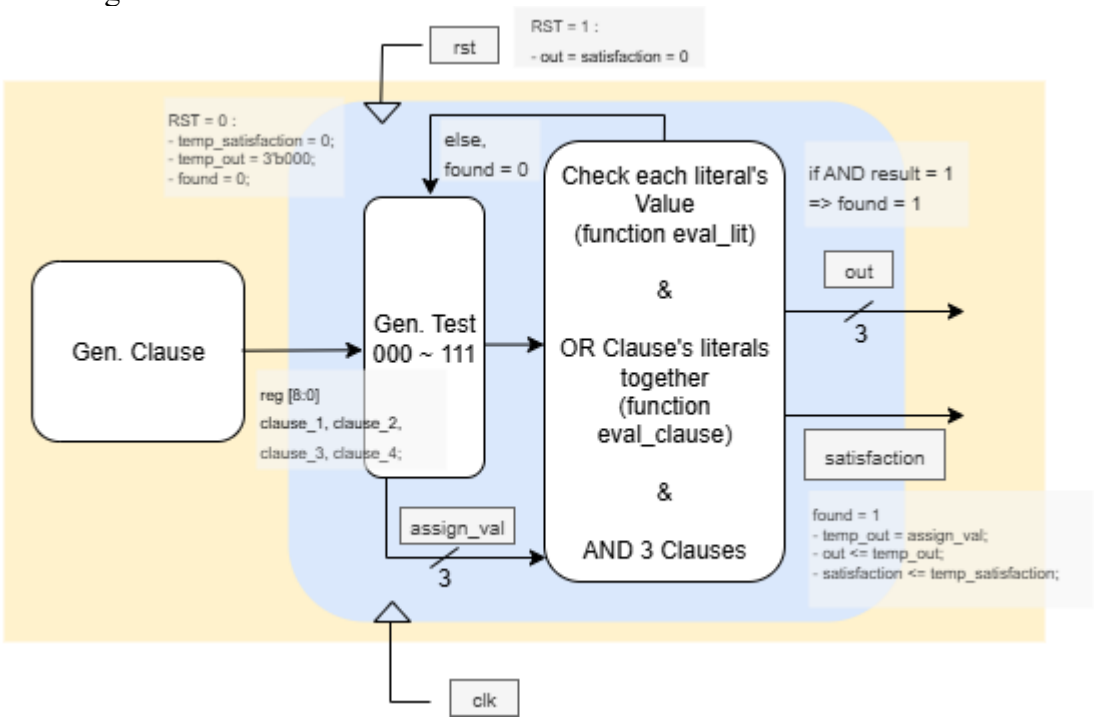
Lab 2

學號: 111062118 姓名: 江佩霖

A. Lab Implementation

1. Lab2_Adv1

(1) Block diagram



(2)-1 透過 function 的方式撰寫重複的檢查/賦值

```
function eval_lit;
       input [2:0] lit;      // literal encoding
       input [2:0] val;      // {A,B,C}
       begin
           case(lit)
               3'b000: eval_lit = val[0];     // A
               3'b100: eval_lit = ~val[0];    // ~A
               3'b001: eval_lit = val[1];     // B
               3'b101: eval_lit = ~val[1];    // ~B
               3'b010: eval_lit = val[2];     // C
               3'b110: eval_lit = ~val[2];    // ~C
               default: eval_lit = 1'b0;      // invalid literal = false
           endcase
       end
   endfunction
```

```
    // function: evaluate one clause
   function eval_clause;
       input [8:0] clause;
```

```
    input [2:0] val;
    reg l1, l2, l3;
    begin
        l1 = eval_lit(clause[8:6], val);
        l2 = eval_lit(clause[5:3], val);
        l3 = eval_lit(clause[2:0], val);
        eval_clause = l1 | l2 | l3;
    end
endfunction
```

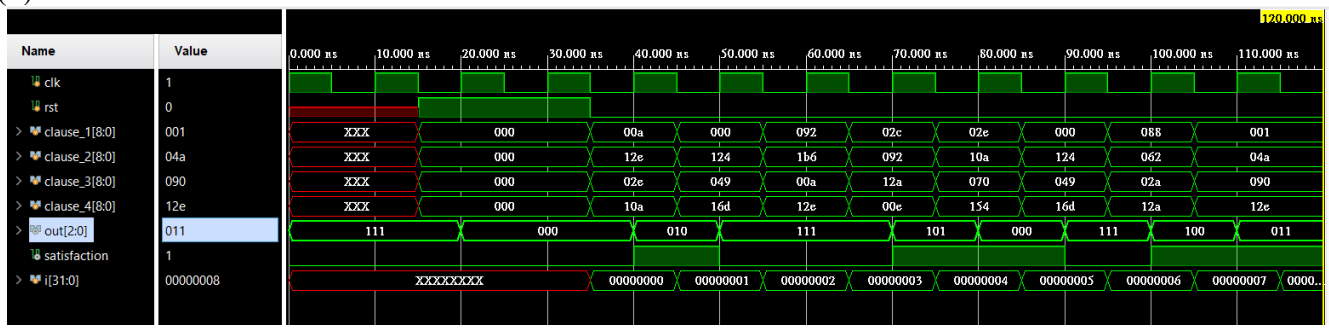
(2)-2 在 test file 中，使用 for 回圈內嵌 case 的方式依序輸入指定的 0~7 clause cases。並且在 initial block 中加上適當的延遲以符合題目的要求。(詳細說明如註解)

```
initial begin
    @(negedge clk); // let rst signal start at 15ns
    @(negedge clk);
    rst = 1;
    clause_1 = 0; clause_2 = 0; clause_3 = 0; clause_4 = 0;
    @(negedge clk);
    @(negedge clk); // let rst signal work for 20ns
    rst = 0;

    for (i = 0; i < 8; i = i + 1) begin
        clause_1 = 0; clause_2 = 0; clause_3 = 0; clause_4 = 0;
        rst = 0;
        case (i)
            0: begin
                clause_1 = 9'b000_001_010;
                clause_2 = 9'b100_101_110;
                clause_3 = 9'b000_101_110;
                clause_4 = 9'b100_001_010;
            end
            1: begin
                clause_1 = 9'b000_000_000;
                clause_2 = 9'b100_100_100;
                clause_3 = 9'b001_001_001;
                clause_4 = 9'b101_101_101;
            end
            //這邊省略 2~6 case code
            7: begin
                clause_1 = 9'b000_000_001;
                clause_2 = 9'b001_001_010;
                clause_3 = 9'b010_010_000;
                clause_4 = 9'b100_101_110;
```

```
            end
        endcase
        @(negedge clk); // wait for a clock cycle
    end
    @(posedge clk); // let simulation run til 120000ns.
    $display("All test cases done.");
    $finish;
end
```

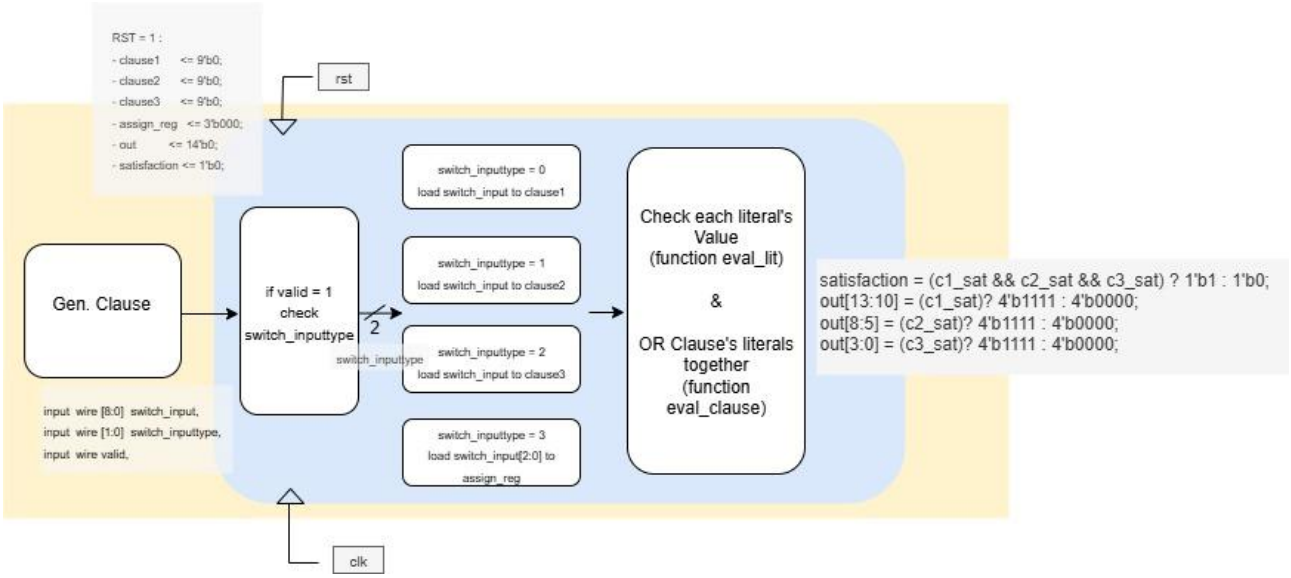
(4)Wave form



- Testing process :
  (i) check rst work
  (ii) check load clause time correct, with function call has right result (by $display value)
  (iii) check output update at right time.
  (ix) check all signal in wave form works correctly

2. Lab2_Adv_2

(1)Block diagram



(2)-1 同 Lab2_1，使用 function 撰寫重複的檢查/賦值(code 略)

(2)-2 如題述，監聽的 clock edge 需同時檢查 clock 以及 reset 的 posedge。

```
always @(posedge clk or posedge rst) begin
// skip code here
end
```

(2)-3 由於 output 的 update 具有時間關係(有順序性，需使用 blocking 方式賦值)

```
case (switch_inputtype)
     2'b00: clause1    = switch_input;
     2'b01: clause2    = switch_input;
     2'b10: clause3    = switch_input;
     2'b11: assign_reg = switch_input[2:0]; // only SW2..SW0 used
     default: ;
   endcase
   // note: outputs are computed in combinational block below
   c1_sat = eval_clause(clause1, assign_reg);
   c2_sat = eval_clause(clause2, assign_reg);
   c3_sat = eval_clause(clause3, assign_reg);
   // final satisfaction output
   satisfaction = (c1_sat && c2_sat && c3_sat) ? 1'b1 : 1'b0;
   out[13:10] = (c1_sat)? 4'b1111 : 4'b0000;
   out[8:5] = (c2_sat)? 4'b1111 : 4'b0000;
   out[3:0] = (c3_sat)? 4'b1111 : 4'b0000;
```

B. Questions and Discussions

1. What are the differences between a combinational circuit and a sequential circuit? Please explain how each of them works in detail.

Ans.

Combinational: Outputs depend only on present inputs. No memory. Example: adder, multiplexer.
Sequential: Outputs depend on present inputs and past state stored in flip-flops/latches. Example: counter, FSM.

2. For an unsatisfiable 3-SAT problem, how would you extend your Verilog design to solve the MAX-SAT problem (find a Boolean assignment that satisfies the maximum number of clauses)?

Ans.

Suppose to use the design stucture below :

- Clause evaluator: For each clause, compute whether it is satisfied.
- Popcount: Count how many clauses are satisfied → score.
- Best register: Store highest score and corresponding assignment.
- Control FSM: Generate candidate assignments (brute force with a counter for small n, or heuristic search like WalkSAT for large n).
- Update: If score > best, save as new best. After search ends, output assignment_best.
- Optimization: Use Gray code to change one variable at a time and update only affected clauses. Parallel evaluation is possible on FPGA for speed.

3. In lab2_adv_2, what unexpected behavior might occur if the valid signal is removed? Explain why?

Ans.

Without valid, inputs from switches are sampled asynchronously.
This can cause glitches, partial updates, or metastability in flip-flops.
Outputs may flicker or show inconsistent results while switches change.
valid ensures inputs are only latched at the right clock edge.

C. Problem Encountered

1. Lab2_1

第一題沒遇到什麼問題，只有最後檢查 wave form 才發現輸入輸出的 clk edge 設定還有整個 simulation 要結束在 120000ns 等等，因此花了一些時間調整，還有發現我的 out 寫反…
(以下是我截圖的 wave form 進展史)。

2. Lab2_2

設計這個電路時，當下沒有特別想要進行 debug（因為寫錯板子就是不會亮 😊，而且也不太想寫 test file 去測試…）。
因此有遇到一個問題，最初的設計用了兩個 always，一個主要負責驅動並更新 out 的值，另一個則依據 clk 與 rst 來決定更新與載入新 clause 的時機。不過在實作後發現這樣很難確認 signal 是否被正確驅動，且板子完全沒有亮。因此做了一些調整，將判斷邏輯集中在同一個 always block 裡，直接使用 clk, rst 驅動所有的信號更新。

D. Suggestions

我覺得很棒，沒什麼建議所以附上笑話：

皮卡丘很愛搗蛋被揍了一頓，會變成什麼?

卡丘，因為他不敢再皮了。