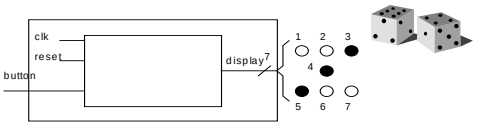
Dice



Design a dice.

- **The clk and reset are internally generated.**
- **A led is ON when the corresponding bit is '1'.**

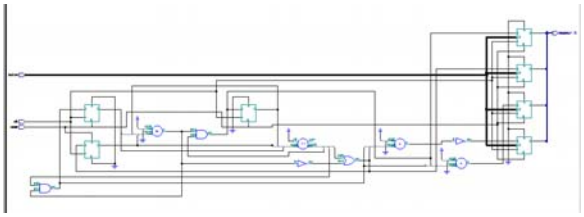
```
ENTITY dice IS
  PORT (clk    : IN std_logic;
        reset  : IN std_logic;
        button : IN std_logic;
        display : OUT std_logic _vector(1 TO 7));
END dice;
```


Dice, behavior (?)

```
ARCHITECTURE demo OF dice IS
BEGIN
 PROCESS(reset,clk)
  VARIABLE dice_value : INTEGER RANGE 1 TO 6;
 BEGIN
  IF reset='1' THEN
    dice_value := 1;
    display <= "0000000";
  ELSIF rising_edge(clk) THEN
    IF dice_value<6 THEN
      dice_value := dice_value + 1;
    ELSE
      dice_value := 1;
    END IF;
  IF button='1' THEN
      CASE dice_value IS      --1234567
        WHEN 1 => display <= "0001000";
        WHEN 2 => display <= "0010100";
        WHEN 3 => display <= "0011100";
        WHEN 4 => display <= "1010101";
        WHEN 5 => display <= "1011101";
        WHEN 6 => display <= "1110111";
      END CASE;
    END IF;
  END IF;
 END PROCESS;
END demo;
```

It seems ok, but is it?

asynchronous inputs

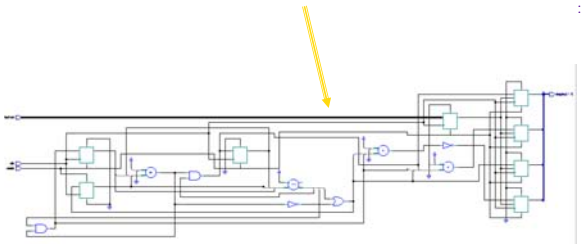


How to solve this problem?

Synchronized input

```
ARCHITECTURE demo OF dice IS
  SIGNAL button_sync : std_logic;
BEGIN
  PROCESS(reset,clk)
    VARIABLE dice_value : INTEGER RANGE 1 TO 6;
  BEGIN
    IF reset='1' THEN
      dice_value := 1;
      display <= "0000000";
      button_sync<='0';
    ELSIF rising_edge(clk) THEN
      button_sync <= button;
      IF dice_value<6 THEN
        dice_value := dice_value + 1;
      ELSE
        dice_value := 1;
      END IF;
    IF button_sync='1' THEN
        CASE dice_value IS      --1234567
          WHEN 1 => display <= "0001000";
          WHEN 2 => display <= "0010100";
          WHEN 3 => display <= "0011100";
          WHEN 4 => display <= "1010101";
          WHEN 5 => display <= "1011101";
          WHEN 6 => display <= "1110111";
        END CASE;
      END IF;
    END IF;
  END PROCESS;
END demo;
```

What happens when the button is pushed?



Often two flipflops are used in stead of one:
- The first flipflip is used to synchronize the asynchronous input
- The second flipflop is used to reduce problems due to metastability

(pessimistic designers even use three flipflops)

Reset

- **The flipflops in an FPGA often has special asynchronous reset and preset inputs.**
- **To prevent that a (p)reset is de-asserted when there is an active edge of the clock the (p)reset could be read at the not active edge of the clock.**

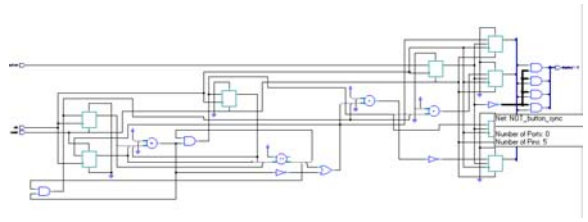
```
process (clk)
begin
  wait until clk='0';
  reset_internal <= reset
end process;
```

Leds off if button is pushed

```
.....

IF button_sync='1' THEN
  CASE dice_value IS   --1234567
    WHEN 1 => display_internal <= "0001000";
    WHEN 2 => display_internal <= "0010100";
    WHEN 3 => display_internal <= "0011100";
    WHEN 4 => display_internal <= "1010101";
    WHEN 5 => display_internal <= "1011101";
    WHEN 6 => display_internal <= "1110111";
  END CASE;
 END IF;
END IF;
END PROCESS;

display <= display_internal WHEN button_sync='0' ELSE
        (OTHERS => '0');
```

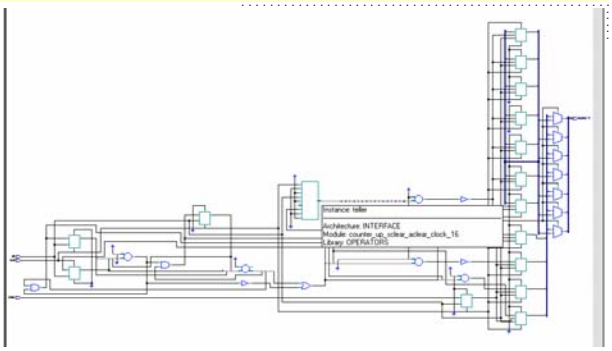



Walking light when button is pushed

```
....

display <= display_internal WHEN button_sync='0' ELSE
        walking_int;

PROCESS(reset,clk)
  CONSTANT wachttijd : integer := (2**16)-1;
  VARIABLE teller : integer RANGE 0 TO wachttijd;
BEGIN
  IF reset='1' THEN
    walking_int<="1000000";
    teller:=0;
  ELSIF rising_edge(clk) THEN
    IF teller < wachttijd THEN
      teller:=teller + 1;
    ELSE
      teller:=0;
      walking_int<=walking_int(2 TO 7) & walking_int(1);
    END IF;
  END IF;
END PROCESS;
```

Unfair behaviour

```
PROCESS(reset,clk)
  VARIABLE dice_value : INTEGER RANGE 1 TO 15;
 BEGIN
  IF reset='1' THEN
    dice_value := 1;
    display <= "0000000";
    button_sync<='0';
  ELSIF rising_edge(clk) THEN
    button_sync <= button;
    IF dice_value<15 THEN
      dice_value := dice_value + 1;
    ELSE
      dice_value := 1;
    END IF;
    IF button_sync='1' THEN
      CASE dice_value IS   --1234567
        WHEN 1     => display <= "0001000";
        WHEN 2     => display <= "0010100";
        WHEN 3     => display <= "0011100";
        WHEN 4     => display <= "1010101";
        WHEN 5     => display <= "1011101";
        WHEN OTHERS => display <= "1110111";
      END CASE;
    END IF;
  END IF;
 END PROCESS;
```