Expr11. Design of Synchronous Sequential Circuits

Name:	王浚哲	**ID:**	3180103011	**Major:**	Computer Science and Technology
Course:	Logic and Computer Design Fundamentals			**Groupmate:**	朱雨轩
Date:	2019-11-27	**Place:**	East 4-509	**Instructor:**	洪奇军

§1 Purposes & Requirements

1. Master the principle and typical design of synchronous sequential circuits.
2. Master the application of excitation functions, state diagrams and state equations of sequential circuits.
3. Master the design, debug and simulation of *finite state machine* with Verilog HDL.
4. Master the implementation of sequential circuits with FPGA.

§2 Principle & Tasks

2.1 Experiment Tasks

1. Implement a 4-bit binary synchronous counter using schematic diagram.
2. Implement a 16-bit reversible binary synchronous counter.

2.2 Experiment Principle

2.2.1 4-bit Binary Synchronous Counter

A *Binary Synchronous Counter* is a sequential circuit which increase its own output binary code by 1 every time the *Clock Signal* is 1.

From its function, we obtain its state table:

	Current State				Next State				Input			
	Q_A	Q_B	Q_C	Q_D	Q_A^{n+1}	Q_B^{n+1}	Q_C^{n+1}	Q_D^{n+1}	D_A	D_B	D_C	D_D
0	0	0	0	0	1	0	0	0	1	0	0	0
1	1	0	0	0	0	1	0	0	0	1	0	0
2	0	1	0	0	1	1	0	0	1	1	0	0
3	1	1	0	0	0	0	1	0	0	0	1	0
4	0	0	1	0	1	0	1	0	1	0	1	0
5	1	0	1	0	0	1	1	0	0	1	1	0
6	0	1	1	0	1	1	1	0	1	1	1	0
7	1	1	1	0	0	0	0	1	0	0	0	1
8	0	0	0	1	1	0	0	1	1	0	0	1
9	1	0	0	1	0	1	0	1	0	1	0	1
10	0	1	0	1	1	1	0	1	1	1	0	1
11	1	1	0	1	0	0	1	1	0	0	1	1
12	0	0	1	1	1	0	1	1	1	0	1	1
13	1	0	1	1	0	1	1	1	0	1	1	1
14	0	1	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	0	0	0	0	0	0	0	0

After K-map simplification of all the 4 outputs, we now have state functions for all next states of the counter:

$$Q_A^{n+1} = \overline{Q_A}$$

$$Q_B^{n+1} = \overline{\overline{Q_A} \otimes \overline{Q_B}}$$

$$Q_C^{n+1} = \overline{(\overline{\overline{Q_A} + \overline{Q_B}}) \otimes \overline{Q_C}}$$

$$Q_D^{n+1} = \overline{(\overline{\overline{Q_A} + \overline{Q_B} + \overline{Q_C}}) \otimes \overline{Q_D}}$$

And the output function for carry *Rc*:

$$R_C = \overline{\overline{Q_A} + \overline{Q_B} + \overline{Q_C} + \overline{Q_D}} = Q_A Q_B Q_C Q_D$$

According to these output functions, we will design the *4-bit Binary Synchronous Counter* using schematic in $_{\S}4$.

2.2.2 4-bit Reversible Binary Synchronous Counter

The *Reversible Binary Synchronous Counter* can be controlled by S to select in forward mode or reversed mode should the counter counts.

- When S = 1, the counter counts in the ascending mode.
- When S = 0, the counter counts in the descending mode.

A 4-bit reversible counter has output functions like this:

$$D_A = \overline{Q_A}$$

$$D_B = \overline{S}(\overline{\overline{Q_A}\oplus\overline{Q_B}}) + S(\overline{\overline{Q_A}\oplus\overline{Q_B}}) = \overline{S\oplus\overline{Q_A}\oplus\overline{Q_B}}$$

$$D_C = \overline{S}[\overline{(\overline{Q_A}\overline{Q_B})\oplus\overline{Q_C}}] + S[\overline{(\overline{Q_A}+\overline{Q_B})\oplus\overline{Q_C}}] = [\overline{S\overline{Q_A}\overline{Q_B}+S(\overline{Q_A}+\overline{Q_B})}]\oplus\overline{Q_C}$$

$$= [\overline{\overline{S}(Q_A+Q_B)+S(\overline{Q_A}+\overline{Q_B})}]\oplus\overline{Q_C}$$

$$D_D = \overline{S}[\overline{(\overline{Q_A}\overline{Q_B}\overline{Q_C})\oplus\overline{Q_D}}] + S[\overline{(\overline{Q_A}+\overline{Q_B}+\overline{Q_C})\oplus\overline{Q_D}}] = [\overline{S\overline{Q_A}\overline{Q_B}\overline{Q_C}+S(\overline{Q_A}+\overline{Q_B}+\overline{Q_C})}]\oplus\overline{Q_D}$$

$$= [\overline{\overline{S}(Q_A+Q_B+Q_C)+S(\overline{Q_A}+\overline{Q_B}+\overline{Q_C})}]\oplus\overline{Q_D}$$

$$R = \overline{S}\,\overline{Q_A}\,\overline{Q_B}\,\overline{Q_C}\,\overline{Q_D} + SQ_AQ_BQ_CQ_D \qquad \text{(Carry, borrow output)}$$

Since all the functions above is definitely making people dizzy and no one is willing to do all those wirings, we'll use *Behavior Description* to implement the reversible counter. See §4.

2.2.3 Auxiliary Module: Frequency Divider

We've known from previous experiments that the SWORD Board generates clock signals at a frequency of **100MHz**, but obviously we don't want our counters do those counts every 1/100,000,000s. Thus we need to design a module that helps us reduce the frequency input to our counters to a human-acceptable range (e.g. 1~10Hz).

The system-generated clock of 100MHz will be divided by 50,000,000 time to obtain a 1Hz square wave which will be used to trigger our counters.

The codes of implementation can be seen in §4.

§3 Main Instruments & Materials

3.1 Experiment Instruments

1. A Computer with ISE 14.7 Installed
2. SWORD Board

3.2 Experiment Materials

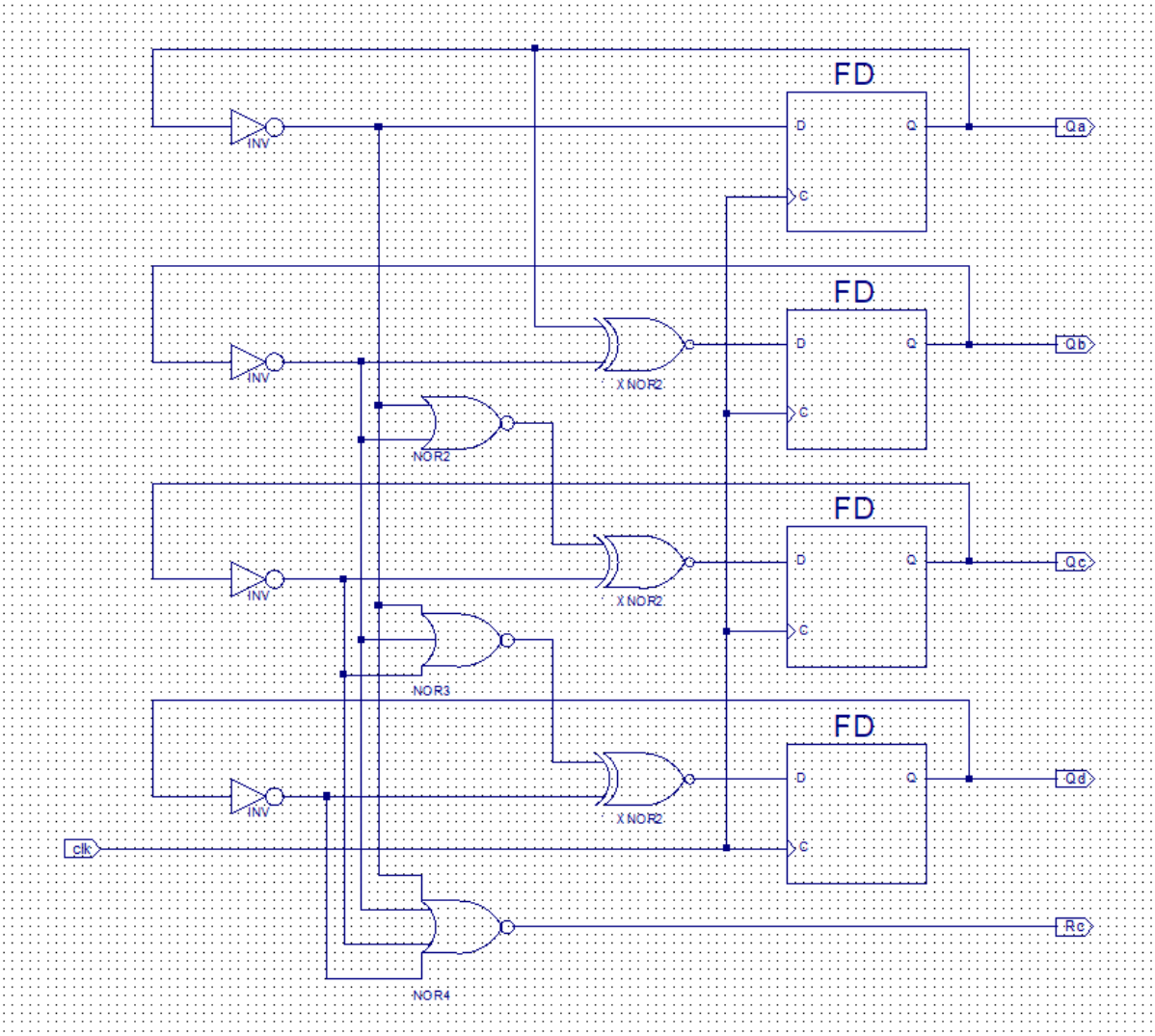
None.

§4 Experiment Procedure & Operations

4.1 Implement a 4-bit binary synchronous counter using schematic diagram

1. Create a new ISE project named "*MyCounter*" with Top Level Source Type *HDL*.
2. Create a new Schematic source file named "*Counter4b*".

3. Design the module according to the schematic diagram.



4. Run simulation on `Counter4b` module, main part of the excitation codes are as follows:

```
// Initialize Inputs
    initial clk = 0;
    always begin
        #20 clk = ~clk;
    end
```

5. Create a new Verilog HDL source file named "*clk_1s.v*" and use *Behavior Description* to implement a *1Hz Oscillation Circuit*. The code is like the follows:

```
module clk_1s(      // Generate 1Hz signal from the on-board 100MHz clock
signal
    input wire clk,
    output reg clk_1s
   );
    reg [31:0] cnt;

    initial begin
        cnt = 0;
```

```
        clk_1s = 0;
    end

    always @ ( posedge clk ) begin
        if ( cnt < 50_000_000 ) cnt <= cnt + 1;
        else begin
            cnt <= 0;
            clk_1s <= ~clk_1s;
        end
    end
endmodule
```

6. Create a new Verilog HDL source file "*Top.v*" and set it as the top module.
    - Invoke the 4-bit counter we just designed and use the 1s-period clock signal generated by `clk_1s` module as its timing input.
    - Invoke `disp_num` module to display the 4-bit binary information on one of the 4 7-segment digital tube displays.
    - And use 1 LED light to indicate the carry output `Rc`.

```
module top(     // Top Module of project "MyCounter"
    input wire clk,
    output wire [7:0] SEG,
    output wire [3:0] AN,
    output wire Rc
   );

    wire clk_1s;
    wire [3:0] Q;

    clk_1s clk1s(clk, clk_1s);

    Counter4b u1(.clk(clk_1s), .Qa(Q[0]), .Qb(Q[1]), .Qc(Q[2]), .Qd(Q[3]),
.Rc(Rc));

    disp_num u2(.clk(clk), .RST(1'b0), .HEXS({12'b0,Q}), .point(4'b0),
.LES(4'b1110), .Segment(SEG), .AN(AN));

endmodule
```

7. Generate Programming File and upload our design to the SWORD Board, check its behavior. A digit which increases by 1 every second and loop between 0~F should be displayed on the rightmost 7-segment display.

4.2 Implement a 16-bit reversible binary synchronous counter

1. Create a new ISE project named "*MyRevCounter*" with Top Level Source Type *HDL*.

2. Create a new Verilog HDL source file named "*CounterRev16b.v*".
3. Use behavior description to implement a 16-bit reversible binary synchronous counter:

```
module CounterRev16b(   // A 16-bit Reversible Binary Synchronous Counter
    input wire clk, S,
    output reg [15:0] cnt,
    output wire Rc      // Carry bit
    );

    initial cnt = 0;

    assign Rc = (~S & (~|cnt)) | (S & (&cnt));

    always @ (posedge clk)
        if (S) cnt <= cnt + 1;
        else cnt <= cnt - 1;

endmodule
```

4. Run simulation on `CounterRev16b` module which should cover both forward and reversed modes. Main part of the excitation codes are as follows:

```
initial begin
    // Initialize Inputs
    clk = 0;
    S = 0;      // Reversed mode

    // Wait 100 ns for global reset to finish
    #32768;     // This interval is large in order to see more data
    S = 1;      // Forward mode
end

always #5 clk = ~clk;
```

5. Since we now have 16 bits storing information, a 1Hz clock is somehow too "slow" for it. We need a higher frequency clock to make things more human-acceptable. Therefore, create a new module `clk_100ms` to generate 100ms-period clock signal for our new counter.

```
module clk_100ms(
    input wire clk,
    output reg clk_100ms
   );
    integer cnt;

    initial begin
        cnt = 0;
        clk_100ms = 0;
```

```
    end

    always @ ( posedge clk ) begin
        if ( cnt < 5_000_000 ) cnt <= cnt + 1;
        else begin
            cnt <= 0;
            clk_100ms <= ~clk_100ms;
        end
    end
endmodule
```

6. Like the previous project, create a new Verilog HDL source file "*Top.v*" and set it as the top module.
    - Invoke the 16-bit reversible counter `CounterRev16b` we just designed and use the 100ms-period clock signal generated by `clk_100ms` module as its timing input.
    - Add an input `SW` to determine the mode our counter would operate on.
    - Invoke `disp_num` module to display the 16-bit binary information on 4 7-segment digital tube displays.
    - And use 1 LED light to indicate the carry output `Rc`.

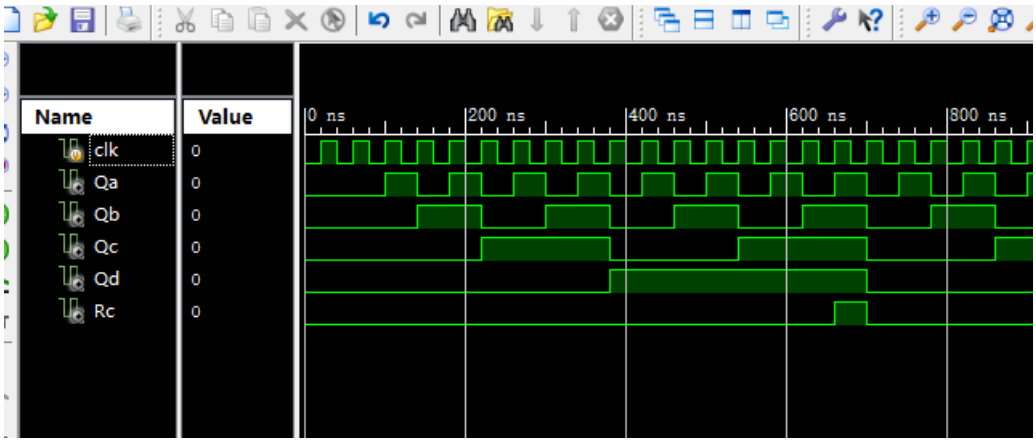
```
module top(
    input wire clk,
    input wire SW,
    output wire [7:0] SEG,
    output wire [3:0] AN,
    output wire Rc
   );

    wire clk_100ms;
    wire [15:0] num;

    clk_100ms m1(clk, clk_100ms);

    CounterRev16b m2(clk_100ms, SW, num, Rc);

    disp_num m3(.clk(clk), .RST(1'b0), .HEXS(num), .point(4'b0010),
.LES(4'b0), .Segment(SEG), .AN(AN));
endmodule
```

7. Generate Programming File and upload our design to the SWORD Board, check its behavior. 4 digits which increases by `4'b0001` every 100ms and loop between "0000"~"FFFF" should be displayed on the 4 7-segment displays.

§5 Results & Analysis

5.1 Implement a 4-bit binary synchronous counter using schematic diagram

1. The simulation result was as the follows:

   From the simulation result we can see that `Counter4b` Module implemented the desired function.

2. The Top Module passed all checks and the programming file was generated successfully.
3. After uploading and operating onto the SWORD Board, it was clear that the Top Module worked correctly. The changing digit indicates that it was an 1s counter.

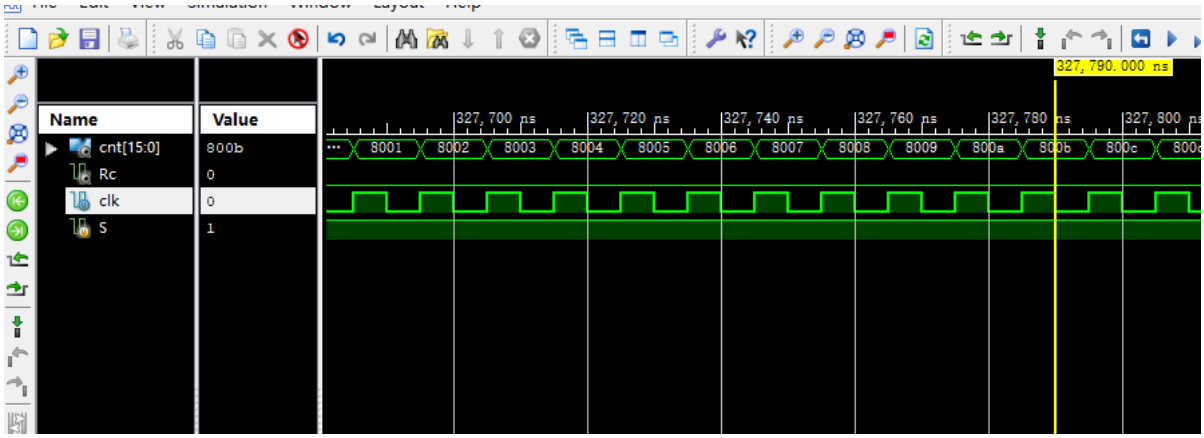
   Note: Since a static image won't help explain anything about this dynamic process of counting, I dropped it out here. But in practice it was REALLY implemented.

Analysis: From the result we can conclude that the `Counter4b` Module has implemented our desired function.

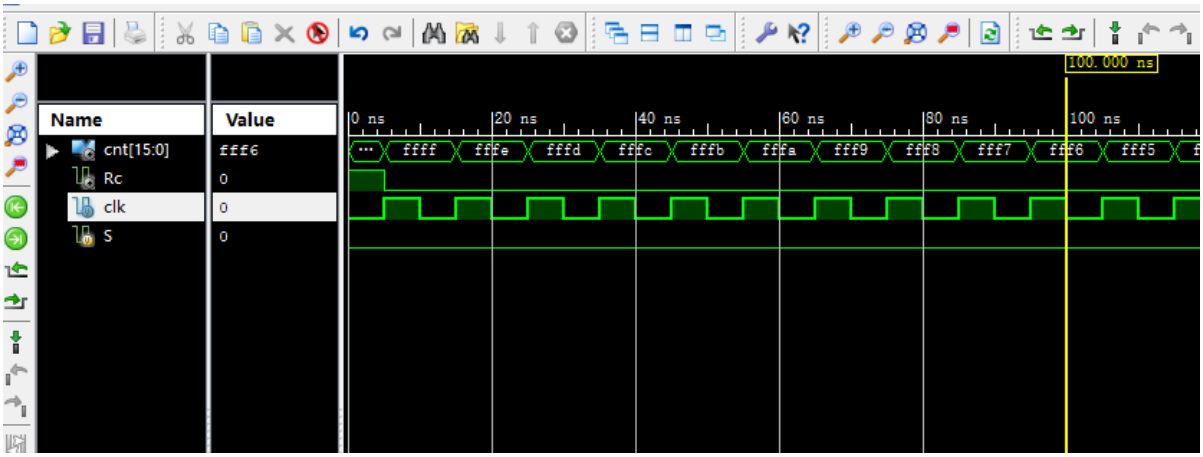
5.2 Implement a 16-bit reversible binary synchronous counter

1. The simulation results were as the follows:

   Increasing Mode:

   Decreasing Mode:



From the simulation result we can see that `CounterRev16b` Module implemented the desired function.

2. The Top Module passed all checks and the programming file was generated successfully.
3. After uploading and operating onto the SWORD Board, it was clear that the Top Module worked correctly. The changing digits indicates that it was an 100ms counter (timer).

   Note: Since a static image won't help explain anything about this dynamic process of counting, I dropped it out here. But in practice it was REALLY implemented.

Analysis: From the result we can conclude that the `CounterRev16b` Module has implemented our desired function.

Overall Experience

In Experiment 9, I learned something really essential for a CPU — the structure and function of an *Arithmetic & Logic Unit* (ALU). Although it was only a 4-bit version, it has deepened my understanding about the fundamentals of computer structure.

From experiment 10, we've entered a brand new realm of the LCDF Course — *Sequential Logic Circuits*. It involved something new: states, timing, latches, triggers and so on. We're now considering TIME, which was never mentioned in the *Combinational Logic Circuits* part. I've learned how a single bit of information is stored in a computer, and how a counter was built up.

Another significant lesson learned was about the Verilog HDL programming language. We were gradually turning from schematic designs into code designs. What I've learnt a lot was the syntax and how powerful the Verilog HDL language is. Besides the fact that the experiments were done successfully, I'd say I've also gained a lot.