writeup - bamboofox2021

□Flag Checker

All sources and solved code is found at here.

> About Verilog hardware-language:
>
> - All statements are executed in parallel
> - Code ordering is flexible.

Understand the flow of program

- Firstly, open all source codes of challenge, we will recognize that **t_chall.v** will call **chall.v** module via this instruction:

```
chall ch(.clk(clk), .rst(rst), .inp(inp), .res(out));
```

- And similarly, **chall.v** will call **magic.v** module by using some following instructions as:

```
magic m0(.clk(clk), .rst(rst), .inp(inp), .val(val0), .res(res0));
magic m1(.clk(clk), .rst(rst), .inp(res0), .val(val1), .res(res1));
magic m2(.clk(clk), .rst(rst), .inp(res1), .val(val2), .res(res2));
magic m3(.clk(clk), .rst(rst), .inp(res2), .val(val3), .res(res3));
```

> - A module in Verilog has module header and module body. A module header contains variables such as input, output, **clk** and **rst**(*rst, clk relate to hardware signal, we can ignore them in this challenge*). And module body will contain some procedures.

> - The definition of constants in Verilog supports the addition of a width parameter. The basic syntax is:
> *<Width in bits>'<**base** letter><number>*
>
> Ex: 8'b01001101 –> 8-bit binary 01001101
> Ex: 8'd182 –> 8-bit decimal 182

magic.v

magic.v

```
module magic(
  input clk,
  input rst,
  input[7:0] inp,
  input[1:0] val,
  output reg[7:0] res
);
  always @(*) begin
    case (val)
      2'b00: res = (inp >> 3) | (inp << 5);
      2'b01: res = (inp << 2) | (inp >> 6);
      2'b10: res = inp + 8'b110111;
      2'b11: res = inp ^ 8'd55;
    endcase
  end
endmodule

```

- Module **magic** is a simple switch-case procedure: base on 2 bits of **val** variable, **inp** will process and store a new result into **res** output.

chall.v

```
  wire[1:0] val0 = inp[1:0];
  wire[1:0] val1 = inp[3:2];
  wire[1:0] val2 = inp[5:4];
  wire[1:0] val3 = inp[7:6];
```

- **chall.v** module receives a parameter(**8-bit** length) **inp** from **t_chall.v** module, then process that input(**cutting into four 2-bit pieces: val3, val2, val1, val0, respectively *from the most* significant bit *to the least* significant bit).**

```
  magic m0(.clk(clk), .rst(rst), .inp(inp), .val(val0), .res(res0));
  magic m1(.clk(clk), .rst(rst), .inp(res0), .val(val1), .res(res1));
  magic m2(.clk(clk), .rst(rst), .inp(res1), .val(val2), .res(res2));
  magic m3(.clk(clk), .rst(rst), .inp(res2), .val(val3), .res(res3));
```

- *inp* and *val0* as two input parameters is passed to *magic module.* After, output received(res0) as a new input in m1 and so on.
- Finally, **res3** is the return value to ***t_chall*** module:

```
    end else begin
      assign res = res3;
```

t_chall.v

```
chall ch(.clk(clk), .rst(rst), .inp(inp), .res(out));
...
#many lines dismiss
...

for (idx = 0; idx < 32; idx++) begin
      inp = flag[idx];
      tmp = target[idx];
      #4;
end

...
#many lines dismiss
...

always @(posedge clk) begin
    #1 ok = ok & (out == tmp);
end
```

```
19
```

- In this module, we recognize each character of flag string is a **inp** variable passing to *chall module* and receive **out** variable in output field.
- Then, **out** is compared to each element of **target** array, respectively.

Summary - solve.py

- Brute-force all elements of **flag** string(appropriate characters are in range from 32 to 127(in ASCII)) and find out original character.
- Because there are more than one satisfied characters for each element of target, so let's choose the most appropriate one to create flag.