**Digital System Design (Fall 2019)**

**Final Report of Term Project**

Group Number (組別) : 02

Group Member 1 (組員1) : Student ID 0613246 Name 李泓賢

Contribution (貢獻度) 45%

Group Member 2 (組員2) : Student ID 0611201 Name 陳彰浩

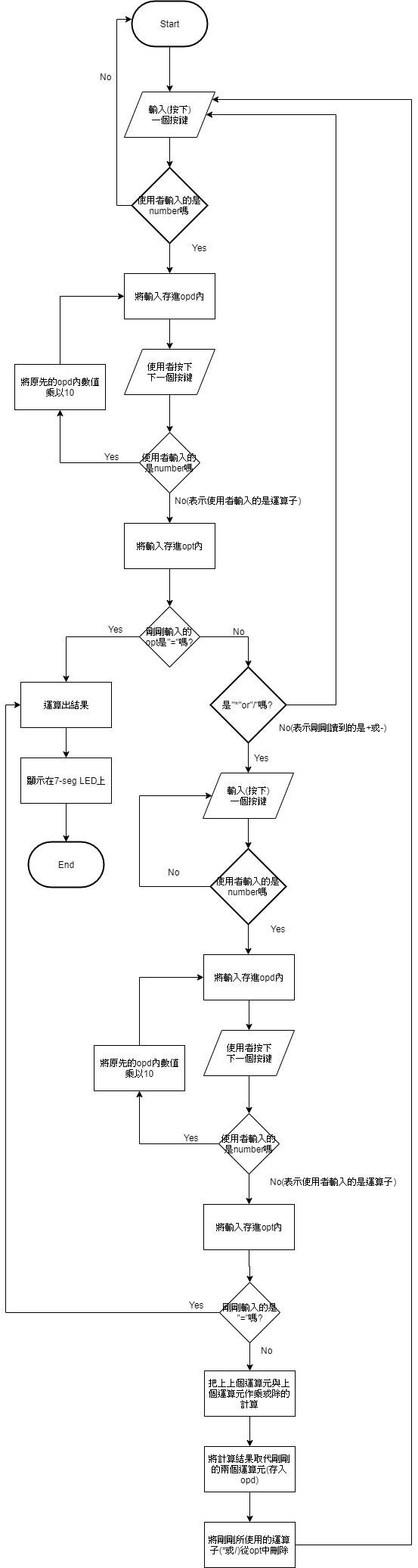
Contribution (貢獻度) 55%

**Title (標題) : 計算機 Calculator**

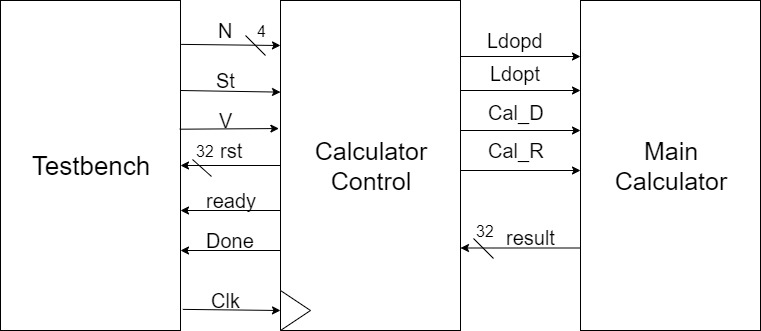
**A. Problem Description (問題敘述):**

模擬按計算機時，一次一個鍵入運算元及運算子，calculator controller控制四則運算的執行順序，Calculator進行加減乘除的運算。

**B. Flowchart or Procedure (流程圖或運作程序) :**



**C. Block Diagram (方塊圖) :**

****

**D. Definition of Inputs, Outputs, Control Signals, and Status Signals (輸入、輸出、控制訊號、及狀態訊號之定義)**

N0~3:鍵盤運算元運算子對應的code。以下為對照表:

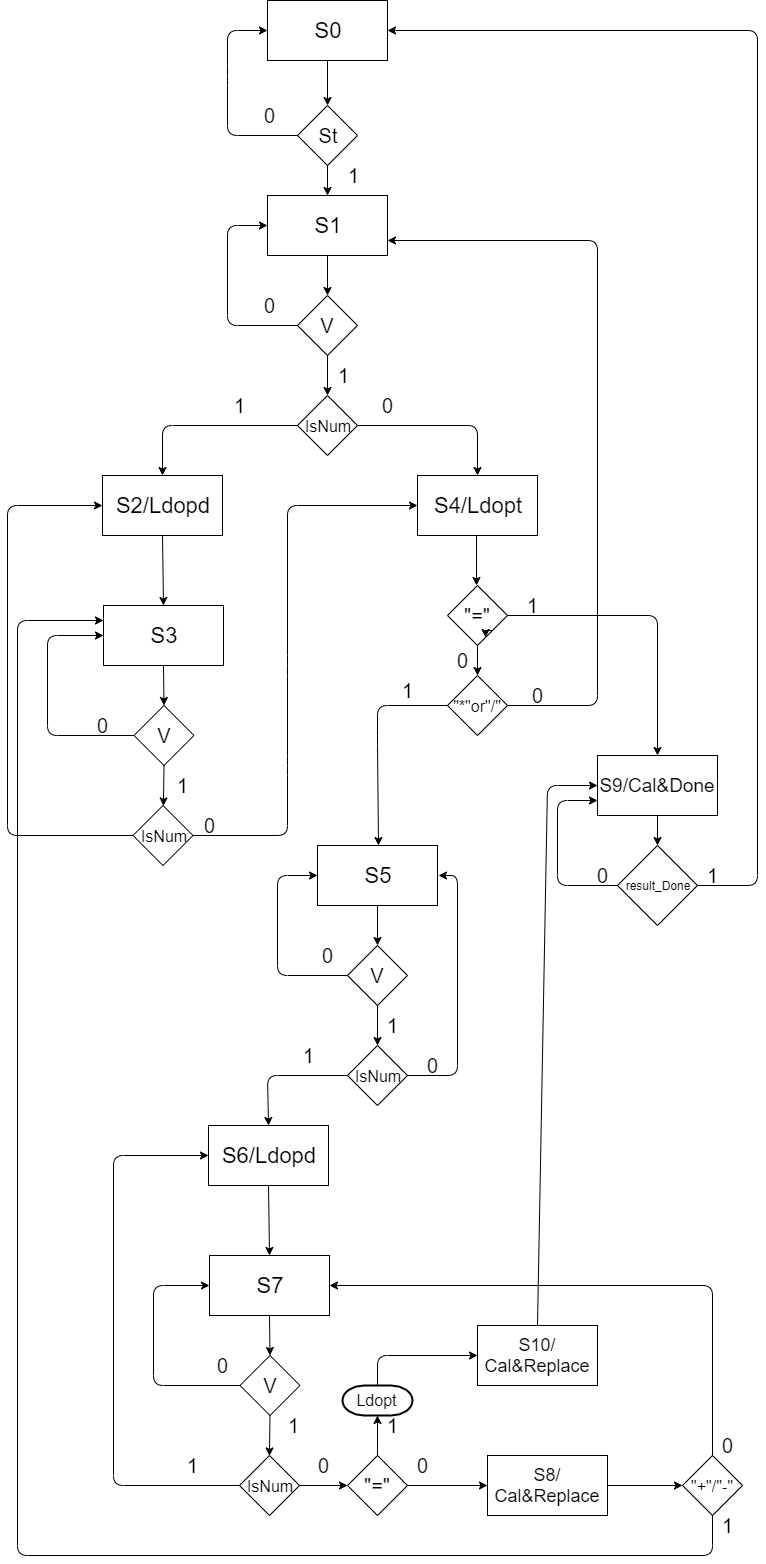
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| operand/operator | N3 | N2 | N1 | N0 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10(+) | 1 | 0 | 1 | 0 |
| 11(-) | 1 | 0 | 1 | 1 |
| 12(\*) | 1 | 1 | 0 | 0 |
| 13(Invalid) | 1 | 1 | 0 | 1 |
| 14(=) | 1 | 1 | 1 | 0 |
| 15(/) | 1 | 1 | 1 | 1 |

1. Clk: 時脈
2. V: 是否為合法的code
3. St: 輸入開始的訊號
4. rst/result: 運算結果
5. ready: 確認是否可以輸入測資
6. Done: 運算完成的訊號
7. Ldopd: 將運算元存入暫存器
8. Ldopt: 將運算子存入暫存器
9. Cal\_R: 遇到乘號或除號時，先進行該運算並且修改前一暫存器中的運

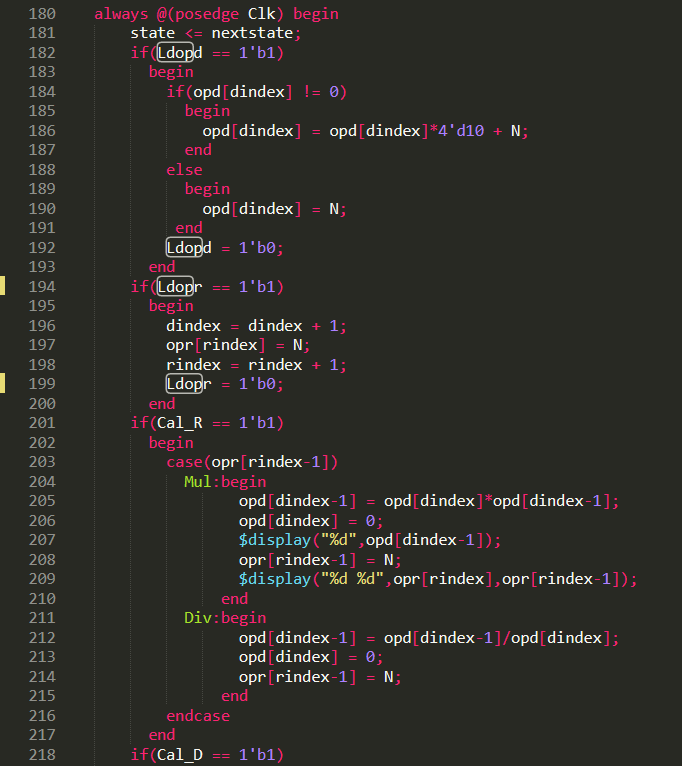
算子。例如:4+3\*2-5在讀取到\*並且讀取到2之後會先算出6，並且把opd陣列中的3以及2 replace成6

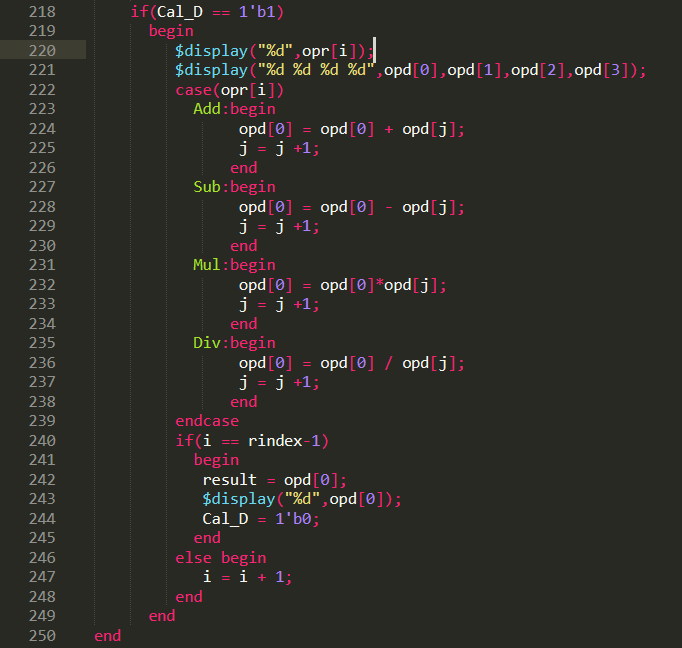
1. Cal\_D: 將運算前往後執行至完成，並輸出完成訊號

**E. State Machine Chart (SM Chart) or State Graph (狀態機器圖或狀態圖):**



**F. Description of Verilog Code (Verilog電路模組說明):**





如上圖，這是我們main code中的第二個always block(第一個always block內容是狀態轉換，應該不須特別說明，幾乎都跟sm chart一樣)，這裡是做load operator以及load operand 還有calculate的功能。

我們每次load運算元以及運算子之後會將其存入opd以及opr的陣列內，並且以dindex以及rindex分別記錄目前存到哪個位置。

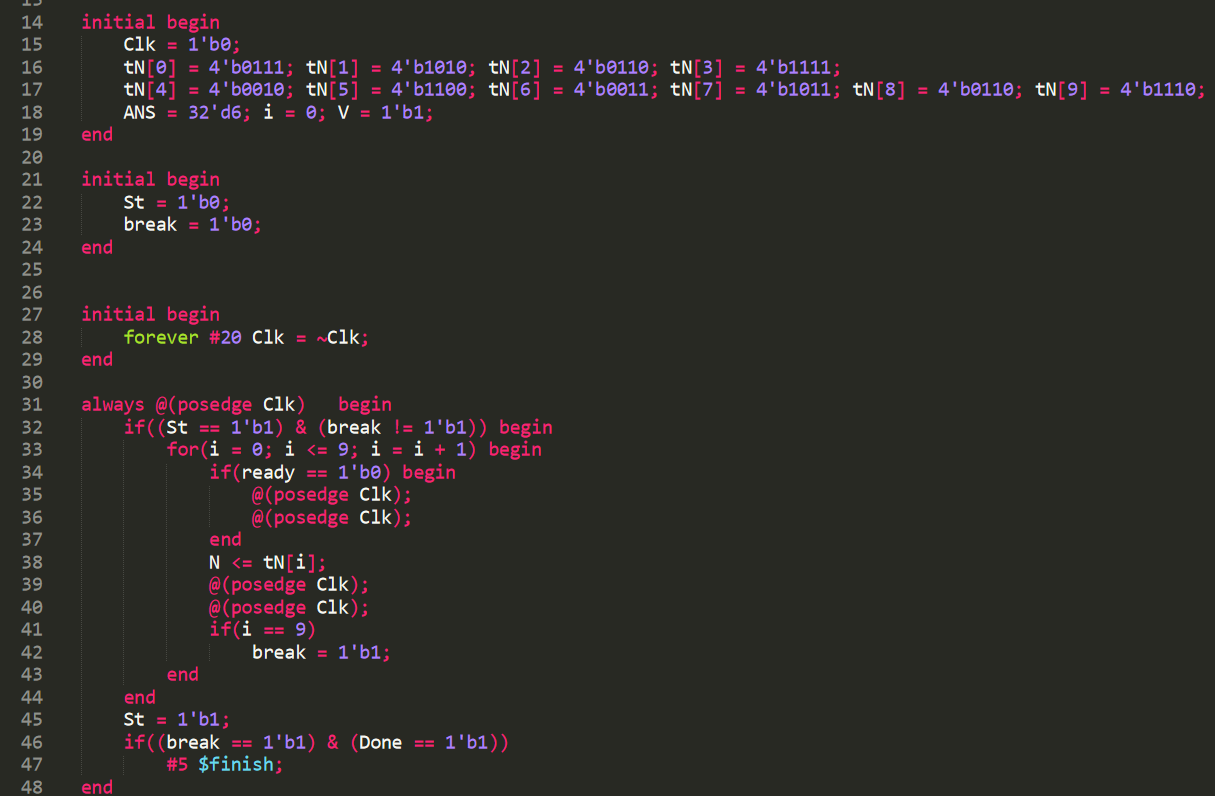
當Mul以及Div時，我們會進opd陣列將opd[dindex-1]更新為opd[dindex-1]乘上或是除以opd[dindex]。並且將opd[dindex]更新為0。

當Add或是Sub時，我們僅會將運算元以及運算子存入opr以及opd陣列。

在讀取到Cal\_Done訊號時，會將opd內的所有數值進行四則運算，最後輸出結果到result。

P.S我們每次四則運算都是把結果存到opd[0]。例如:4+3-2，會先做4+3並且把結果存到opd[0](因此4被7覆蓋了)，接著再做7-2，並且把結果存到opd[0](因此7被5覆蓋了)

**G. Description of Test Bench (Verilog測試模組說明):**



如上圖，我們的tN[]是用來儲存讀取到鍵盤上的code，接著利用for迴圈來讀取。其中，最重要的是@posedge CLK的設定，因為讀加減以及乘除運算所經過的State數量不同(加減只需要讀取數字並且儲存，只需兩個Clk，乘除卻需要在讀取數字之後事先做計算並且儲存，共需要四個Clk)。

P.S. 當系統做完乘除運算之後，會回傳ready = 1’b1

因此我們在35 36行多等待了兩次Clk cycle，才能成功地讓計算機做完乘除運算之後，下一個tN[]才會讀取進來(以免跳過一個N沒有讀到)。

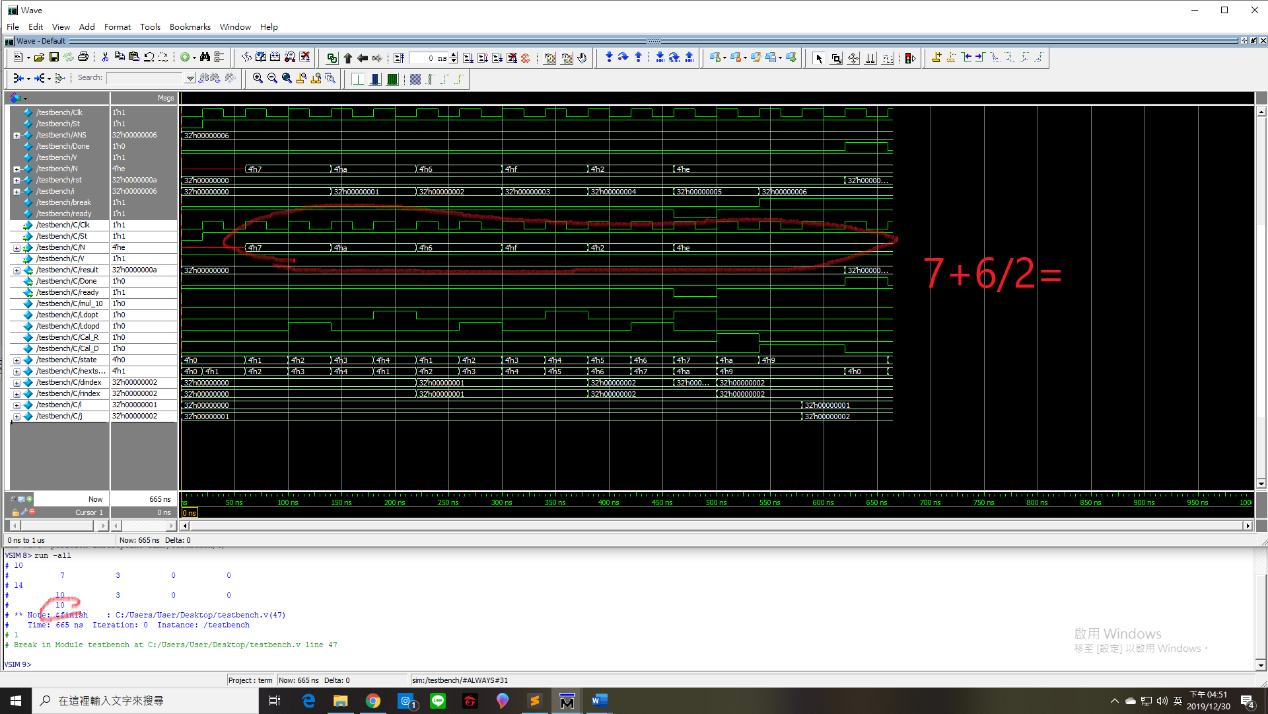
而39 40行等待的兩次Clk cycle便是上述我所說的，最少需要兩個cycle才能做加減的數字讀取以及儲存。

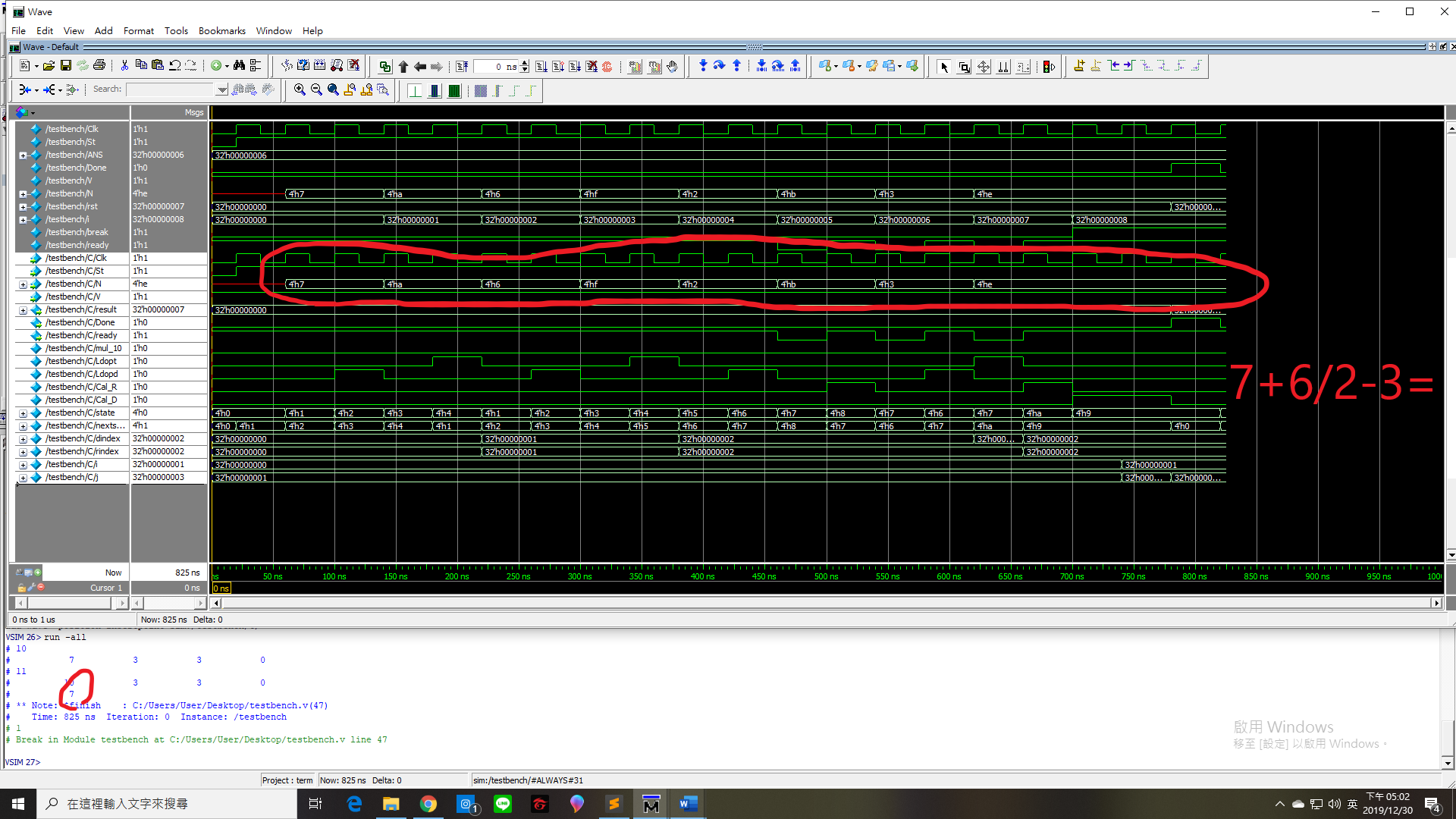
另外，在最後一個N進來之後，有個flag叫做”break”，會讓系統不會再繼續讀進測資(for迴圈可能會重啟)。

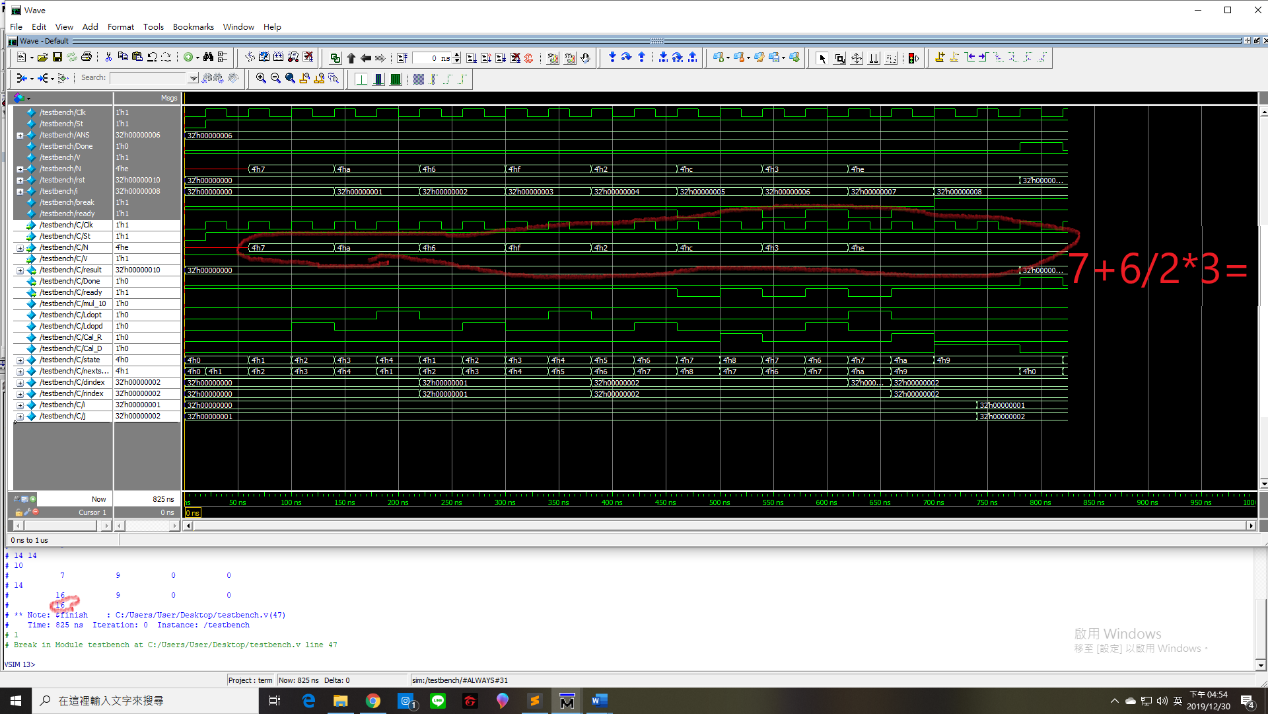
在計算機做完result的計算之後，會回傳Done = 1b’1，因此當break以及Done都等於1時，會結束模擬。

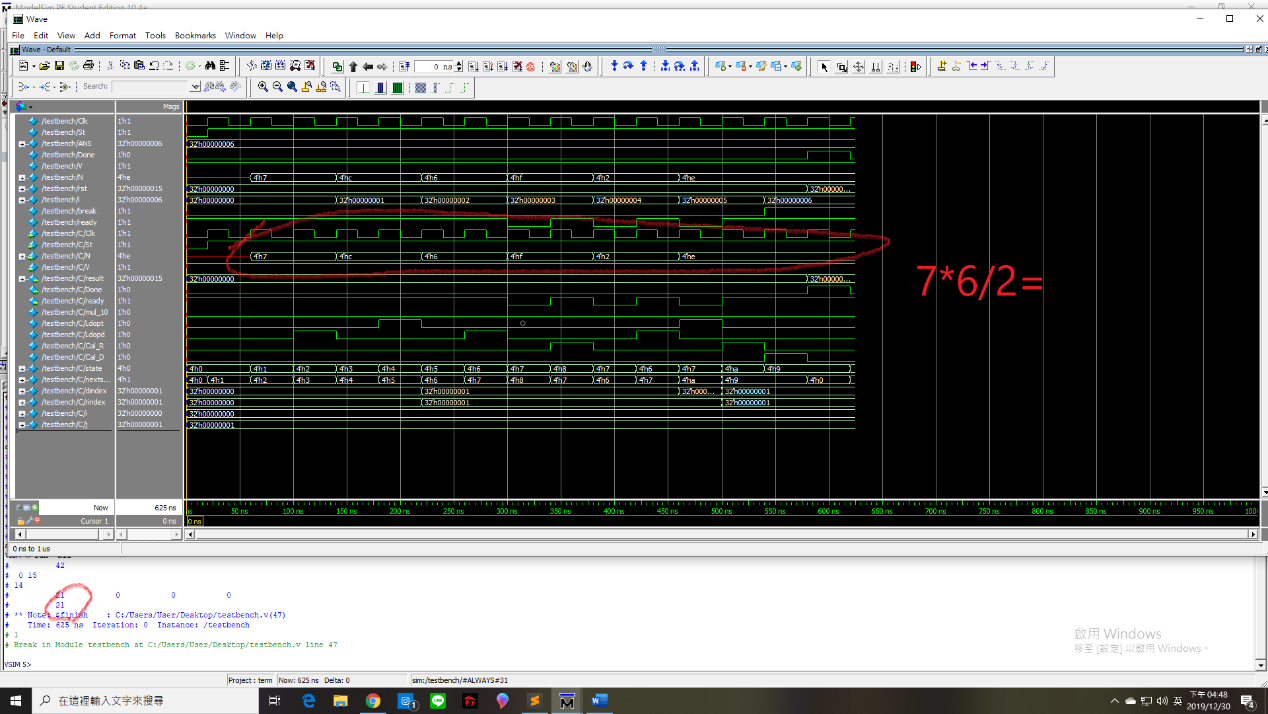
**H. Simulation Results (模擬結果):**

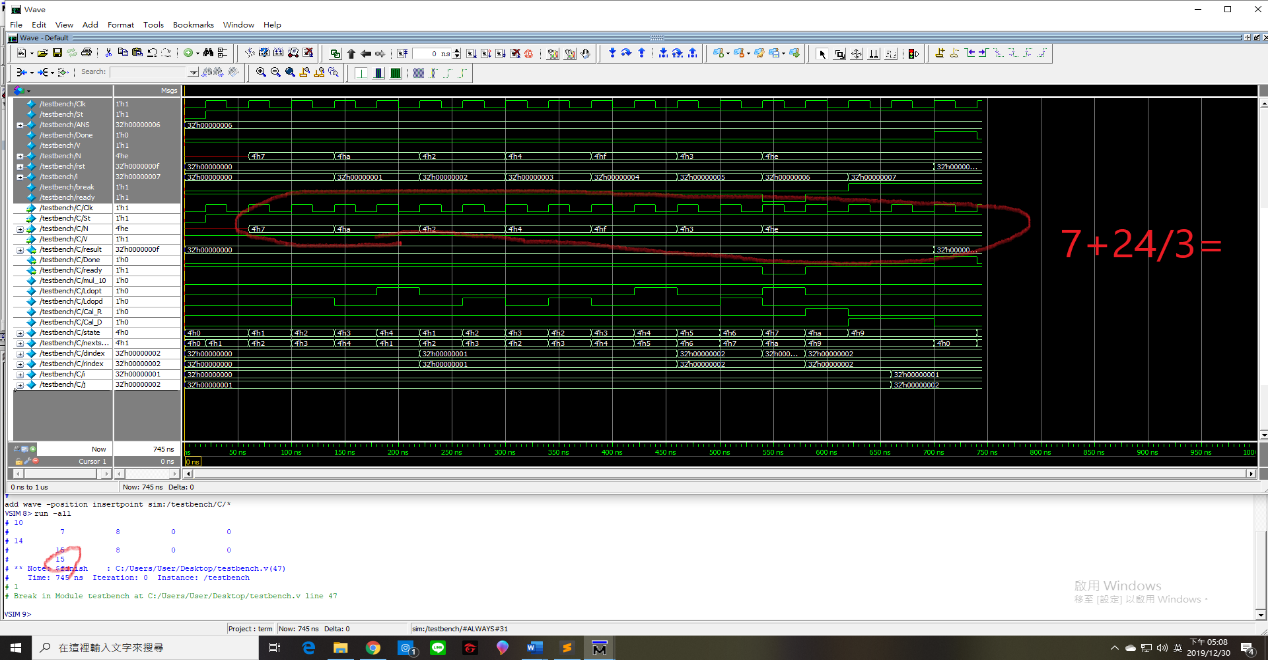
以下我做了許多測資進行測試: 有連續乘除的計算，還有加減與乘除的混和計算；以及最重要的，二位數字的讀取及運算











**I. Conclusions and Discussions (心得、感想、結論、及討論):**

這次project整體而言弄得相當趕，因為我們這組兩人都是大三必修loading重，所以可能在與其他課程作業考試的時間分配上沒能掌控好，最後的project的進行相當匆促，結果其實不盡理想。當然過程中所參考的一些資料以及寫code的技巧都對提升自己有所幫助，所以未能在此次project中將其徹底活用並完成預期結果，不免有些遺憾。

Discussion:

1. 我們在計算上的邏輯是遇到乘除便先將其完成計算，最後讀到等於符號時由前往後運算，因此在如何控制運算元和運算子的暫存器指標是本次project中我們所面臨的最大難題。
2. 跟其他程式語言不同，verilog中需要考慮時脈的問題，在本次project中我們使每個輸入都至少維持2個Cycle，因為至少有兩個state需要依賴輸入進行狀態轉換的判斷並得保證每個輸入都會被讀取到。
3. 類似上述，在撰寫testbench時我們便得考慮輸入以及clock的問題，因為我們的project每次讀取輸入值的週期不相同(在**Description of Test Bench (Verilog測試模組說明)有詳述**)，因此我們花了非常多的時間在使modelsim能夠在正確的state之間變換。

**References (參考資料) :** (請說明各參考項目對你的專題提供那方面資料)

1. 原本要參考課本的keypad scanner 可是發現那個testbench是錯的…

Clock 根本對不起來。(課本的K與Kd用來debounce，可是K與Kd根本不可能同時等於1…)。