



# A 24-Hour Digital Clock Specification and Design Example

---

Digilent NEXYS4 DDR version

Morihiro KUGA  
Kumamoto University

# Training Board (NEXYS4 DDR)

Download cable  
connector (USB)

Reset Switch

FPGA: Xilinx Artix7  
XC7A100T-CSG324

8 7-segment LEDs

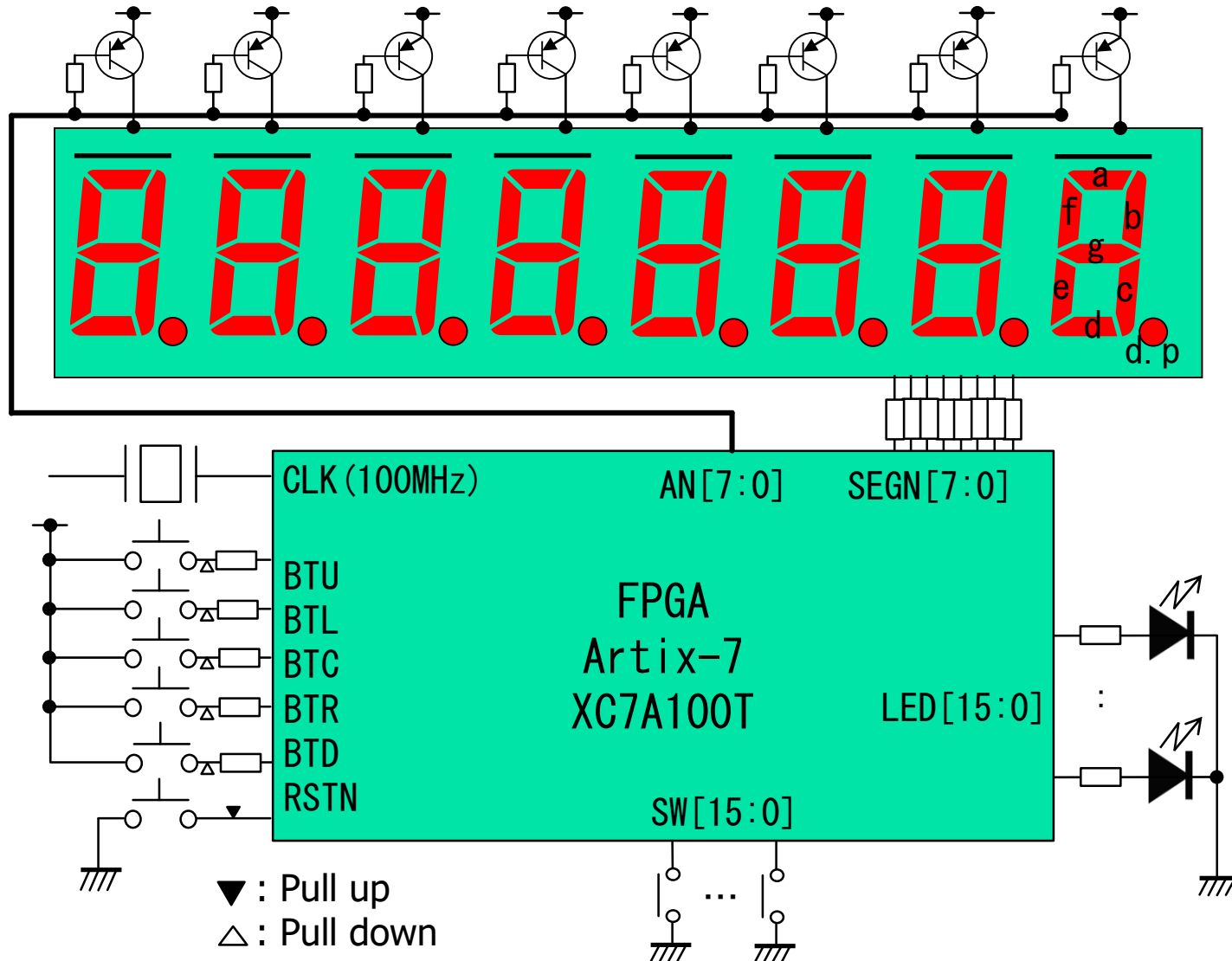
16 LEDs

5 Push switches

16 slide switches

Design & Implementation of 24-Hour Digital Clock

# Block Diagram of the Training Board





# FPGA Pins

- All the pins are 3.3V Low Voltage CMOS signals
- Last 'N' letter means negative logic signal

Name	Dir.	PIN	Purpose
CLK	in	E3	Clock(100MHz)
RSTN	in	C12	Reset(Negative)
BTU	in	M18	Push switch (Positive)
BTL	in	P17	Push switch (Positive)
BTC	in	N17	Push switch (Positive)
BTR	in	M17	Push switch (Positive)
BTD	in	P18	Push switch (Positive)

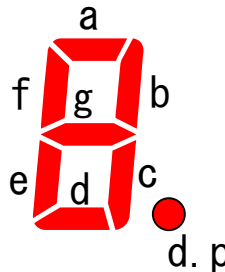
# FPGA Pins

## Output pins for 7-segment LEDs

- All the signals are negative logic

Name	Seg.	PIN
SEGN[7]	a	T10
SEGN[6]	b	R10
SEGN[5]	c	K16
SEGN[4]	d	K13
SEGN[3]	e	P15
SEGN[2]	f	T11
SEGN[1]	g	L18
SEGN[0]	d.p	H15

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



Name	PIN	Digit
AN[7]	U13	Most Significant Digit
AN[6]	K2	
AN[5]	T14	
AN[4]	P14	
AN[3]	J14	
AN[2]	T9	
AN[1]	J18	
AN[0]	J17	Least Significant Digit

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)

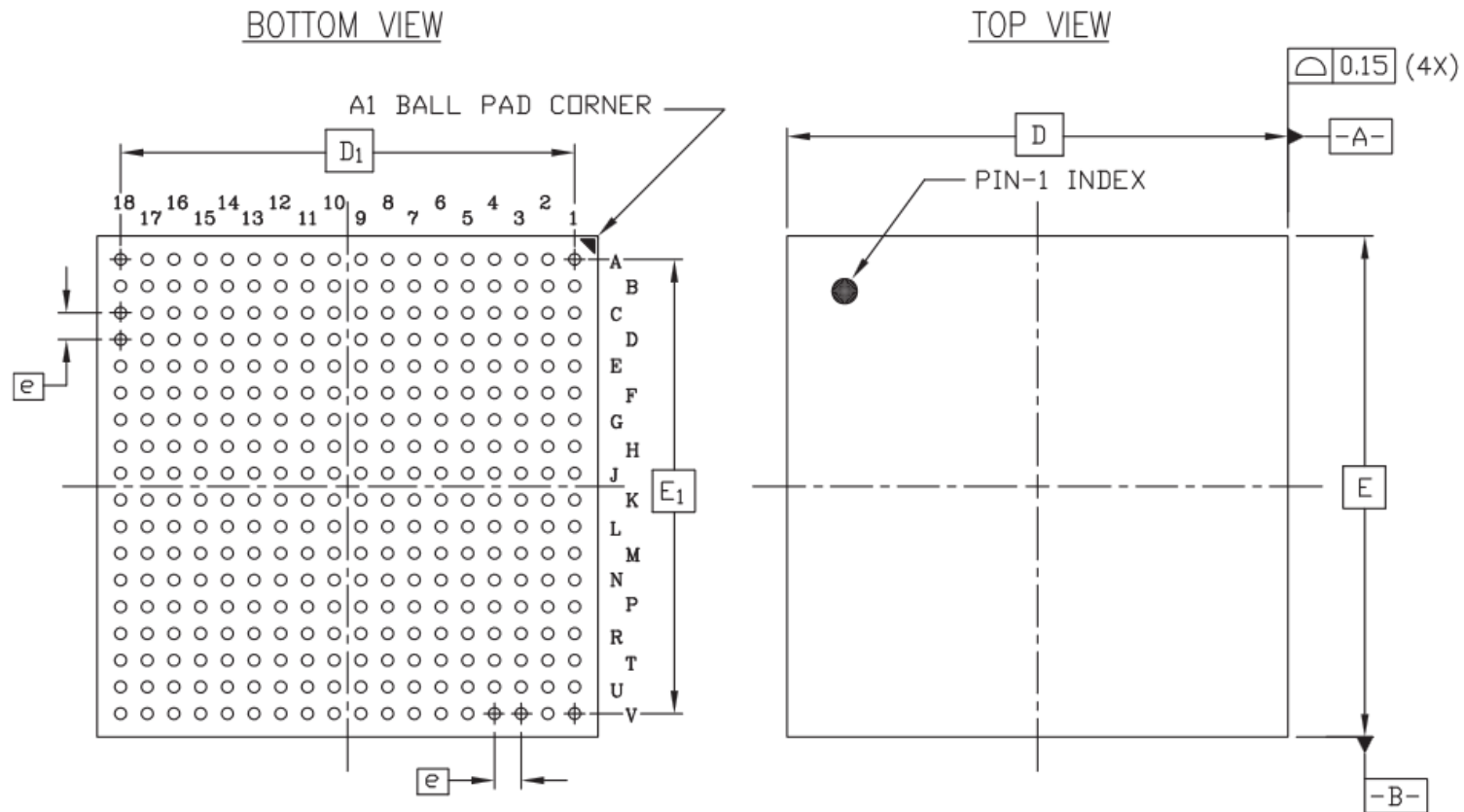
Name	Dir.	PIN	Digit
SW[15]	in	V10	Most Significant Bit
SW[14]	in	U11	
SW[13]	in	U12	
SW[12]	in	H6	
SW[11]	in	T13	
SW[10]	in	R16	
SW[9]	in	U8	
SW[8]	in	T8	
SW[7]	in	R13	
SW[6]	in	U18	
SW[5]	in	T18	
SW[4]	in	R17	
SW[3]	in	R15	
SW[2]	in	M13	
SW[1]	in	L16	
SW[0]	in	J15	Least Significant Bit

# FPGA Pins(LEDs, Positive)

Name	Dir.	PIN	Digit
LED[15]	out	V11	Most Significant Bit
LED[14]	out	V12	
LED[13]	out	V14	
LED[12]	out	V15	
LED[11]	out	T16	
LED[10]	out	U14	
LED[9]	out	T15	
LED[8]	out	V16	
LED[7]	out	U16	
LED[6]	out	U17	
LED[5]	out	V17	
LED[4]	out	R18	
LED[3]	out	N14	
LED[2]	out	J13	
LED[1]	out	K15	
LED[0]	out	H17	Least Significant Bit

# Package view of Artix-7 FPGA

CS/CSG324 および CS/CSG325 ワイヤボンド チップスケール BGA  
(Artix-7 FPGA) (0.8mm ピッチ)



Xilinx:7シリーズFPGAパッケージおよびピン配置,UG457(v1.13),2014/11/13



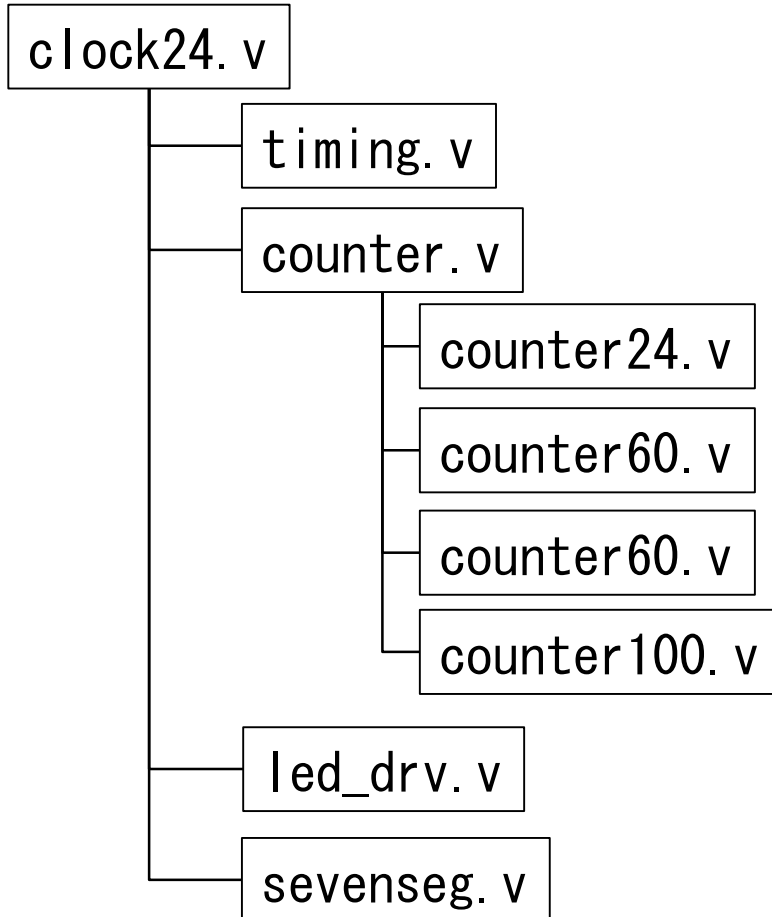


# Specification (Design sample)

---

- Input Signals
  - CLK: Clock 100MHz
  - RSTN: Reset for initialization (negative)
  - SETH: Increment hour counter every 1s for adjustment
  - SETM: Increment minute counter every 1s for adjustment
  - SCLR: Clear second and millisecond counters
- Counters
  - hh: Hour counter, Base 24 counter
  - mm: Minute counter, Base 60 (sexagesimal) counter
  - ss: Second counter, Base 60 (sexagesimal) counter
  - uu: 10 millisecond counter, Base 100 (centesimal) counter
- Millisecond counter is incremented every 10ms
- 7-segment LEDs are dynamic driven by 1kHz
- Note that I/O is either positive or negative logic value

# Hierarchy of Modules



clock24.v: Top most module

timing.v: Timing generator, 1kHz, 100Hz

counter.v: Top module for counters

counter24.v: Base 24 counter

Count hours

counter60.v: Base 60 counter

Count seconds and minutes

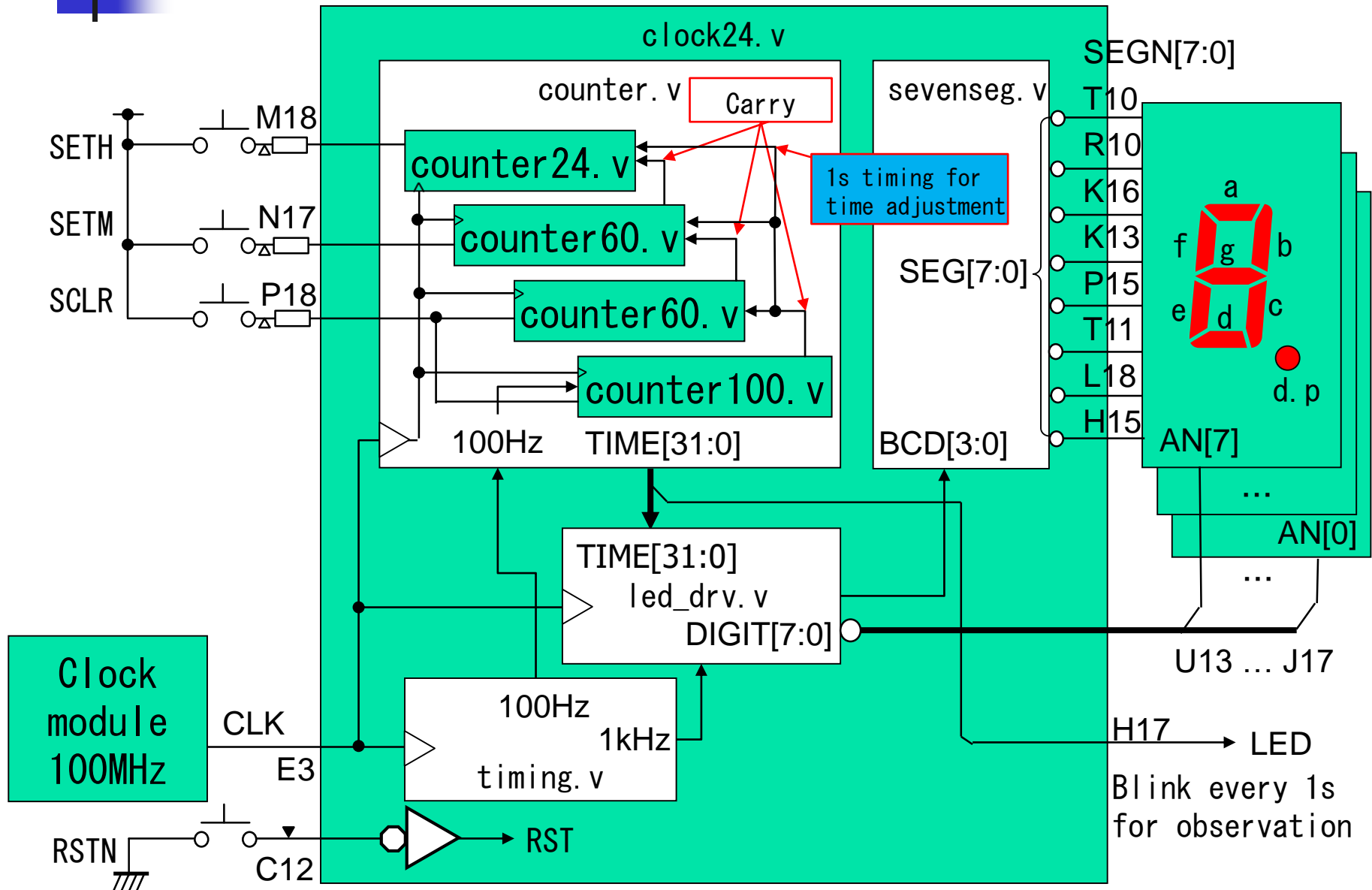
counter100.v: Base 60 counter

Count 100 and 10 milliseconds

led\_dev.v: Dynamic drive controller  
for 7-segment LEDs

sevensseg.v: Decoder from BCD  
to 7-segment display

# Block diagram of 24-hour digital clock



# User Constraint File: clock24.xdc (I)

```
## Configuration options, can be used for all designs
set_property CFGBVS VCC0 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]

## Clock signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports {CLK}];
create_clock -add -name CLK_pin -period 10.00 -waveform {0 5} [get_ports {CLK}];

## RESET
set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports {RSTN}];

## LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports {LED}];

##Buttons
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports {SETM}];
set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports {SETH}];
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports {SCLR}];
```

# User Constraint File: clock24.xdc (II)

## ##7 segment display (SEGN)

```
set_property -dict {PACKAGE_PIN T10  IOSTANDARD LVCMOS33} [get_ports {SEGN[7]}};  
set_property -dict {PACKAGE_PIN R10  IOSTANDARD LVCMOS33} [get_ports {SEGN[6]}};  
set_property -dict {PACKAGE_PIN K16  IOSTANDARD LVCMOS33} [get_ports {SEGN[5]}};  
set_property -dict {PACKAGE_PIN K13  IOSTANDARD LVCMOS33} [get_ports {SEGN[4]}};  
set_property -dict {PACKAGE_PIN P15  IOSTANDARD LVCMOS33} [get_ports {SEGN[3]}};  
set_property -dict {PACKAGE_PIN T11  IOSTANDARD LVCMOS33} [get_ports {SEGN[2]}};  
set_property -dict {PACKAGE_PIN L18  IOSTANDARD LVCMOS33} [get_ports {SEGN[1]}};  
set_property -dict {PACKAGE_PIN H15  IOSTANDARD LVCMOS33} [get_ports {SEGN[0]}};
```

## ##7 segment display (AN)

```
set_property -dict {PACKAGE_PIN J17  IOSTANDARD LVCMOS33} [get_ports {AN[0]}};  
set_property -dict {PACKAGE_PIN J18  IOSTANDARD LVCMOS33} [get_ports {AN[1]}};  
set_property -dict {PACKAGE_PIN T9    IOSTANDARD LVCMOS33} [get_ports {AN[2]}};  
set_property -dict {PACKAGE_PIN J14  IOSTANDARD LVCMOS33} [get_ports {AN[3]}};  
set_property -dict {PACKAGE_PIN P14  IOSTANDARD LVCMOS33} [get_ports {AN[4]}};  
set_property -dict {PACKAGE_PIN T14  IOSTANDARD LVCMOS33} [get_ports {AN[5]}};  
set_property -dict {PACKAGE_PIN K2    IOSTANDARD LVCMOS33} [get_ports {AN[6]}};  
set_property -dict {PACKAGE_PIN U13  IOSTANDARD LVCMOS33} [get_ports {AN[7]}};
```

# 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
timing    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(____));

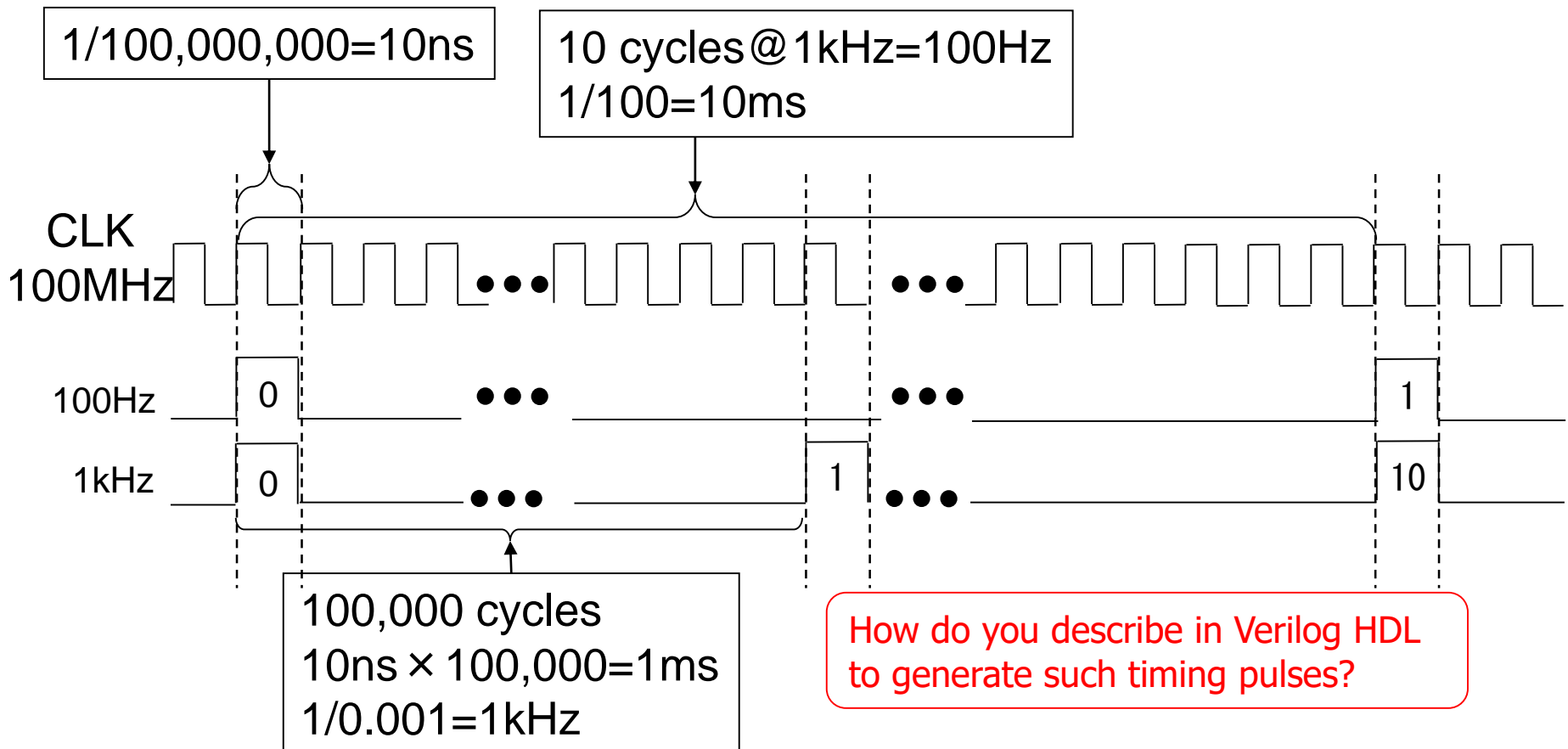
assign SEGN = _____; // negative signal

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

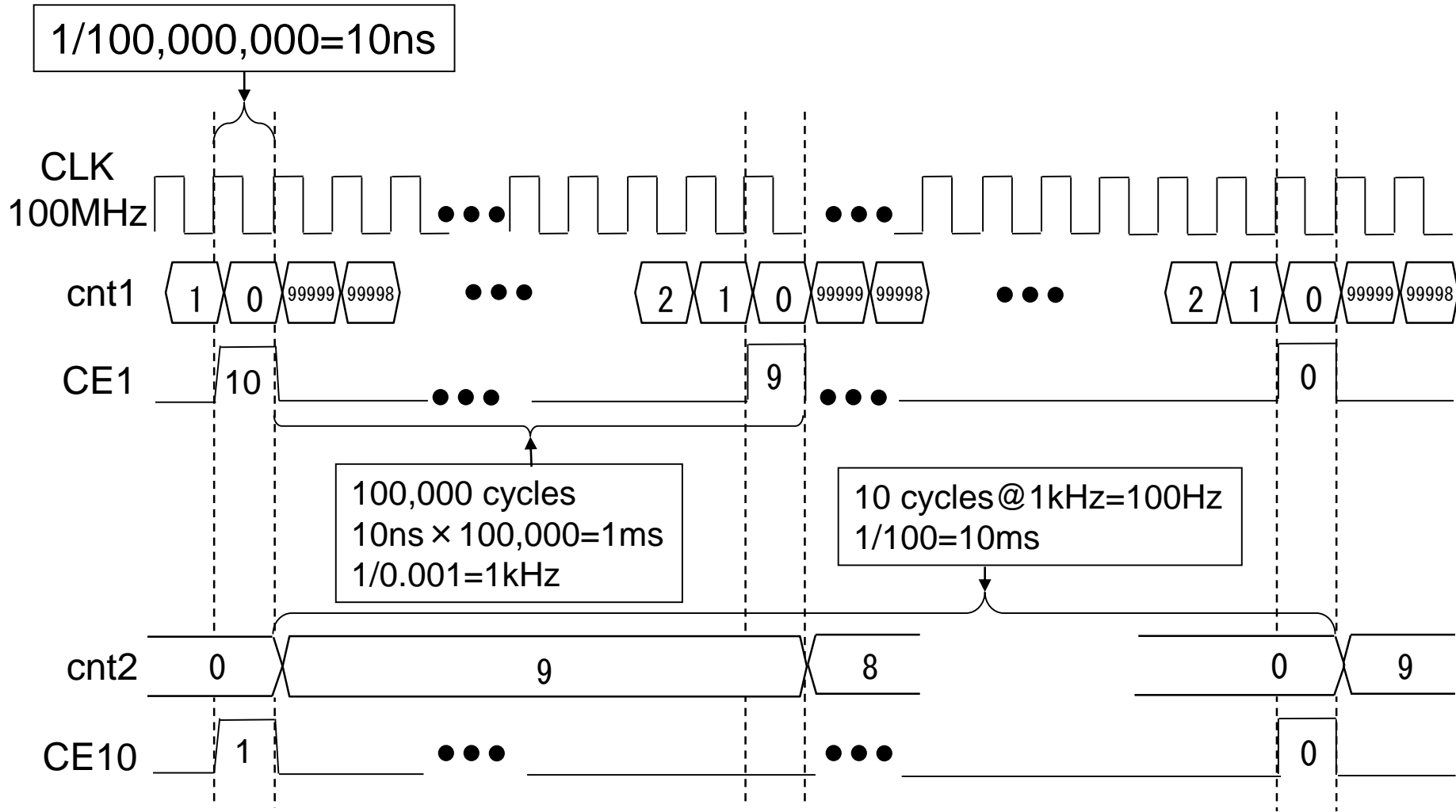
endmodule
```

# Timing Pulses in "timing.v"

- The 1kHz and the 100Hz pulses are timing pulses such that there are '1' only one period of 100MHz every one period of 1ms and 10ms, respectively.



# Detailed Waveform in "timing.v"





# Timing Pulse Generator: timing.v

```
module timing ( 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?)

1 hundred thousand counter

Decimal counter:  
Count up if 1 hundred thousand counter generates the carry.

10ns pulse of 1ms period.

10ns pulse of 10ms period

# Specification of Clock Counter

- Need counters for counting time
  - 10 millisecond counter
    - Count 10 millisecond and 100 millisecond
    - Base 100 (centesimal) counter
    - Clear by RST
    - Clear by SCLR for time adjustment
  - Second counter
    - Base 60 (sexagesimal) counter
    - Clear by RST
    - Clear by SCLR for time adjustment
  - Minute counter
    - Base 60 (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 & millisecond
    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(____), .CNT(____),.UP(____));
    counter60  c60s (.CLK(CLK),.RST(____ || ____),.CE(____), .CNT(____),.UP(____));
    counter60  c60m (.CLK(CLK),.RST(RST), .CE(____ || ____ && ____),.CNT(____),.UP(____));
    counter24  c24  (.CLK(CLK),.RST(RST), .CE(____ || ____ && ____),.CNT(____));

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

endmodule
```

# Base 100 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'.**

# Base 60 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
```

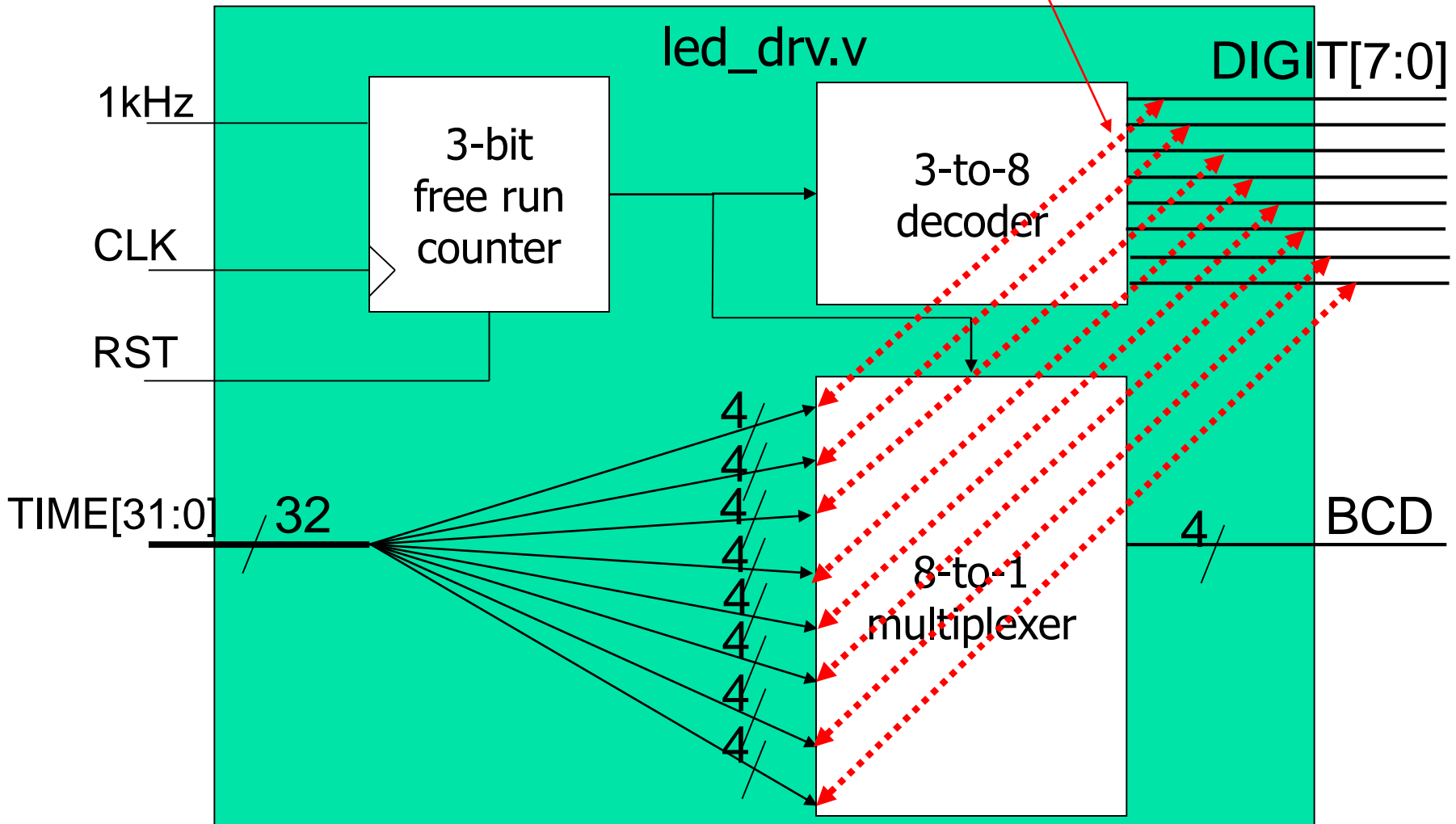
Reset

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.

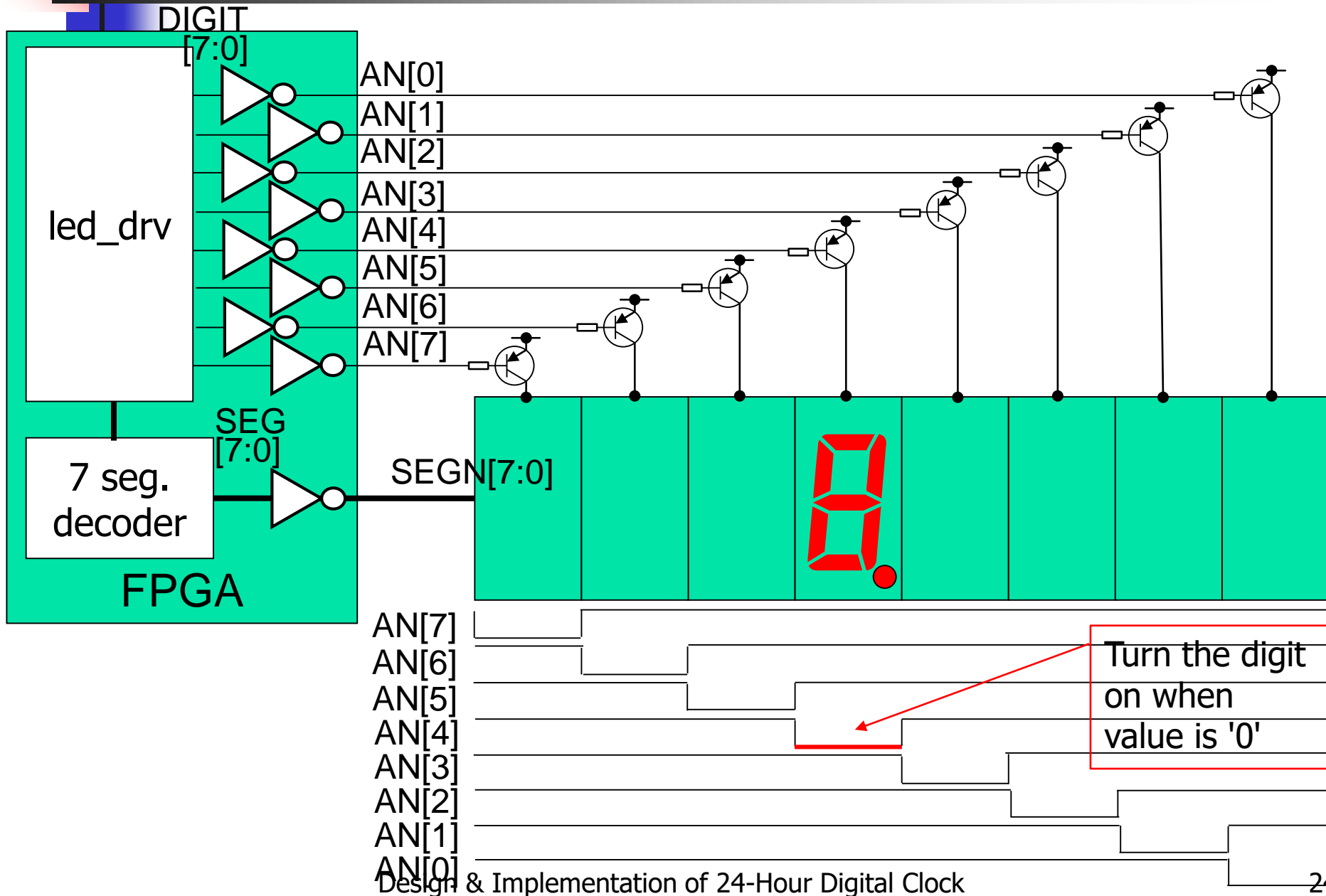
# Design entry (LED driver: led\_drv.v)

## ■ Block diagram

Digit position and its display timing should be matched.



# 7-segment LED driver: led\_drv.v





# 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

```
always @(_____)
```

3-to-8 decoder

```
begin
    case(_____)
        3'b000 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b001 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b010 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b011 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b100 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b101 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b110 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        3'b111 :DIGIT<=_____;
```

\_\_\_\_\_;

```
        default:DIGIT<=_____;
```

\_\_\_\_\_;

```
    endcase
end
```

```
always @(_____ or _____)
```

8-to-1 multiplexer

```
begin
    case(_____)
        3'b000 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b001 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b010 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b011 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b100 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b101 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b110 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        3'b111 :BCD<=TIME [____:____];
```

\_\_\_\_\_;

```
        default:BCD<=_____;
```

\_\_\_\_\_;

```
    endcase
end
endmodule
```

# 7-segment decoder: sevenseg.v

```
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<=_____;
```

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

```
            4'h8: SEG<=_____;  
            4'h9: SEG<=_____;  
            default: SEG<=_____;  
        endcase  
    endmodule
```

# Test Bench: clock24\_test.v (for iverilog)

```
`timescale 1ns/1ns          // Unit time 1ns, precision time 1ns
module clock24_test;
    reg          CLK, RSTN, SETH, SETM, SCLR;
    wire [7:0] SEGN, AN;
    wire          LED;

    initial
    begin
        $monitorh( $time, " ", dut.CE1, " ", dut.CE10, " ", dut.TIME ); // for example
    end

    clock24 dut ( .CLK(CLK), .RSTN(RSTN), .SETH(SETH), .SETM(SETM), .SCLR(SCLR),
                  .SEGN(SEGN), .AN(AN), .LED(LED) );

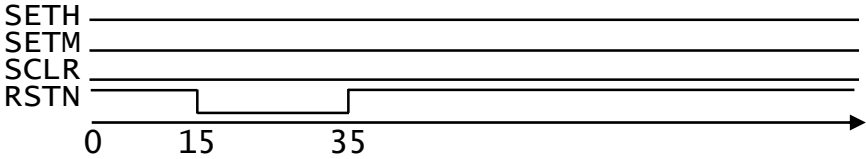
    initial begin
        CLK = 0;
        while( 1 )
            CLK = # 5 ~CLK;
    end

    end
    initial begin
        SETH = 0;
        SETM = 0;
        SCLR = 0;
        RSTN = 1;
        RSTN = #15 0;
        RSTN = #20 1;
    end
endmodule
```

You should enumerate the adequate variables to observe its behavior.

\$monitorh( ) displays the signal values in hexadecimal number.

Test vector



RSTN = #20 1;  
This assignment means that RSTN will be set "1" after 20ns.

# Test Bench: clock24\_test.v (for gtkwave)

```
`timescale 1ns/1ns          // Unit time 1ns, precision time 1ns
module clock24_test;
    reg          CLK, RSTN, SETH, SETM, SCLR;
    wire [7:0] SEGN, AN;
    wire          LED;

    initial
    begin
        $dumpfile( "clock24.vcd" );
        $dumpvars( 0, dut );
    end

    clock24 dut ( .CLK(CLK), .RSTN(RSTN), .SETH(SETH), .SETM(SETM), .SCLR(SCLR),
                  .SEGN(SEGN), .AN(AN), .LED(LED) );

    initial begin
        CLK = 0;
        while( 1 )
            CLK = # 5 ~CLK;
    end

    initial begin
        SETH = 0;
        SETM = 0;
        SCLR = 0;
        RSTN = 1;
        RSTN = #15 0;
        RSTN = #20 1;
    end

end
endmodule
```

Output the VCD file for waveform viewer

# Test Bench: clock24\_test.v (for Vivado)

```
`timescale 1ns/1ns          // Unit time 1ns, precision time 1ns
module clock24_test;
    reg          CLK, RSTN, SETH, SETM, SCLR;
    wire [7:0] SEGN, AN;
    wire          LED;

    clock24 dut ( .CLK(CLK), .RSTN(RSTN), .SETH(SETH), .SETM(SETM), .SCLR(SCLR),
                  .SEGN(SEGN), .AN(AN), .LED(LED) );

    initial begin
        CLK = 0;
        while( 1 )
            CLK = # 5 ~CLK;
    end

    initial begin
        SETH = 0;
        SETM = 0;
        SCLR = 0;
        RSTN = 1;
        RSTN = #15 0;
        RSTN = #20 1;
    end
end
endmodule
```



# Test Vector: clock24\_test.v

Right side code is a part of test vector in "clock24\_test.v".  
This vector generates 100MHz infinite clock and power on reset.

Note that you need 100 million clocks for 1 second simulation. It takes long time for simulation.

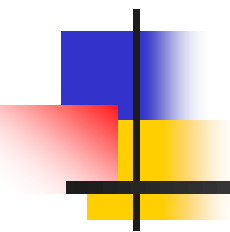
So, you had better to change temporally 100,000-counter to small one for accelerating simulation.

For example, if you change temporally 100,000-counter to 10-counter in "timing.v", you can simulate 10,000 times faster than original circuit.

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 have to test for time adjustment functions.

```
initial begin
    CLK = 0;
    while( 1 )
        CLK = # 5 ~CLK;
end

initial begin
    SETH = 0;
    SETM = 0;
    SCLR = 0;
    RSTN = 1;
    RSTN = #15 0;
    RSTN = #100 1;
end
```



# How to simulate by iverilog (CUI version)

---



# Workflow on ST terminal

---

1. Create working directory. (e.g.: `mkdir clock24`)
2. Change directory for working. (e.g.: `cd clock24`)
3. Create all source code to fill into blank parts.
4. Create test bench code "clock24\_test.v" with observing variables in `$monitor( )` task.
5. Start compile by iverilog command;  
`iverilog -o clock24_test clock24_test.v clock24.v ...`
6. Start simulation by `./clock24_test` command.
7. Consider the simulation result to verify the behavior.



# Simulation Result Example

Compile

```
$ iverilog clock24_test.v
clock24.v timing.v counter.v
counter24.v counter100.v
counter60.v seven_seg.v
led_drv.v -o clock24_test
$ ./clock24_test
```

Start simulation

```
0 x x xxxxxxxx
15 0 0 00000000
1000015 1 0 00000000
1000025 0 0 00000000
2000015 1 0 00000000
2000025 0 0 00000000
3000015 1 0 00000000
3000025 0 0 00000000
4000015 1 0 00000000
4000025 0 0 00000000
5000015 1 0 00000000
5000025 0 0 00000000
6000015 1 0 00000000
6000025 0 0 00000000
7000015 1 0 00000000
7000025 0 0 00000000
8000015 1 0 00000000
8000025 0 0 00000000
9000015 1 0 00000000
...
```

Simulation time (ns)

dut.CE1

dut.CE10

dut.TIME

```
...
9000025 0 0 00000000
10000015 1 1 00000000
10000025 0 0 00000001
11000015 1 0 00000001
98000025 0 0 00000009
99000015 1 0 00000009
99000025 0 0 00000009
100000015 1 1 00000009
100000025 0 0 00000010
101000015 1 0 00000010
101000025 0 0 00000010
102000015 1 0 00000010
102000025 0 0 00000010
103000015 1 0 00000010
103000025 0 0 00000010
104000015 1 0 00000010
104000025 0 0 00000010
```

```
^C** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 104228475
ticks.
> finish
** Continue **
$
```

Ctrl+C: stop simulation

Exit from iverilog



## How to use Vivado Tool (CUI version)

---

"Vivado" is a design tool for Xilinx's FPGA.



# Workflow on ST terminal

1. Create working directory. (e.g.: `mkdir clock24`)
2. Change directory for working. (e.g.: `cd clock24`)
3. Create source code directory. (e.g.: `mkdir srcs`)
4. Copy all source code into "srcs" directory.
5. Copy constraint file "clock24.xdc" into "srcs".
6. Prepare "clock24.tcl" file for vivado script.
7. Start compile by vivado command;  
**`source ~kuga/setup/vivado2019.2`**  
**`vivado -mode tcl -source clock24.tcl`**
8. Check error log in "vivado.log".
9. Check report file in impl\_1 directory.
10. Check an existence of **"clock24.bit"**.

```
./srcs/  
clock24.v  
timing.v  
counter.v  
    counter24.v  
    counter60.v  
    counter100.v  
led_drv.v  
seven_seg.v  
clock24.xdc  
./clock24.tcl
```



# Vivado script "clock24.tcl"

---

1. `create_project clock24 ./clock24 -part xc7a100tcsg324-1 -force`
2. `set_property board_part digilentinc.com:nexys4_ddr:part0:1.1 [current_project]`
3. `add_files { ./srcs/seven_seg.v ./srcs/counter.v ./srcs/counter24.v  
./srcs/counter100.v ./srcs/clock24.v ./srcs/led_drv.v ./srcs/counter60.v  
./srcs/timing.v }`
4. `update_compile_order -fileset sources_1`
5. `add_files -fileset constrs_1 -norecurse ./srcs/clock24.xdc`
6. `launch_runs impl_1 -to_step write_bitstream -jobs 2`
7. `wait_on_run impl_1`
8. `exit`

When you use GUI of Vivado, above commands are executed in background.  
You can execute these commands as a script command.

1. Create project named adder8.
2. Set board information with FPGA device.
3. Add source files.
4. Update compile order
5. Add constraint file
7. Launch logic synthesis, placing, routing, generate bit stream.
8. Wait on run.
9. Exit form Vivado.



# Submit your designed files

- You MUST submit your source codes, simulation result summary and bitstream file on Moodle.

## 「課題1-2: 24時間デジタル時計の設計データ提出」

- Source codes
  - clock24.v, timing.v, counter.v, counter24.v, counter60.v, counter100.v, led\_drv.v, seven\_seg.v
- Simulation result summary
  - Summarize your simulation result as a PDF file.
    - Don't need all simulation result.
    - Pick up an important point to explain the correct behavior, including following points;
      - include of from 0.99s to 1.00s in TIME result
      - include of time adjustment function by SCLR to clear sec and msec counters
- Bitstream file
  - clock24.bit



## How to use Vivado Tool (GUI version)

---

"Vivado" is a design tool for Xilinx's FPGA.



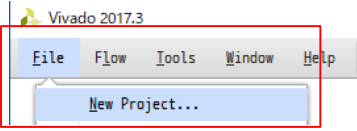
# Invoke Vivado

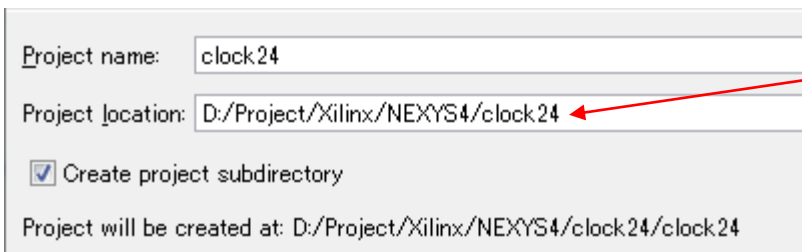
---

1. Create a working directory (e.g. mkdir clock24)
2. Create "srcs" directory in working dir.
3. Copy 10 files in "srcs" directory
4. Setup environment by following commands;  
`source ~kuga/setup/vivado2019.2`
6. Change working dir. (e.g. cd clock24)
7. Invoke FPGA design tool named Vivado;  
`vivado &`

```
clock24_test.v
clock24.v
  timing.v
  counter.v
    counter24.v
    counter60.v
    counter100.v
  led_drv.v
  seven_seg.v
clock24.xdc
```

# Create Project (I)

1. Select  to create new project
2. Click "Next" in "Create a New Vivado Project"
3. Set "Project Name" and "Project location" in "Project Name" window



Project name: clock24

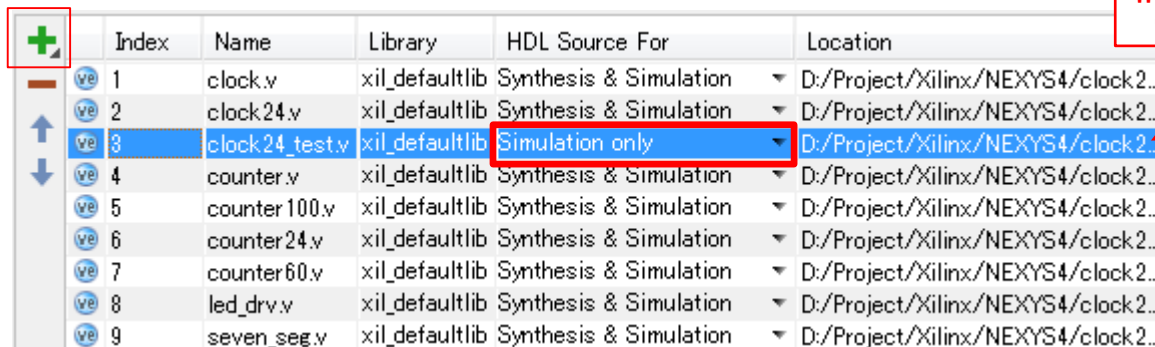
Project location: D:/Project/Xilinx/NEXYS4/clock24


☒ Create project subdirectory

Project will be created at: D:/Project/Xilinx/NEXYS4/clock24/clock24

Select your working directory

4. Click "Next" in "Project Type" after set "RTL Project"
5. Set design files in "Add Sources" window



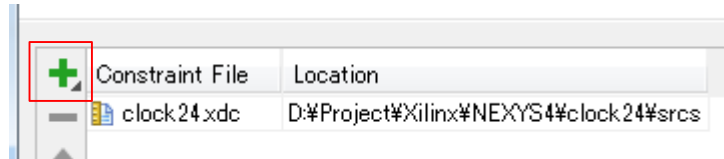
	Index	Name	Library	HDL Source For	Location
	1	clock.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	2	clock24.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	3	clock24_test.v	xil_defaultlib	Simulation only	D:/Project/Xilinx/NEXYS4/clock2...
	4	counter.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	5	counter100.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	6	counter24.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	7	counter60.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	8	led_drv.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...
	9	seven_seg.v	xil_defaultlib	Synthesis & Simulation	D:/Project/Xilinx/NEXYS4/clock2...

Don't forget to change "Simulation only".

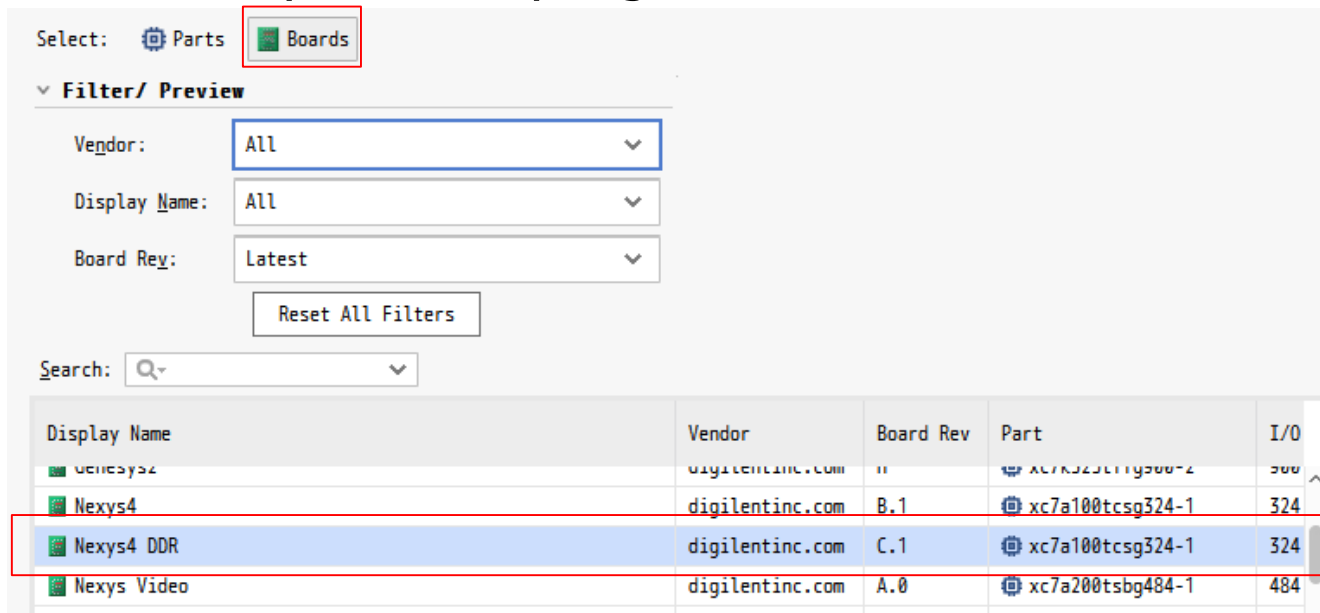


# Create Project (II)

- Set constraint file in "Add Constraints" window



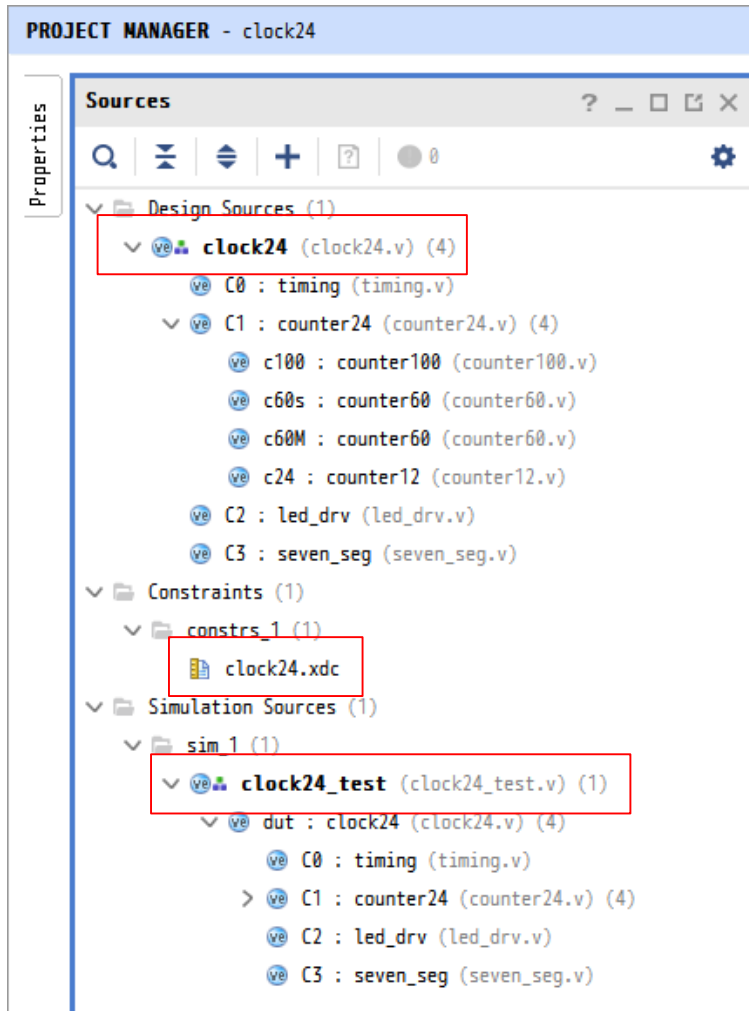
- Select a target FPGA board
  - Select Nexys4 DDR by Digilent inc.



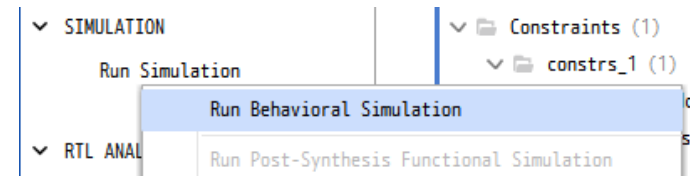
- Click "Next" and click "Finish" after checking "New Project Summary"

# Functional Simulation (I)

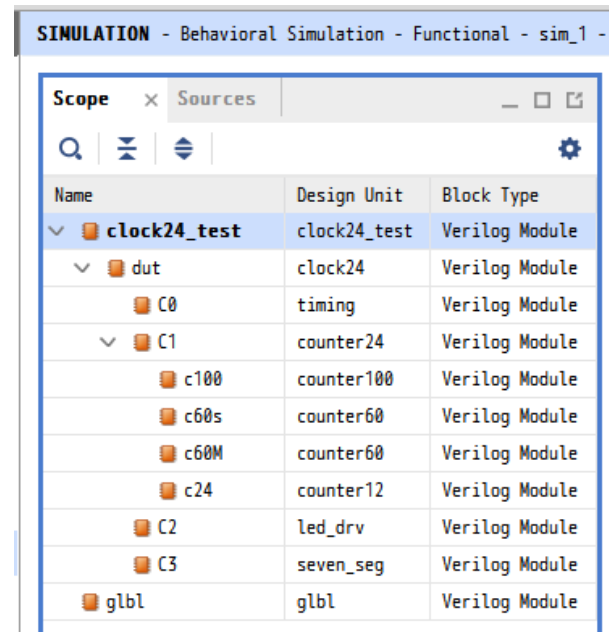
1. Check registered files in "Sources" box



2. Click "Run Behavioral Simulation" after click "Run Simulation" to invoke simulator



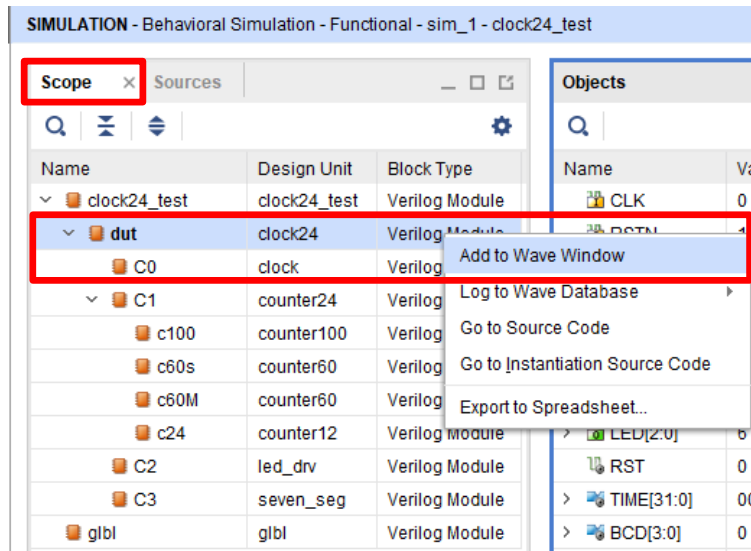
3. Simulator window will arise



# Functional Simulation (II)

## 4. Register observation signals on "waveform viewer"

1. Select "dut" in "Name" box
2. Click right SW on "dut" and "Add to Wave Window" for observing signals on "waveform viewer"
3. Some signals in "dut" module are added on "waveform viewer"

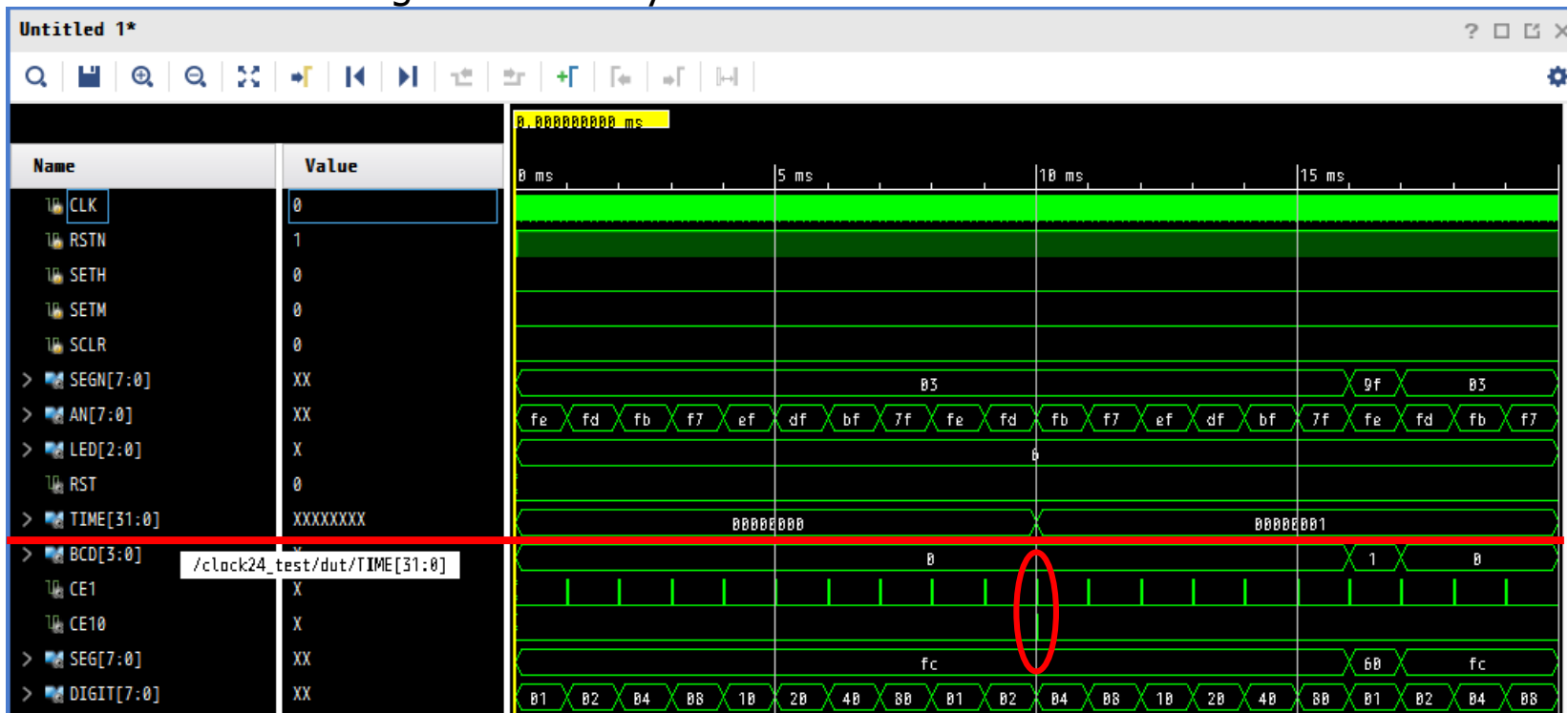
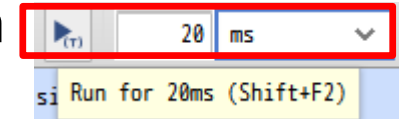
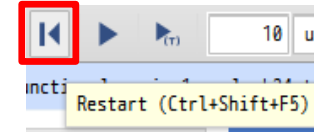


Name	Value	Data T...
CLK	0	Logic
RSTN	1	Logic
SETH	0	Logic
SETM	0	Logic
SCLR	0	Logic
SEGN[7:0]	03	Array
AN[7:0]	fe	Array
LED[2:0]	6	Array
RST	0	Logic
TIME[31:0]	00000000	Array
BCD[3:0]	0	Array
CE1	0	Logic
CE10	0	Logic
SEG[7:0]	fc	Array
DIGIT[7:0]	01	Array

Name	Value
CLK	0
RSTN	1
SETH	0
SETM	0
SCLR	0
SEGN[7:0]	03
AN[7:0]	fe
LED[2:0]	6
RST	0
TIME[31:0]	00000000
BCD[3:0]	0
CE1	0
CE10	0
SEG[7:0]	fc
DIGIT[7:0]	01

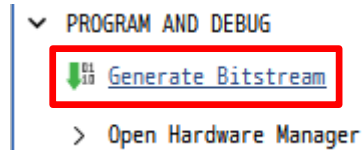
# Functional Simulation (III)

5. Reset simulation to click "Restart" icon
6. Start 20 milliseconds simulation to click "Run" icon
7. You can see simulation result in waveform viewer
  - "TIME" variable indicates time
  - It is incremented by CE10 timing pulse every 10ms
  - It takes long time to verify for 1s simulation



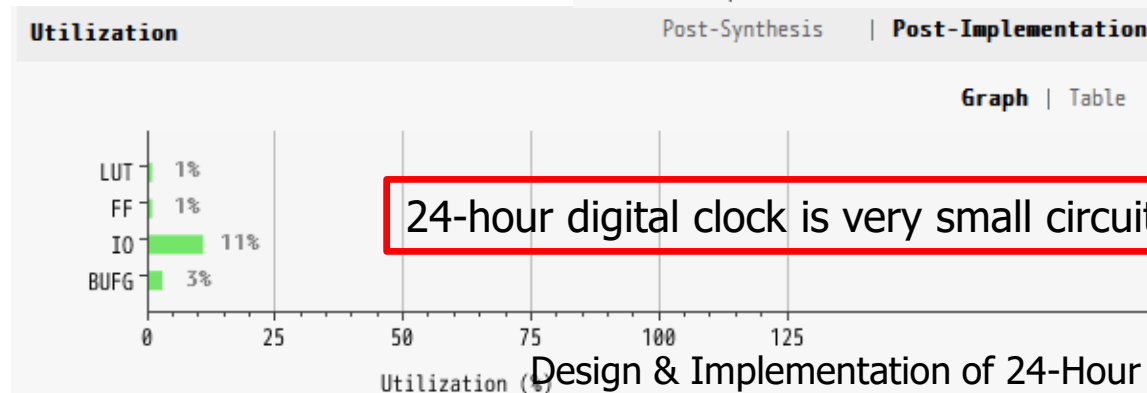
# Implementation

1. Click "Generate Bitstream"  
to create bitstream file (clock24.bit)
  - All layout process, such as logic synthesis, optimization, placing, routing and timing analysis are invoked
2. After layout process, "Bitstream Generation Completed" window arises
3. Click "OK" to check layout view
4. If error occurs, debug your code

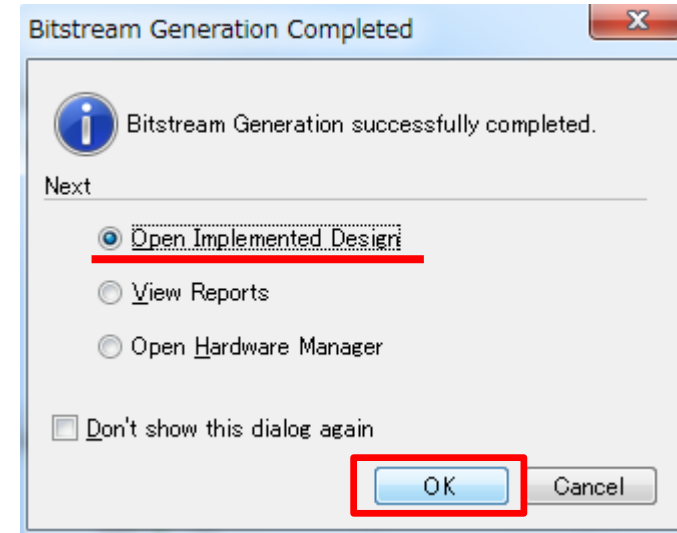


Synthesis	
Status:	✓ Complete
Messages:	4 warnings
Part:	xc7a100tcs9324-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation	
Status:	✓ Complete
Messages:	No errors or warnings
Part:	xc7a100tcs9324-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental compile:	None

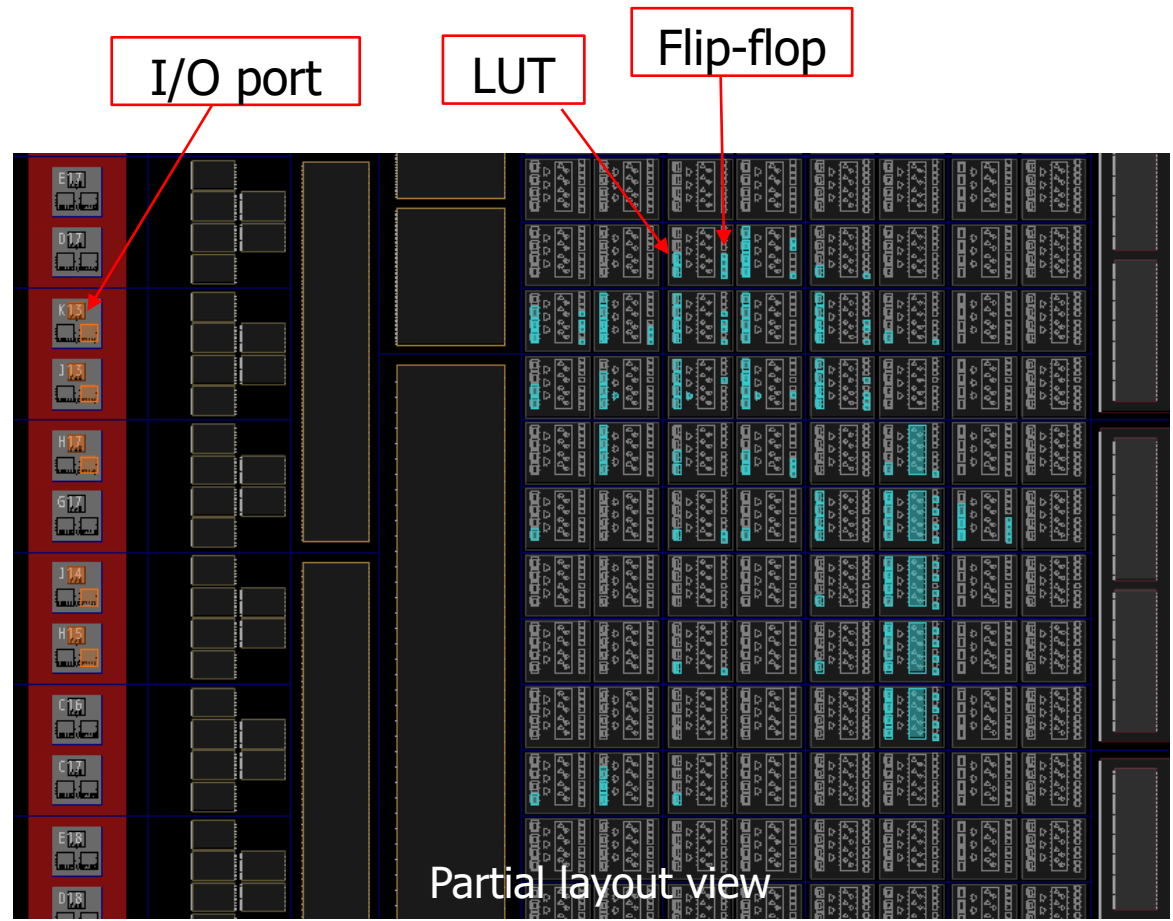
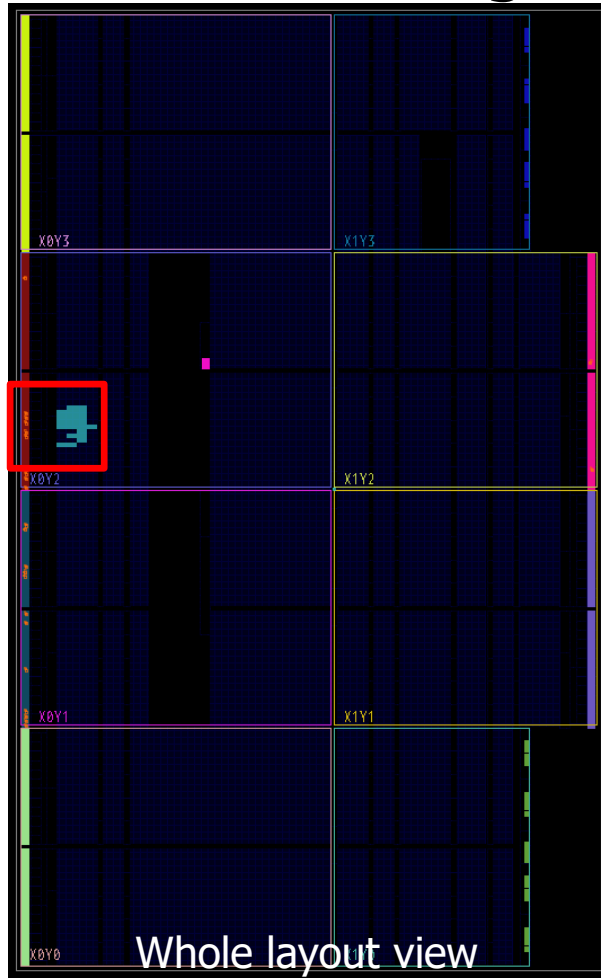


24-hour digital clock is very small circuit



# Implemented Design (Layout View)

- All the logics are mapped to FPGA resources





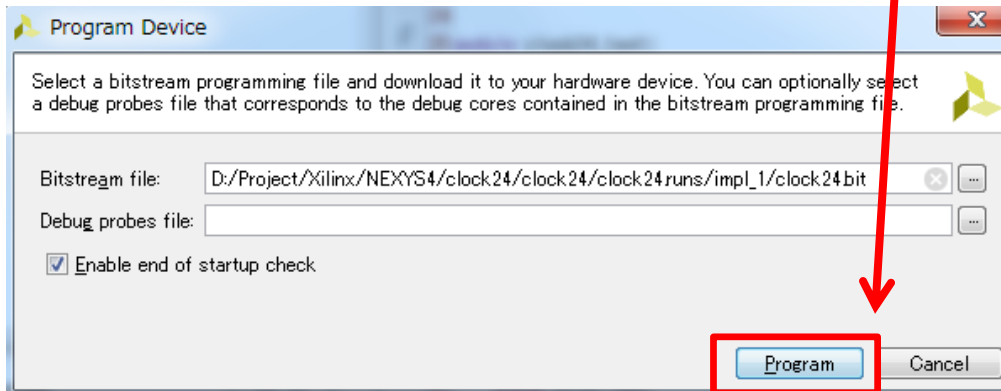
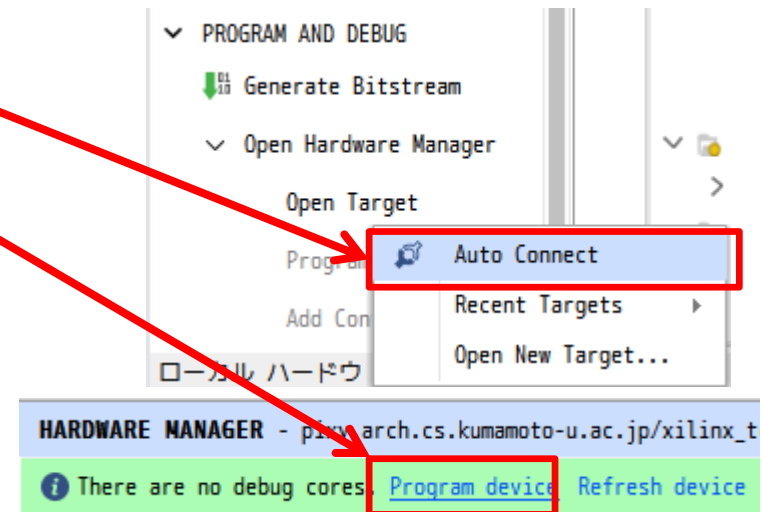
# How to Validate Circuit Behavior

---

- FPGA requires a Bitstream file to configure on FPGA circuits.
- To configure (download), use “Hardware Manager” one of the Vivado’s command.
- Please contact to teaching staff for validating your design.

# FPGA Board on Local Host

1. Invoke "Hardware Manager" by  
Open Hardware Manager  $\Rightarrow$  Open Target
2. Click "Auto Connect"
3. Click "Program device"
4. Finally, click "Program"
5. Check circuit behavior





# FPGA Board on Remote Host

1. Invoke "Hardware Manager" by  
Open Hardware Manager  $\Rightarrow$  Open New Target
2. Click "Next"
3. Select "Remote server",  
Set Host name "172.28.103.81"
4. Click "Next", "Next" and "Finish"
5. Click "Program device"
6. Finally, click "Program"
7. Check circuit behavior

There are 2 host PCs.

DELL: 172.28.103.81

HP: 172.28.103.82

