2022 Digital IC Design Homework 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NAME | | 柳譯筑 | | | | | |
| Student ID | | NE6101034 | | | | | |
| **Simulation Result** | | | | | | | |
| Functional simulation | Pass/ Fail (encoder) | | Pass/ Fail (decoder) | Gate-level simulation | Pass/ Fail (encoder) | | Pass/ Fail (decoder) |
| (your pre-sim result) encoder/decoder  img0 | | | | (your post-sim result) encoder/decoder  img0 | | | |
| (your pre-sim result) encoder/decoder  Img1 | | | | (your post-sim result) encoder/decoder  img1 | | | |
| (your pre-sim result) encoder/decoder  img2 | | | | (your post-sim result) encoder/decoder  img2 | | | |
| **Synthesis Result** | | | | **encoder** | | **decoder** | |
| Total logic elements | | | |  | | 140/68416 | |
| Total memory bit | | | |  | | 0/1152000 | |
| Embedded multiplier 9-bit element | | | |  | | 0/300 | |
| Simulation time img0 | | | | 125760(ns) | | 61620(ns) | |
| Simulation time img1 | | | | 123120(ns) | | 61620(ns) | |
| Simulation time img2 | | | | 86820(ns) | | 61620(ns) | |
| (your flow summary) encoder | | | | (your flow summary) decoder | | | |
| **Description of your design** | | | | | | | |
| **Encoder:**  *首先關於訊號*：  我的finish跟valid 跟encode直接用wire assign 為：  assign encode = 1'd1;  assign finish = (state==OUT & cnt==12'd2056) ? 1'b1 : 1'b0;  assign valid = (state == OUT) ? 1'b1 : 1'b0;  search buffer： 跟題目敘述一樣，是一個 9(個字元)\*8(bit)的2d register  lookahead buffer：我把lookahead擴大變成一個 8+9(個字元)\*8(bit)的2d register  offset： match到最大的length字串的第一位元是位於search buffer的哪一個位置  cnt：一開始是用來計算input到第幾個char的reg, 後來用來計算現在img第幾個char進入到lookahead buffer了。  一開始reset訊號起來的時候state進入Idle, image全部先initial為 0， search buffer, lookahead buffer, offset, match\_len, char\_nxt, cnt全部初始化為0的數值/陣列。用sequential的方式update這些訊號跟buffer。  *接著關於State*:  我的state總共有四個：IDLE, INPUT, CALC, OUT，  IDLE初始化buffer跟訊號，Input的時候把整張image讀進來，  INPUT會count到 2048，進入CALC。  CALC為整個計算的主要State:  第一個迴圈j指標會iterate search buffer尋找從哪一個地方開始，從0->8遞迴，若search buffer[j] == lookahead[0] 就進入算match length 的迴圈。  第二個迴圈k計算match的長度，假設對到為目前的max length則max\_len = k, max\_index = j, offset則是8 - max\_index（原本陣列相反過來計算）。  我用break\_j, break\_k控制是否跳出內、外迴圈。  如果k==8了，就兩個都設為1->跳出內外迴圈。  如果只是裡面迴圈的match結束了，就跳出內部迴圈。  比對的部分我拆為在search buffer/ lookahead buffer VS lookahead buffer。  如果j+k<9 -> 在search buffer跟lookahead buffer計算match 長度。  否則就在lookahead buffer跟lookahead buffer比對長度。  OUT狀態會output CALC的結果，然後更新下一輪的buffer給CALC繼續際算。  下一輪的lookahead buffer在 j（現在在更新哪一個位置）+match\_len<7的時候會等於lookahead[match\_len+j+1], 否則就是img[cnt+match\_len+j-6]。  下一輪的search buffer會是search[j+match\_len+j+1] 當j（現在在更新哪一個位置）<8, 否則就是lookahead[j+match\_len-8]。  Encoder有latch所以無法合成, 目前還沒有想到可以不latch的合成，所以encoder就先到這邊。  **Decoder:**  Decoder只有兩個State: Idle, Calc，因為一開始的版本如果分成Idle, Calc, Output好像很容易會用到產生 latch的寫法。  Input的code pos, code len, char\_nxt要用register存起來，因為他在負緣給資料很容易影響訊號正確性。  首先是兩個訊號直接wire assign：assign encode = 1'd0;  assign n\_finish = (char\_nxt == EndToken) ? 1'b1 : 1'b0;  我的calculate一個clk就推一次buffer，推到cnt==len就進入下一個output輪迴，  每次推的新的data是現在位在offset位置的data，buffer其他位置則是上一次buffer的下一格。這樣offset會一直指著該進入的下一個正確數字。  前面幾個版本想做一次在calculate state算完所有buffer該填的數值，但結論是幾乎都可以合成，但都有訊號會latch  最後這個是最精簡最少判斷條件的，這個作業真的有點難，但感覺進步好多，讚。 | | | | | | | |

*Scoring = (Total logic elements + total memory bit + 9\*embedded multiplier 9-bit element)*