




2023 Digital IC Design Homework 3

NAME	林宜謙																														
Student ID	N16100250																														
Simulation Result																															
Functional simulation	100	Gate-level simulation	100																												
<div><p>Congraultaions!!! You past all patterns! Your score is 100. Total use 2128 cycles to complete simulation.</p><p>Note: \$finish : C:/Users/lin/Documents/modelsim/verilog_practice/HW3/testfi Time: 212800 ns Iteration: 1 Instance: /testfixture</p><p>Clock width = 100(original)</p><p>Congraultaions!!! You past all patterns! Your score is 100. Total use 2128 cycles to complete simulation.</p></div>		<div><p>Congraultaions!!! You past all patterns! Your score is 100. Total use 2128 cycles to complete simulation.</p><p>Note: \$finish : C:/Users/lin/Documents/modelsim/verilog_practice/HW3/testfix Time: 38304 ns Iteration: 1 Instance: /testfixture</p><p>Clock width = 18</p><p>Congraultaions!!! You past all patterns! Your score is 100. Total use 2128 cycles to complete simulation.</p></div>																													
Synthesis Result																															
Total logic elements	456																														
Total memory bits	0																														
Embedded multiplier 9-bit elements	1																														
Total cycle used	2128																														
Clock width	18																														
<div><div>Flow Summary</div><div> <<Filter>></div><table><tr><td>Flow Status</td><td>Successful - Tue Apr 25 17:02:34 2023</td></tr><tr><td>Quartus Prime Version</td><td>20.1.1 Build 720 11/11/2020 SJ Lite Edition</td></tr><tr><td>Revision Name</td><td>AEC</td></tr><tr><td>Top-level Entity Name</td><td>AEC</td></tr><tr><td>Family</td><td>Cyclone IV E</td></tr><tr><td>Device</td><td>EP4CE55F23A7</td></tr><tr><td>Timing Models</td><td>Final</td></tr><tr><td>Total logic elements</td><td>456 / 55,856 (< 1 %)</td></tr><tr><td>Total registers</td><td>199</td></tr><tr><td>Total pins</td><td>19 / 325 (6 %)</td></tr><tr><td>Total virtual pins</td><td>0</td></tr><tr><td>Total memory bits</td><td>0 / 2,396,160 (0 %)</td></tr><tr><td>Embedded Multiplier 9-bit elements</td><td>1 / 308 (< 1 %)</td></tr><tr><td>Total PLLs</td><td>0 / 4 (0 %)</td></tr></table></div>				Flow Status	Successful - Tue Apr 25 17:02:34 2023	Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition	Revision Name	AEC	Top-level Entity Name	AEC	Family	Cyclone IV E	Device	EP4CE55F23A7	Timing Models	Final	Total logic elements	456 / 55,856 (< 1 %)	Total registers	199	Total pins	19 / 325 (6 %)	Total virtual pins	0	Total memory bits	0 / 2,396,160 (0 %)	Embedded Multiplier 9-bit elements	1 / 308 (< 1 %)	Total PLLs	0 / 4 (0 %)
Flow Status	Successful - Tue Apr 25 17:02:34 2023																														
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition																														
Revision Name	AEC																														
Top-level Entity Name	AEC																														
Family	Cyclone IV E																														
Device	EP4CE55F23A7																														
Timing Models	Final																														
Total logic elements	456 / 55,856 (< 1 %)																														
Total registers	199																														
Total pins	19 / 325 (6 %)																														
Total virtual pins	0																														
Total memory bits	0 / 2,396,160 (0 %)																														
Embedded Multiplier 9-bit elements	1 / 308 (< 1 %)																														
Total PLLs	0 / 4 (0 %)																														

Description of your design

主要分為 4 個 state:

Data_IN: 收 input 進來的資料。

CHECK_DATA: 確認每一個收進來的資料，分別判斷要 pop / push / 比較優先度 / 遇到 '(' / 遇到 ')', 都在這個階段中完成。

CHECK_STACK_EMPTY: 在確認完每一個 input 資料之後，判斷 stack 裡面是否還有運算符號還沒有被 pop 出來(要跳過 '(')。

CALCULATE: postfix 轉換完成後進行計算，計算完成後拉高 valid = 1，輸出 stack 最後的數字為最後的計算結果，並再下一個 cycle 把值 reset。

--- 流程說明 ---

增加一個非同步 reset 訊號，等待 reset 訊號將所需要的變數與 register 歸零。

DATA_IN 階段:

將 valid 高低為 0，這個階段要不斷收 input 的值，直到遇到 "=="，判斷值的方式分為 "=="、運算符號(包含括號)、0~15 的數字，而由於所有 input 的 ASCII 碼都小於 127 並且 output 值也是 7-bit，所以在整個程式的陣列中都設為 7-bit，所以在 ASCII 碼中只會用到 8-bit 中的後 7-bit，因此判斷都用 `ascii[6:0]` 取 input 的值。

- 第一層判斷 input 收到 "==" 為止，並跳到 CHECK_DATA 階段，並將 data 陣列的 index 歸零，作為下一階取值用的 index。
- 第二層判斷 input 的資料是否為那五個運算符號，根據 ascii 碼的十進位值發現都比 48(7'b011_0000)小，比較小的就會直接存 ascii 碼的值到 data 陣列中。
- 第三層再判斷完前兩層後，所剩下的 input 數值只會剩下 0~15 的 ascii 碼，再來就是透過一個運算方式將 0~15 的 ascii 數值統一整理成 7-bit 0~15 的數值，這樣整理可以找一層 if-else 判斷減少 Total Logic element 的成本，運算方式如下:

- 觀察 0~9 的 7-bit ascii 碼組成為 7'b011_0000(7'd48)加上 0~9 的數值
- 觀察 10~15 的 7-bit ascii 碼組成為 7'b110_0000(7'd96)加上 1~6 的數值，如果要後面 4-bit 弄成 10~15 的數值，將 1~6 + 9，而 9 的二進位為 4'b1001，所以發現在[3]、[0]的位置為 1，所以這兩個 1 可以從 ascii 中的前 4 位 bit(4'b0011 與 4'b0110)會有兩個 bit 不一樣，所以透過 XOR 與 AND 的方式將這兩個 bit 取出。

計算方式為：

$\text{Ascii}[4] \wedge 1'b0$ 與 $\text{Ascii}[6] \& 1'b0$ ，再把這兩個結果與 2'b00 集束得到 $\{\text{Ascii}[6] \& 1'b0, 2'b00, \text{Ascii}[4] \wedge 1'b0\}$ ，所以 0~9 就會計算出 4'b0000，a~f(10~15)會計算到 4'b1001，所以就可以湊出 9 的數值，使 0~9 與 a~f 透過這樣的計算方式得到 0~15。

CHECK_DATA 階段:

判斷每一個收到的資料要 pop/push/ '('/ ')' 不同狀態，直到輸入資料的次數 (data_num)，而陣列 index 從 0 開始所以多+1 判斷，而目前的資料會分為數字 0~15 與五個運算子，另外為減少成本，將 postfix 的結果直接放在 data 陣列中，並用 postfix_index 紀錄，因為 postfix 的資料與 stack 每一次判斷所放的資料數量一定不超過 data 的總數量，所以不會干擾到。

另外 stack 陣列的總數量為 8 個，原因是在這個 postfix 的演算法中，在 data 數量不超過 16 個情況下，所 push 進 stack 數量一定不超過 8 個。
(減少了 100 多 total logic element)

利用 4 層的 if-else 判斷每一個資料:

- ✓ 以 postfix_idx 變數表示 pop 出來的資料在 data 陣列的位置
- ✓ 以 data_arr_idx 變數表示需要進行處理 input 資料在 data 陣列的位置
- ✓ 以 stack_index 變數表示 stack 的位置
- 第一層判斷，判斷是否為數字，判斷方式為數值小於 16(7'b001_0000)，如果是數字的話就直接 pop 到 data_arr 中。
- 第二層判斷是否要將 data push 進 stack 中，而因為第一層已經將數字分開了，所以這個階段只會有五個運算子在判斷，可以用最後三個 bit([2:0])判斷就好，條件分為三個 or 去判斷，只要符合其中一個就好，而且會按照順序判斷:
 1. 如果 stack 是空的時候，直接 push 進 stack
 2. 如果遇到 '(' 直接 push， '(' 最後 3 個 bit 為 3'b000
 3. 判斷運算子的優先順序，如果 data 的優先度大於 top stack 的資料，才能 push，再來是判斷優先度的方式，利用兩個比較方式製造出優先度，以 '*' 為標準，因為其優先度最高，判斷方式:
 - ✓ 若最後 3-bit 等於 3'b010，也就是等於 '*' 為 1
 - ✓ 若最後 3-bit 大於 3'b001，也就是大於 '(' 與 ')' 為 1透過這樣的判斷方式，集束這兩個 bit 可以分別對這五個運算子缺分成三種優先度的數值:
 - '*' 會得到 2'b11 (最大)
 - '+', '-' 會得到 2'b01 (次中)
 - '(', ')' 會得到 2'b00 (最低)由此可以比較出 data 與 top stack 的資料優先度，決定要不要 push
- 第三層根據前兩層判斷結果，表示剩下的就是優先度一樣的運算子與 ')', 要做的事情都是把 stack 的資料 pop 出來，但是遇到 ')' 的情況不一樣，若

在 pop 的時候要 pop 出 '(' 的時候要把 '(' ' ' 消掉，所以直接 data_arr_idx+1 忽略 ')' 與 stack_index-1 忽略 '('，並且利用 pop_time-2 表示在計算階段少兩次，因為少了兩個 data。

- 第 4 層剩下的條件就是把 top stack 大於與等於 data 優先度的資料 pop 到 data_arr 中以 postfix_idx 紀錄。

CHECK_STACK_EMPTY 階段:

在進這個階段的時候將原本 data_arr_idx 拉到 0 作為下一計算階段 postfix 結果的 index 使用。

在前一個階段判斷完所有 data 的時候，有可能 stack 裡面還有未 pop 出來的資料，所以要持續 pop 到 stack 清空，但是有一個情況會是 stack 裡面還有 '(' 的時候，遇到的時候不能 pop 要直接跳過，並且 pop_time 總計算次數要少兩次表示有 '(' ')' 對消的情況發生。

CALCULATE 階段:

包含 reset 所以用 valid 是否等於 1(表示計算是否結束)決定要不要 reset，再來就是根據 pop_time 的數字表示 postfix 要 push/運算子計算的總數量，postfix 的資料只剩下數字與 '+', '-', '*'，分別 push 到 stack 或運算子計算，用一個 case 判斷 '+', '-', '*'，將 stack 的兩個數字計算，其餘就是把數字 push 到 stack 中，所有次數做完後將 valid 拉高為 1，把 stack 最後的數字也就是計算結果的 stack[0] 數字輸出到 result 中完成計算，在下一個 clock 進行 reset 的動作。

*Scoring = Area cost * Timing cost*

*Area cost = Total logic elements + Total memory bits + 9*Embedded multipliers 9-bit elements*

*Timing cost = Total cycle used * Clock width*

*** Total logic elements must not exceed 1500.**

(Scoring) 15832320 = 465 * 34048

(Area cost) 465 = 456 + 0 + 9 * 1

(Timing cost) 34048 = 2128 * 16