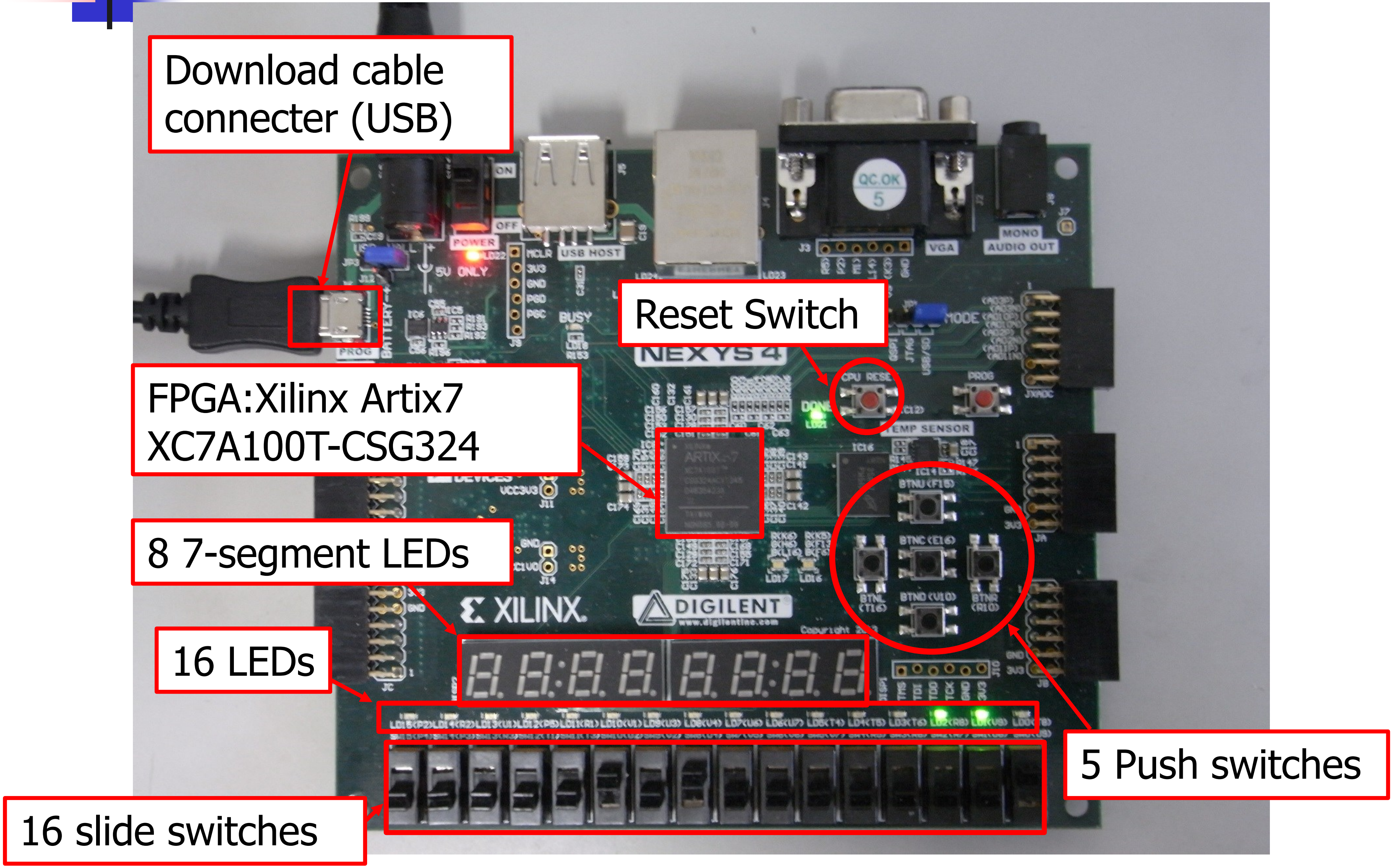
# Skill Up Course

## A 24-Hour Digital Clock
## Specification and Design Example

# Overview of Training Board（NEXYS4)



Download cable connecter (USB)

Reset Switch

FPGA:Xilinx Artix7 XC7A100T-CSG324

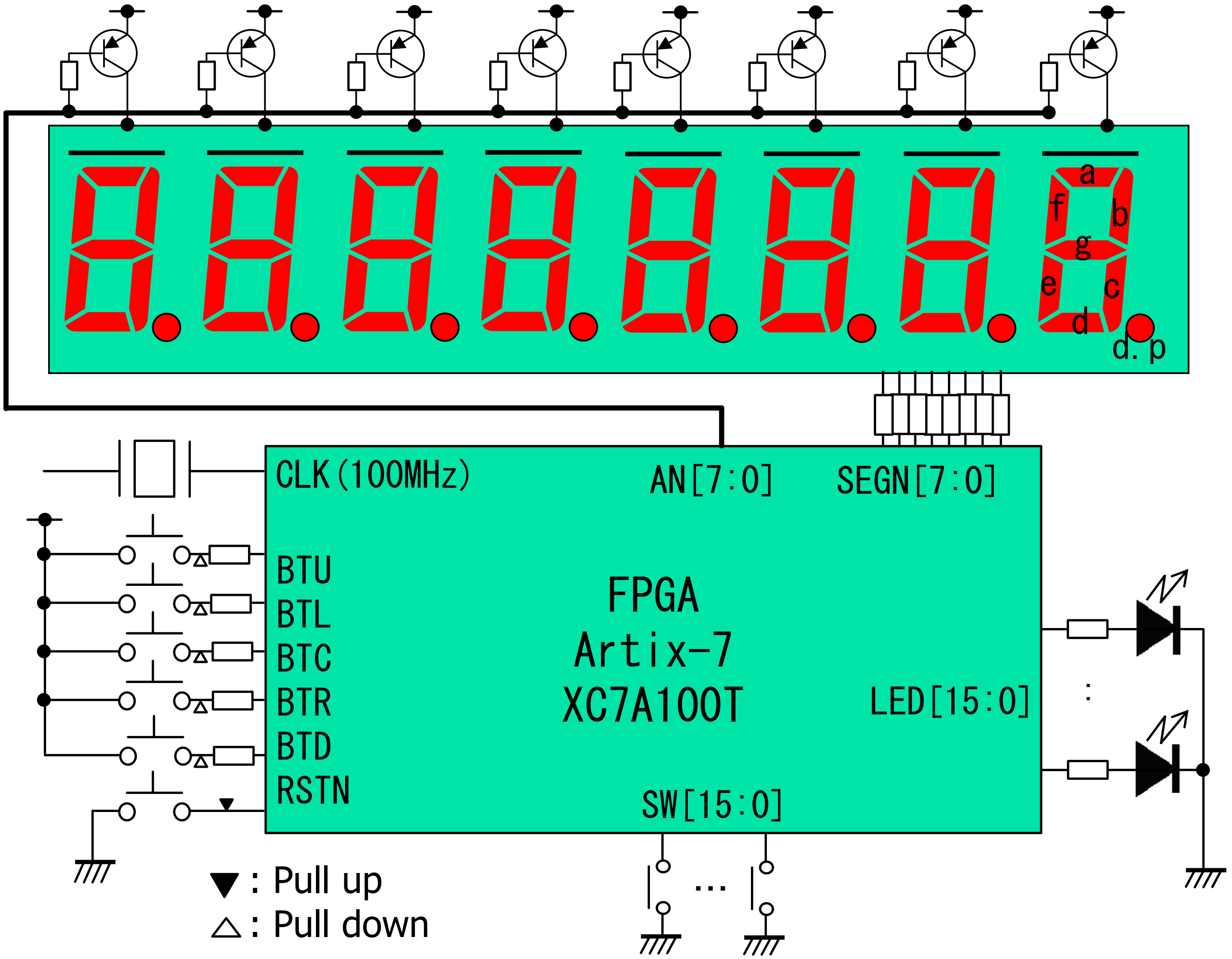
8 7-segment LEDs

16 LEDs

5 Push switches

16 slide switches

# Block Diagram of the Training Board



▼: Pull up
△: Pull down

# FPGA Pins

- All pin is 3.3V Low Voltage CMOS signal
- Last "N" letter signals are negative logic value

| Name | Dir. | PIN | Purpose |
|---|---|---|---|
| CLK | in | E3 | Clock（100MHz） |
| RSTN | in | C12 | Reset（Negative） |
| BTU | in | F15 | Push switch （Positive） |
| BTL | in | T16 | Push switch （Positive） |
| BTC | in | E16 | Push switch （Positive） |
| BTR | in | R10 | Push switch （Positive） |
| BTD | in | V10 | Push switch （Positive） |

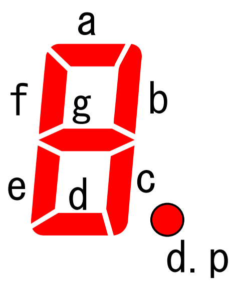
# FPGA Pins

- Output pins for 7-segment LEDs
- All signal is negative logic value

| Name | Seg. | PIN |
|---|---|---|
| SEGN[7] | a | L3 |
| SEGN[6] | b | N1 |
| SEGN[5] | c | L5 |
| SEGN[4] | d | L4 |
| SEGN[3] | e | K3 |
| SEGN[2] | f | M2 |
| SEGN[1] | g | L6 |
| SEGN[0] | d.p | M4 |

| Name | PIN | Digit |
|---|---|---|
| AN[7] | M1 | Most Significant Digit |
| AN[6] | L1 | |
| AN[5] | N4 | |
| AN[4] | N2 | |
| AN[3] | N5 | |
| AN[2] | M3 | |
| AN[1] | M6 | |
| AN[0] | N6 | Least Significant Digit |

SEGN signals control each segment of 7-segment LED. When signal is '0', then LED is turned on.



AN signals control each anode common pin of 7-segment LED. When signal is '0', then LED is turned on.

# FPGA Pins（Slide SWs，Positive）

| Name | Dir. | PIN | Digit |
|---|---|---|---|
| SW[15] | in | P4 | Most Significant Bit |
| SW[14] | in | P3 | |
| SW[13] | in | R3 | |
| SW[12] | in | T1 | |
| SW[11] | in | T3 | |
| SW[10] | in | U2 | |
| SW[9] | in | V2 | |
| SW[8] | in | U4 | |
| SW[7] | in | V5 | |
| SW[6] | in | V6 | |
| SW[5] | in | V7 | |
| SW[4] | in | R5 | |
| SW[3] | in | R6 | |
| SW[2] | in | R7 | |
| SW[1] | in | U8 | |
| SW[0] | in | U9 | Least Significant Bit |

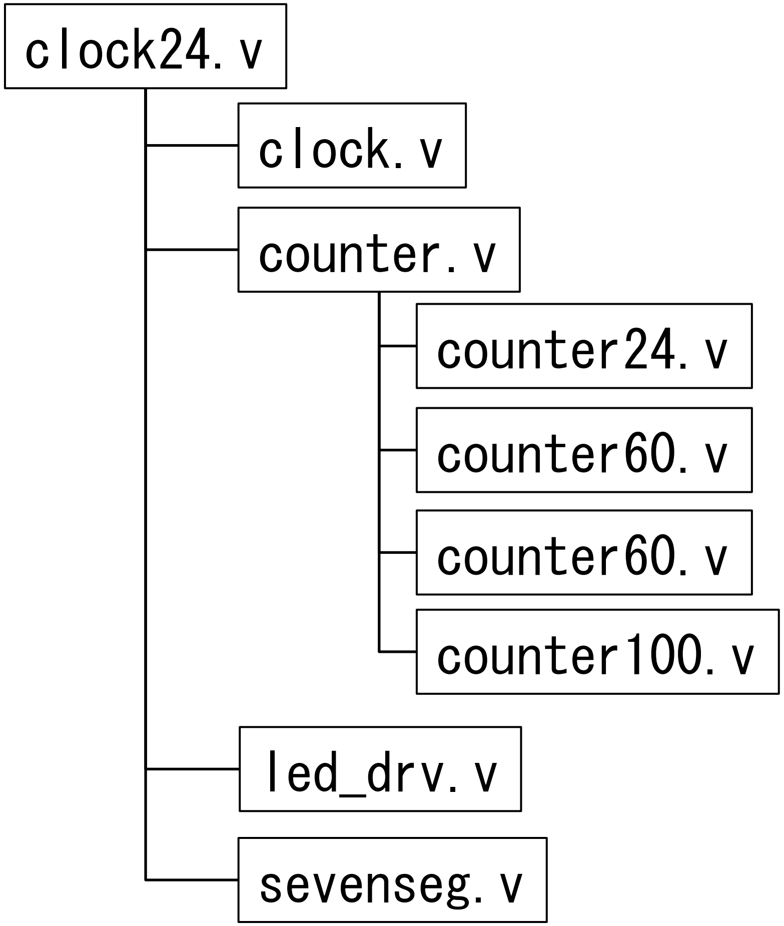
# FPGA Pins（LEDs,　Positive）

| Name | Dir. | PIN | Digit |
|---|---|---|---|
| LED[15] | out | P2 | Most Significant Bit |
| LED[14] | out | R2 | |
| LED[13] | out | U1 | |
| LED[12] | out | P5 | |
| LED[11] | out | R1 | |
| LED[10] | out | V1 | |
| LED[9] | out | U3 | |
| LED[8] | out | V4 | |
| LED[7] | out | U6 | |
| LED[6] | out | U7 | |
| LED[5] | out | T4 | |
| LED[4] | out | T5 | |
| LED[3] | out | T6 | |
| LED[2] | out | R8 | |
| LED[1] | out | V9 | |
| LED[0] | out | T8 | Least Significant Bit |

# Specification (Design sample)

- Input Signals
  - CLK: Clock 100MHz
  - RSTN: Reset for initialization
  - SETH: Increment hour counter every 1s for adjustment
  - SETM: Increment minute counter every 1s for adjustment
  - SCLR: Clear second and milli-second counters
- Counters
  - hh： Hour counter, Base 24 counter
  - mm：Minute counter, Sexagesimal counter
  - ss： Second counter, Sexagesimal counter
  - uu： milli-counter, Centesimal counter
- Milli-second counter is incremented every 10ms
- 7-segment LEDs are dynamic driven by 1kHz
- Note that I/O is either positive or negative logic value

# Module hierarchie



clock24.v：Top most module

clock.v：Timing generator, 1kHz, 100Hz

counter.v：Top module for counters

counter24.v：Base 24 counter
Count hours

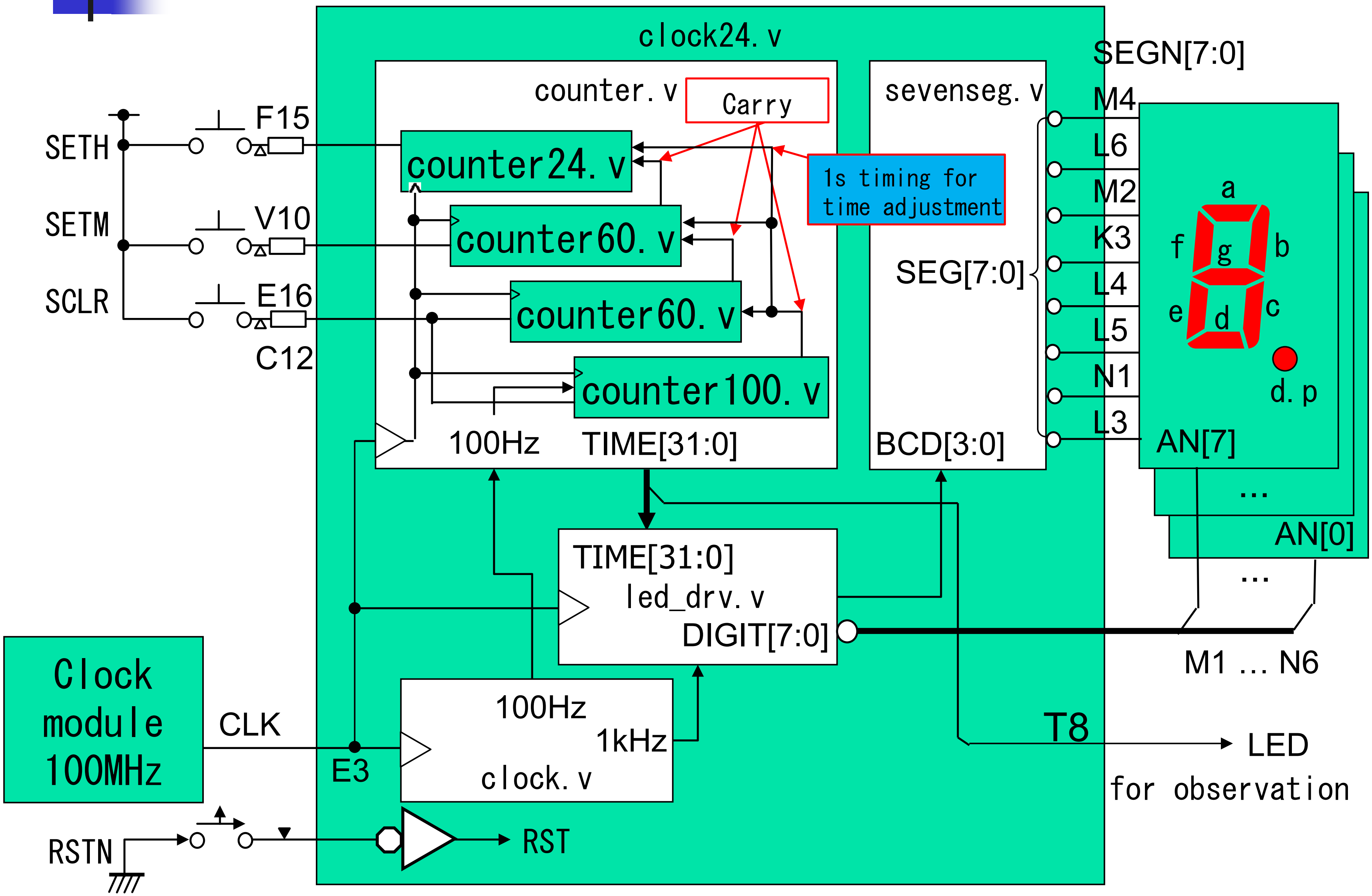
counter60.v： Sexagesimal counter
Count seconds and minutes

counter100.v： Sexagesimal counter
Count 100 and 10 milli-seconds

led_dev.v：Dynamic drive controller
for 7-segment LEDs

sevenseg.v：Converter from BCD
to 7-segment display

# Block diagram of 24-hour digital clock



# Design entry（Top module: clock24.v）

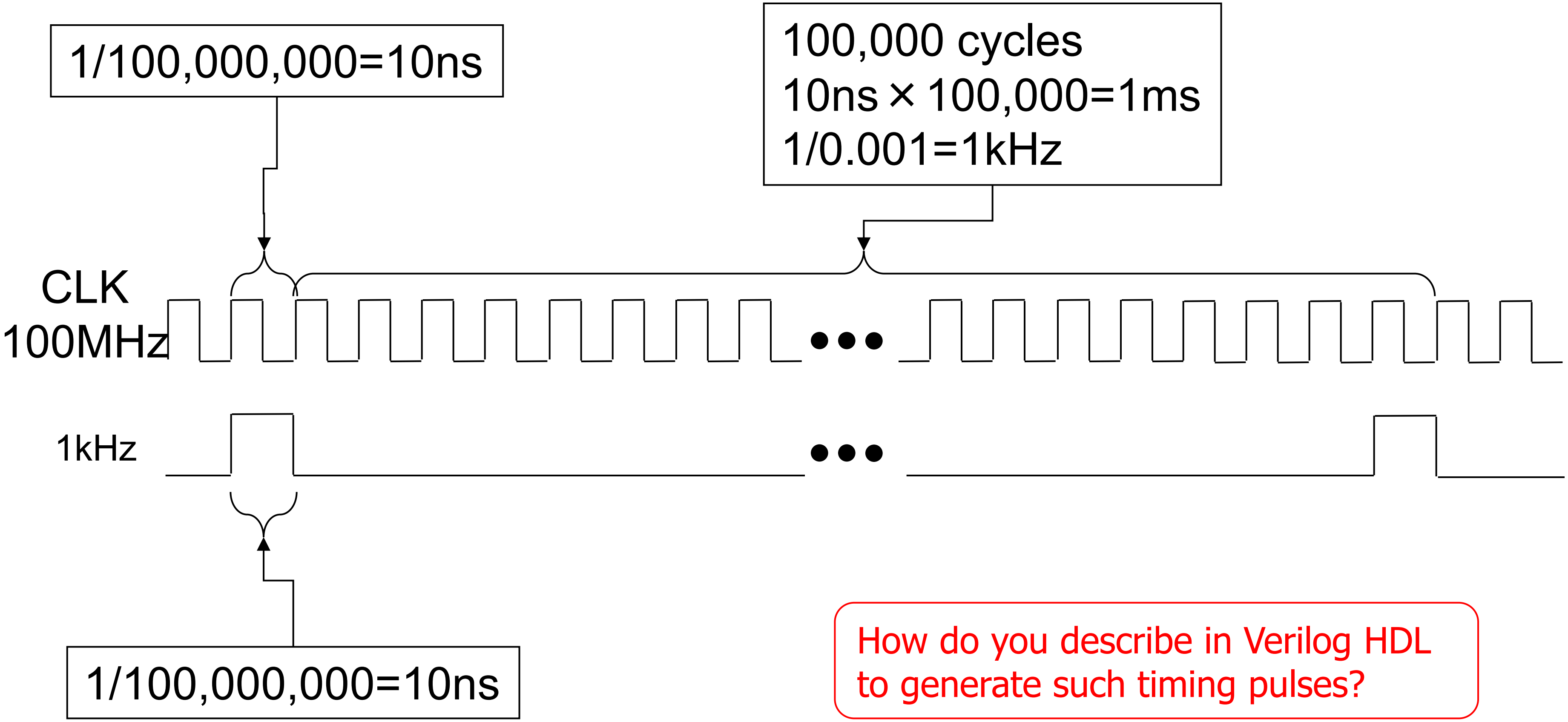
```
module clock24 ( CLK, RSTN, SETH, SETM, SCLR, SEGN, AN, LED );
  input        CLK;   // Clock (100MHz)
  input        RSTN;  // Reset (Low active)
  input        SETH;  // Set hour (High active)
  input        SETM;  // Set minute (High active)
  input        SCLR;  // Clear sec and msec (high active)
  output [7:0] SEGN;  // segment for 7 segment LED (Low active)
  output [7:0] AN;    // Digit enable for 7 segment LED (Low active)
  output       LED;   // LED (High active)

  // internal wire
  wire        _____;  // Reset (High active)
  wire [31:0] _____;  // HH:MM:ss:mm
  wire [ 3:0] _____;  // BCD value of TIME digit
  wire        _____;  // Clock enable  1ms = 1,000Hz
  wire        _____;  // Clock enable 10ms =   100Hz
  wire [ 7:0] _____;  // Segment data
  wire [ 7:0] _____;  // Digit position

  assign ____ = _______;    // Internal signals should be unified to positive signal
                            // in order to avoid errors
  clock    C0 (.CLK(___),.RST(___),.CE10(___),.CE1(___) );
  counter  C1 (.CLK(___),.RST(___),.CE10(___),.SETH(___),.SETM(___),.SCLR(___),.TIME(___));
  led_drv  C2 (.CLK(___),.RST(___),.CE(___),  .TIME(___),.BCD(___), .DIGIT(___) );
  sevenseg C3 (.BCD(___),.SEG(___));

  assign SEGN = __________; // negative signal

  assign AN  = __________;   // negative signal
  assign LED = __________;   // You had better output one second pulse for observation

endmodule
```

# User Cnstraint File: counter24.ucf

```
## Clock signal
NET "CLK"   LOC = "E3"       | IOSTANDARD = "LVCMOS33";
NET "CLK" TNM_NET = CLK_pin;
TIMESPEC TS_CLK_pin = PERIOD CLK_pin 100 MHz HIGH 50%;

## 7 segment display
NET "SEGN<7>"                LOC = "L3"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<6>"                LOC = "N1"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<5>"                LOC = "L5"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<4>"                LOC = "L4"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<3>"                LOC = "K3"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<2>"                LOC = "M2"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<1>"                LOC = "L6"     | IOSTANDARD = "LVCMOS33";
NET "SEGN<0>"                LOC = "M4"     | IOSTANDARD = "LVCMOS33";
NET "AN<0>"                  LOC = "N6"     | IOSTANDARD = "LVCMOS33";
NET "AN<1>"                  LOC = "M6"     | IOSTANDARD = "LVCMOS33";
NET "AN<2>"                  LOC = "M3"     | IOSTANDARD = "LVCMOS33";
NET "AN<3>"                  LOC = "N5"     | IOSTANDARD = "LVCMOS33";
NET "AN<4>"                  LOC = "N2"     | IOSTANDARD = "LVCMOS33";
NET "AN<5>"                  LOC = "N4"     | IOSTANDARD = "LVCMOS33";
NET "AN<6>"                  LOC = "L1"     | IOSTANDARD = "LVCMOS33";
NET "AN<7>"                  LOC = "M1"     | IOSTANDARD = "LVCMOS33";
## LED
NET "LED"                    LOC = "T8"     | IOSTANDARD = "LVCMOS33";
## Buttons
NET "RSTN"                   LOC = "C12"    | IOSTANDARD = "LVCMOS33"; # Reset (N)
NET "SCLR"                   LOC = "E16"    | IOSTANDARD = "LVCMOS33"; # Center
NET "SETH"                   LOC = "F15"    | IOSTANDARD = "LVCMOS33"; # Up
NET "SETM"                   LOC = "V10"    | IOSTANDARD = "LVCMOS33"; # Down
```

# Timing signal in clock.v

- The 1kHz timing pulse is the signal such that '1' only one period of 100MHz every one period of 1ms.



How do you describe in Verilog HDL to generate such timing pulses?

# Timing Pulse Generator：clock.v

```
module clock ( CLK, RST, CE10, CE1 );
  input  CLK;   // Clock
  input  RST;   // Reset
  output CE10;  // Clock enable 10ms (100Hz)
  output CE1;   // Clock enable  1ms (1kHz)
  reg [__:__] cnt1;
  reg [__:__] cnt2;

  always @( ____________ or ____________ )
    begin
      if(_____) cnt1 <= ____________; else
      if(_____) cnt1 <= ____________; else
                cnt1 <= ____________;
    end

  always @( ____________ or ____________ )
    begin
      if(_____) cnt2 <= ____________; else
      if(_____)
        begin
          if(_____) cnt2 <= ___________; else
                    cnt2 <= ___________;
        end
    end

  assign CE1  = ______________; // Clock enable  1ms = 1,000Hz
  assign CE10 = ______________; // Clock enable 10ms =   100Hz

endmodule
```

These counters are designed decrement counter，if possible. (Why?)

Mega-counter

Decimal counter:
Count up if mega-counter generates a carry.

10ns pulse of 1ms period.

10ns pulse of 10ms period

# Specification of Clock Counter

- Need counters for counting time
  - Milli-second counter
    - Count 10 milli-second and 100 milli-second
    - Centesimal counter
    - Clear by RST
    - Clear by SCLR for time adjustment
  - Second counter
    - Sexagesimal counter
    - Clear by RST
    - Clear by SCLR for time adjustment
  - Minute counter
    - Sexagesimal counter
    - Clear by RST
    - Increment every 1 second by SETM for time adjustment
  - Hour counter
    - base 24 counter
    - Clear by RST
    - Increment every 1 second by SETH for time adjustment

Counter may be either binary or BCD counter.
However, need the converter from binary to BCD format for 7-segment display in using binary counter.

# Top module of Clock Counter: counter.v

```
module counter ( CLK, RST, CE10, SETH, SETM, SCLR, TIME );
  input  CLK;         // Clock
  input  RST;         // Reset
  input  CE10;        // Clock enable 10ms
  input  SETH;        // Set Hour
  input  SETM;        // Set Minute
  input  SCLR;        // Clear second & milli-second
  output [31:0] TIME; // Time value

  wire      _____; // 1s timing
  wire      _____; // 1m timing
  wire      _____; // 1h timing
  wire [7:0] ____, ____, ____,  ___; // Return value from each counter

  counter100 c100 (.CLK(CLK),.RST(________),.CE(____),           .UP(____),.CNT(____));
  counter60  c60s (.CLK(CLK),.RST(________),.CE(____),           .UP(____),.CNT(____));
  counter60  c60M (.CLK(CLK),.RST(RST),     .CE(_______________),.UP(____),.CNT(____));
  counter24  c24  (.CLK(CLK),.RST(RST),     .CE(_______________),          .CNT(____));

  assign TIME = { ____, ____, ____, ____ };

endmodule
```

# Centesimal BCD counter: counter100.v

```
module counter100 ( CLK, RST, CE, CNT, UP );
  input        CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0] CNT;          // Output time
  output       UP;           // Carry
  reg    [3:0] d1, d0;       // Counter

  always @( ___________ or ___________ )
    begin
       if( _______ )
         begin
           _______;
           _______;
         end
       else if( ____ )
         begin
           if( _______ )
             begin
               ________;
               if( _______ ) _______;
               else          _______________;
             end
           else
             _______________;
         end
    end

 assign CNT = { ___, ___ }; // Output time
 assign UP  = ( __________ && __________ && ___ ) ? 1'd1 : 1'd0;

endmodule
```

Reset

Algorithm :
If unit digit value is 9, then clear unit digit and increment tens digit. Otherwise increment unit digit, though clear tens digit if tens digit value is 9.

If counter is 99 and carry up timing is next posedge, then UP is '1'.

# Sexagesimal BCD counter: counter60.v

```
module counter60 ( CLK, RST, CE, CNT, UP );
  input         CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0]  CNT;          // Output time
  output        UP;           // Carry
  reg    [3:0]  d1, d0;       // Counter

  always @( ___________ or ___________ )
    begin
       if( _______ )
         begin
           _______;
           _______;
         end
       else if( ____ )
         begin
           if( _______ )
             begin
               ________;
                if( _______ ) _______;
                else          _______________;
             end
           else
             _______________;
         end
    end

 assign CNT = { ___, ___ }; // Output time
 assign UP  = ( __________ && __________ && ___ ) ? 1'd1 : 1'd0;

endmodule
```

Reset

Algorithm：
If unit digit value is 9, then clear unit digit and increment tens digit. Otherwise increment unit digit, though clear tens digit if tens digit value is 5.

If counter is 59 and carry up timing is next posedge, then UP is '1'.

# Base 24 BCD counter: counter24.v

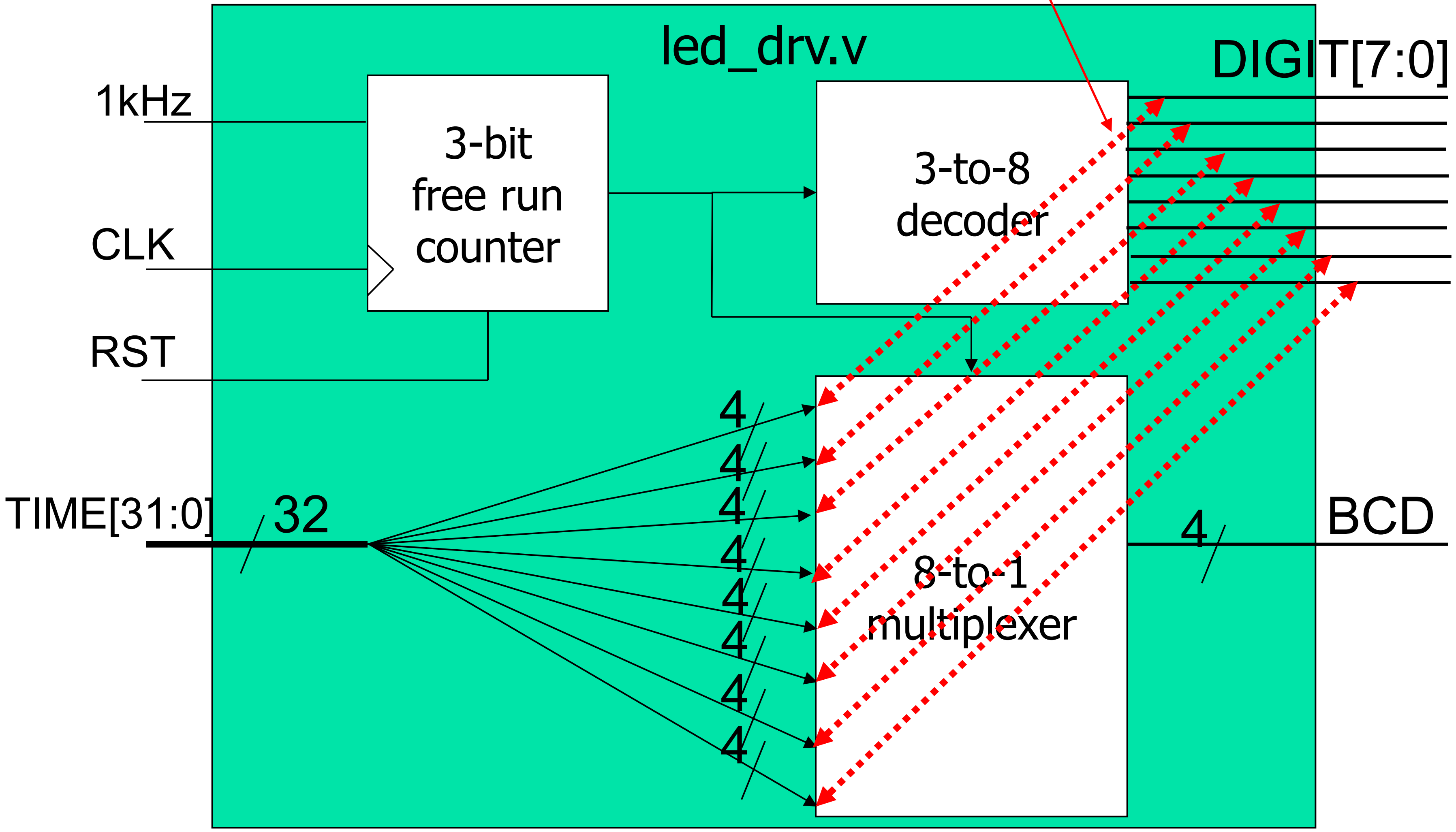
```
module counter24 ( CLK, RST, CE, CNT );
  input        CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0] CNT;          // Output time
  reg    [3:0] d1, d0;       // Counter

  always @( ___________ or ___________ )
    begin
      if( _______ )
        begin
          _______;
          _______;
        end
      else if( ___ )
        begin
          if( _______ )
            begin
              _______;
              _______________;
            end
          else if( __________________ )
            begin
              _______;
              _______;
            end
          else
            _______________;
        end
    end

  assign CNT = { ___, ___ }; // Output time
endmodule
```

Reset

Algorithm :
If unit digit value is 9, then clear unit digit and increment tens digit. Otherwise increment unit digit, though clear counter if counter value is 23.

# Design entry (LED driver: led_drv.v)

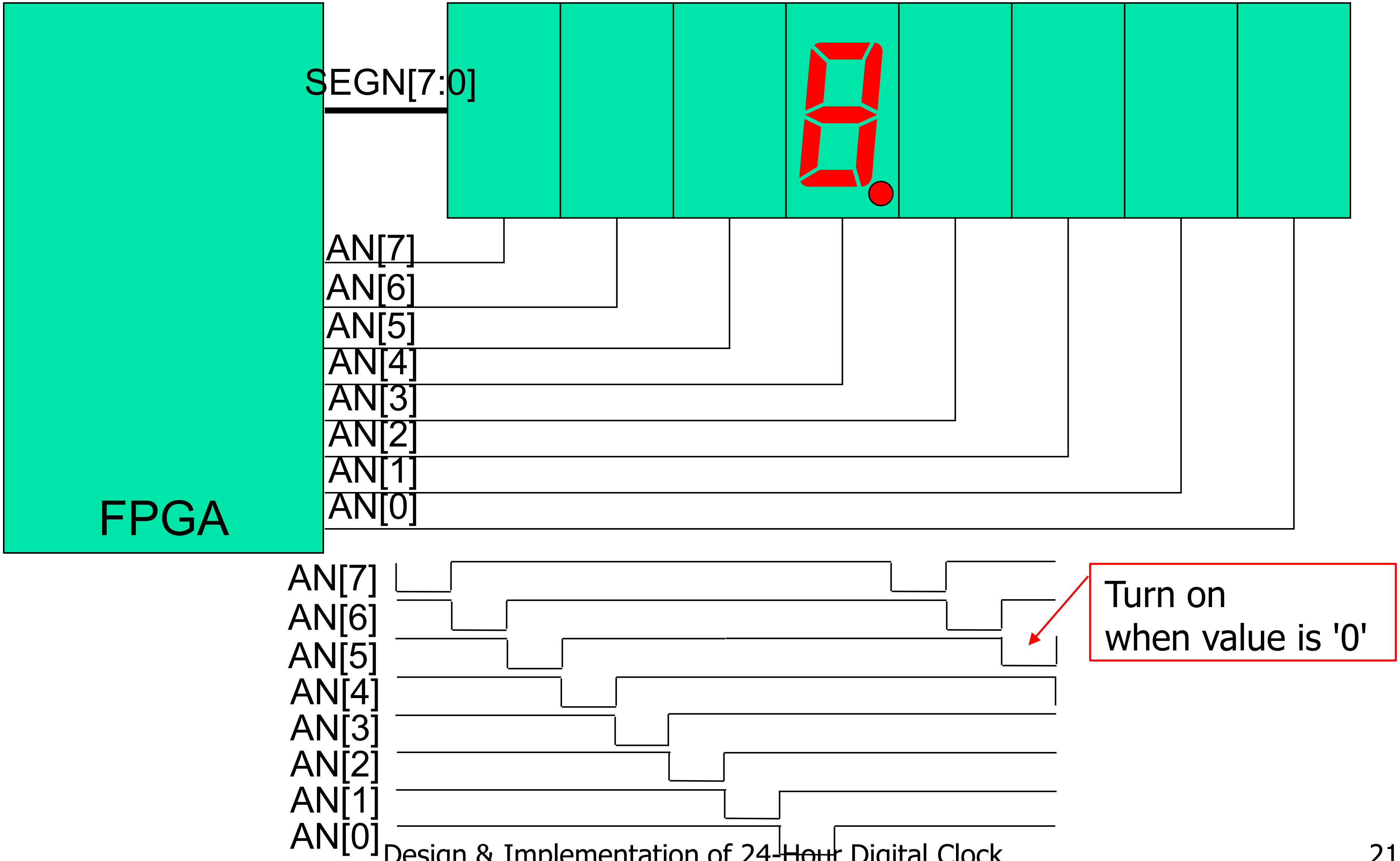
- Block diagram



Digit position and its display timing should be matched.

# 7-segment LED driver: led_drv.v

- Waveform of 7-segment LED driver.



Turn on
when value is '0'

# led_drv.v

```
module led_drv ( CLK, RST, CE, TIME,
                 BCD, DIGIT );
  input  ___________;
  input  ___________;
  input  ___________; // clock enable
  input  ___________;
  output ___________;
  output ___________;

  reg    ___________;
  reg    ___________;
  reg    ___________;

  always @(________ or ________)
  begin
    if(___) _____ <= __________; else
    if(___) _____ <= __________;
  end
```

Generate 3-bit free run counter

3-to-8 decoder

```
always @(_______)
  begin
    case(_______)
    4'b000 :DIGIT<=_________;
    4'b001 :DIGIT<=_________;
    4'b010 :DIGIT<=_________;
    4'b011 :DIGIT<=_________;
    4'b100 :DIGIT<=_________;
    4'b101 :DIGIT<=_________;
    4'b110 :DIGIT<=_________;
    4'b111 :DIGIT<=_________;
    default:DIGIT<=_________;
    endcase
  end
```

8-to-1 Multiplexer

```
always @(_______)
  begin
    case(_______)
    4'b000 :BCD<=TIME[__:__];
    4'b001 :BCD<=TIME[__:__];
    4'b010 :BCD<=TIME[__:__];
    4'b011 :BCD<=TIME[__:__];
    4'b100 :BCD<=TIME[__:__];
    4'b101 :BCD<=TIME[__:__];
    4'b110 :BCD<=TIME[__:__];
    4'b111 :BCD<=TIME[__:__];
    default:BCD<=___________;
    endcase
  end
endmodule
```

# 7-segment decoder: sevenseg.v

SEG[7:0] controls each segment
(a, b, c, d, e, f, g, d.p )
of 7-segment LED.

```
module sevenseg (BCD, SEG);
  input  ____________;
  output ____________;
  reg    ____________;
  always @(_____)
    case(______)
    4'h0:    SEG<=8'b11111100;
    4'h1:    SEG<=___________;
    4'h2:    SEG<=___________;
    4'h3:    SEG<=___________;
    4'h4:    SEG<=___________;
    4'h5:    SEG<=___________;
    4'h6:    SEG<=___________;
    4'h7:    SEG<=___________;
    4'h8:    SEG<=_________;
    4'h9:    SEG<=_________;
    default:SEG<=_________;
    endcase
endmodule
```

# Test Bench: clock24_test.v

```
`timescale 1ns/1ns       // Unit time 1ns, precision time 1ns
module clock24_text ;
  reg          CLK;      // define input variables to DUT with register type
  reg          RSTN;
  reg          SETH;
  reg          SETM;
  reg          SCLR;
  wire [7:0] SEGN;       // define output variables from DUT with wire type
  wire [7:0] AN;
  wire          LED;

  initial
    begin
      $shm_open("waves.shm");
      $shm_probe("as");
    end

  `include "clock24_test.vct"

  // Instantiate DUT module
  clock24 unit ( .CLK(CLK), .RSTN(RSTN), .SETH(SETH), .SETM(SETM), .SCLR(SCLR),
                 .SEGN(SEGN), .AN(AN), .LED(LED) );

endmodule
```

Directive to store the simulation result for wave viewer, simvision, in Verilog-XL simulator.

Include test vector file

# Test Vector: clock24_test.txt

A sample test vector is generated from right side script. This vector generates 100MHz clock and 10 million nano-second after power on reset. However, 10 million nano-second is

$$10^{7} \times 10^{-9}[s] = 10^{-2}[s] = 10[\mathrm{ms}].$$

Thus, this vector is only 0.01 seconds simulation time.

If you have further extend the simulation time, it takes too long time for simulation. So, you had better to change temporally 100,000 counter to small one for accelerating simulation. For example, if you change temporally  100,000 counter to 10 counter in "clock.v", you can simulate 10,000 times faster.

Note that don't forget write back changes, when you implement your design into FPGA.  And the right side script is eliminated the test pattern for time adjustment functions. Of course, you must test for time adjustment functions.

```
# input
RSTN
SETH
SETM
SCLR

# clock
CLK 10

# testvector
# RSTN SETH SETM SCLR
5  1       0    0    0
10 0       0    0    0
10 1       0    0    0
10000000 1 0    0    0
```

The "make_vector.pl" command translates from "clock24_test.txt" script  to "clock24_test.vct" vector file for Verilog-XL simulator.

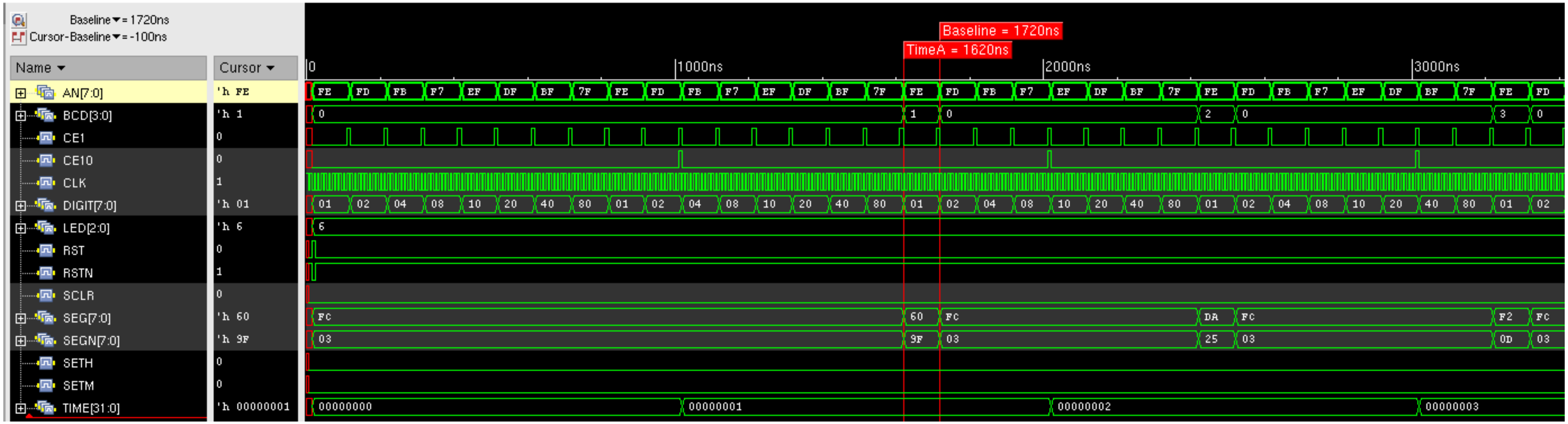
# Simulation Result

The figure shows simulation result.

This simulation result shows 10,000 times accelerated simulation, because of modifying 100,000 counter to 10 counter in "clock.v".

In the neighbor of cursors, display time of one digit is 1,720[ns]-1,620[ns]=100[ns]. It indicates 1[ms](1kHz) in real time because of 10,000 times acceleration.

"TIME[31:0]" is incremented about 1,000[ns], it indicates counting up at 10[ms]. "BCD[3:0]" becomes "1" at 1,620[ns], it indicates that "1" of the digit of 10[ms] is displayed at right most 7-segment LED.


Baseline = 1720ns
Cursor-Baseline = -100ns
Name
Cursor
Baseline = 1720ns
TimeA = 1620ns
0
1000ns
2000ns
3000ns
AN[7:0] 'h FE
BCD[3:0] 'h 1
CE1 0
CE10 0
CLK 1
DIGIT[7:0] 'h 01
LED[2:0] 'h 6
RST 0
RSTN 1
SCLR 0
SEG[7:0] 'h 60
SEGN[7:0] 'h 9F
SETH 0
SETM 0
TIME[31:0] 'h 00000001
FE FD FB F7 EF DF BF 7F
01 02 04 08 10 20 40 80
FC 60 FC DA FC F2 FC
03 9F 03 25 03 0D 03
00000000 00000001 00000002 00000003