Department of Computer Science and Information Engineering
National Cheng Kung University

數位IC設計 作業三詳解

Note

- 本次作業分為Encoder端與Decoder端，在本投影片中會分別對其Verilog code進行說明
- 作業三中，LZ77的search buffer長度為9，look-ahead buffer長度為8，請同學了解此兩者長度對程式撰寫的影響，以利於準備期末考試
- State Machine為電路運作的一大重點，請同學務必了解本作業State的跳轉
- 範例程式將一同公告於Moodle，期末考時同學可參考自己撰寫的程式與助教提供的範例

Department of Computer Science and Information Engineering
National Cheng Kung University

數位IC設計 作業三詳解

Encoder 端

LZ77_Encoder.v

- Module與IO腳位宣告
  - 各腳位詳細功能請參考作業三之題目說明(2022_hw3.pdf)

```
1  module LZ77_Encoder(clk,reset,chardata,valid,encode,finish,offset,match_len,char_nxt);
2
3  input               clk;        // 時脈訊號，本電路同步於時脈正緣
4  input               reset;      // 高位準非同步重置訊號，此訊號為高電位時進行重置
5  input       [7:0]   chardata;   // 待編碼字元，除了終止符號$外前4 bits固定為0
6  output reg          valid;      // 輸出編碼有效訊號
7  output              encode;     // Encoder端此訊號固定為高電位
8  output reg          finish;     // 編碼結束訊號，此訊號拉高後結束模擬
9  output reg  [3:0]   offset;     // 匹配字串開頭字元與search buffer起始位置間的offset
10 output reg  [2:0]   match_len;  // 匹配字串長度，最大為look-ahead buffer長度-1
11 output reg  [7:0]   char_nxt;   // 匹配字串後的下一個字元
```

- 變數宣告

```
14  reg         [1:0]   current_state, next_state;
15  reg         [11:0]  counter;
16  reg         [3:0]   search_index;               // 記錄匹配字串起始位置
17  reg         [2:0]   lookahead_index;            // 記錄look-ahead buffer匹配字串結束位置
18  reg         [3:0]   str_buffer  [2047:0];       // 儲存輸入字元，其開頭同時作為look-ahead buffer
19  reg         [3:0]   search_buffer   [8:0];      // Search buffer
20
21  wire                equal   [7:0];              // 判斷待匹配字串與look-ahead buffer中的字串是否相符
22  wire        [11:0]  current_encode_len;         // 當前完成編碼的長度
23  wire        [2:0]   curr_lookahead_index;       // look-ahead buffer匹配字串結束的下一個位置
24  wire        [3:0]   match_char [6:0];           // 待匹配字串，可能來自search buffer或look-ahead buffer
25
26  parameter [1:0] IN=2'b00, ENCODE=2'b01, ENCODE_OUT=2'b10, SHIFT_ENCODE=2'b11; // 本電路包含4個State
```

LZ77_Encoder.v

- 電路重置
  - 當reset訊號為high時需立刻進行非同步重置

```
55  always @(posedge clk or posedge reset)
56  begin
57      if(reset)
58      begin
59          current_state <= IN;
60          counter <= 12'd0;
61          search_index <= 4'd0;
62          lookahead_index <= 3'd0;
63          valid <= 1'b0;
64          finish <= 1'b0;
65          offset <= 4'd0;
66          match_len <= 3'd0;
67          char_nxt <= 8'd0;
68
69          search_buffer[0] <= 4'd0;
70          search_buffer[1] <= 4'd0;
71          search_buffer[2] <= 4'd0;
72          search_buffer[3] <= 4'd0;
73          search_buffer[4] <= 4'd0;
74          search_buffer[5] <= 4'd0;
75          search_buffer[6] <= 4'd0;
76          search_buffer[7] <= 4'd0;
77          search_buffer[8] <= 4'd0;
78      end
```

同步於時脈正緣

高位準非同步重置

對register進行初始化

若未進行初始化且該register會影響到電路的control path時可能會發生錯誤，建議在對data path與control path的區別不熟悉時所有register都一律初始化

LZ77_Encoder.v

- reset結束後透過state machine控制電路功能
  - 利用組合電路判斷next_state，並在時脈正緣時將判斷結果傳給current_state(循序電路)

```
79      else
80      begin
81          current_state <= next_state;
```

循序電路 (always@(posedge clk or posedge reset))

```
141 always @(*)
142 begin
143     case(current_state)
144         IN:
145         begin
146             next_state = (counter==2047) ? ENCODE : IN;
147         end
148         ENCODE:
149         begin
150             next_state = (search_index==15 || match_len==7) ? ENCODE_OUT : ENCODE;
151         end
152         ENCODE_OUT:
153         begin
154             next_state = SHIFT_ENCODE;
155         end
156         SHIFT_ENCODE:
157         begin
158             next_state = (lookahead_index==0) ? ENCODE : SHIFT_ENCODE;
159         end
160         default:
161         begin
162             next_state = IN;
163         end
164     endcase
165 end
```

組合電路 (always@(*))

LZ77_Encoder.v

- State Machine - IN
  - 循序電路負責讀取testbench輸入的字元

```
83      case(current_state)
84          IN:
85          begin
86              str_buffer[counter] <= chardata[3:0];
87              counter <= (counter==2047) ? 0 : counter+1;
88          end
```

將testbench傳入之chardata存入str_buffer，因前4 bits固定為0，只儲存後4 bits (chardata[3:0])

透過counter記錄接收的字元數量，0~2047共2048個字元，若輸入字元數不同則須更改判斷條件與counter的bit數(本範例中counter後續需數到2049，故宣告為12 bits)

  - 組合電路中判斷state machine跳轉條件

```
143     case(current_state)
144         IN:
145         begin
146             next_state = (counter==2047) ? ENCODE : IN;
147         end
```

如果已讀完2048個字元(counter == 2047)，則進入下一個state(ENCODE)，否則留在當前state(IN)

LZ77_Encoder.v

- State Machine - ENCODE
  - 負責判斷匹配情況，並進行編碼(更改char_nxt, match_len, offset)，同時利用lookahead_index記錄look-ahead buffer中有幾個字元被匹配，後續需將匹配數量的字元移至search buffer

```
89              ENCODE:
90              begin
91                  if(equal[match_len]==1 && search_index < counter && current_encode_len <= 2048)
92                  begin
93                      char_nxt <= str_buffer[curr_lookahead_index];
94                      match_len <= match_len+1;
95                      offset <= search_index;
96
97                      lookahead_index <= curr_lookahead_index;
98                  end
99                  else
100                 begin
101                     search_index <= (search_index==15) ? 0 : search_index-1;
102                 end
103             end
```

匹配成功：
進行編碼並修改lookahead_index的值

匹配失敗：
修改search_index，從search buffer的不同起始位置開始匹配
當search_index等於15時，代表其已經從0減1而underflow，已完成所有起始位置的判斷

  - 每次匹配成功，match_len就會加1
  - 藉由equal[match_len]判斷新字串的匹配情況 (1代表匹配成功)
  - 匹配失敗時則需從不同的search_index以相同的match_len進行匹配
  - 因search_buffer左邊的字元有優先匹配的權力，故search_index會由8逐步減少 (詳見後面的search buffer與look-ahead buffer示意圖)

LZ77_Encoder.v

- State Machine – ENCODE

```
33  assign  match_char[0] = search_buffer[search_index];
34  assign  match_char[1] = (search_index >= 1) ? search_buffer[search_index-1] : str_buffer[search_index];
35  assign  match_char[2] = (search_index >= 2) ? search_buffer[search_index-2] : str_buffer[1-search_index];
36  assign  match_char[3] = (search_index >= 3) ? search_buffer[search_index-3] : str_buffer[2-search_index];
37  assign  match_char[4] = (search_index >= 4) ? search_buffer[search_index-4] : str_buffer[3-search_index];
38  assign  match_char[5] = (search_index >= 5) ? search_buffer[search_index-5] : str_buffer[4-search_index];
39  assign  match_char[6] = (search_index >= 6) ? search_buffer[search_index-6] : str_buffer[5-search_index];
40
41  assign  equal[0] = (search_index <= 8) ? ((match_char[0]==str_buffer[0]) ? 1'b1 : 1'b0) : 1'b0;
42  assign  equal[1] = (search_index <= 8) ? ((match_char[1]==str_buffer[1]) ? equal[0] : 1'b0) : 1'b0;
43  assign  equal[2] = (search_index <= 8) ? ((match_char[2]==str_buffer[2]) ? equal[1] : 1'b0) : 1'b0;
44  assign  equal[3] = (search_index <= 8) ? ((match_char[3]==str_buffer[3]) ? equal[2] : 1'b0) : 1'b0;
45  assign  equal[4] = (search_index <= 8) ? ((match_char[4]==str_buffer[4]) ? equal[3] : 1'b0) : 1'b0;
46  assign  equal[5] = (search_index <= 8) ? ((match_char[5]==str_buffer[5]) ? equal[4] : 1'b0) : 1'b0;
47  assign  equal[6] = (search_index <= 8) ? ((match_char[6]==str_buffer[6]) ? equal[5] : 1'b0) : 1'b0;
48  assign  equal[7] = 1'b0;
```

equal[0]為1表示待匹配字串第一個字元匹配正確

equal[1]為1表示待匹配字串第一與第二個字元匹配正確

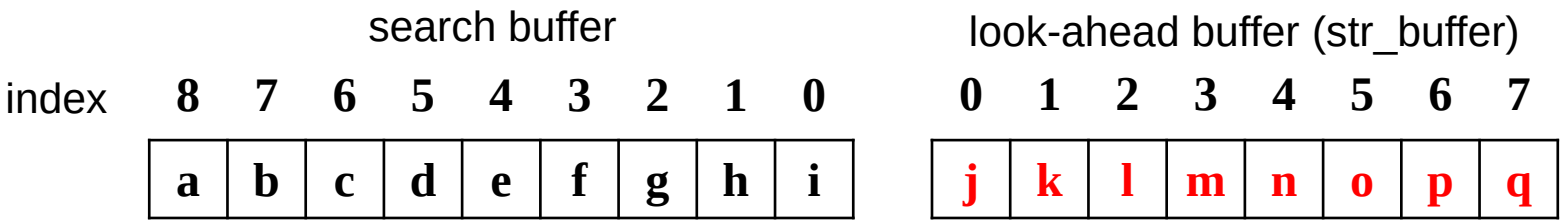
以此類推

- 匹配是否成功需判斷equal[match_len]是否等於1
- match_char[0]~match_char[6]為待匹配字串，str_buffer[0]~str_buffer[6]則為look-ahead buffer開頭的字串
- 因look-ahead buffer長度為8，因此匹配長度最長為7(匹配7個字元+1個next_char)，equal[7]默認為0，若look-ahead buffer長度改變，則需增加match_char與equal
- equal[n]為1的條件除了匹配正確外(match_char[n]==str_buffer[n])，equal[n-1]~equal[0]也必須都為1，同時search_index不可超過search_buffer長度(search_index <= 8)
- match_len會記錄此輪編碼中最長的匹配數量，直接判斷equal[match_len]即可知道當前進行匹配的字串是否超過先前成功匹配的長度

LZ77_Encoder.v

- State Machine – ENCODE

```
33  assign  match_char[0] = search_buffer[search_index];
34  assign  match_char[1] = (search_index >= 1) ? search_buffer[search_index-1] : str_buffer[search_index];
35  assign  match_char[2] = (search_index >= 2) ? search_buffer[search_index-2] : str_buffer[1-search_index];
36  assign  match_char[3] = (search_index >= 3) ? search_buffer[search_index-3] : str_buffer[2-search_index];
37  assign  match_char[4] = (search_index >= 4) ? search_buffer[search_index-4] : str_buffer[3-search_index];
38  assign  match_char[5] = (search_index >= 5) ? search_buffer[search_index-5] : str_buffer[4-search_index];
39  assign  match_char[6] = (search_index >= 6) ? search_buffer[search_index-6] : str_buffer[5-search_index];
```



- search_index起始的字串不足7個字元時，由look-ahead buffer(str_buffer)中的字元補足
- Ex: search_index = 4
  - match_char[0]~match_char[4]即為search_buffer[4]~search_buffer[0]
  - match_char[5]~match_char[6]則為str_buffer[0]~str_buffer[1]
  - 最終的待匹配字串match_char為efghijk

LZ77_Encoder.v

- State Machine - ENCODE
  - 匹配判斷條件說明

```
89              ENCODE:
90              begin
91                  if(equal[match_len]==1 && search_index < counter && current_encode_len <= 2048)
92                  begin
93                      char_nxt <= str_buffer[curr_lookahead_index];
94                      match_len <= match_len+1;
95                      offset <= search_index;
96
97                      lookahead_index <= curr_lookahead_index;
98                  end
```

在IN以外的state中，counter代表已完成編碼並輸出的字元數量

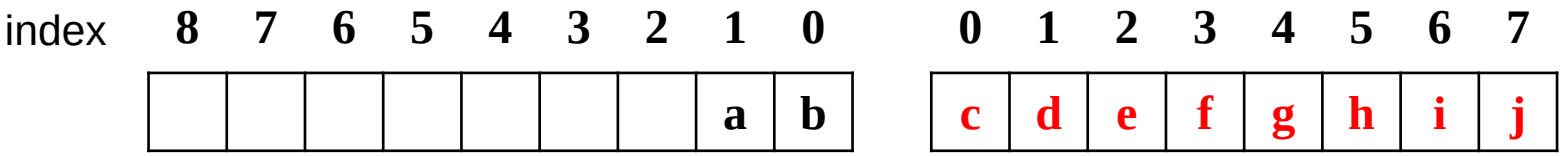
current_encode_len為已完成編碼並輸出的字元數量，再加上當前進行編碼中字元數量

```
51    assign  current_encode_len = counter+match_len+1;
```

因在匹配字串後還有next_char會一同被編碼，故需加1

search buffer

look-ahead buffer (str_buffer)



- search_index >= counter代表當前search_buffer中的字元數量不足，因此匹配失敗
- 以上圖為例，此時完成編碼的字元僅有a,b兩個，counter = 2，若search_index = 2~8則匹配失敗
- current_encode_len > 2048時代表所有輸入都已完成編碼，因此編碼結束
- 匹配成功時，equal[0]~equal[match_len]等於1，實際上有match_len+1個字元被匹配，因此match_len需加1

LZ77_Encoder.v

- State Machine – ENCODE state跳轉條件

```
148         ENCODE:
149         begin
150             next_state = (search_index==15 || match_len==7) ? ENCODE_OUT : ENCODE;
151         end
```

組合電路 (always@(*))

  - 當search_index等於15時，代表其已經從0減1而underflow，已完成所有起始位置的判斷
  - 當match_len等於7時，已達到最長匹配長度，因search_index已有考慮優先順序，故可直接結束ENCODE階段
  - 上述兩種情形任一項符合時，由ENCODE state跳轉至ENCODE_OUT state，否則留在當前state

LZ77_Encoder.v

- State Machine – ENCODE_OUT
  - 負責輸出編碼結果

```
104             ENCODE_OUT:
105             begin
106                 valid <= 1;
107                 // offset <= offset;
108                 // match_len <= match_len;
109                 char_nxt <= (current_encode_len==2049) ? 8'h24 : (match_len==0) ? str_buffer[0] : char_nxt;
110                 counter <= current_encode_len;
111             end
```

循序電路

```
152         ENCODE_OUT:
153         begin
154             next_state = SHIFT_ENCODE;
155         end
```

組合電路 (always@(*))

  - 輸出時valid須設為1
  - match_len與offset在ENCODE state已記錄完成，故維持原值
  - char_nxt的值分為三種情況
    - current_encode_len等於2049，代表所有字元皆已編碼，char_nxt等於8'h24($字符)
    - match_len等於0，表示沒有任何字元匹配，char_nxt為look-ahead buffer的第一個字元(str_buffer[0])
    - 若非上述兩種情況，char_nxt為ENCODE state記錄的值
  - ENCODE_OUT state固定為1個cycle，無須判斷直接跳轉至SHIFT_ENCODE state

LZ77_Encoder.v

- State Machine – SHIFT_ENCODE
  - 負責重置register，將已編碼字元移至search_buffer，並且將str_buffer中的字元往前移

```
112             SHIFT_ENCODE:
113             begin
114                 finish <= (counter==2049) ? 1 : 0;
115                 offset <= 0;
116                 valid <= 0;
117                 match_len <= 0;
118                 search_index <= 8;
119                 lookahead_index <= (lookahead_index==0) ? 0 : lookahead_index-1;
120
121                 search_buffer[8] <= search_buffer[7];
122                 search_buffer[7] <= search_buffer[6];
123                 search_buffer[6] <= search_buffer[5];
124                 search_buffer[5] <= search_buffer[4];
125                 search_buffer[4] <= search_buffer[3];
126                 search_buffer[3] <= search_buffer[2];
127                 search_buffer[2] <= search_buffer[1];
128                 search_buffer[1] <= search_buffer[0];
129                 search_buffer[0] <= str_buffer[0];
130
131                 for (i=0; i<2047; i=i+1) begin
132                     str_buffer[i] <= str_buffer[i+1];
133                 end
134             end
```

每個cycle往前移動一個字元，藉由lookahead_index記錄共須移動幾個字元

將 search_buffer[7]~search_buffer[0] 往前移至 search_buffer[8]~search_buffer[1]，look-ahead buffer第一個字元(str_buffer[0])則存入search_buffer[0]

str_buffer中的所有字元往前移動一個位置
verilog中for迴圈會複製電路，因此共會產生2047個shift電路(str_buffer[i] <= str_buffer[i+1])

須在確定要產生多份相似電路時才可使用for迴圈語法，否則應以多個cycle完成task來減少資源消耗，請謹慎使用

LZ77_Encoder.v

- State Machine – SHIFT_ENCODE
  - 負責重置register，將已編碼字元移至search_buffer，並且將str_buffer中的字元往前移

```
112             SHIFT_ENCODE:
113             begin
114                 finish <= (counter==2049) ? 1 : 0;
115                 offset <= 0;
116                 valid <= 0;
117                 match_len <= 0;
118                 search_index <= 8;
119                 lookahead_index <= (lookahead_index==0) ? 0 : lookahead_index-1;
```

counter等於2049表示所有字元完成編碼，finish拉高結束模擬

valid須設為0，否則會繼續輸出編碼

search_index設為8，由search buffer左邊到右邊進行匹配

```
156         SHIFT_ENCODE:
157         begin
158             next_state = (lookahead_index==0) ? ENCODE : SHIFT_ENCODE;
159         end
```

組合電路 (always@(*))

  - lookahead_index等於0代表所有完成編碼的字元皆已移入search buffer，因此回到ENCODE state進行下一輪編碼，否則維持原state (SHIFT_ENCODE)
  - 不可在輸出最後一組編碼時拉高finish
    - 在ENCODE_OUT state輸出最後一組編碼後，於SHIFT_ENCODE state拉高finish

Department of Computer Science and Information Engineering

National Cheng Kung University

數位IC設計 作業三詳解

Decoder 端

LZ77_Decoder.v

- Module與IO腳位宣告
  - 各腳位詳細功能請參考作業三之題目說明(2022_hw3.pdf)

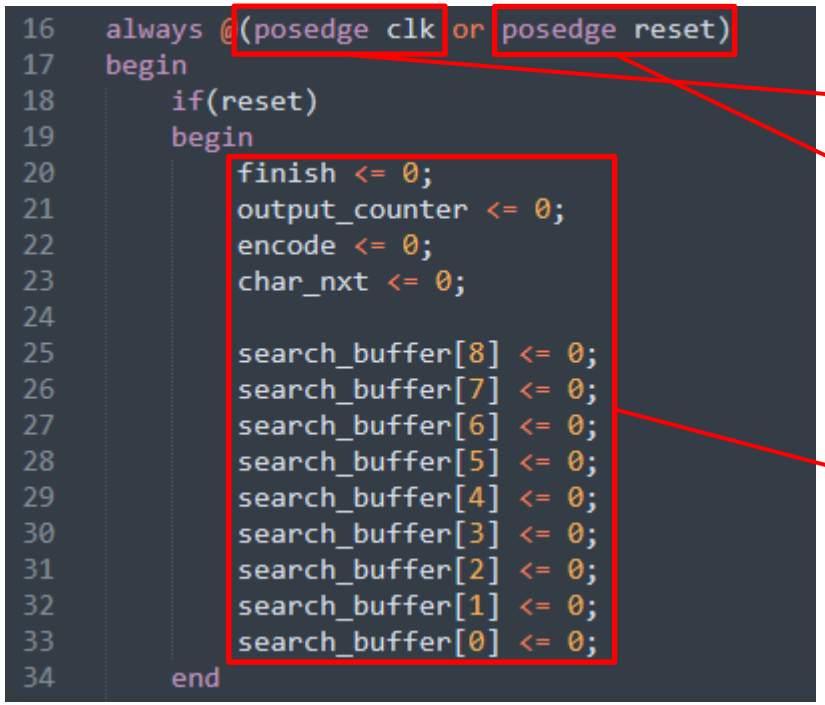
```
1   module LZ77_Decoder(clk,reset,code_pos,code_len,chardata,encode,finish,char_nxt);
2
3   input               clk;        // 時脈訊號，本電路同步於時脈正緣
4   input               reset;      // 高位準非同步重置訊號，此訊號為高電位時進行重置
5   input       [3:0]   code_pos;   // 匹配字串在search buffer中的起始位置
6   input       [2:0]   code_len;   // 匹配字串長度
7   input       [7:0]   chardata;   // 匹配字串後的下一個字元
8   output  reg         encode;     // Decoder端此訊號固定為低電位
9   output  reg         finish;     // 解碼結束訊號，此訊號拉高後結束模擬
10  output  reg [7:0]   char_nxt;   // 解碼得到的字元
```

- 變數宣告

```
12   reg         [2:0]   output_counter;     // 記錄解碼出的字串長度
13   reg         [3:0]   search_buffer[8:0]; // Search buffer
```

LZ77_Decoder.v

- 電路重置
  - 當reset訊號為high時需立刻進行非同步重置



```
16  always @(posedge clk or posedge reset)
17  begin
18      if(reset)
19      begin
20          finish <= 0;
21          output_counter <= 0;
22          encode <= 0;
23          char_nxt <= 0;
24
25          search_buffer[8] <= 0;
26          search_buffer[7] <= 0;
27          search_buffer[6] <= 0;
28          search_buffer[5] <= 0;
29          search_buffer[4] <= 0;
30          search_buffer[3] <= 0;
31          search_buffer[2] <= 0;
32          search_buffer[1] <= 0;
33          search_buffer[0] <= 0;
34      end
```

同步於時脈正緣

高位準非同步重置

對register進行初始化

若未進行初始化且該register會影響到電路的control path時可能會發生錯誤，建議在對data path與control path的區別不熟悉時所有register都一律初始化

LZ77_Decoder.v

- reset結束後於每個cycle正緣輸出解碼得到的字元(char_nxt)

```
35      else
36      begin
37          char_nxt <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
38          search_buffer[8] <= search_buffer[7];
39          search_buffer[7] <= search_buffer[6];
40          search_buffer[6] <= search_buffer[5];
41          search_buffer[5] <= search_buffer[4];
42          search_buffer[4] <= search_buffer[3];
43          search_buffer[3] <= search_buffer[2];
44          search_buffer[2] <= search_buffer[1];
45          search_buffer[1] <= search_buffer[0];
46
47          search_buffer[0] <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
48          output_counter <= (output_counter == code_len) ? 0 : output_counter+1;
49          finish <= (char_nxt==8'h24) ? 1 : 0;
50      end
```

  - char_nxt之值分為兩種情況
    - 當output_counter不等於code_len時，代表search_buffer中有匹配的字串還未完成輸出，char_nxt的值為search_buffer[code_pos]
    - 當output_counter等於code_len時，代表匹配字串已完成輸出，char_nxt的值為匹配字串後的下一個字元(chardata)

LZ77_Decoder.v

- reset結束後於每個cycle正緣輸出解碼得到的字元(char_nxt)

```
35      else
36      begin
37          char_nxt <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
38          search_buffer[8] <= search_buffer[7];
39          search_buffer[7] <= search_buffer[6];
40          search_buffer[6] <= search_buffer[5];
41          search_buffer[5] <= search_buffer[4];
42          search_buffer[4] <= search_buffer[3];
43          search_buffer[3] <= search_buffer[2];
44          search_buffer[2] <= search_buffer[1];
45          search_buffer[1] <= search_buffer[0];
46
47          search_buffer[0] <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
48          output_counter <= (output_counter == code_len) ? 0 : output_counter+1;
49          finish <= (char_nxt==8'h24) ? 1 : 0;
50      end
```

  - 當前輸出的編碼需存入search_buffer中
  - 每個cycle將search_buffer[7]~search_buffer[0]移動至search_buffer[8]~search_buffer[1]
  - search_buffer[0]存入解碼得到的字元(判斷同char_nxt)
  - output_counter記錄當前解碼出的字串長度，等於code_len時歸零以準備處理下一組碼
  - 當上一個cycle輸出的字元為$字符時(char_nxt等於8'h24)，拉高finish訊號結束模擬
    - cycle n執行char_nxt <= 8'h24後，cycle n+1時char_nxt會等於8'h24

LZ77_Decoder.v

- reset結束後於每個cycle正緣輸出解碼得到的字元(char_nxt)

```
35      else
36      begin
37          char_nxt <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
38          search_buffer[8] <= search_buffer[7];
39          search_buffer[7] <= search_buffer[6];
40          search_buffer[6] <= search_buffer[5];
41          search_buffer[5] <= search_buffer[4];
42          search_buffer[4] <= search_buffer[3];
43          search_buffer[3] <= search_buffer[2];
44          search_buffer[2] <= search_buffer[1];
45          search_buffer[1] <= search_buffer[0];
46
47          search_buffer[0] <= (output_counter == code_len) ? chardata : search_buffer[code_pos];
48          output_counter <= (output_counter == code_len) ? 0 : output_counter+1;
49          finish <= (char_nxt==8'h24) ? 1 : 0;
50      end
```

  - 因search_buffer中的字串會不斷向前推，因此輸出匹配字串時只需要輸出固定位置的字元(search_buffer[code_pos])
  - 詳見下一頁的解碼過程範例

LZ77_Decoder.v

- 解碼範例(假設search_buffer中的字串為"112a"，輸入編碼為(3, 4, 2)，code_pos = 3，chardata = 2)