# Capitolo 2: reti sequenziali elementari

## Esercizio 5: Cronometro

#### Progetto e architettura

Per la realizzazione del cronometro, si è adottato un approccio strutturale nel quale tre contatori distinti, uno dedicato a monitorare i secondi, uno i minuti e un altro le ore, sono opportunamente connessi in uno schema seriale. Ogni contatore è realizzato come una specifica istanza di un’entità contatore il cui modulo è parametrizzato, ovvero si definisce il generic **N**. Per i contatori **sec\_counter** e **min\_counter**, N è stato impostato a 59 dato che esso rappresenta il massimo valore consentito per i secondi e i minuti, mentre per **ore\_counter**, N è 23 poichè questa è l’ora massima consentita nel sistema orario a 24 ore. Nel contatore (**contatore**) all’impulso di conteggio successivo al raggiungimento del picco il valore (**count**) si azzera.

Per poter precaricare un valore di conteggio nei contatori si sfrutta un segnale di abilitazione **set**. Quando **set** è attivo, ogni contatore imposta il valore intero di conteggio a uno fornito dall’esterno (**init\_sec** per sec\_counter, **init\_min** per min\_counter**,** e **init\_ore** per **ore\_counter**).

Solo il primo contatore, ovvero quello dei secondi, prende in ingresso il segnale di sincronismo **clock**, tale che ad ogni colpo di clock **sec\_count** incrementi il suo valore.

min\_counter, invece,prende in ingresso come segnale di tempificazione il segnale **clock\_tick\_min** il quale, in modo sincrono a clock,si alza sono nel momento in cui il valore in uscita a sec\_counter è il massimo, ovvero 59, altrimenti è basso. In seguito a transizioni di clock\_tick\_min del tipo 0🡪1🡪0, **min\_count** si incrementa di 1.

Analogamenteore\_counter ha come clock il segnale **clock\_tick\_ore**, che viene abilitato sono quando sec\_count e min\_count raggiunto entrambi il massimo valore di 59, altrimenti è ‘0’. Un impulso di questo tipo genera un incremento di **ore\_count**.

Di seguito è possibile visualizzare l’architettura del cronometro.

Immagine che contiene testo, diagramma, schizzo, Disegno tecnico

Descrizione generata automaticamente

#### Implementazione

**cronometro.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity cronometro is

    port (

        clk : in std\_logic;

        reset : in std\_logic;

        set : in std\_logic;

        init\_sec : in integer range 0 to 59;

        init\_min : in integer range 0 to 59;

        init\_ore : in integer range 0 to 23;

        sec : out integer range 0 to 59;

        min : out integer range 0 to 59;

        ore : out integer range 0 to 23

    );

end entity;

architecture Structural of cronometro is

    component contatore

        generic (

            MAX\_VALUE : integer := 59

        );

        port (

            clk : in std\_logic;

            reset : in std\_logic;

            set : in std\_logic;

            init\_count : in integer range 0 to MAX\_VALUE;

            count : out integer range 0 to MAX\_VALUE

        );

    end component;

    signal sec\_count : integer range 0 to 59;

    signal min\_count : integer range 0 to 59;

    signal ore\_count : integer range 0 to 23;

    signal clock\_tick\_min : std\_logic := '0';

    signal clock\_tick\_ore : std\_logic := '0';

begin

    sec\_counter : contatore

        generic map (MAX\_VALUE => 59)

        port map (

            clk => clk,

            reset => reset,

            set => set,

            init\_count => init\_sec,

            count => sec\_count

        );

    min\_counter : contatore

        generic map (MAX\_VALUE => 59)

        port map (

            clk => clock\_tick\_min,

            reset => reset,

            set => set,

            init\_count => init\_min,

            count => min\_count

        );

    ore\_counter : contatore

        generic map (MAX\_VALUE => 23)

        port map (

            clk => clock\_tick\_ore,

            reset => reset,

            set => set,

            init\_count => init\_ore,

            count => ore\_count

        );

    process(clk, reset)

    begin

        if rising\_edge(clk) and reset='0' then

            if sec\_count = 59 then

                clock\_tick\_min <= '1';

            else

                clock\_tick\_min <= '0';

                clock\_tick\_ore <= '0';

            end if;

            if min\_count = 59 and sec\_count = 59 then

                clock\_tick\_ore <= '1';

            end if;

        end if;

    end process;

    sec <= sec\_count;

    min <= min\_count;

    ore <= ore\_count;

end architecture;

**contatore.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity contatore is

    generic (

        MAX\_VALUE : integer := 59

    );

    port (

        clk : in std\_logic;

        reset : in std\_logic;

        set : in std\_logic;

        init\_count : in integer range 0 to MAX\_VALUE;

        count : out integer range 0 to MAX\_VALUE

    );

end entity;

architecture Behavioral of contatore is

    signal counter : integer range 0 to MAX\_VALUE :=0;

begin

    process(clk, reset, set)

    begin

        if reset = '1' then

                counter <= 0;

        elsif set='1' then

                counter <= init\_count;

        elsif rising\_edge(clk) then

            if counter < MAX\_VALUE then

                counter <= counter + 1;

            else

                counter <= 0;

            end if;

        end if;

    end process;

    count <= counter;

end architecture;

Sintesi su board di sviluppo

#### Simulazione

**cronometro\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity cronometro\_tb is

end entity;

architecture tb of cronometro\_tb is

    signal clk : std\_logic := '0';

    signal reset : std\_logic := '0';

    signal set : std\_logic := '0';

    signal init\_sec : integer := 0;

    signal init\_min : integer := 0;

    signal init\_ore : integer := 0;

    signal sec : integer;

    signal min : integer;

    signal ore : integer;

begin

    crono\_metro : entity work.cronometro

        port map (

            clk => clk,

            reset => reset,

            set => set,

            init\_sec => init\_sec,

            init\_min => init\_min,

            init\_ore => init\_ore,

            sec => sec,

            min => min,

            ore => ore

        );

    process

    begin

        wait for 5 ns;

        clk <= not clk;

    end process;

    process

    begin

        init\_sec <= 50;

        init\_min <= 59;

        init\_ore <= 4;

        set <= '1';

        wait for 10 ns;

        set <= '0';

        wait for 200 ns;

        reset <= '1';

        wait for 10 ns;

        reset <= '0';

        wait for 650 ns;

        init\_sec <= 58;

        init\_min <= 59;

        init\_ore <= 23;

        set <= '1';

        wait for 10 ns;

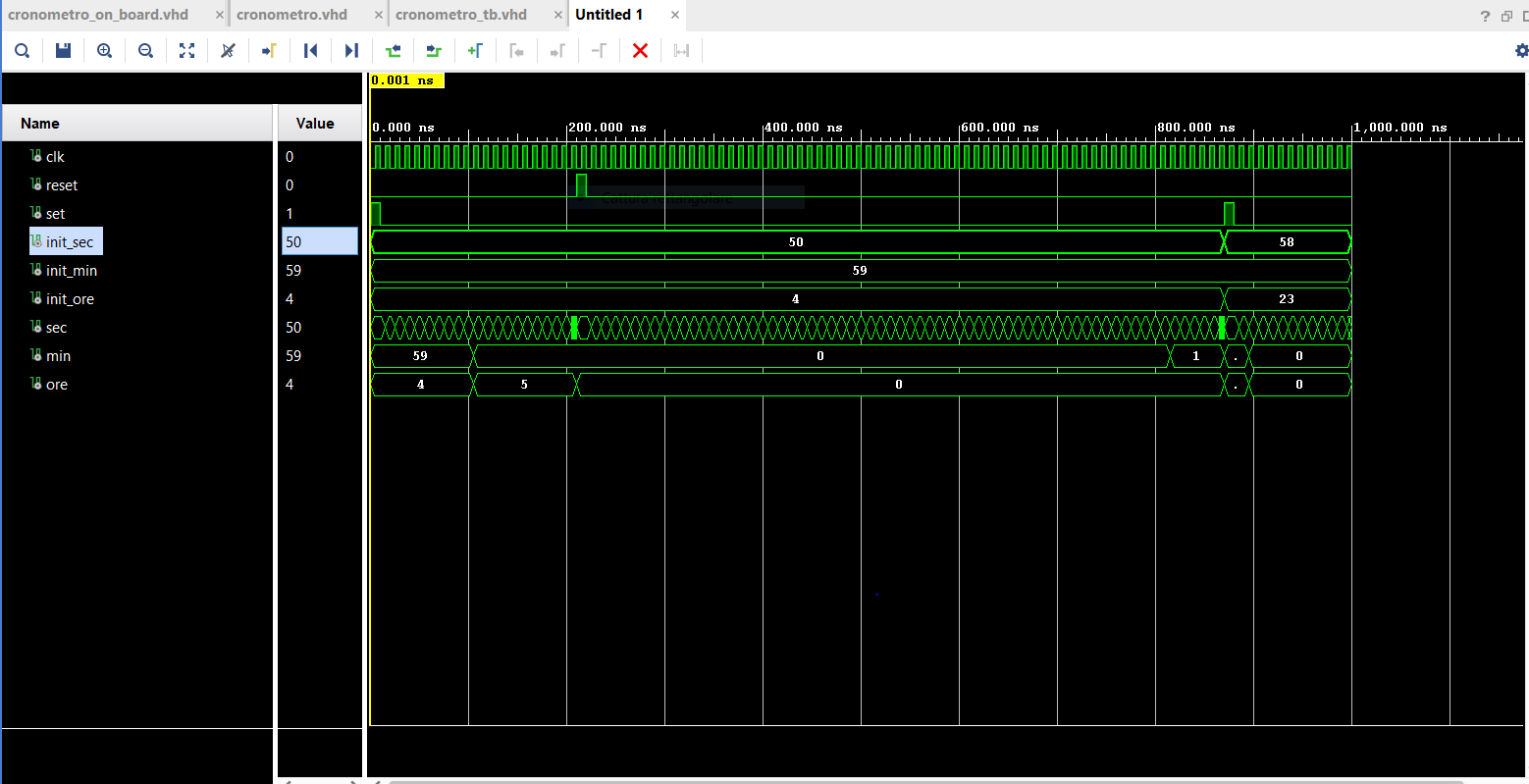
        set <= '0';

        wait for 100 ns;

        wait;

    end process;

end tb;



#### Sintesi su board di sviluppo

La sintesi sul board Nexys A7-100T richiede la scrittura di un modulo dedicato. L'entity **cronometro\_on\_board** presenta diverse porte di input e output necessarie per interfacciarsi con il board di sviluppo:

* **reset** è collegato al bottone di pin P17.
* **input** è un vettore di 6 bit collegato ai primi 6 switch da **J15** a **T18**, i quali vengono utilizzati per inserire i dati in ingresso sec\_init, min\_init e ore\_init. In realtà, per quanto riguarda le ore, sono sufficienti i primi 5 switch.
* **input\_ore** (**U12**), **input\_min** (**U11**), **input\_sec** (**V10**) sono associati a switch; quando uno solo di essi è alto, si fa l’acquisizione da “input” del valore iniziale associato.
* **input\_count** (bottone **M18**) è il segnale di set per il caricamento dei valori iniziali nel cronometro.
* **stop** è associato al bottone P18; alla sua pressione si esegue l’acquisizione di un intertempo.
* Le prime sei cifre del display a 7 segmenti sono utilizzate per visualizzare il tempo corrente nel formato del tipo “23.59.59”, compreso di dots.
* **view** è connesso al bottone N17 il quale, quando è premuto, causa la visualizzazione sul display di un intertempo precedentemente catturato invece che delle uscite del cronometro.

Per poter eseguire l’acquisizione e la visualizzazione degli intertempi è stato necessario progettare un nuovo modulo ossia una memoria MEM che potesse contenere 8 word da 17 bit (5 per l’ora, 6 per i minuti, 6 per i secondi).

MEM prende in ingresso come clock la **OR** tra il segnale stope view così che essa si attivi sia per le operazioni di lettura che scrittura. Per scandire gli indirizzi a cui accedere si sono istanziati due contatori di tipo “**counter\_mod8**.vhd”: **counter\_addr\_0** che fornire l’indirizzo in cui scrivere il nuovo intertempo catturato e **counter\_addr\_1** per fornisce l’indirizzo da cui leggere il prossimo intertempo.

Quindi, se si preme stopviene scritta nella cella di memoria all’indirizzo **address\_0** il valore corrente del cronometro dato dal segnale **tempo** da 17 bit(concatenazione vettori **ore, min** e **sec**); invece se viene pressato view si visualizza sul display il valore dell’intertempo all’indirizzo **address\_1**.

Il display a 7 segmenti è chiamato così perché ogni cifra del display è composta da 7 led che possono essere individualmente accesi per un totale di 128 possibili combinazioni. Gli anodi dei 7 segmenti sono tutti collegati tra loro mentre i catodi sono separati. I catodi di segmenti “simili” su tutte le cifre del display sono collegati a sette nodi di circuito denominati da CA a CG. Il circuito di controllo del display (**display\_seven\_segments**) può essere utilizzato per visualizzare un numero a 8 cifre su questo display. Questo circuito pilota i segnali di anodo e i modelli di catodo corrispondenti di ogni cifra in una successione continua e ripetitiva, a una frequenza di aggiornamento più rapida della risposta dell'occhio umano.

Immagine che contiene testo, diagramma, Piano, Carattere

Descrizione generata automaticamente

Il display\_seven\_segments è composto da un **counter\_mod8, cathodes\_manager, andoes\_manager** e un **clock\_filter**. L’uscita del **clock\_filter**, ovvero **clock\_filter\_out** va in ingresso come segnale di abilitazione al contatore\_mod8. Il valore in uscita dal contatore a sua volta va in ingresso al gestore dei catodi e anodi per selezionare il segmento da accendere. Sarà cathodes\_manager a prendere in ingresso il valore temporale(**value**)da mostrare. value può essere l’uscita del cronometro oppure un intertempo a seconda del caso.

Si sono introdotti degli alias per suddividere il valore in ingresso al display: **sec** rappresenta i primi 6 bit, **min** comprende i bit dal sesto all'undicesimo, e **ore** spazia dai bit 12 ai 16. Sono, inoltre, stati definiti 6 segnali che rappresentano le cifre da visualizzare: **cifra\_0** e **cifra\_1** sono quelle dei secondi, **cifra\_2** e **cifra\_3** sono quelle dei minuti, **cifra\_4** e **cifra\_5** sono quello delle ore.

Per ottenere le due cifre dei secondi, si converte il vettore di bit sec, di tipo std\_logic\_vector, in un numero intero senza segno. Successivamente, per ricavare la cifra meno significativa è sufficiente calcolare il resto della divisione di questo numero per 10; invece il risultato della divisione dell’intero sempre per 10 rappresenta la cifra più significativa. Il processo verrà replicato per le cifre dei minuti e delle ore.

**cronometro\_on\_board.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity cronometro\_on\_board is

port (

    CLK : in std\_logic;

    stop : in std\_logic; -- pulsante per catturare gli intertempi

    view : in std\_logic; -- pulsante per visualizzare gli intertempi

    input : in std\_logic\_vector(5 downto 0);

    input\_sec : in std\_logic;

    input\_min : in std\_logic;

    input\_ore : in std\_logic;

    anodes\_out : out  STD\_LOGIC\_VECTOR (7 downto 0);

    cathodes\_out : out  STD\_LOGIC\_VECTOR (7 downto 0);

    input\_count : in std\_logic;

    reset : in std\_logic

);

end cronometro\_on\_board;

architecture Behavioral of cronometro\_on\_board is

    component display\_seven\_segments

    generic(

        CLKIN\_freq : integer := 100000000;

        CLKOUT\_freq : integer := 500

                );

    port(

        CLK : IN std\_logic;

        RST : IN std\_logic;

        VALUE : IN std\_logic\_vector(16 downto 0);

        ENABLE : IN std\_logic\_vector(7 downto 0);

        DOTS : IN std\_logic\_vector(7 downto 0);

        ANODES : OUT std\_logic\_vector(7 downto 0);

        CATHODES : OUT std\_logic\_vector(7 downto 0)

        );

    end component;

    component cronometro is

    port (

        clk : in std\_logic;

        reset : in std\_logic;

        set : in std\_logic;

        init\_sec : in integer range 0 to 59;

        init\_min : in integer range 0 to 59;

        init\_ore : in integer range 0 to 23;

        sec : out integer range 0 to 59;

        min : out integer range 0 to 59;

        ore : out integer range 0 to 23

    );

    end component;

    component counter\_mod8 is

    port (

        clock : in  STD\_LOGIC;

           reset : in  STD\_LOGIC;

           enable : in STD\_LOGIC;

           counter : out  STD\_LOGIC\_VECTOR (2 downto 0)

    );

    end component;

    component MEM is

    port (

        clk     : in  std\_logic;

        address : in  std\_logic\_vector(2 downto 0);

        d\_in    : in  std\_logic\_vector(16 downto 0);

        read\_write   : in  std\_logic; -- se � basso di legge, se � alto si scrive

        d\_out   : out std\_logic\_vector(16 downto 0)

    );

    end component;

     signal value : std\_logic\_vector(16 downto 0); -- ingresso a display\_seven\_segments

     signal tempo : std\_logic\_vector(16 downto 0); -- uscita di cronometro

     signal sec, min, ore : integer := 0;

     signal sec\_init, min\_init, ore\_init: integer := 0;

     signal counter : integer := 0;

     signal clock : std\_logic := '0';

     signal address\_0, address\_1, address : std\_logic\_vector(2 downto 0);

     signal intertempo : std\_logic\_vector(16 downto 0);

     signal stop\_view : std\_logic;

     signal r\_w : std\_logic; -- read o write?

begin

    new\_clk : process(CLK) -- realizza un clock con periodo di 1 sec a partire da CLK

    begin

        if rising\_edge(CLK) then

            counter <= counter + 1;

            if counter = (100\_000\_000)/2 then

                counter <= 0;

                clock <= not clock;

            end if;

        end if;

    end process;

    tempo <= std\_logic\_vector(to\_unsigned(ore,5)) & std\_logic\_vector(to\_unsigned(min,6))  & std\_logic\_vector(to\_unsigned(sec,6));

    cronom : cronometro port map (

        clk => clock,

        reset => reset,

        set => input\_count,

        init\_sec => sec\_init,

        init\_min => min\_init,

        init\_ore => ore\_init,

        sec => sec,

        min => min,

        ore => ore );

     init\_counter : process(CLK) -- per settare il conteggio iniziale

     begin

        if(input\_sec='1' and input\_min='0' and input\_ore='0') then

            sec\_init <= to\_integer(unsigned(input));

        elsif(input\_sec='0' and input\_min='1' and input\_ore='0') then

            min\_init <= to\_integer(unsigned(input));

        elsif(input\_sec='0' and input\_min='0' and input\_ore='1') then

            ore\_init <= to\_integer(unsigned(input(4 downto 0)));

        end if;

     end process;

     counter\_addr\_0 : counter\_mod8 port map ( -- per ricavare l'indirizzo nel quale scrivere il prossimo intertempo nella memoria

        clock => stop,

        reset => reset,

        enable => '1',

        counter => address\_0

     );

     counter\_addr\_1 : counter\_mod8 port map ( -- per ricavare l'indirizzo da cui leggere un successivo intertempo, in precedenza scritto nella memoria

        clock => view,

        reset => reset,

        enable => '1',

        counter => address\_1

     );

     stop\_view <= stop or view;

     address <= address\_1 when view='1' else address\_0;

     gestione\_intertempi : process(address)

     begin

        if view='1' then

            r\_w <= '0';

            value <= intertempo;

        else

            r\_w <= '1';

            value <= tempo;

        end if;

     end process;

     intertempi\_mem : MEM port map ( -- memoria che contiene gli intertempi

        clk => stop\_view,

        address => address,

        read\_write => r\_w,

        d\_in => tempo,

        d\_out   => intertempo

     );

     seven\_segment\_array: display\_seven\_segments

     generic map(

        CLKIN\_freq => 100000000,

        CLKOUT\_freq => 500 )

     port map(

        CLK => CLK,

        RST => reset,

        value => value,

        enable => "00111111",

        dots => "00010100",

        anodes => anodes\_out,

        cathodes => cathodes\_out

     );

end Behavioral;

**cathodes\_manager.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity cathodes\_manager is

    Port ( counter : in  STD\_LOGIC\_VECTOR (2 downto 0);

           value : in   std\_logic\_vector(16 downto 0); --dato di mostrare sugli 8 display

           dots : in  STD\_LOGIC\_VECTOR (7 downto 0); --configurazione punti da accendere

           cathodes : out  STD\_LOGIC\_VECTOR (7 downto 0)); --sono i 7 catodi pi� il punto

end cathodes\_manager;

architecture Behavioral of cathodes\_manager is

constant zero   : std\_logic\_vector(6 downto 0) := "1000000";

constant one    : std\_logic\_vector(6 downto 0) := "1111001";

constant two    : std\_logic\_vector(6 downto 0) := "0100100";

constant three  : std\_logic\_vector(6 downto 0) := "0110000";

constant four   : std\_logic\_vector(6 downto 0) := "0011001";

constant five   : std\_logic\_vector(6 downto 0) := "0010010";

constant six    : std\_logic\_vector(6 downto 0) := "0000010";

constant seven  : std\_logic\_vector(6 downto 0) := "1111000";

constant eight  : std\_logic\_vector(6 downto 0) := "0000000";

constant nine   : std\_logic\_vector(6 downto 0) := "0010000";

alias sec is value (5 downto 0);

alias min is value (11 downto 6);

alias ore is value (16 downto 12);

signal cifra\_0 : std\_logic\_vector(3 downto 0);

signal cifra\_1 : std\_logic\_vector(3 downto 0);

signal cifra\_2 : std\_logic\_vector(3 downto 0);

signal cifra\_3 : std\_logic\_vector(3 downto 0);

signal cifra\_4 : std\_logic\_vector(3 downto 0);

signal cifra\_5 : std\_logic\_vector(3 downto 0);

signal cathodes\_for\_digit : std\_logic\_vector(6 downto 0) := (others => '0');

signal nibble :std\_logic\_vector(3 downto 0) := (others => '0');

signal dot :std\_logic := '0'; --stabilisce se il punto relativo alla cifra visualizzata deve essere acceso o spento

                              --nota: dot=1 significa che deve essere acceso, ma il segnale deve essere negato per andare sui catodi

begin

    cifra\_0 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(sec)) rem 10),4));

    cifra\_1 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(sec)) / 10),4));

    cifra\_2 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(min)) rem 10),4));

    cifra\_3 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(min)) / 10),4));

    cifra\_4 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(ore)) rem 10),4));

    cifra\_5 <= std\_logic\_vector(to\_unsigned((to\_integer(unsigned(ore)) / 10),4));

-- questo processo multiplexa le cifre da mostrare

digit\_switching: process(counter)

begin

    case counter is

        when "000" =>

            nibble <= cifra\_0;

            dot <= dots(0);

        when "001" =>

            nibble <= cifra\_1;

            dot <= dots(1);

        when "010" =>

            nibble <= cifra\_2;

            dot <= dots(2);

        when "011" =>

            nibble <= cifra\_3;

            dot <= dots(3);

        when "100" =>

            nibble <= cifra\_4;

            dot <= dots(4);

        when "101" =>

            nibble <= cifra\_5;

            dot <= dots(5);

        when "110" =>

            nibble <= "0000";

            dot <= dots(6);

        when "111" =>

            nibble <= "0000";

            dot <= dots(7);

        when others =>

            nibble <= (others => '0');

            dot <= '0';

    end case;

end process;

seven\_segment\_decoder\_process: process(nibble)

  begin

    case nibble is

      when "0000" => cathodes\_for\_digit <= zero;

      when "0001" => cathodes\_for\_digit <= one;

      when "0010" => cathodes\_for\_digit <= two;

      when "0011" => cathodes\_for\_digit <= three;

      when "0100" => cathodes\_for\_digit <= four;

      when "0101" => cathodes\_for\_digit <= five;

      when "0110" => cathodes\_for\_digit <= six;

      when "0111" => cathodes\_for\_digit <= seven;

      when "1000" => cathodes\_for\_digit <= eight;

      when "1001" => cathodes\_for\_digit <= nine;

        when others => cathodes\_for\_digit <= (others => '0');

    end case;

  end process seven\_segment\_decoder\_process;

cathodes <= (not dot)&cathodes\_for\_digit;

end Behavioral;

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { CLK }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {CLK}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { input[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { input[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { input[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { input[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { input[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { input[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

##7 segment display

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[0] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { cathodes\_out[7] }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { anodes\_out[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { view }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { input\_count }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { stop }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd