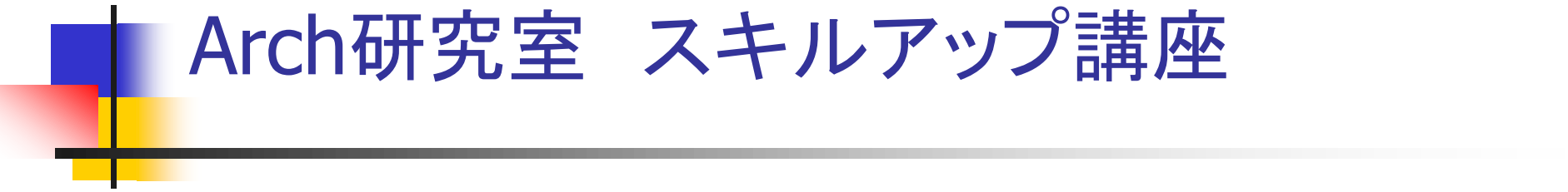
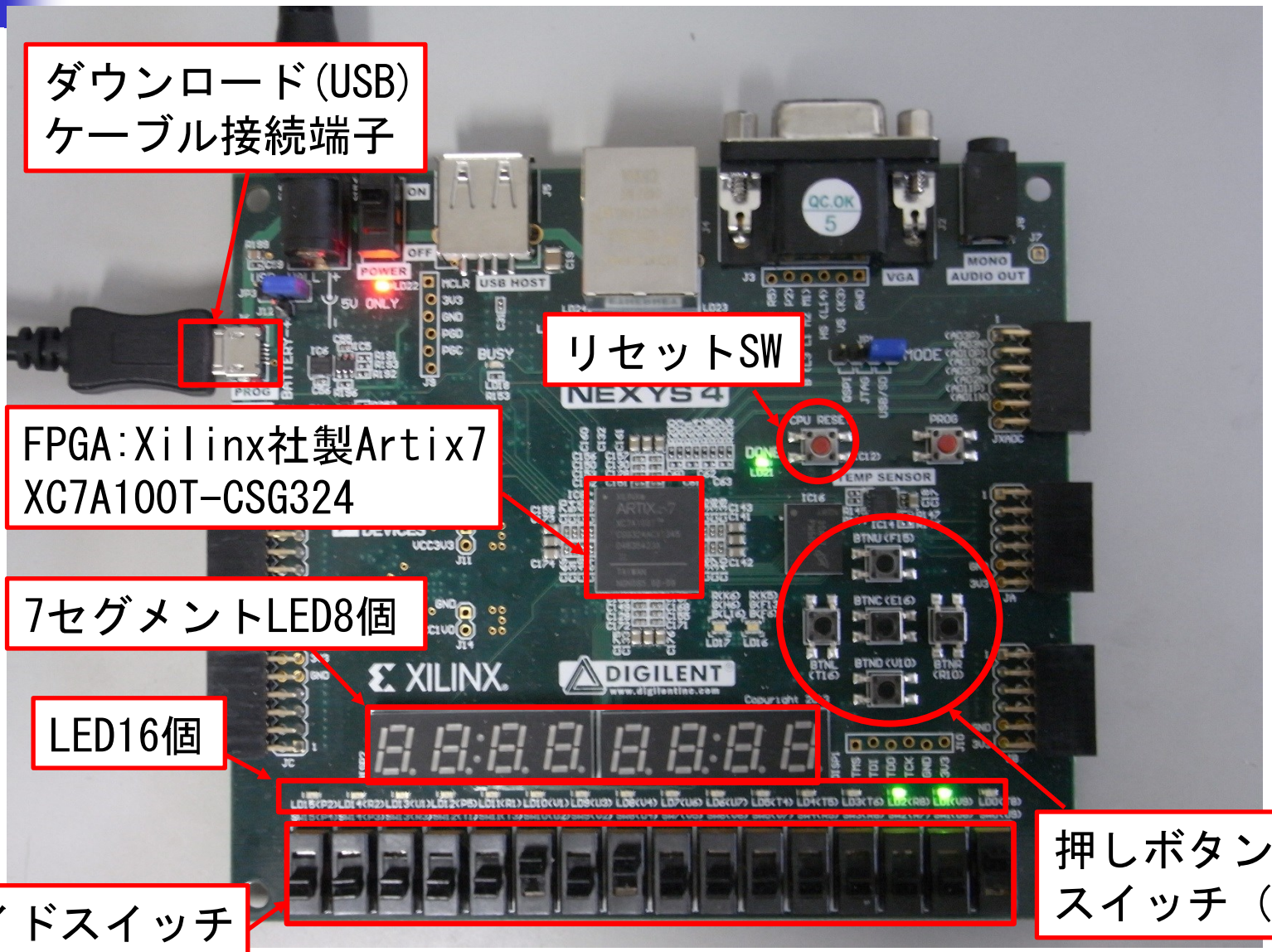


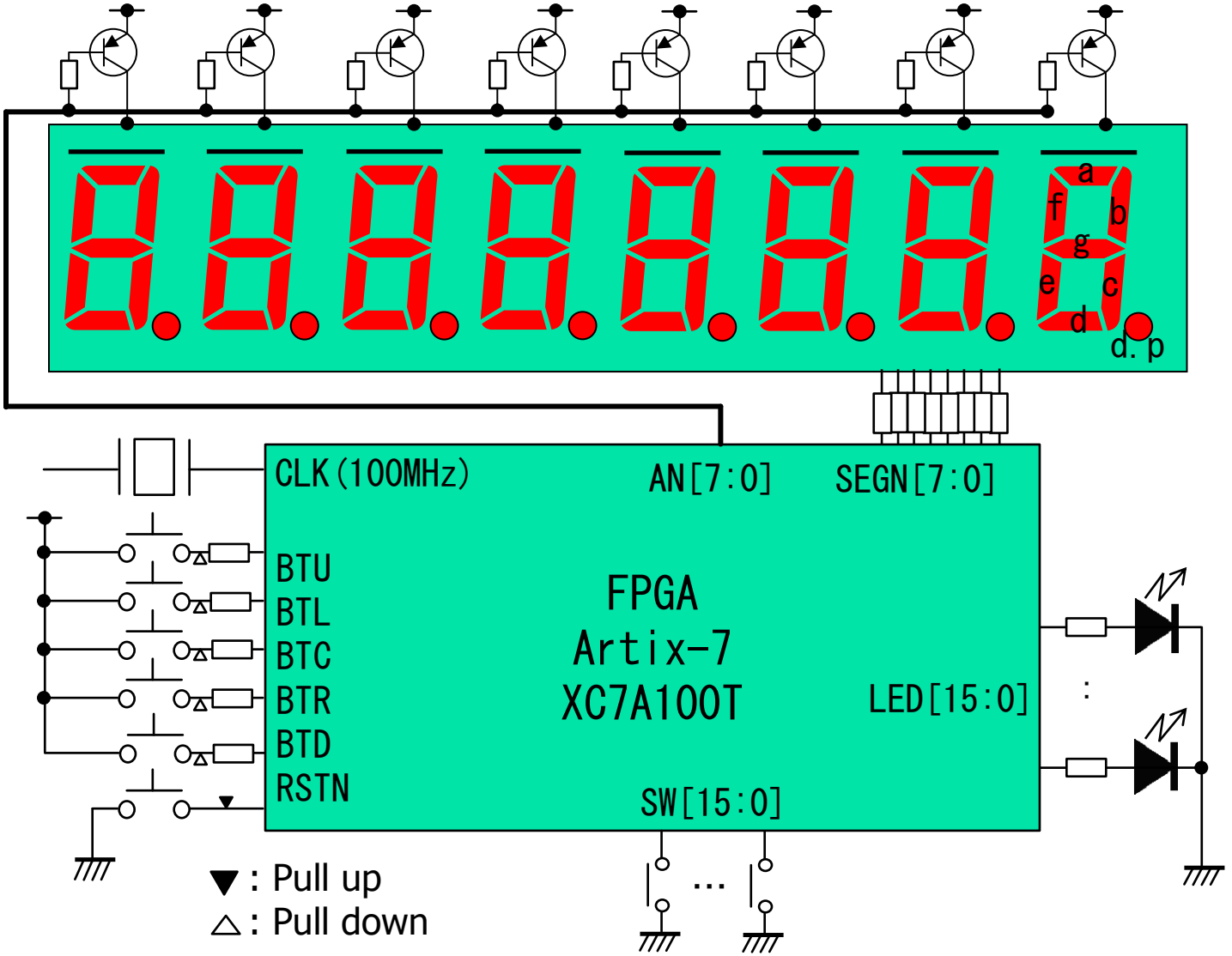
Arch研究室　スキルアップ講座

NEXYS4による24時間時計
仕様書および設計例

実験ボード(NEXYS4)外観



実験ボードブロック図


a
f
b
g
e
c
d
d.p
CLK(100MHz)
AN[7:0]
SEGN[7:0]
BTU
BTL
BTC
BTR
BTD
RSTN
FPGA
Artix-7
XC7A100T
LED[15:0]
SW[15:0]
▼: Pull up
△: Pull down

FPGA端子

・信号レベルは全て3.3V LVCMOS
・Nで終わる信号は負論理

信号名	方向	PIN	用途
CLK	in	E3	クロック（100MHz）
RSTN	in	C12	リセット（負論理）
BTU	in	F15	押しボタンスイッチ（正論理）
BTL	in	T16	押しボタンスイッチ（正論理）
BTC	in	E16	押しボタンスイッチ（正論理）
BTR	in	R10	押しボタンスイッチ（正論理）
BTD	in	V10	押しボタンスイッチ（正論理）

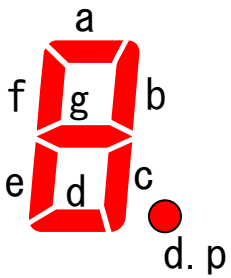
FPGA端子

7セグメントLED関連出力端子（全て負論理）

信号名	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

信号名	PIN	桁
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信号は7セグメントLEDの各セグメントを制御
負論理なので‘0’の時発光



AN信号は7セグメントLEDのアノードコモンを制御
負論理なので‘0’の時に発光する桁を選択

FPGA端子（スライドスイッチ，正論理）

信号名	方向	PIN	桁
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

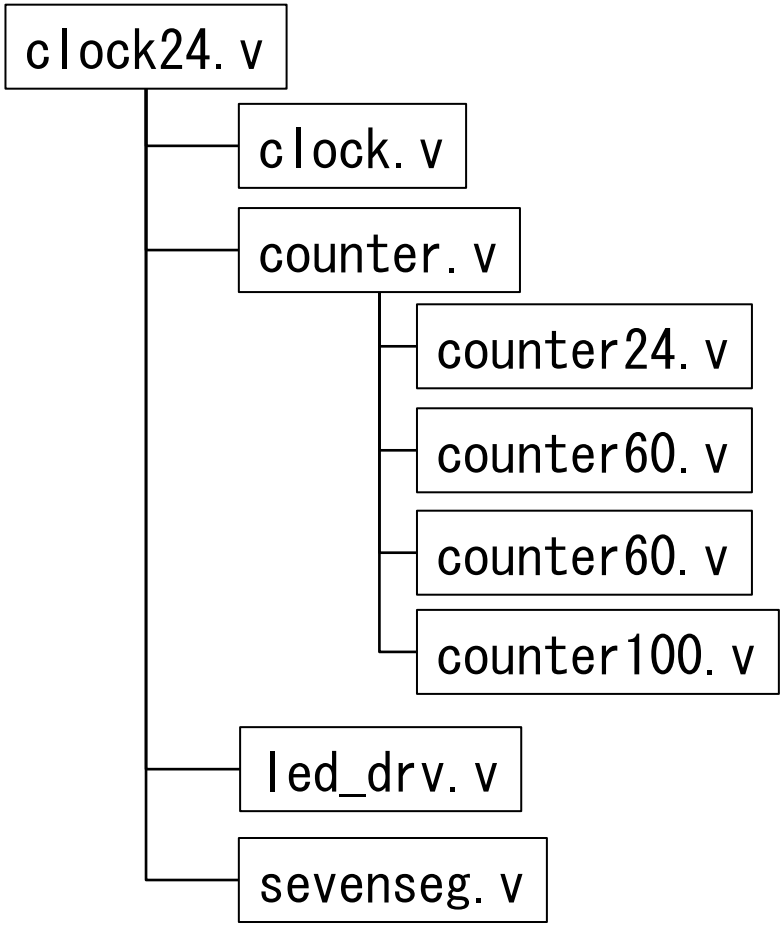
FPGA端子（LED，正論理）

信号名	方向	PIN	桁
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

24時間時計の仕様（設計例）

- 入力信号
  - クロック（CLK）として100MHzを入力
  - リセット（RSTN）は初期リセット（内部状態の全クリア）
  - SETHにより時カウンタの時刻合わせ（1秒毎に＋１）
  - SETMにより分カウンタの時刻合わせ（1秒毎に＋１）
  - SCLRにより秒，0.1秒，0.01秒のリセット
- カウンタの構成
  - hh：　時カウンタ・24進カウンタ
  - mm：分カウンタ・60進カウンタ
  - ss：　秒カウンタ・60進カウンタ
  - uu：1/100秒，1/10秒カウンタ・100進カウンタ
- カウンタのインクリメント（＋１）周波数は100Hz
- 7セグメントLEDはダイナミック点灯（1kHzを利用）
- 入出力値は論理レベルに注意

24時間時計のモジュール(設計例)



clock24.v:最上位階層

clock.v:1kHz, 100Hzのパルス生成

counter.v:時計用カウンタの上位階層

counter24.v:24進カウンタ

　時間をカウントするカウンタ

counter60.v:60進カウンタ

　分および秒をカウントするカウンタ

　ファイルは1個で実現可能

　(別ファイルでも構わない)

counter100.v:100進カウンタ

　100m秒，10m秒をカウントする

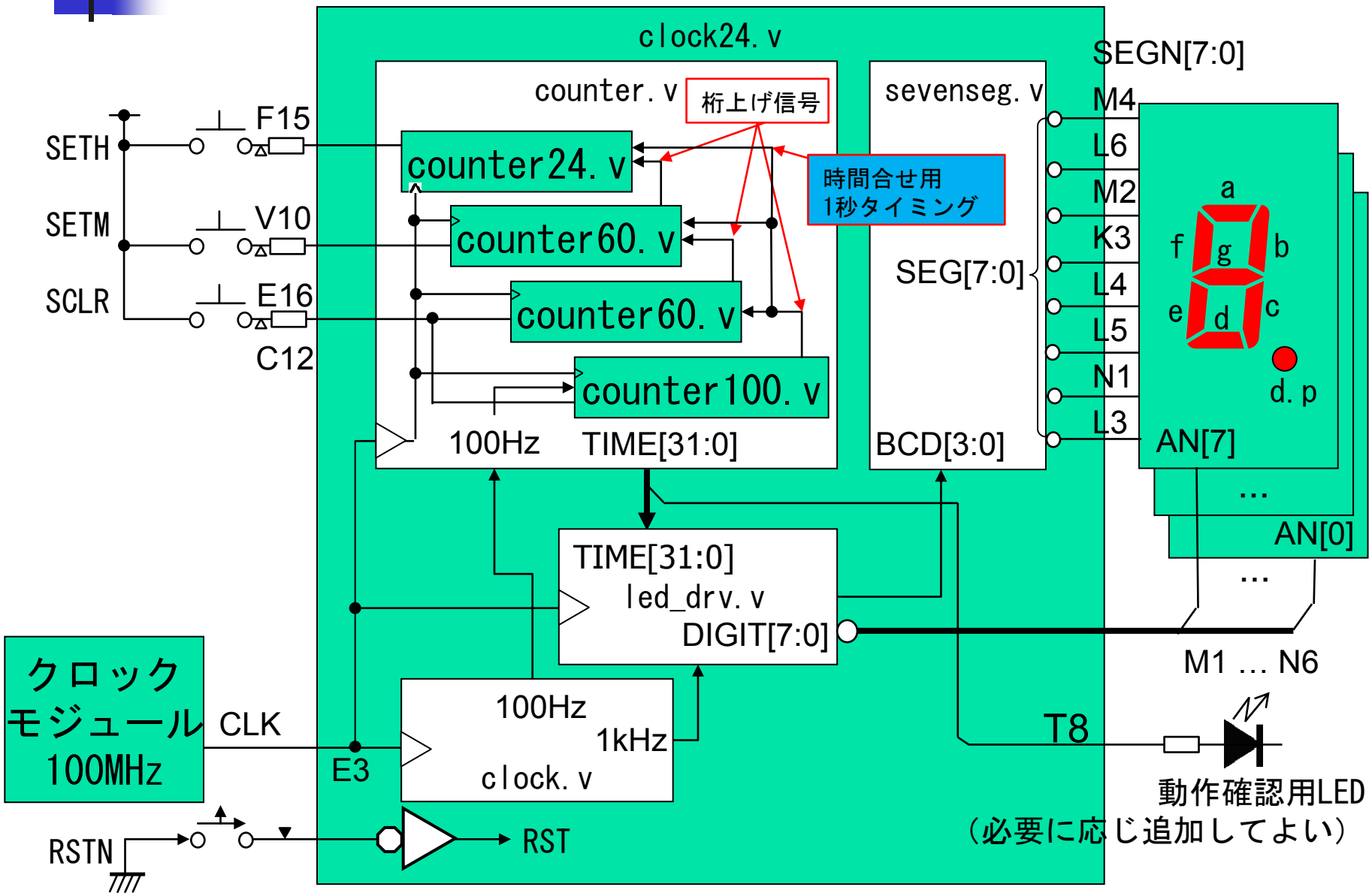
led_dev.v:7セグメントLEDのダイナミック

　点灯用制御信号生成

sevenseg.v:BCDコードから7セグメント

　LEDを表示するために必要なデコーダ

24時間時計の仕様（サンプルブロック図）



設計入力(最上位階層: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 ____ = _______;    // 負論理信号は内部では正論理で統一

  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 = ___________; // 負論理で出力

  assign AN  = ___________;  // 負論理で出力
  assign LED = ___________;  // 動作確認用に1秒の信号などを出力しておくとよい

endmodule
```

外部信号が負論理の場合でも，内部信号は正論理で統一した方が分かりやすい

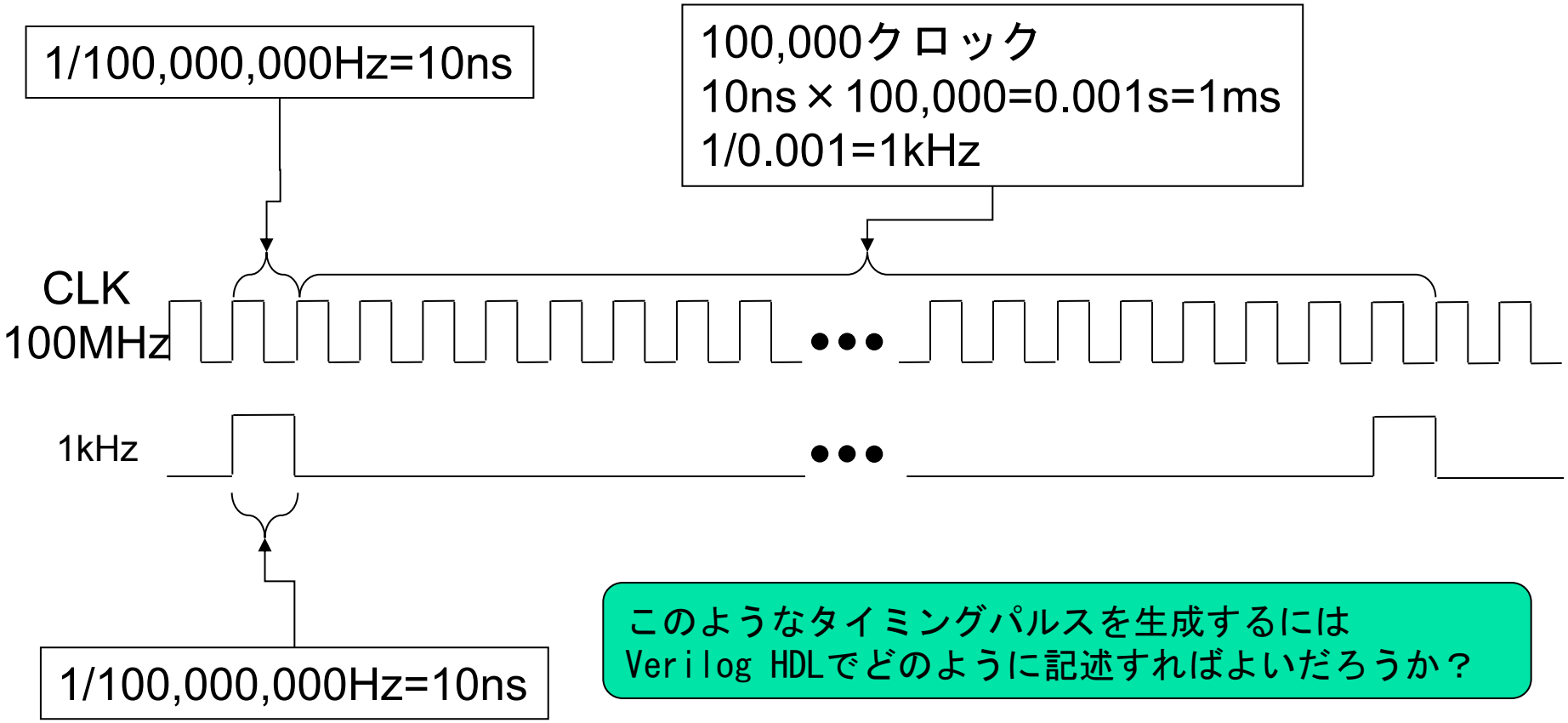
コメントに漢字は使えません

制約条件ファイル: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
```

clock.vにおけるタイミング信号の考え方

- 1kHzのタイミングパルスは1msの周期で
  100MHzの1周期だけ1になるような信号



このようなタイミングパルスを生成するには
Verilog HDLでどのように記述すればよいだろうか？

基準タイミング生成：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
```

可能であれば，このような分周カウンタはインクリメント（+1）カウンタではなくデクリメント（−1）カウンタで実現した方が良い（なぜでしょう？）

100,000カウンタ

10カウンタ：100,000カウンタを数えきった際にカウントアップする

1ms周期で10ns幅のパルス

10ms周期で10ns幅のパルス

時間用カウンタの考え方

- 時間をカウントするために以下のカウンタを要する
  - 100ms, 10msカウンタ
    - 1/10秒，1/100秒をカウントする．100進カウンタが必要
    - RSTにより0に初期化できるとする
    - SCLRにより時間合わせのために0に初期化できるとする
  - 秒カウンタ
    - 60進カウンタが必要
    - RSTにより0に初期化できるとする
    - SCLRにより時間合わせのため，0に初期化できるとする
  - 分カウンタ
    - 60進カウンタが必要
    - RSTにより0に初期化できるとする
    - SETMにより時間合わせのため，1秒ごとにカウントアップできるとする
  - 時間カウンタ
    - 24進カウンタが必要
    - RSTにより0に初期化できるとする
    - SETHにより時間合わせのため，1秒ごとにカウントアップできるとする

カウンタは2進カウンタ，BCDカウンタのどちらでもよい．
但し，2進カウンタの場合は表示のために2進→BCD変換回路を要する．

時計用カウンタ上位階層: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 時間時刻合わせ
  input  SETM; // Set Minuite 分時刻合わせ
  input  SCLR; // Clear ss & mm, 秒, 1/10秒,1/00秒リセット
  output [31:0] TIME; // 時間出力

  wire       _____; // 1秒タイミング信号
  wire       _____; // 1分タイミング信号
  wire       _____; // 1時間タイミング信号
  wire [7:0] _____, _____, _____,  ___; // 時刻用変数

  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
```

100進BCDカウンタ:counter100.v

```
module counter100 ( CLK, RST, CE, CNT, UP );
  input        CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0] CNT;          // 時間出力
  output       UP;           // 桁上げ
  reg    [3:0] d1, d0;       // カウンタ変数

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

 assign CNT = { ___, ___ }; // 時間出力
 assign UP  = ( ___________ && ___________ && ___ ) ? 1'd1 : 1'd0;

endmodule
```

リセット時

考え方：
1の位が9ならば0にして10の位を桁上げ，そうでなければ1の位を+1．但し，10の位も9であるときは0にする．

カウンタが99で，桁上げが必要なタイミングに桁上げ信号を出力

60進BCDカウンタ:counter60.v

```
module counter60 ( CLK, RST, CE, CNT, UP );
  input        CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0] CNT;          // 時間出力
  output       UP;           // 桁上げ
  reg    [3:0] d1, d0;       // カウンタ変数

  always @( ___________ or ___________ )
    begin
      if( ________ )
        begin
          ________;
          ________;
        end
      else if( ____ )
        begin
          if( ________ )
            begin
              _________;
               if( ________ ) ________;
               else           ________________;
            end
          else
            ________________;
        end
    end

 assign CNT = { ___, ___ }; // 時間出力
 assign UP  = ( ___________ && ___________ && ___ ) ? 1'd1 : 1'd0;

endmodule
```

リセット時

考え方：
1の位が9ならば0にして10の位を桁上げ，そうでなければ1の位を+1．但し，10の位が5であるときは0にする．

カウンタが59で，桁上げが必要なタイミングに桁上げ信号を出力

24進BCDカウンタ:counter24.v

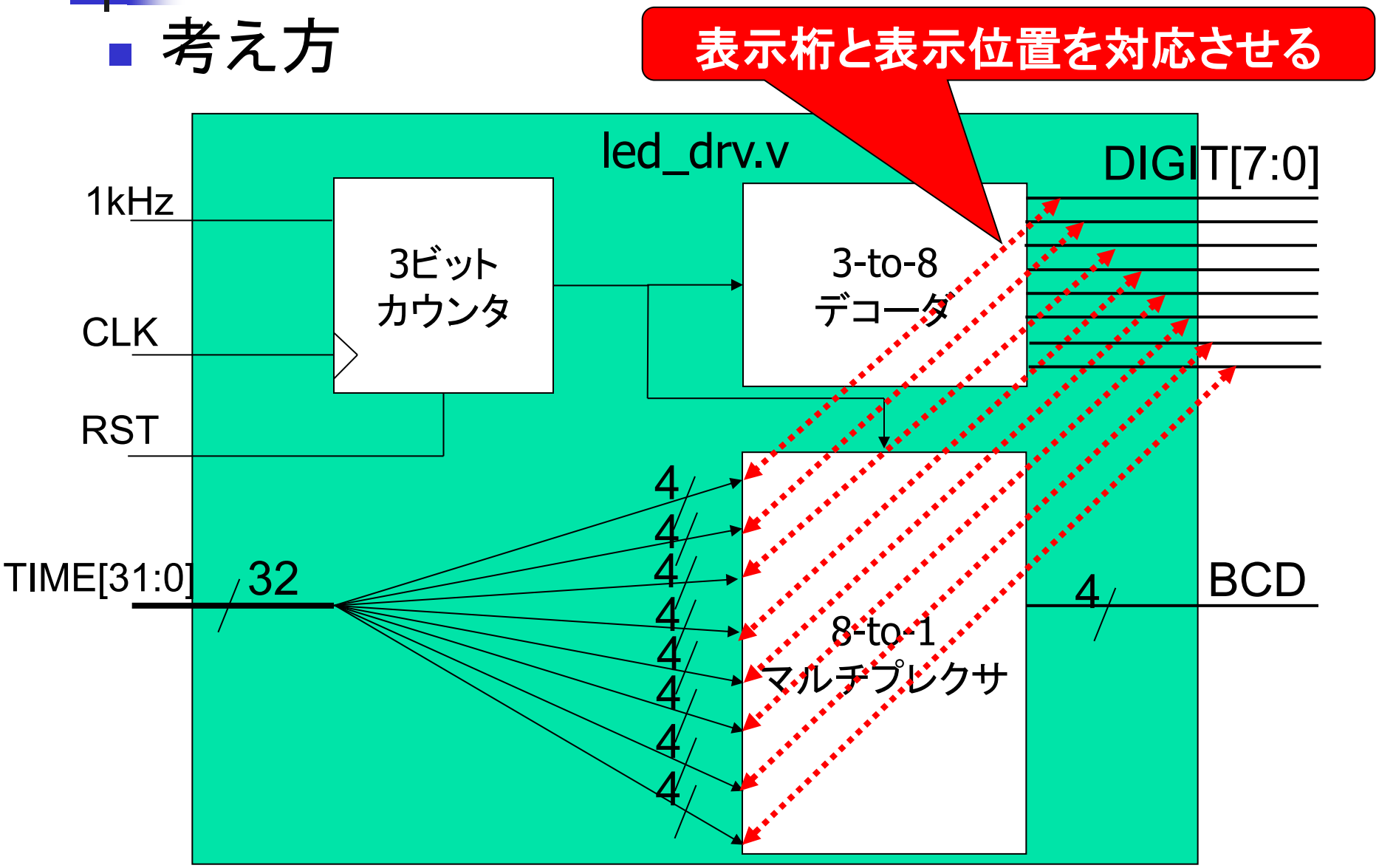
```
module counter24 ( CLK, RST, CE, CNT );
  input         CLK, RST, CE; // Clock, Reset, Clock Enable
  output [7:0] CNT;           // 時間出力
  reg    [3:0] d1, d0;        // カウンタ変数

  always @( ____________ or ____________ )
    begin
      if( ________ )
        begin
          ________;
          ________;
        end
      else if( ___ )
        begin
          if( ________ )
            begin
              ________;
              ____________________;
            end
          else if( ____________________ )
            begin
              ________;
              ________;
            end
          else
              ____________________;
        end
    end

  assign CNT = { ___, ___ }; // 時間出力
endmodule
```

リセット時

考え方：
1の位が9のときは0にして10の位を桁上げ.
カウンタ値がBCDで23であるときは0.
そうでなければ1の位を+1にする.

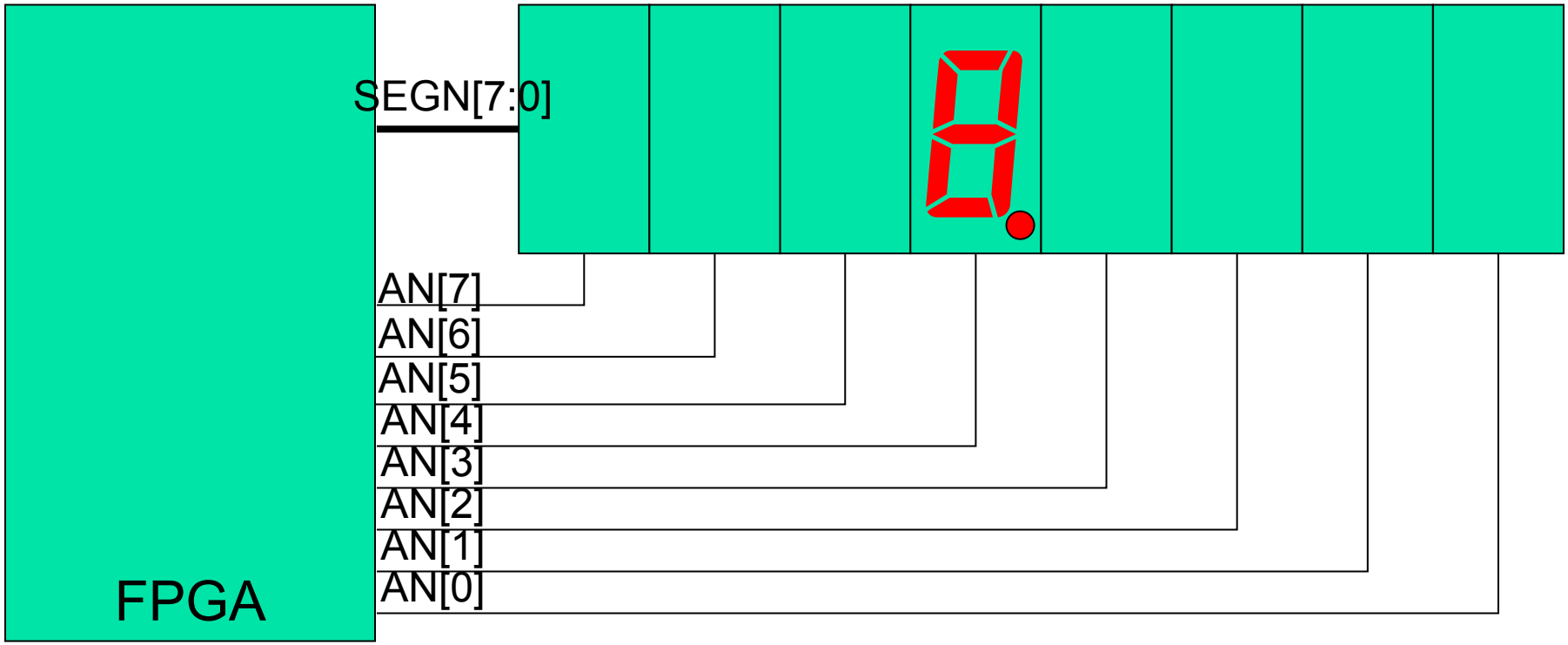
設計入力(LEDドライバ:led_drv.v)

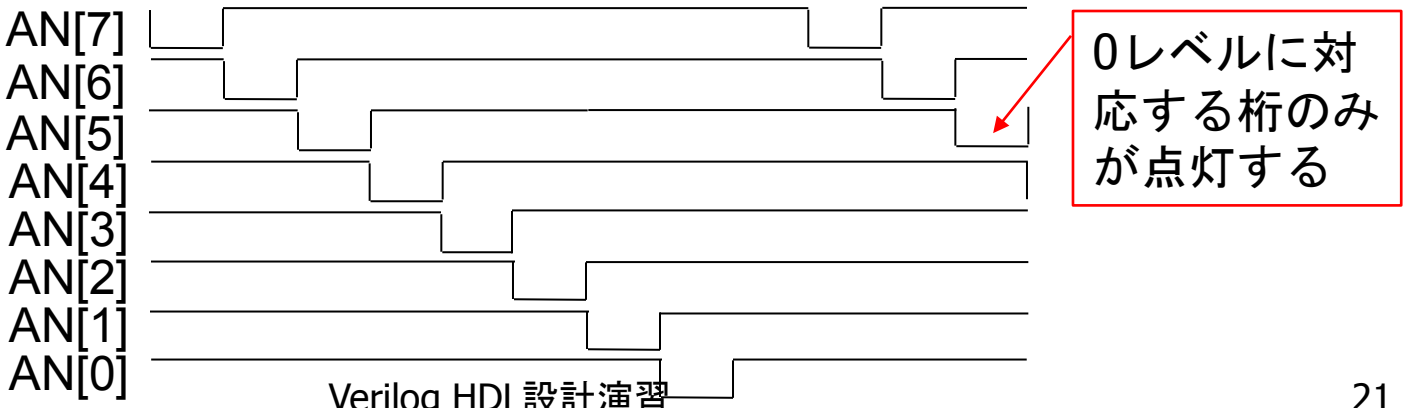
- 考え方



設計入力(LEDドライバ:led_drv.v)

- AN[7:0]信号の変化の考え方





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
```

3bitのfree run counterを生成する

```
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
```

3-to-8 デコーダ

```
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
```

8-to-1 マルチプレクサ

設計入力(7セグメントデコーダ: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<=_____________;
    4'h8:   SEG<=_____________;
    4'h9:   SEG<=_____________;
    default:SEG<=_____________;
    endcase
endmodule
```

SEG[7:0]は7セグメントLEDの各セグメントa, b, c, d, e, f, g, d.pの順に対応している

テストベンチ:clock24_test.v

```
`timescale 1ns/1ns        // シミュレーションの時間単位を1ns，精度を1nsにする
module clock24_text ;
  reg         CLK;        // テスト回路への入力変数はregを使用
  reg         RSTN;
  reg         SETH;
  reg         SETM;
  reg         SCLR;
  wire [7:0] SEGN;        // テスト回路からの出力変数はwireを使用
  wire [7:0] AN;
  wire        LED;

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

  `include "clock24_test.vct"

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

endmodule
```

Verilog-XLシミュレータにおいて
simvision波形ビューアを使用する際の
波形情報保存指定

テストベクタの読込み

テストベクタ:clock24_test.txt

テストベクタのサンプルを示す.

このテストベクタは，クロックとして10ns（100MHz）を使用し，リセットをかけた後に1千万ns時間進めている． 1千万ns時間は

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

であり，0.01秒の時間しかシミュレーションしないことになる．なお，これ以上時間をかけてもシミュレーションに多大な時間をかけることになるため，clockモジュールの100,000カウンタを一時的に変更するなどしてシミュレーション時間を加速した方が良い．例えばclockモジュールの100,000カウンタを10カウンタとすれば，10,000倍の加速シミュレーションを行うことになる．

なお，シミュレーション後は変更箇所を元に戻すことを忘れないこと．また，右記テストベクタでは時間合わせ機能のテストを省略している．もちろん，時間わせ機能のテストも必要である．

```
# 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
```

Verilogシミュレータに入力する実際のテストベクタ（clock24_test.vct）は make_vector.plコマンドを使用してclock24_test.txtから変換する.

シミュレーション結果（例）

シミュレーション結果の例を示す．

clockモジュールの100,000カウンタを10カウンタとしているため，10,000倍の加速シミュレーションを行った結果である．カーソル付近において，LED表示の1セグメント分の表示期間が1,720[ns]-1,620[ns]=100[ns]となっているが，10,000倍なので実際の時間は1[ms](1kHz)に相当する．

TIME[31:0]も約1,000[ns]の所で+1されており，10[ms]でカウントアップされていることが分かる．BCD[3:0]が1,620[ns]で"1"となっていることから，10[ms]の桁の"1"が7セグメントLEDに表示されようとしていることが分かる．