Lab13--Implementation & Application of Timer / Counter

姓名：潘子曰 学号：3180105354 专业：计算机科学与技术
课程名称：逻辑与计算机设计基础实验 同组学生姓名：张佳文
试验时间：2019-12-19 实验地点：紫金港东4-509 指导老师：洪奇军 ## 1. Objectives & Requirements

> - Master the working principles and design of **binary timer / counter**
> - Master the concept and method of dividing clock signal with counter
> - Know the concept of **Program Counter (PC)** in computer system
> - Know the concept of **Baud** of transferring serial data

2. Contents & Principles

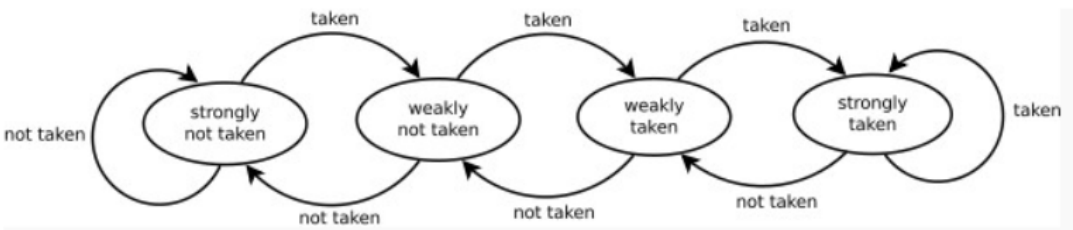
Tasks

> - Implement counter with different **positional notation**
> - Implement **timer with alarming**
> - Implement 24'h **wall clock module**
> - Do physical test

Principles

General counter

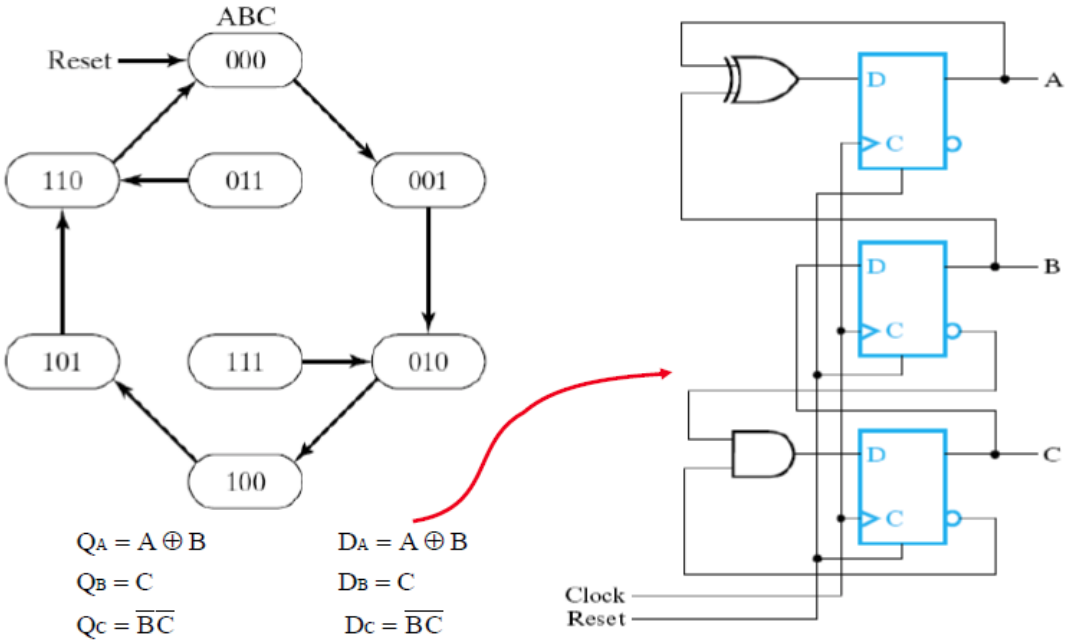
- Counter
  - traverse sequence of states
  - not necessarily the numerical sequence
  - not necessarily form a cycle
    - saturating counter: stop at the initial or final state



saturating counter

- Modulo-N counter
  - the general form of a cyclic counter
  - resets and output a carry at a chosen number

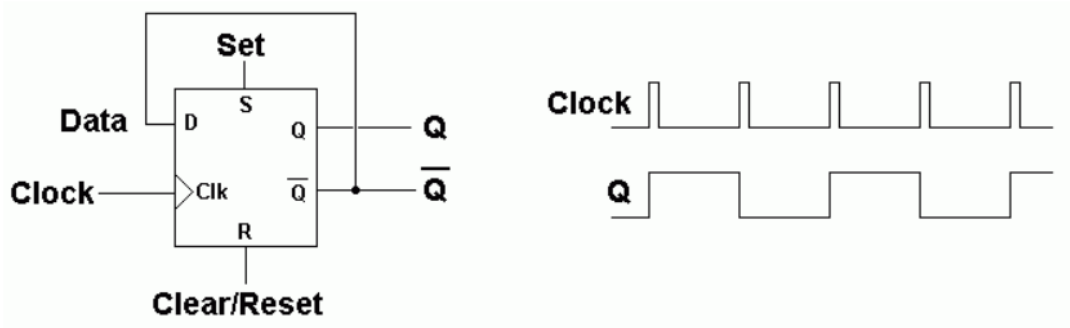
Modulo-6 counter



This is an example of a modulo-N counter. The above is the state diagram and circuit diagram.

Divide frequency

- Dividing frequency is to lower the frequency to what we really need.
  - Dividing the frequency by 2 is to lower the frequency by 2 times.
- Basically, the frequency-dividing coefficient is 2^n
- This a binary dividing of frequency with a D flip-flop



3. Major Experiment Instruments

Equipment
Computer (Intel Core i7-9750H, 16GB memory)
Sword circuit design box
Xilinx ISE 14.7

4. Experiment Procedure

Before our implementation, I have some different

Task1: Timer

> Basically what we are going to implement is a timer with alarm. The input clock signal is a millisecond.

```
module Timer(
 input clk,
 input Up,
 input Load,
 input Start,
 input [31:0] Timing_constants,
 output reg [31:0] cnt,
 output reg Alarm
);

reg[1:0] go;

always @(posedge clk or posedge Start) begin
 if (Start) begin
     go <= 2'b01;
     Alarm <= 0;
     cnt <= Timing_constants;
 end
 else begin
     if(Load) cnt <= Timing_constants;
     else begin
         if(go == 2'b01) begin
             Alarm <= 0;
             if(Up) cnt <= cnt + 1;
             else cnt <= cnt -1;
         end
         else if((!cnt)==0 | (&cnt==1)) begin
             Alarm <= 1;
             go <= 0;
         end
     end
 end
end

endmodule
```

Task2: Wall Clock

> - Function:
>   - 24 hour time display: hours, minutes, seconds, milliseconds
>   - reset time
>   - with behavioral description
>   - BCD display

Binary to BCD module

Since we are going to implement a wall clock, we have to convert the binary data into the commonly-used **BCD** code. Therefore, we have to design a binary-to-BCD converter. And what we need is exactly a shift register, which carries the overflowed bit to the correct position.

```
module B2BCD(
   // I/O Signal Definitions
   input  [11:0] number,
   output reg [3:0] hundreds,
   output reg [3:0] tens,
   output reg [3:0] ones
    );

   // Internal variable for storing bits
   reg [23:0] shift;
   integer i;

   always @(number) begin
      // Clear previous number and store new number in shift register
      shift[23:12] = 0;
      shift[11:0] = number;

      // Loop eight times
      for (i=0; i<12; i=i+1) begin
         if (shift[15:12] >= 5)
            shift[15:12] = shift[15:12] + 3;

         if (shift[19:16] >= 5)
            shift[19:16] = shift[19:16] + 3;

         if (shift[23:20] >= 5)
            shift[23:20] = shift[23:20] + 3;

         // Shift entire register left once
         shift = shift << 1;
      end

      // Push decimal numbers to output
      hundreds = shift[23:20];
      tens     = shift[19:16];
      ones     = shift[15:12];
   end

endmodule
```

My clock

With a Binary-to-BCD module, we can implement our wall clock readily. Some details will be listed below:

- format: hh.mm.ss.top 2 bit of milliseconds
- 24'h system
- behavioral description
- Output flashing bits and points control signal

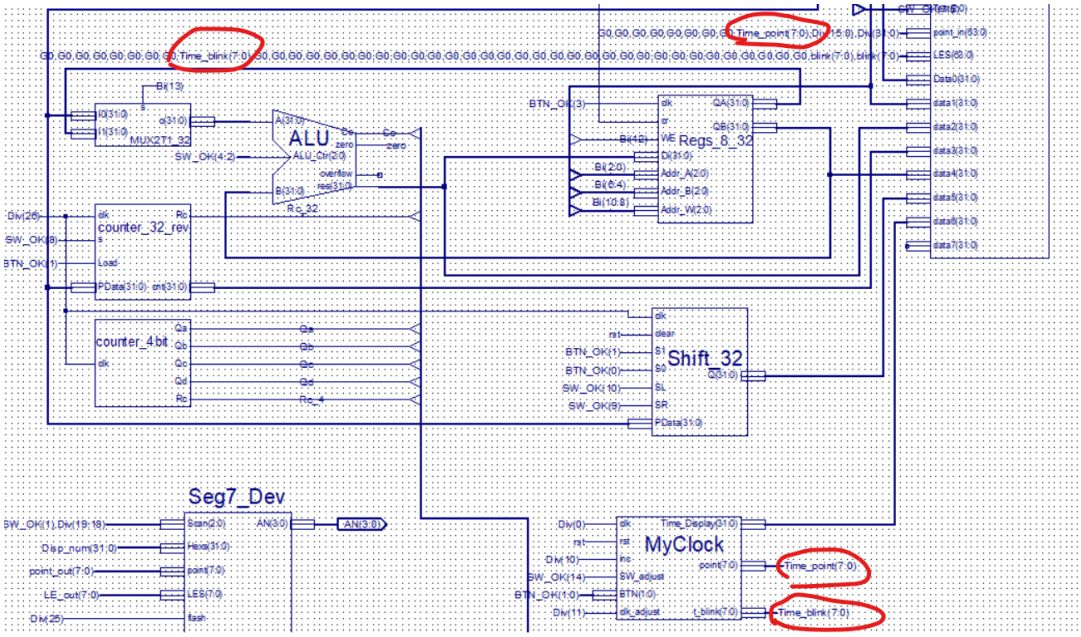
```
module MyClock(
     input clk_adjust,
    input clk, rst, inc,
    input SW_adjust,
    input [1:0] BTN,
    output [31:0] Time_Display,
    output [7:0] point,
    output [7:0] t_blink
    );

assign point = 8'b01010100;
wire [3:0] AE;                          //Adjust Enable
reg clk_h, clk_m, clk_s, clk_ms;        //clock signal

wire wire_h, wire_m, wire_s, wire_ms, clk_ok;
reg[7:0] cnt_m = 0, cnt_s = 0, cnt_h=0; //clock count
reg[11:0] cnt_ms = 0;
reg[15:0] cnt_clk = 0;
reg [1:0] adjust_pos = 2'b11;                   //position code

assign AE[0] = SW_adjust && adjust_pos == 2'b00 && BTN[1]? 1:0;
assign AE[1] = SW_adjust && adjust_pos == 2'b01 && BTN[1]? 1:0;
assign AE[2] = SW_adjust && adjust_pos == 2'b10 && BTN[1]? 1:0;
assign AE[3] = SW_adjust && adjust_pos == 2'b11 && BTN[1]? 1:0;

assign wire_h = (clk_h & ~SW_adjust)|(AE[3] & SW_adjust);
assign wire_m = (clk_m & ~SW_adjust)|(AE[2] & SW_adjust);
assign wire_s = (clk_s & ~SW_adjust)|(AE[1] & SW_adjust);
assign wire_ms = (clk_ms & ~SW_adjust)|(AE[0] & SW_adjust);
assign clk_ok = clk & ~SW_adjust;

//millisecond signal
always@(posedge clk_ok or posedge rst)begin
    if(rst) begin
        cnt_clk <= 0;
        clk_ms <= 0;
    end
    else if(cnt_clk >= 16'hC34F) begin
        cnt_clk <= 0;
        clk_ms <= 1;
    end
    else begin
        cnt_clk <= cnt_clk +1;
        clk_ms <= 0;
    end
end

//second signal & millisecond count
always@(posedge wire_ms or posedge rst)begin
    if(rst) begin
        cnt_ms <= 0;
        clk_s <= 0;
    end
    else begin
         if(cnt_ms >= 12'h3E7) begin
              cnt_ms <= 0;
              clk_s <= 1;
         end
```

```
          else begin
               cnt_ms <= cnt_ms +1;
               clk_s <= 0;
          end
     end
end

//minute signal & second count
always@(posedge wire_s or posedge rst)begin
     if(rst) begin
         cnt_s <= 0;
         clk_m <= 0;
     end
      else begin
          if(cnt_s >= 8'h3B) begin
               cnt_s <= 0;
               clk_m <= 1;
          end
          else begin
               cnt_s <= cnt_s +1;
               clk_m <= 0;
          end
      end
end

//hour signal & minute count
always@(posedge wire_m or posedge rst)begin
     if(rst) begin
         cnt_m <= 0;
         clk_h <= 0;
     end
     else begin
         if(cnt_m >= 8'h3B) begin
             cnt_m <= 0;
             clk_h <= 1;
         end
         else begin
             cnt_m <= cnt_m +1;
             clk_h <= 0;
         end
      end
end

//hour count
always@(posedge wire_h or posedge rst)begin
     if(rst) cnt_h <= 0;
     else begin
          if(cnt_h >= 8'h17) cnt_h <= 0;
          else cnt_h <= cnt_h +1;
     end
end

//control position
always@(posedge BTN[0]) begin
     adjust_pos <= adjust_pos-1;
end

//binary to BCD display
```

```
117 B2BCD   BCD_Hour(.number({4'h0,cnt_h}), .tens(Time_Display[31:28]),
    .ones(Time_Display[27:24])),
118         BCD_Minute(.number({4'h0,cnt_m}),
    .tens(Time_Display[23:20]), .ones(Time_Display[19:16])),
119         BCD_Second(.number({4'h0,cnt_s}),
    .tens(Time_Display[15:12]), .ones(Time_Display[11:8])),
120         BCD_Millisecond(.number(cnt_ms),
    .hundreds(Time_Display[7:4]), .tens(Time_Display[3:0]));
121
122 //time blink
123 assign t_blink = SW_adjust ? (adjust_pos[1]?(adjust_pos[0]?
    8'hC0:8'h30):(adjust_pos[0]?8'h0C:8'h03)) : 8'h00;
124 assign wire_clk_h = clk_h;
125 assign wire_clk_m = clk_m;
126 assign wire_clk_s = clk_s;
127 assign wire_clk_ms = clk_ms;
128
129 endmodule
```

Physical Test

The physical Test environment can be a bit of complicated:



We have to modify the wire assignment into the `multi8ch32` module: Time_point[7:0] and Time_blink[7:0], which specifies the assignment for flashing bits and point control.

5. Experiment Results and Analyses

Problems

At first time, I use complicated combination assignment within a clock duration. To be more specific, I made judgement for branch at each positive edge of clock signal.

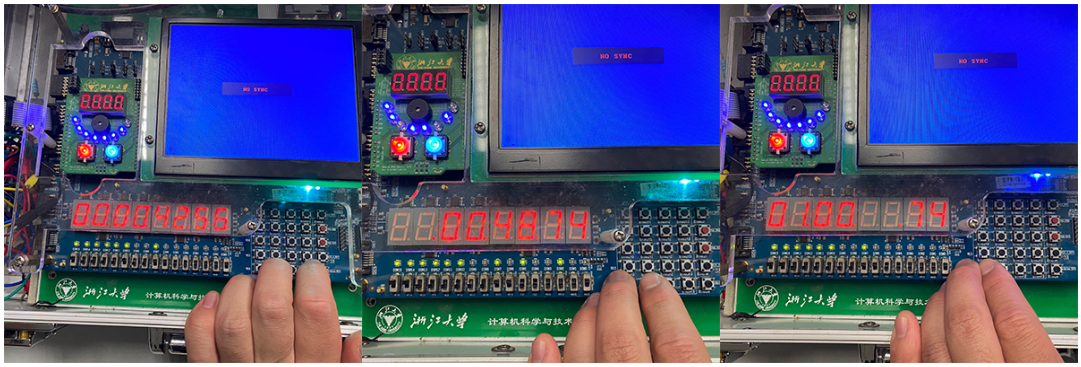
But a complicated problem during synthesis phase. This the error log:

```
1 assignment under multiple single edges is not supported for synthesis
```

> Possibility is that **ISE** cannot afford to optimize complex combinational logic with time sequential control. According to the solution from xilinx forum, I made some change to my original code: using an **AND** gate to integrate clock and enable signal. And thus avoiding building complex combinational logic:

```
assign wire_h = (clk_h & ~SW_adjust)|(AE[3] & SW_adjust);
assign wire_m = (clk_m & ~SW_adjust)|(AE[2] & SW_adjust);
assign wire_s = (clk_s & ~SW_adjust)|(AE[1] & SW_adjust);
assign wire_ms = (clk_ms & ~SW_adjust)|(AE[0] & SW_adjust);
assign clk_ok = clk & ~SW_adjust;
```

Wall Clock

> The function can be specified as following:
>
> - BTN_OK[0]: change the adjusting bits
> - BTN_OK[1]: increment the adjusting numbers
> - SW_OK[14]: control adjusting or not
>
> 
>
> Therefore, the wall clock function is well verified.

6. Discussion and Conclusion

> Actually, the reason why I don't want to follow Professor Shi's experiment procedure is that his design seems much abstract from what I have made. As a developer, it should always be our first to consider to make our design intuitive and dependable. Therefore, I made such a project, which is completely my own design.