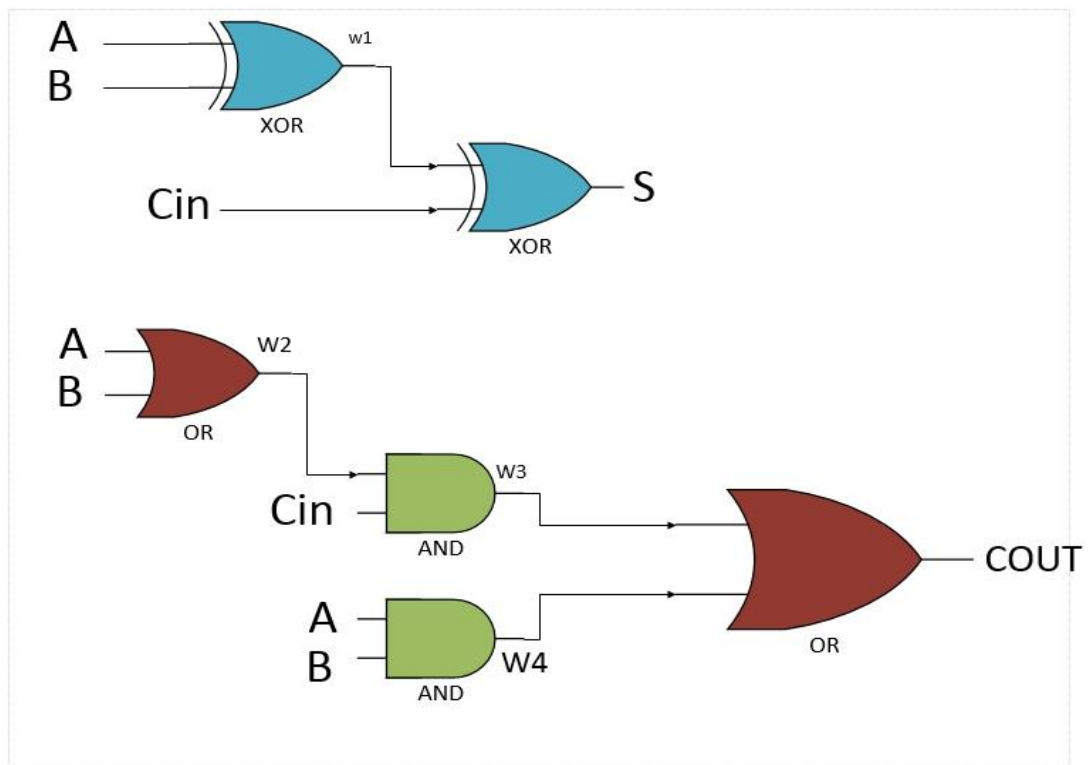
- ALU31:**
  - Inputs: 0, A31, B31, Less, Ai, Bi, Ci+1, Ci, SEL, INV, Set, Si.
  - Output: S31.
- ALU1:**
  - Inputs: A1, B1, Ci+1, Ci, SEL, INV, Si.
  - Output: S1.
- ALU0:**
  - Inputs: Less, Ai, Bi, Ci+1, Ci, SEL, INV, Si.
  - Output: S0.

Control signals and data flow are indicated by arrows:

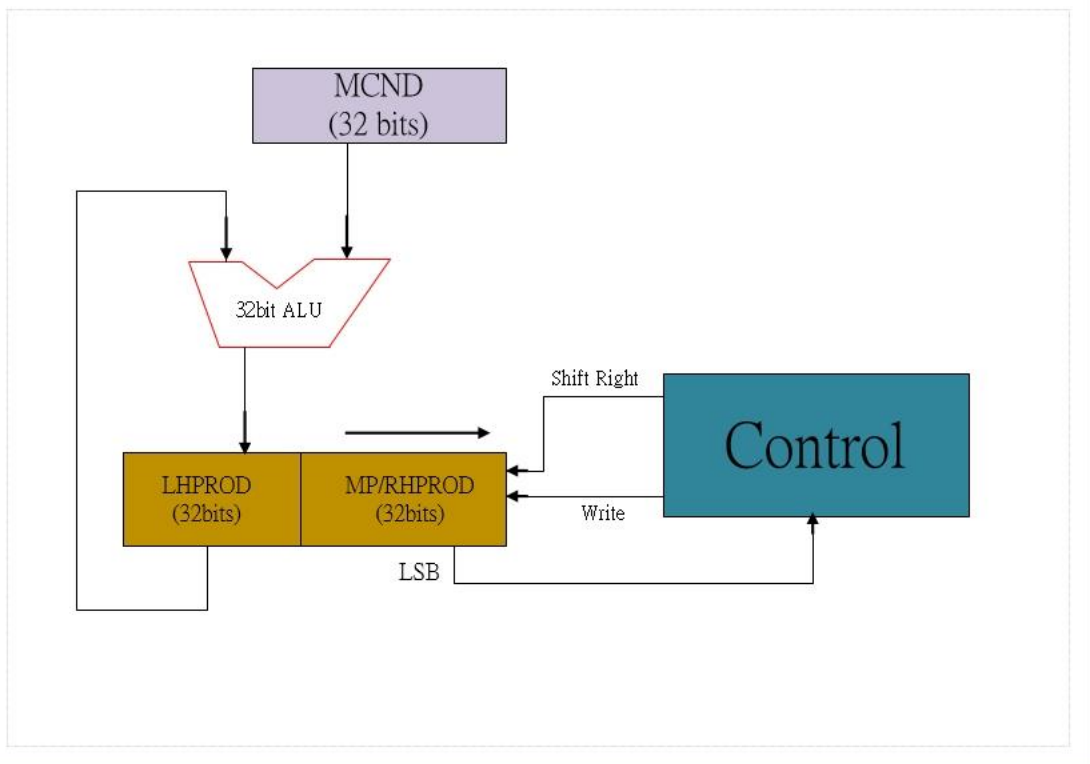
- Operation:** A signal that branches to the 'Less' input of ALU31 and the 'Less' input of ALU0.
- Carry In:** A signal that branches to the 'Ci+1' input of ALU31 and the 'Ci+1' input of ALU0.
- Binvert:** A signal that branches to the 'INV' input of ALU31 and the 'INV' input of ALU0.
- Data Flow:**
  - The output S31 of ALU31 is connected to the 'A1' input of ALU1.
  - The output S1 of ALU1 is connected to the 'A0' input of ALU0.
  - The output S0 of ALU0 is connected to the 'A31' input of ALU31.



**FA**



## Multiplier



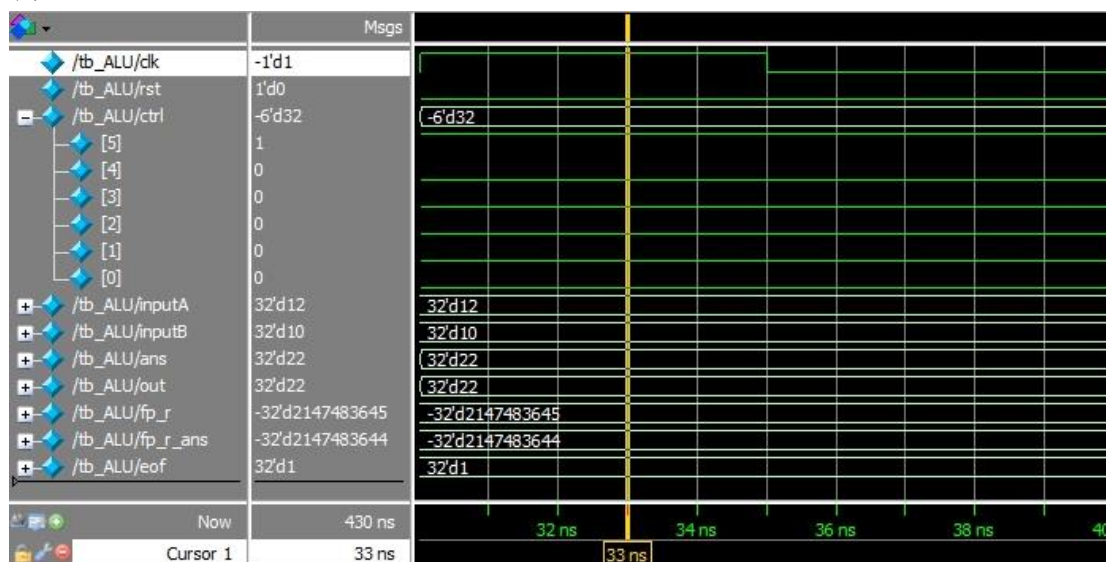
1. ALU :  
此 ALU 有以下功能:32-bits AND, OR, ADD, SUB, SLT , 以 Full-Adder 做起並以 Ripple-Carry 的進位方式連接 32 個 1-Bit ALU Bit Slice , 使之變為 32-bits ALU 。
2. Multiplier:  
使用第三版乘法器來設計, 先將乘數放入 PROD 右半部, 然後檢查 PROD 最右邊位數是否為 0, 如果為 0 則直接將 PROD 向右位移 1bit , 反之則被乘數加 PROD 左半部並放入左半部, 然後 PROD 再右位移 1bit , 之後檢查是否重負 32 次, 最後再進行一次將值傳送給 HiLo Reg. 。
3. Shifters :  
以 Data Flow Modeling、組合邏輯的方式設計之 32 bits 的 Barrel Shifter 。  
以 2 對 1 多工器(Mux for Shifter)來組合  $2^5$  個 Gate 進行移位。
4. HiLo Reg. :  
在乘法器計算完畢之後, 將結果儲存到兩個 32 bits 暫存器。
5. Mux :  
以 Data Flow Modeling、組合邏輯的方式設計, 整合四組訊號後則一輸出。
6. ALU Control :  
根據輸入的 6 bits 訊號, 決定要執行哪個運算, 並將 6 bits 的訊號解析成兩組 2 bits 訊號( SignalToALU\_Operation<ALU 功能限定>、SignalToMUX\_Operation ) 及一組 1 bit 訊號 SignalToALU\_Binvert <ALU 功能限定> 。
7. Testbench :  
以讀檔將測試資料讀入, 驗證各項 module 之功能正確性。
8. TotalALU :  
將各項 module 建立並執行。

### 三、結果

#### 1. 加法

輸入:12, 10

輸出:22

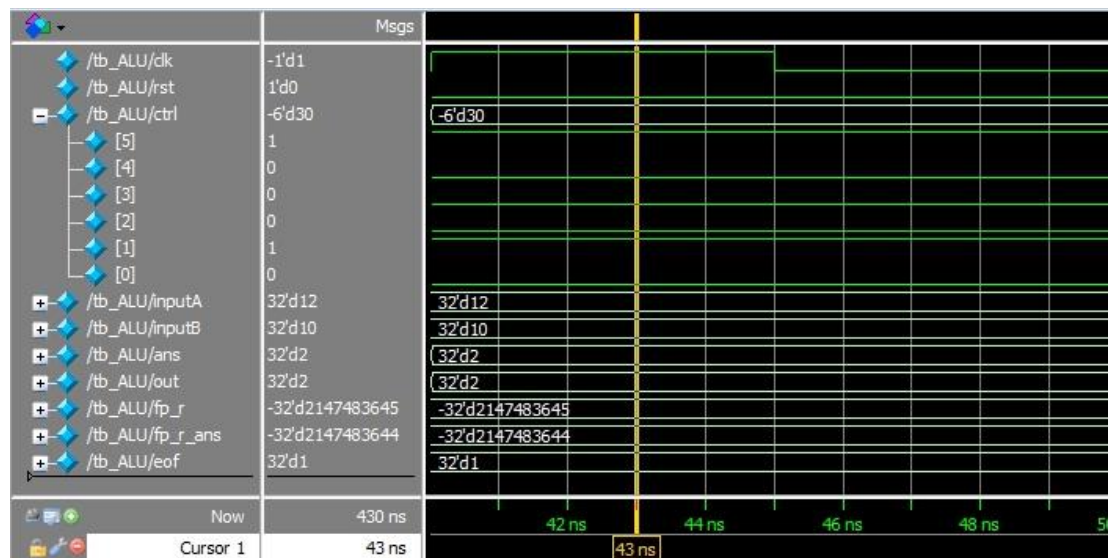


驗證:

當 ctrl 訊號為 32(ADD), 執行加法。

輸入 A 為 12、B 為 10 而結果 ans 輸出值為 22，故執行正確。

#### 2. 減法



輸入:12,10

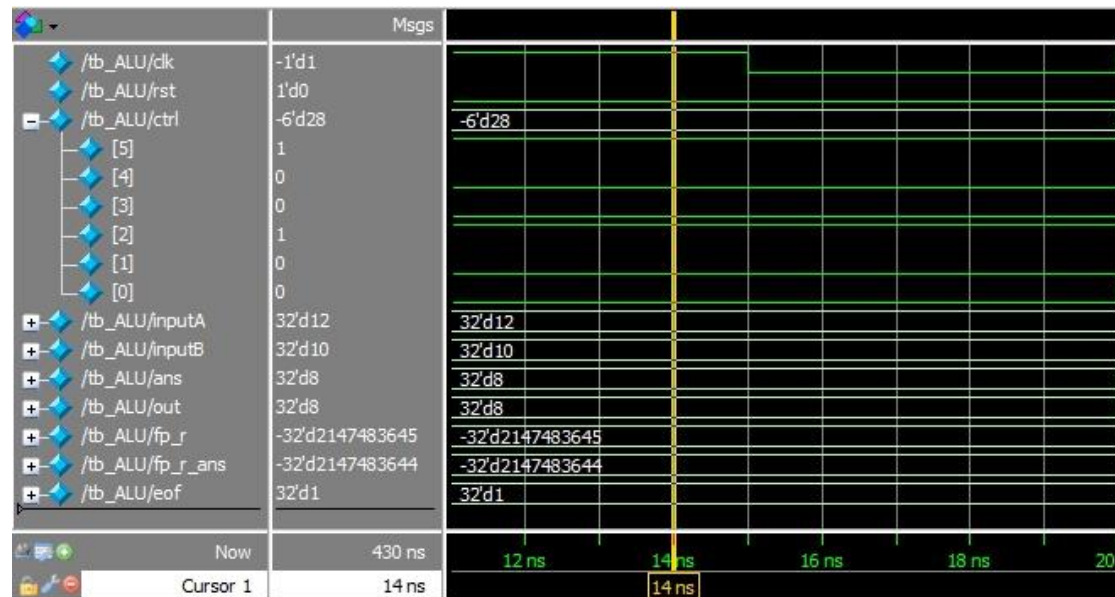
輸出:2

驗證:

當輸入訊號為 34(減法), ALU 執行減法動作。

理論結果應該是  $12 - 10 = 2$ ，而實際檢查 ans 值確實是 2，故結果正確。

### 3. AND



輸入:12, 10

輸出:8

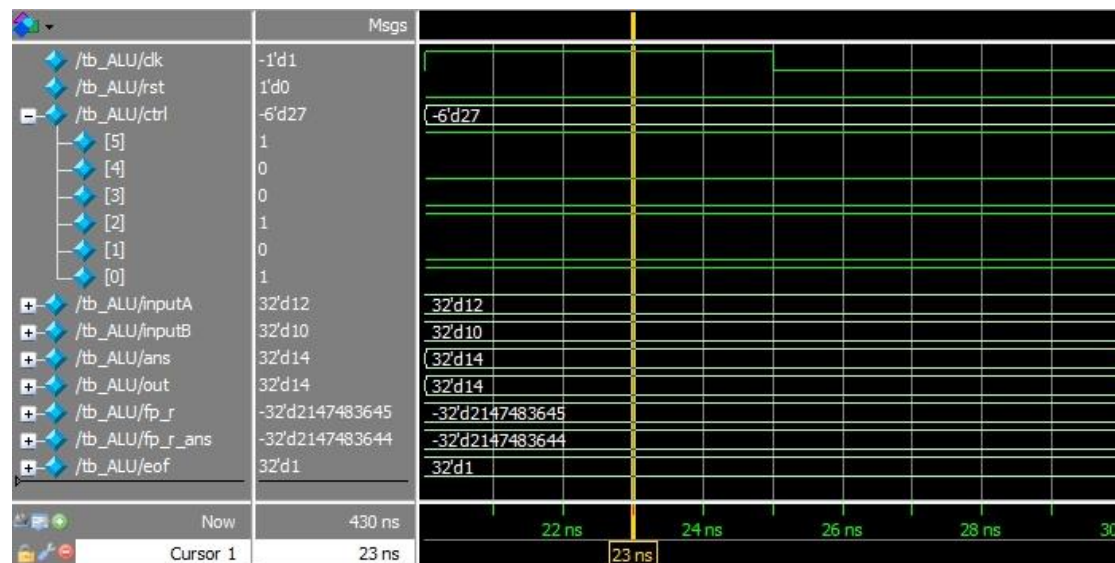
驗證:觀察 ctrl 訊號為 32，表示 AND 操作。

12 的 2 進位表示為 1100，10 的 2 進位表示為 1010。

將每一個 bit 各取 AND 結果為 1000，答案為 8。

觀察 and 訊號結果為 8，故執行正確。

### 4. OR



輸入:12, 10

輸出:14

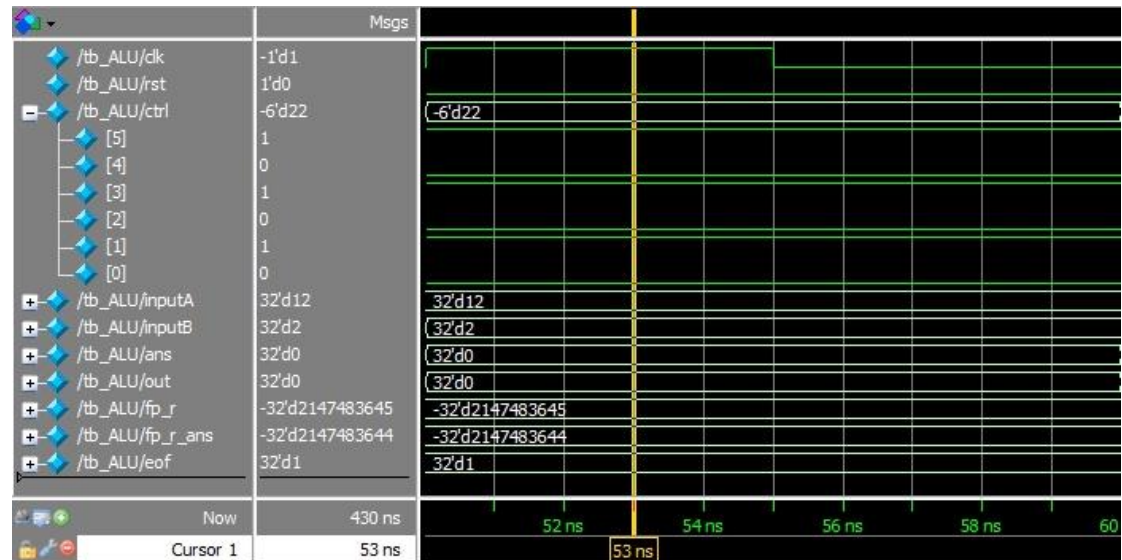
驗證：觀察 ctrl 訊號為 37，執行 OR 運算。

12 的 2 進位表示為 1100，10 的 2 進位表示為 1010。

將每一個 bit 取 OR 結果為 1110，答案為 14。

觀察 ans 訊號為 14，故結果正確。

## 5. SLT



輸入:12, 2

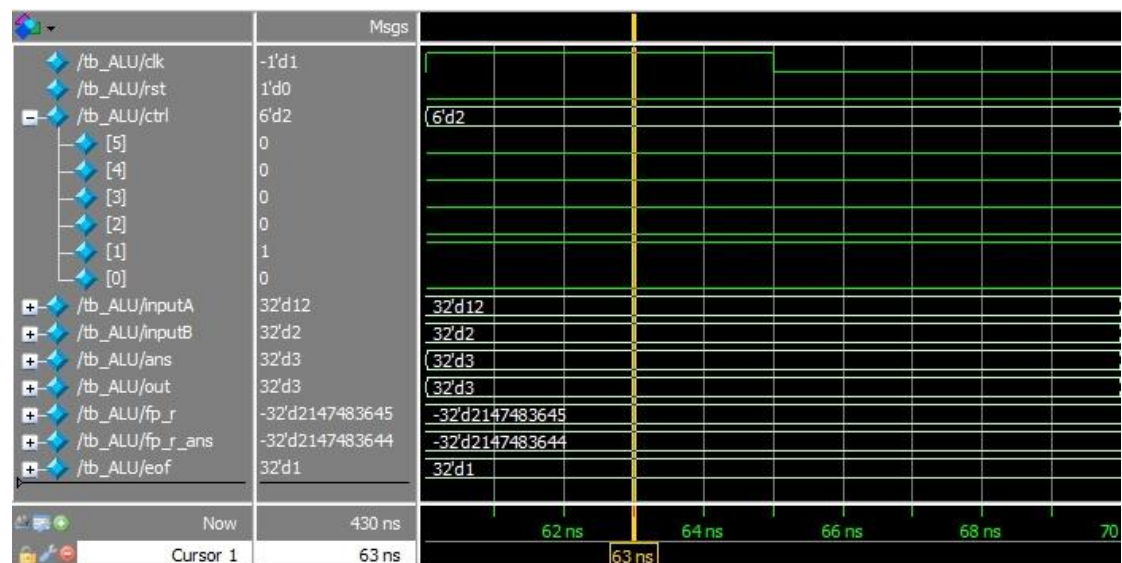
輸出:0

驗證：觀察 ctrl 訊號為 42，執行 SLT 運算。

輸入為 12、2，12 小於 2 理論輸出應為 0。

觀察 ans 訊號確實為 0，故執行結果正確。

## 6. SRL





輸入:12,2

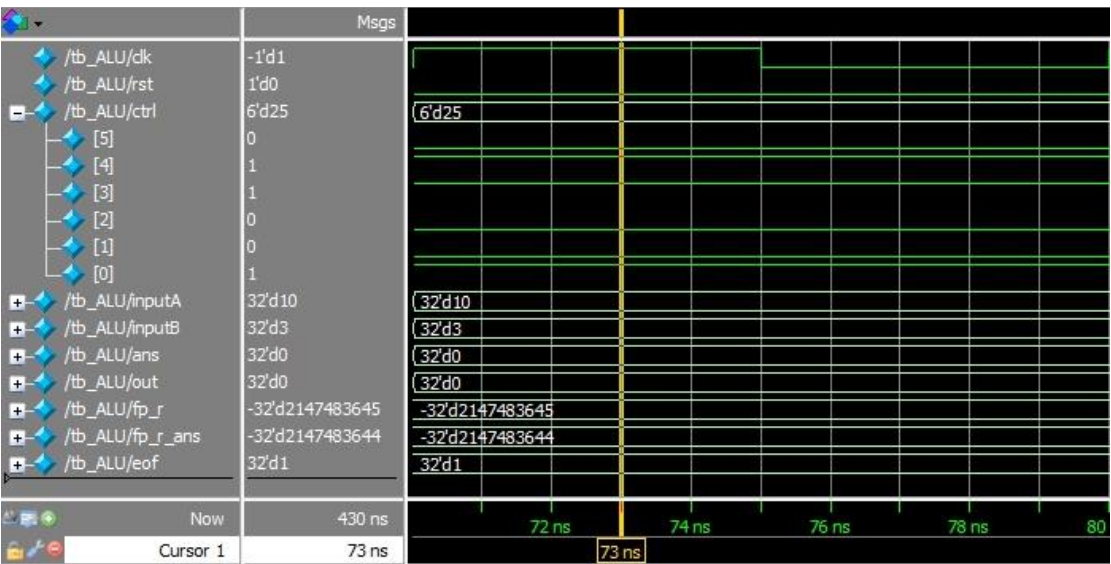
輸出:3

驗證: 觀察 ctrl 訊號為 2，執行 SRL 運算。

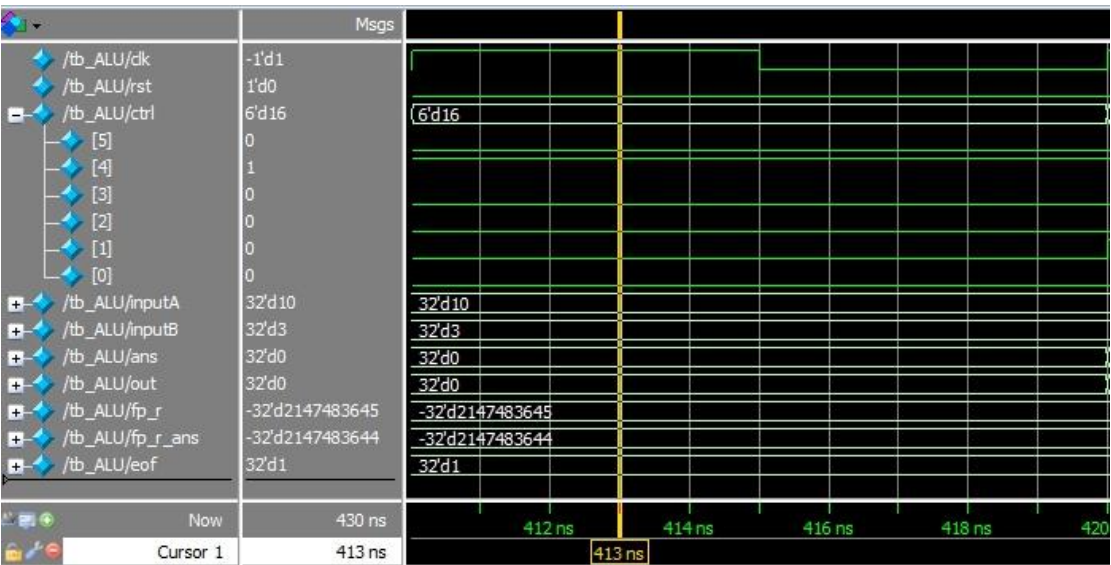
輸入 12、2，執行邏輯右移理論答案應為 3。

觀察 ANS 訊號確實為 3，故結果正確。

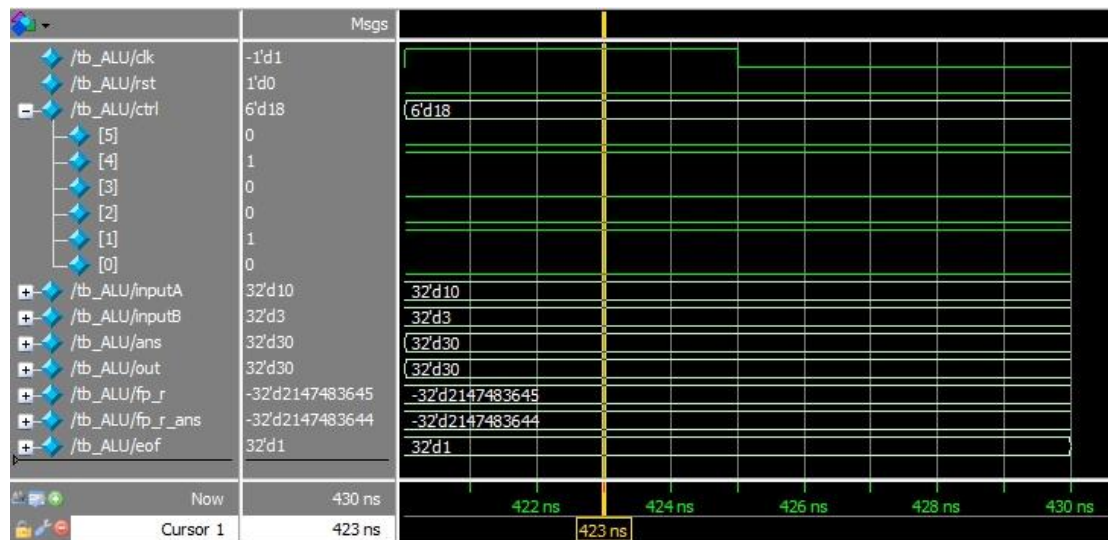
7. MULTU



Hi



Lo



輸入:10,3

輸出:30

驗證:先觀察 ctrl 訊號，一開始為 25，執行乘法運算。

之後乘法運算結束後將結果放入 HiLo 暫存器內。

之後，ctrl 訊號分別為 16(Move from Hi)與 18(Move from Lo)所以分別取出 HiLo 暫存器的值。

觀察 HiLo 的 ans 訊號分別為 0 與 30。理論上 Lo 暫存器應存放乘法結果，比對輸出發現相符，故執行正確。