Digital System Design (Fall 2019)

Final Report of Term Project

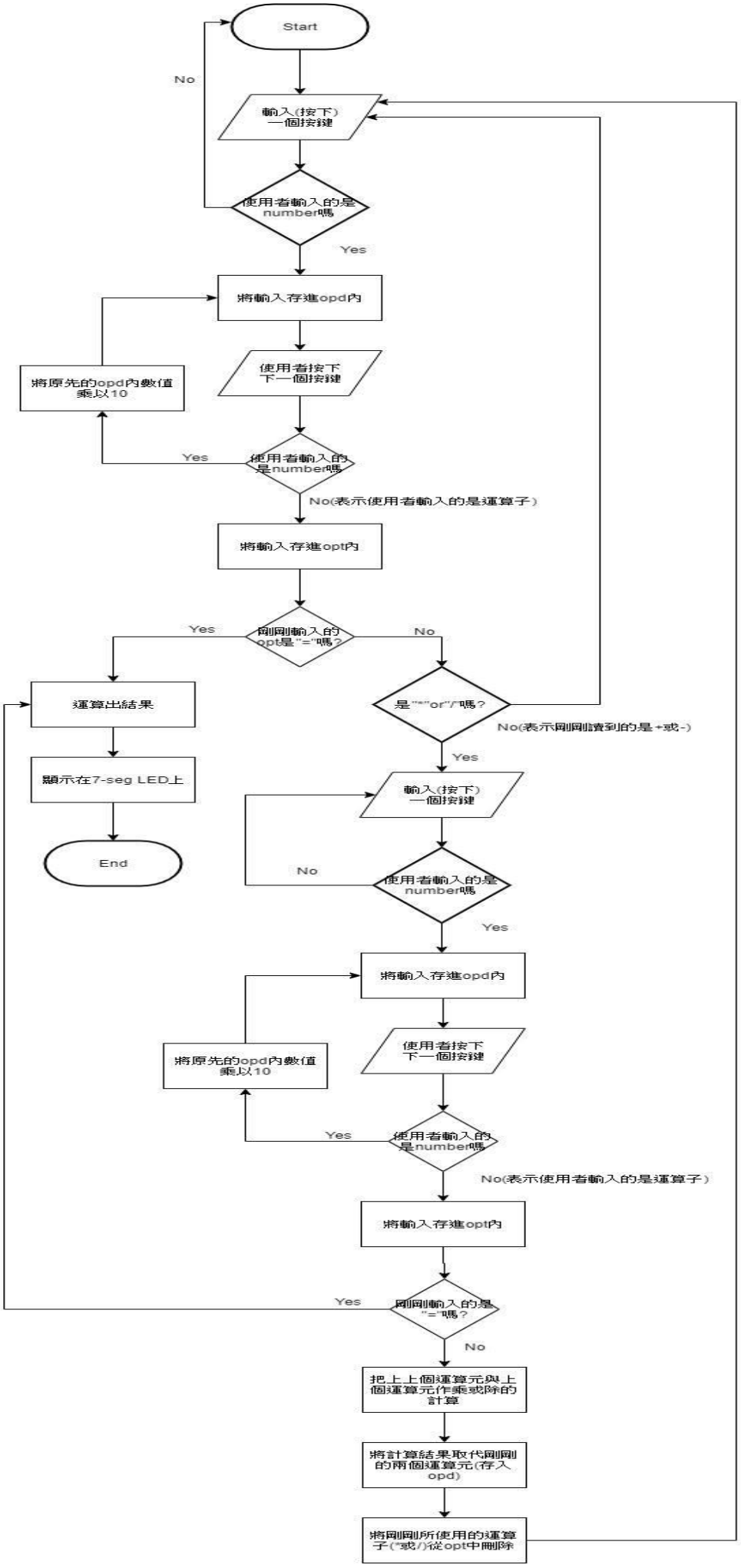
Group Number (組別) : 02
Group Member 1 (組員 1) : Student ID 0613246 Name 李泓賢
Contribution (貢獻度) 45%
Group Member 2 (組員 2) : Student ID 0611201 Name 陳彰浩
Contribution (貢獻度) 55%

Title (標題) : 計算機 Calculator

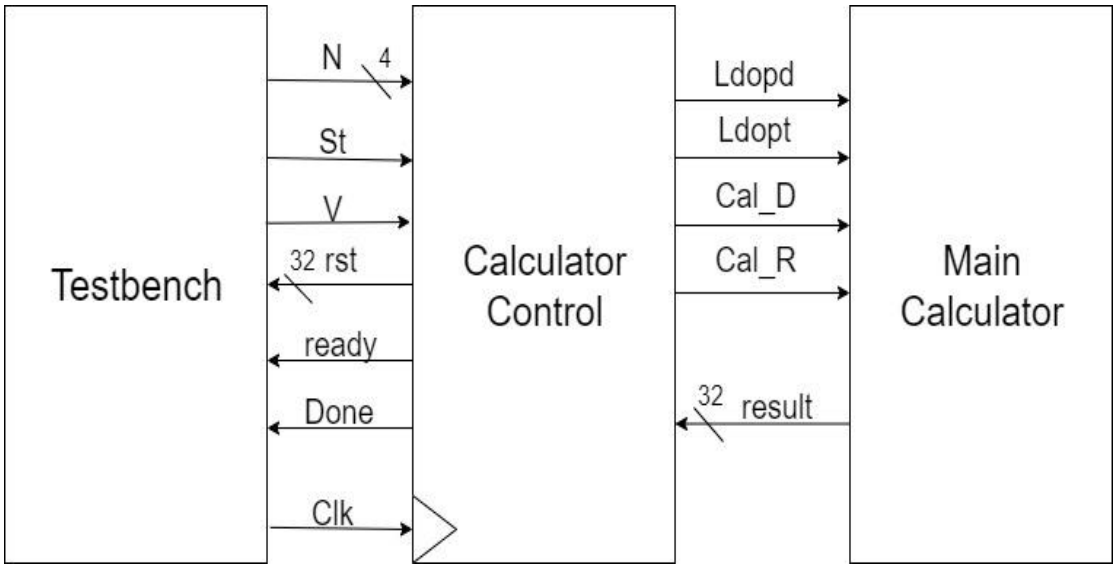
A. Problem Description (問題敘述):

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) :


Start
No
輸入(按下)
一個按鍵
使用者輸入的是
number嗎
Yes
將輸入存進opd內
使用者按下
下一個按鍵
將原先的opd內數值
乘以10
Yes
使用者輸入的
是number嗎
No(表示使用者輸入的是運算子)
將輸入存進opt內
Yes
剛剛輸入的
opt是"="嗎?
No
運算出結果
顯示在7-seg LED上
End
是"*"or"/"嗎?
No(表示剛剛讀到的是+或-)
Yes
輸入(按下)
一個按鍵
No
使用者輸入的是
number嗎
Yes
將輸入存進opd內
使用者按下
下一個按鍵
將原先的opd內數值
乘以10
Yes
使用者輸入的
是number嗎
No(表示使用者輸入的是運算子)
將輸入存進opt內
Yes
剛剛輸入的是
"="嗎?
No
把上上個運算元與上
個運算元作乘或除的
計算
將計算結果取代剛剛
的兩個運算元(存入
opd)
將剛剛所使用的運算
子(*或/)從opt中刪除

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

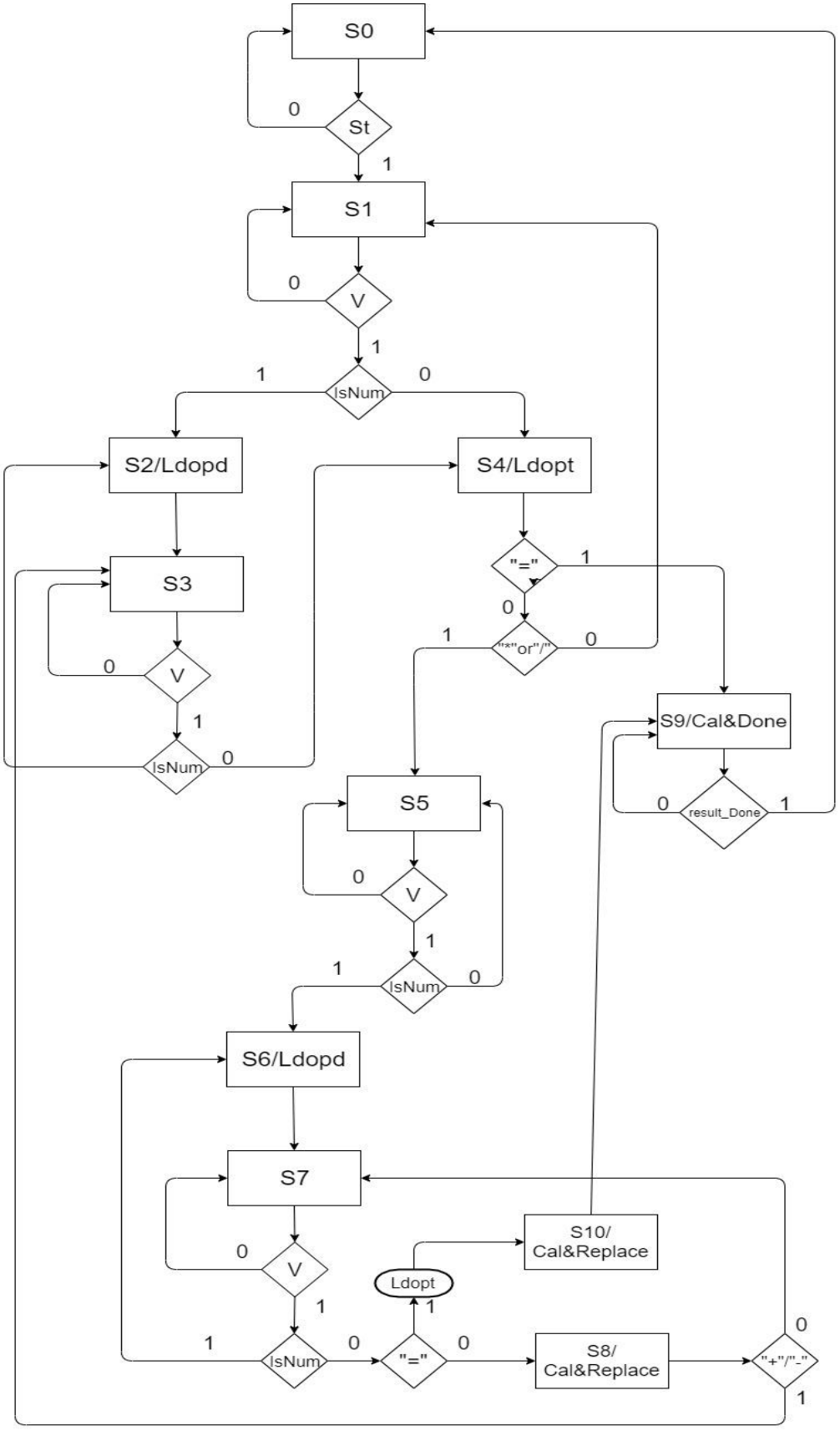
(7) Ldopd: 將運算元存入暫存器

(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

(10) 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):


S0
St
S1
V
IsNum
S2/Ldopd
S4/Ldopt
S3
V
IsNum
"="
"*"or"/"
S9/Cal&Done
result_Done
S5
V
IsNum
S6/Ldopd
S7
V
IsNum
"="
Ldopt
S10/
Cal&Replace
S8/
Cal&Replace
"+"/"-"
0
1

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

```
180     always @(posedge Clk) begin
181         state <= nextstate;
182         if(Ldopd == 1'b1)
183           begin
184             if(opd[dindex] != 0)
185               begin
186                 opd[dindex] = opd[dindex]*4'd10 + N;
187               end
188             else
189               begin
190                 opd[dindex] = N;
191               end
192             Ldopd = 1'b0;
193           end
194         if(Ldopr == 1'b1)
195           begin
196             dindex = dindex + 1;
197             opr[rindex] = N;
198             rindex = rindex + 1;
199             Ldopr = 1'b0;
200           end
201         if(Cal_R == 1'b1)
202           begin
203             case(opr[rindex-1])
204               Mul:begin
205                     opd[dindex-1] = opd[dindex]*opd[dindex-1];
206                     opd[dindex] = 0;
207                     $display("%d",opd[dindex-1]);
208                     opr[rindex-1] = N;
209                     $display("%d %d",opr[rindex],opr[rindex-1]);
210                   end
211               Div:begin
212                     opd[dindex-1] = opd[dindex-1]/opd[dindex];
213                     opd[dindex] = 0;
214                     opr[rindex-1] = N;
215                   end
216             endcase
217           end
218         if(Cal_D == 1'b1)
```

```
218         if(Cal_D == 1'b1)
219           begin
220             $display("%d",opr[i]);
221             $display("%d %d %d %d",opd[0],opd[1],opd[2],opd[3]);
222             case(opr[i])
223               Add:begin
224                     opd[0] = opd[0] + opd[j];
225                     j = j +1;
226                   end
227               Sub:begin
228                     opd[0] = opd[0] - opd[j];
229                     j = j +1;
230                   end
231               Mul:begin
232                     opd[0] = opd[0]*opd[j];
233                     j = j +1;
234                   end
235               Div:begin
236                     opd[0] = opd[0] / opd[j];
237                     j = j +1;
238                   end
239             endcase
240             if(i == rindex-1)
241               begin
242                 result = opd[0];
243                 $display("%d",opd[0]);
244                 Cal_D = 1'b0;
245               end
246             else begin
247                 i = i + 1;
248               end
249           end
250     end
```

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):

```
14  initial begin
15      Clk = 1'b0;
16      tN[0] = 4'b0111; tN[1] = 4'b1010; tN[2] = 4'b0110; tN[3] = 4'b1111;
17      tN[4] = 4'b0010; tN[5] = 4'b1100; tN[6] = 4'b0011; tN[7] = 4'b1011; tN[8] = 4'b0110; tN[9] = 4'b1110;
18      ANS = 32'd6; i = 0; V = 1'b1;
19  end
20
21  initial begin
22      St = 1'b0;
23      break = 1'b0;
24  end
25
26
27  initial begin
28      forever #20 Clk = ~Clk;
29  end
30
31  always @(posedge Clk)   begin
32      if((St == 1'b1) & (break != 1'b1)) begin
33          for(i = 0; i <= 9; i = i + 1) begin
34              if(ready == 1'b0) begin
35                  @(posedge Clk);
36                  @(posedge Clk);
37              end
38              N <= tN[i];
39              @(posedge Clk);
40              @(posedge Clk);
41              if(i == 9)
42                  break = 1'b1;
43          end
44      end
45      St = 1'b1;
46      if((break == 1'b1) & (Done == 1'b1))
47          #5 $finish;
48  end
```

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

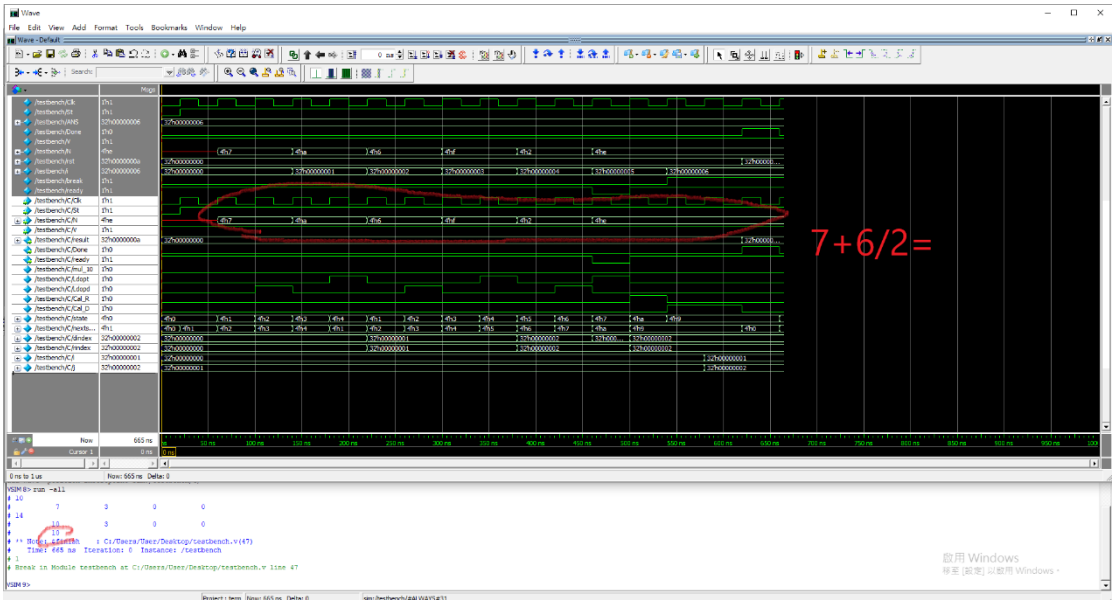
做加減的數字讀取以及儲存。

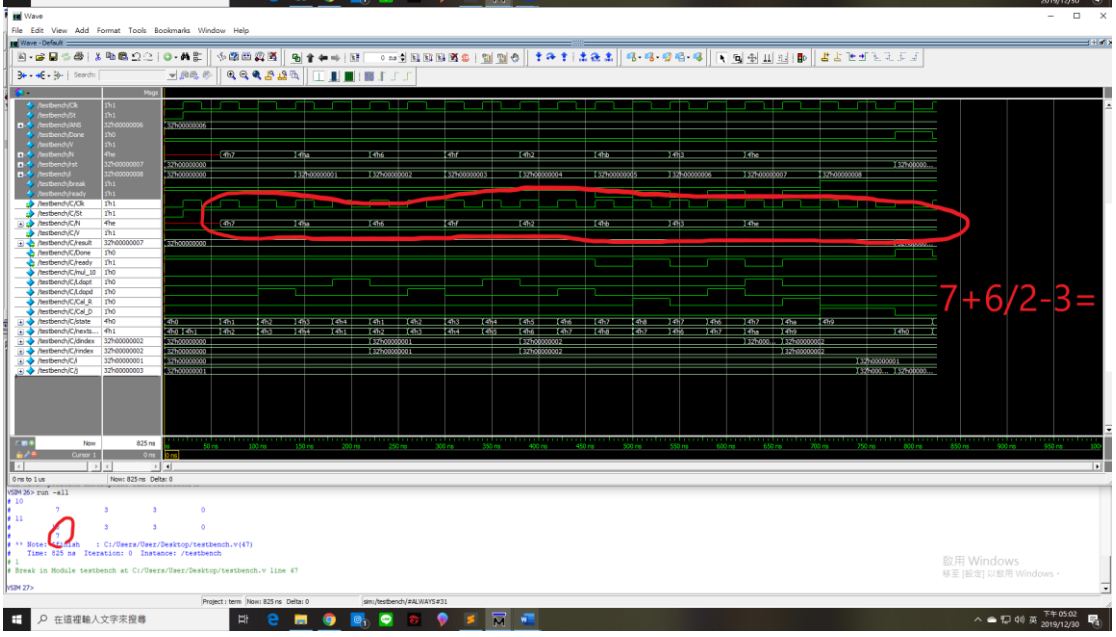
另外，在最後一個 N 進來之後，有個 flag 叫做”break”，會讓系統不會再繼續讀進測資(for 迴圈可能會重啟)。

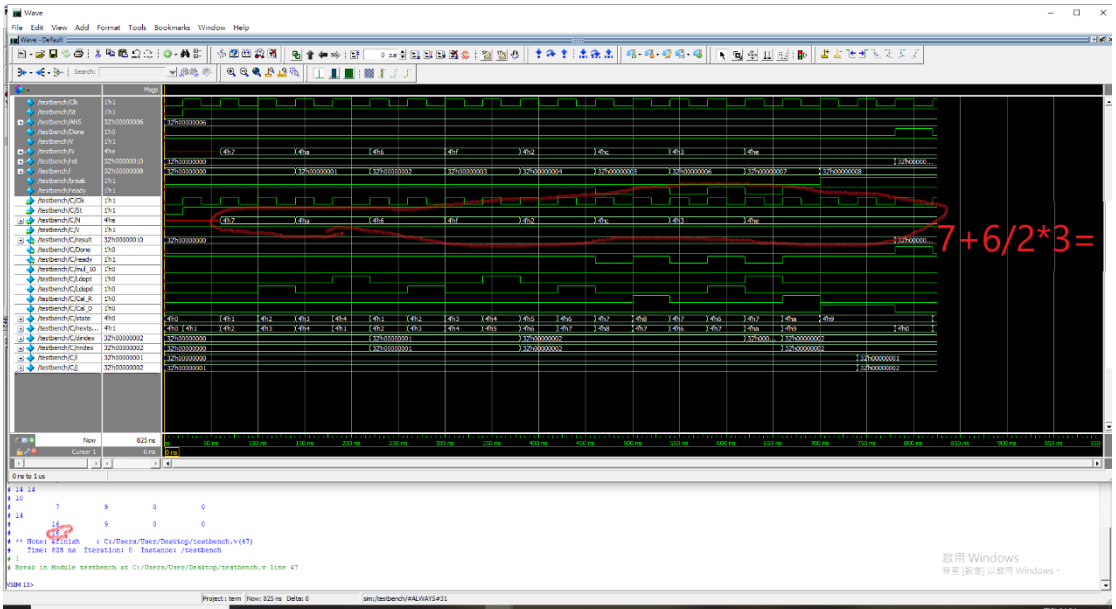
在計算機做完 result 的計算之後，會回傳 Done = 1b’1，因此當 break 以及 Done 都等於 1 時，會結束模擬。

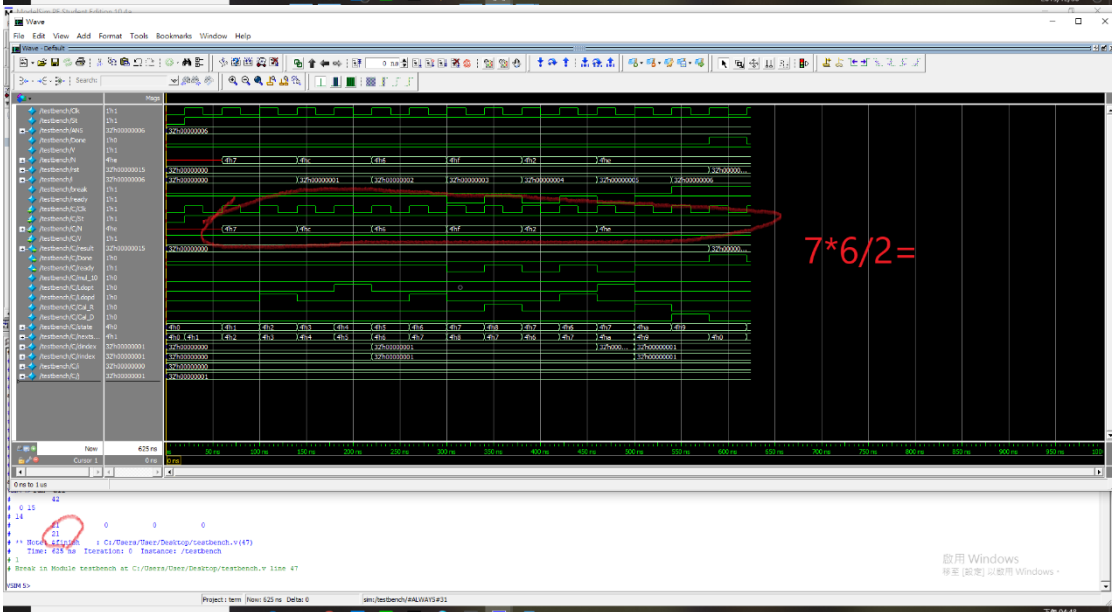
H. Simulation Results (模擬結果):

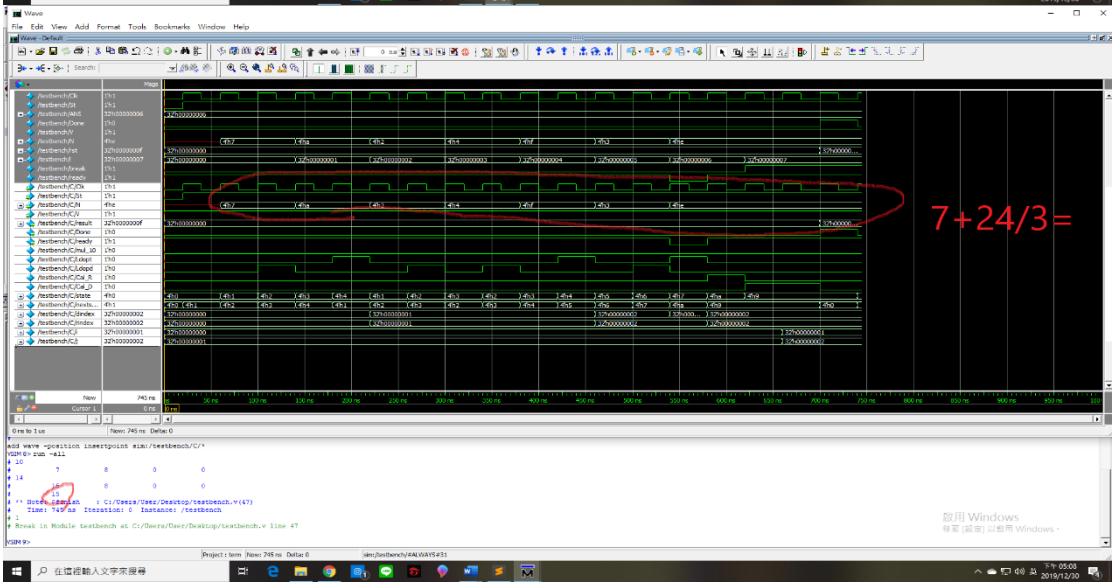
以下我做了許多測資進行測試: 有連續乘除的計算，還有加減與乘除的混和計算；以及最重要的，二位數字的讀取及運算






7+6/2*3=


7*6/2=


7+24/3=

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…)。