2023 Digital IC Design Homework 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | 林宜謙 | | | | |
| Student ID | N16100250 | | | | |
| **Simulation Result** | | | | | |
| Functional simulation | | 100 | | Gate-level simulation | 100 |
| Clock width = 100(original) | | | | Clock width = 18 | |
| **Synthesis Result** | | | | | |
| Total logic elements | | | 456 | | |
| Total memory bits | | | 0 | | |
| Embedded multiplier 9-bit elements | | | 1 | | |
| Total cycle used | | | 2128 | | |
| Clock width | | | 18 | | |
|  | | | | | |
| **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 :  Ascii[4] ^ 1’b0 與Ascii[6] & 1’b0，再把這兩個結果與2’b00集束得到  { Ascii[6] & 1’b0, 2’b00, Ascii[4] ^ 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**