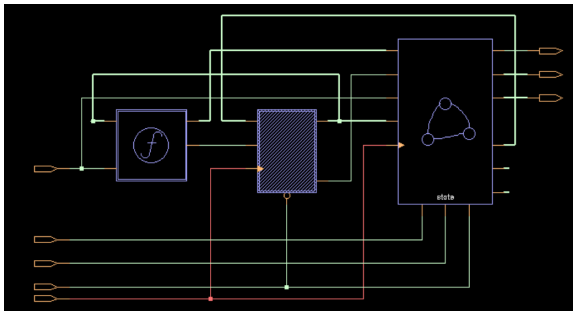
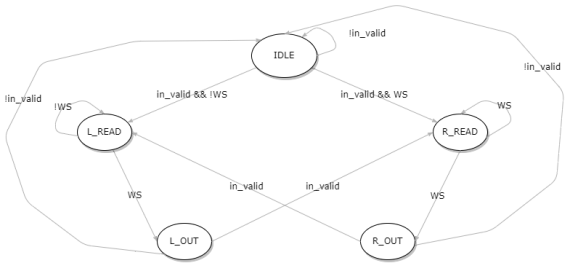
數電HW2結報

此電路為一個有限狀態機，分為IDLE(停止狀態)、R_READ/L_READ(輸入狀態，分為WS=1和WS=0兩種，以便之後輸出時決定輸出的port為哪一個)、R_OUT/L_OUT(用右邊或左邊的port輸出)

Block diagram & state diagram





在輸入資料方面，IDLE、R_READ、L_READ都有可能出現in_valid == 0或in_valid==1的情況，所以三個狀態都要根據in_valid的值去判斷是否要輸入SD的資料到input_buffer中，並將下個clk cycle 的input_buffer值存在next_val中。若現在輸入不合法則將next_val設為0。

```
if(!in_valid) begin
  next_val = 0;
end
else  begin
  next_val = shifted_input_1; // shifted_input_1 = {input_buffer[30:0], SD};
end
```

在狀態為 L_OUT, R_OUT 時，如果 in_valid == 1，代表狀態轉為 L_OUT, R_OUT 的該clk cycle中的SD的值應該要在輸出完成後被存到 input_buffer 中，但因為 out_left 和 out_right 輸出時是直接從input_buffer中取值，若在輸出的該時脈週期就輸入SD，改變 input_buffer ，會導致輸出的值錯誤。所以我們需要一個額外的 DFF—SD_buffer 來儲存輸出時的SD輸入值。當輸出完畢，狀態從L_OUT/R_OUT 轉為 R_READ/L_READ 時將上一個時脈的SD值和此時脈的SD值一起存進 input_buffer。

```
casez(in_valid)
  0: begin
    next_val = 0;
  end
  1: begin
```

```
    next_val = {30'd0, SD_buffer, SD};
  end
  default: begin
    next_val = 32'bx;
  end
endcase
```

輸出值的管理則相對簡單，如果狀態為 L_OUT, R_OUT ，則直接輸出input_buffer的數值到out_left或out_right，並拉起out_valid，否則三者皆為0。若出現未被考慮的state則會導致unknown，讓我開發時可以檢視程式的邏輯是否有缺陷。

```
casez(state)
  IDLE: begin
    /* output handling*/
  end
  L_READ: begin
    /* output handling*/
  end
  R_READ: begin
    /* output handling*/
  end
  L_OUT: begin
    /* output handling*/
  end
  R_OUT: begin
    /* output handling*/
  end
  default: begin // Unexpected case will result in unkonown output
    out_valid = 1'bx;
    out_left = 32'bx;
    out_right = 32'bx;
    next = 3'bx;
    next_val = 32'bx;
  end
endcase
```

心得:

這次作業我一開始並不是用有限狀態機實現的，因此coding之前花了一整個下午整理出所有的判斷條件，結果面積不甚理想，後來換成狀態機後，輸出的邏輯判斷瞬間變得簡單許多，也使面積降低一半。通過這次作業，我對有限狀態機的設計有了更深刻的理解，學會了如何在設計中充分考慮各種情況並予以處理。