Capitolo 3: Macchine Aritmetiche

Esercizio 7: Moltiplicatore di Booth

Progetto e architettura

Il moltiplicatore di Booth (**molt\_booth**) realizza il prodotto tra due operandi con segno attraverso la composizione di due entità principali: **operational\_unit** e **control\_unit**. La macchina è modellata come un automa a stati finti: l'unità di controllo (control\_unit) realizza il diagramma a stati che, attraverso la definizione di segnali di controllo, guida il flusso di esecuzione della parte operativa (operational\_unit), la quale svolge le operazioni necessarie per ottenere il risultato della moltiplicazione.

Di seguito è presentata l'architettura complessiva del sistema molt\_booth.

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

L'entity **control\_unit** è caratterizzata da un'architettura comportamentale che implementa l'algoritmo di Booth per la moltiplicazione di due numeri interi con segno. L'algoritmo, basato sulla codifica di Booth, e riportato successivamente e consente l'implementazione di un moltiplicatore seriale piuttosto che parallelo.

BoothMultiplier: (in : INBUS; out : OUTBUS)

register A[7:0], M[7:0], Q[7:-1], COUNT[2:0];

bus INBUS[7:0], OUTBUS[7:0];

Begin: A := 0; COUNT := 0;

Input: M := INBUS; Q[7:0] := INBUS; Q[-1] :=0;

Scan: if Q[0]Q[-1] = 01

then A[7:0] := A[7:0] + M[7:0];

else if Q[0]Q[-1] = 10

then A[7:0] := A[7:0] - M[7:0];

RightShift: A[7] := A[7]; A[6:0].Q := A.Q[7:0];

Increment: COUNT := COUNT + 1; goto Scan;

Test: if COUNT<8 then go to Scan;

Output: OUTBUS := A;

OUTBUS := Q[7:0];

End BoothMultiplier;

La soluzione adottata prevede la definizione di due processi.

1. **res\_stato**: Questo processo si attiva solo in corrispondenza di eventi del clock e, se si è osservato proprio un fronte di salita, imposta lo stato corrente (**current\_state**) a quello di riposo (**idle**) se il segnale di **reset** è alto, altrimenti aggiorna current\_state al valore dello stato successivo (**next\_state**).
2. **Comb**: Questo process include nella sensitivity list lo stato corrente (current\_state), il segnale d’inizio (**start**) e il conteggio (**conteggio**) e, in risposta a variazioni dei valori di tali segnali, modifica opportunamente lo stato successivo (next\_state) e i segnali di controllo (**count\_en**, **subtract**, **sel\_SR**, **load\_SR**, **load\_M**, **stop\_cu** e **shift**). Si osservi che ogni volta che il processo in questione si “risveglia” vengono azzerati i segnali di count\_en, subtract, sel\_SR, load\_SR, load\_M, stop\_cu e shift per essere poi eventualmente alzati se lo stato lo prevede.

Il diagramma degli stati finiti qui di seguito mostra in dettaglio tutti i possibili stati in cui la macchina può trovarsi, le transizioni tra gli stati in risposta agli eventi, e le azioni svolte in ciascuno di essi.

Immagine che contiene testo, diagramma, disegno

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, disegno

Descrizione generata automaticamente

L'entity operational\_unit è invece realizzata attraverso un approccio strutturale. Le componenti istanziate sono uno shift register (**shift\_register**) a 17 bit, rappresentante la concatenazione delle parole A da 8 bit e Q da 9 bit, un registro (**M**) da 8 bit per contenere l'operando Y, un contatore modulo 9 (**counter**), un sommatore/sottrattore (**parallel\_adder**) di due interi a 8 bit basato sul ripple carry adder, e infine un multiplexer 4 a 1 (**mux\_41**) per selezionare la word da scrivere parallelamente nello shift register.

* L'entità shift\_register ha cinque porte: **parallel\_in** per il caricamento parallelo dei dati, **clock** per il segnale di clock, **reset** per il reset, **load** per il comando di inserimento dei dati, **shift** per eseguire uno shift verso destra di una posizione, e **parallel\_out** per la lettura parallela dei dati.

L’architettura "Behavioural" include un processo che gestisce il comportamento del registro a scorrimento. Quando il segnale di clock è in salita (**clock'event and clock='1'**), il processo esegue delle operazioni in base ai segnali di reset, load e shift: se il reset è attivo, il registro viene azzerato; se il segnale di load è attivo, il registro viene caricato con i dati presenti su parallel\_in; se, invece, il segnale di shift è alto, viene fatto uno shift a destra di un bit dei dati nel registro.

* L'entità M presenta i terminali **Y\_in** per l'input del moltiplicando, **clk** per il segnale di clock, **rst** per il segnale di reset, **load** per il caricamento dei dati, e **Y\_out** per l'output del moltiplicando.

L'architettura "Behavioural" implementa un processo nel quale, quando si ha un fronte di salita del segnale di clock, se il reset è attivo, il registro viene azzerato; invece, se è il segnale di load ad essere uno, sull’uscita Y\_out vengono caricati i dati presenti sull’ingresso Y\_in.

* L'entità counter ha i ports: **clock** per il segnale di clock, **reset** per il segnale di reset, **enable** per abilitare il contatore, e **count** per l'uscita a 4 bit, ossia il conteggio.

Il contatore comportamentale è sincronizzato sulla transizione da 0 a 1 del segnale di clock: se il segnale di reset è alto, il contatore viene azzerato; in caso contrario, se il segnale di abilitazione è attivo, il contatore viene incrementato di uno. Quando il contatore raggiunge il valore massimo 9 viene riportato a 0.

* L'entità parallel\_adder ha i terminati **X** e **Y** come data input, **cin** come input per il bit di carry in (riporto in ingresso), **Z** come output per il risultato della somma/sottrazione, e **cout** come output per il bit di carry out (riporto in uscita).

L'architettura "Structural" implementa il sommatore/sottrattore utilizzando un componente interno di tipo **ripple\_carry**. Questo componente è un sommatore a 8 bit che riceve come operandi **X** e (**not(Y)) xor (cin)** (ovvero Y se cin=0, altrimenti il complemento da 2 di Y, ovvero -Y, se cin=1) e come carry in proprio **cin**. Se **cin** è alto il ripple carry adder esegue un’operazione di somma, mentre se **cin** è basso una sottrazione.

Si noti che nel sommatore di tipo ripple carry il secondo operando viene calcolato mediante il costrutto **for…generate** e che il modulo è a sua volta realizzato per composizione di 8 **full\_adder**.

* L'entity mux\_41 ha un parametro generico **width** che specifica la larghezza dei vettori di ingresso e di uscita. Le porte del modulo includono quattro ingressi (**x0**, **x1**, **x2**, **x3**) di lunghezza width, un selettore (**s)** a due bit e un'uscita (**y**).

L'architettura comportamentale contiene un processo sensibile ai segnali: s, x0, x1, x2 e x3. La struttura del multiplexer è realizzata tramite l’uso della clausola **case** la quale assegna all'uscita y uno degli ingressi in base al valore di s. I casi sono esaustivi ovvero coprono tutte le possibili combinazioni del segnale selettore.

Tutte le componenti dell'unità operativa sono interconnesse, come mostrato nel disegno sottostante.

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

Implementazione

Qui sono riportati i file in formato VHDL delle principali componenti del sistema.

**molt\_booth.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity molt\_booth is

     port( clock, reset, start: in std\_logic;

           X, Y: in std\_logic\_vector(7 downto 0);

           P: out std\_logic\_vector(15 downto 0);

           stop\_cu: out std\_logic);

end molt\_booth;

architecture Structural of molt\_booth is

    component control\_unit is

        port( q0, q\_1, clock, reset, start: in std\_logic;

          count: in std\_logic\_vector(3 downto 0);

          load\_M, count\_en, load\_SR, shift: out std\_logic;

          subtract, stop\_cu: out std\_logic;

          sel\_SR : out std\_logic\_vector(1 downto 0));

    end component;

    component operational\_unit is

       port( X, Y: in std\_logic\_vector(7 downto 0);

          clock, reset: in std\_logic;

          load\_SR, shift, load\_M, subtract, count\_en: in std\_logic;

          sel\_SR : in std\_logic\_vector(1 downto 0);

          count: out std\_logic\_vector(3 downto 0);

          P: out std\_logic\_vector(16 downto 0));

    end component;

    signal temp\_q0,temp\_q\_1  : std\_logic :='0';

    signal temp\_sel\_M, temp\_subtract, temp\_load\_SR: std\_logic;

    signal temp\_sel\_SR: std\_logic\_vector(1 downto 0);

    signal temp\_count: std\_logic\_vector(3 downto 0);

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

    signal temp\_count\_en: std\_logic;

    signal temp\_shift: std\_logic;

    signal temp\_load\_M: std\_logic;

begin

    unita\_di\_controllo: control\_unit port map

       (q0 => temp\_q0,

       q\_1 => temp\_q\_1,

       clock => clock,

       reset => reset,

       start => start,

       count => temp\_count,

       load\_M => temp\_load\_M,

       count\_en => temp\_count\_en,

       load\_SR => temp\_load\_SR,

       shift => temp\_shift,

       subtract => temp\_subtract,

       stop\_cu => stop\_cu,

       sel\_SR => temp\_sel\_SR);

    unita\_operativa: operational\_unit port map

       (X => X,

       Y => Y,

       clock => clock,

       reset => reset,

       load\_SR => temp\_load\_SR,

       shift => temp\_shift,

       load\_M => temp\_load\_M,

       subtract => temp\_subtract,

       count\_en => temp\_count\_en,

       sel\_SR => temp\_sel\_SR,

       count => temp\_count,

       P => temp\_p);

    temp\_q0 <= temp\_p(1);

    temp\_q\_1 <= temp\_p(0);

    P <= temp\_p(16 downto 1);

end Structural;

**control\_unit.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity control\_unit is

    port( q0, q\_1, clock, reset, start: in std\_logic;

          count: in std\_logic\_vector(3 downto 0);

          load\_M, count\_en, load\_SR, shift: out std\_logic;

          subtract, stop\_cu: out std\_logic;

          sel\_SR : out std\_logic\_vector(1 downto 0));

end control\_unit;

architecture structural of control\_unit is

    type state is (idle, acquisisci\_op, fine\_acquisisci, avvia\_somma, avvia\_shift, incr\_count, fine);

    signal current\_state,next\_state: state;

begin

    reg\_stato: process(clock)

    begin

       if(clock'event and clock='1') then

           if(reset='1') then

              current\_state <=idle;

           else

              current\_state <=next\_state;

           end if;

       end if;

    end process;

    comb: process(current\_state, start, count)

    begin

        count\_en <='0';

        subtract <='0';

        sel\_SR <= "00";

        load\_SR <='0';

        load\_M <='0';

        stop\_cu <='0';

        shift <='0';

        case current\_state is

          when idle =>

            if(start='1') then

              next\_state <= acquisisci\_op;

            else

              next\_state <= idle;

            end if;

          when acquisisci\_op =>

            load\_M <='1';

            load\_SR <='1';

            next\_state <= fine\_acquisisci;

          when fine\_acquisisci =>

            next\_state <= avvia\_somma;

          when avvia\_somma =>

            if(count="1000") then -- 9

              stop\_cu <= '1';

              sel\_SR <= "10";

              next\_state <= fine;

            elsif(q0='0' and q\_1='1') then

              load\_SR <='1';

              sel\_SR <= "01";

              subtract <= '0';

              next\_state <= avvia\_shift;

            elsif(q0='1' and q\_1='0') then

              load\_SR <='1';

              sel\_SR <= "01";

              subtract <= '1';

              next\_state <= avvia\_shift;

            else -- q0=0=q\_1 oppure q0=1=q\_1

              load\_SR <='1';

              sel\_SR <= "10";

              next\_state <= avvia\_shift;

            end if;

          when avvia\_shift =>

             shift <= '1';

             next\_state <= incr\_count;

          when incr\_count =>

             count\_en <= '1';

             next\_state <= avvia\_somma;

          when fine =>

             if(start/='1') then

                next\_state <= fine;

             else next\_state <= idle;

             end if;

        end case;

    end process;

 end structural;

**operational\_unit.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity operational\_unit is

    port( X, Y: in std\_logic\_vector(7 downto 0); --moltiplicatore e moltiplicando

          clock, reset: in std\_logic;

          load\_SR, shift, load\_M, subtract, count\_en: in std\_logic;

          sel\_SR : in std\_logic\_vector(1 downto 0);

          count: out std\_logic\_vector(3 downto 0);

          P: out std\_logic\_vector(16 downto 0));

end operational\_unit;

architecture Structural of operational\_unit is

    component parallel\_adder is

    port( X, Y: in std\_logic\_vector(7 downto 0);

          cin: in std\_logic;

          Z: out std\_logic\_vector(7 downto 0);

          cout: out std\_logic);

    end component;

    component M is

        port( Y\_in: in std\_logic\_vector(7 downto 0);

          clk, rst, load: in std\_logic;

          Y\_out: out std\_logic\_vector(7 downto 0));

    end component;

    component shift\_register is

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

          clock, reset, load, shift: in std\_logic;

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

    end component;

    component counter is

    port( clock,  reset: in std\_logic;

          enable: in std\_logic;

          count: out std\_logic\_vector(3 downto 0));

    end component;

    component mux\_41 is

    generic (width : integer range 0 to 32 := 17);

    port( x0, x1,x2,x3: in std\_logic\_vector(width-1 downto 0);

          s: in std\_logic\_vector(1 downto 0);

          y: out std\_logic\_vector(width-1 downto 0));

    end component;

    signal op\_2: std\_logic\_vector(7 downto 0);

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

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

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

    signal sum: std\_logic\_vector(7 downto 0);

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

    signal carry\_out: std\_logic; -- non utilizzato

begin

    moltiplicando: M port map(

       Y\_in => Y,

       clk => clock,

       rst => reset,

       load => load\_M,

       Y\_out => op\_2);

    SR\_init <= "00000000" & X & "0";

    SR\_sum <= sum & SR\_out(8 downto 0);

    MUX\_SR\_parallel\_in : mux\_41 generic map (width => 17) port map(

        x0 => SR\_init,

        x1 => SR\_sum,

        x2 => SR\_out,

        x3 => "00000000000000000", -- mai selezionato

        s => sel\_SR,

        y => SR\_in);

    registro\_shift: shift\_register port map(

       parallel\_in => SR\_in,

       clock => clock,

       reset => reset,

       load => load\_SR,

       shift => shift,

       parallel\_out => SR\_out);

    sommatore\_sottrattore: parallel\_adder port map(

       X => SR\_out(16 downto 9),

       Y => op\_2,

       cin => subtract,

       Z => sum,

       cout => carry\_out);

    contatore\_mod\_9: counter port map(

       clock => clock,

       reset => reset,

       enable => count\_en,

       count => count);

    P <= SR\_out;

end Structural;

**shift\_register.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity shift\_register is

    port( parallel\_in: in std\_logic\_vector(16 downto 0); -- (8 bit per A) & (9 bit per Q)

          clock, reset, load, shift: in std\_logic;

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

end shift\_register;

architecture Behavioural of shift\_register is

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

begin

    registro\_a\_scorrimento: process(clock)

    begin

       if(clock'event and clock='1') then

          if(reset='1') then

              temp <= (others=>'0');

          else

              if(load='1') then

                 temp <= parallel\_in;

              elsif(shift='1') then

                 temp(14 downto 0) <= temp(15 downto 1);

              end if;

          end if;

       end if;

    end process;

    parallel\_out <= temp;

end Behavioural;

**M.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity M is  -- registro che mantiene il moltiplicando Y

    port( Y\_in: in std\_logic\_vector(7 downto 0);

          clk, rst, load: in std\_logic;

          Y\_out: out std\_logic\_vector(7 downto 0));

end M;

architecture Behavioural of M is

    signal y: std\_logic\_vector(7 downto 0);

begin

    molt: process(clk)

    begin

        if(clk'event and clk='1') then

          if(rst='1') then

             y <= (others=>'0');

          else

            if(load='1') then

               y <= Y\_in;

            end if;

          end if;

        end if;

    end process;

    Y\_out<=y;

end Behavioural;

**counter.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity counter is -- modulo 9

    port(

        clock, reset: in std\_logic;

        enable: in std\_logic;

        count: out std\_logic\_vector(3 downto 0)

    );

end counter;

architecture Behavioural of counter is

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

begin

    contatore\_mod\_9: process(clock)

    begin

        if(rising\_edge(clock)) then

            if(reset='1') then

                c <= (others=>'0');

            else

                if(enable = '1') then

                    if (unsigned(c) = 9) then

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

                    else

                        c <= std\_logic\_vector(unsigned(c) + 1);

                    end if;

                end if;

            end if;

        end if;

    end process;

    count <= c;

end Behavioural;

**parallel\_adder.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity parallel\_adder is -- sommatore & sottrattore

    port( X, Y: in std\_logic\_vector(7 downto 0);

          cin: in std\_logic;

          Z: out std\_logic\_vector(7 downto 0);

          cout: out std\_logic);

end parallel\_adder;

architecture Structural of parallel\_adder is

    component ripple\_carry is

    port( X, Y: in std\_logic\_vector(7 downto 0);

          c\_in: in std\_logic;

          c\_out: out std\_logic;

          Z: out std\_logic\_vector(7 downto 0));

    end component;

    signal compl\_y: std\_logic\_vector(7 downto 0);

begin

    complemento\_y: for i in 0 to 7 generate

                   compl\_y(i)<=Y(i) xor cin;

                   end generate;

    ripple\_carry\_adder: ripple\_carry port map(

       X=>X,

       Y=>compl\_y,

       c\_in=>cin,

       c\_out=>cout,

       Z=>Z);

end structural;

**ripple\_carry.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ripple\_carry is

    port( X, Y: in std\_logic\_vector(7 downto 0);

          c\_in: in std\_logic;

          c\_out: out std\_logic;

          Z: out std\_logic\_vector(7 downto 0));

end ripple\_carry;

architecture Structural of ripple\_carry is

    component full\_adder is port(

        a, b: in std\_logic;

        cin: in std\_logic;

        cout, s: out std\_logic);

    end component;

    signal temp: std\_logic\_vector(7 downto 0);

    begin

    full\_adder\_0: full\_adder port map(a=>X(0), b=>Y(0), cin=>c\_in, cout=>temp(0), s=>Z(0));

    full\_adder\_1\_to\_6: for i in 1 to 6 generate

       f\_a: full\_adder port map(a=>X(i), b=>Y(i), cin=>temp(i-1), cout=>temp(i), s=>Z(i));

       end generate;

    full\_adder\_7: full\_adder port map(a=>X(7), b=>Y(7), cin=>temp(6), cout=>c\_out, s=>Z(7));

end Structural;

Simulazione

È stato creato un singolo testbench (**molt\_booth\_tb**) per verificare il corretto funzionamento dell'intero sistema di moltiplicazione.

**molt\_booth\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity molt\_booth\_tb is

end molt\_booth\_tb;

architecture Behavioral of molt\_booth\_tb is

    signal clock : std\_logic := '0';

    signal reset : std\_logic := '0';

    signal start : std\_logic := '0';

    signal X, Y : std\_logic\_vector(7 downto 0);

    signal P : std\_logic\_vector(15 downto 0);

    signal stop\_cu : std\_logic;

    constant CLOCK\_PERIOD : time := 10 ns;

begin

    uut: entity work.molt\_booth

        port map (

            clock => clock,

            reset => reset,

            start => start,

            X => X,

            Y => Y,

            P => P,

            stop\_cu => stop\_cu

        );

    clk\_process: process

    begin

        wait for CLOCK\_PERIOD / 2;

        clock <= not clock;

    end process;

    stim: process

    begin

        start <= '0';

        X <= "00000110"; -- 6

        Y <= "11111011"; -- -5

        wait for CLOCK\_PERIOD;

        start <= '1';

        wait for CLOCK\_PERIOD;

        start <= '0';

        while stop\_cu /= '1' loop -- aspetta che la moltiplicazione finisca

            wait for CLOCK\_PERIOD;

        end loop;

        wait for 3\*CLOCK\_PERIOD;

        reset <= '1';

        wait for 10 ns;

        reset<='0';

        X <= "00000101"; -- 5

        Y <= "00000010"; -- 2

        start <= '1';

        wait for CLOCK\_PERIOD;

        start <= '0';

        while stop\_cu /= '1' loop

            wait for CLOCK\_PERIOD;

        end loop;

        wait for 3\*CLOCK\_PERIOD;

        reset <= '1';

        wait for 10 ns;

        reset<='0';

        X <= "11111011"; -- -5

        Y <= "11111011"; -- -5

        start <= '1';

        wait for CLOCK\_PERIOD;

        start <= '0';

        while stop\_cu /= '1' loop

            wait for CLOCK\_PERIOD;

        end loop;

        wait;

    end process;

end Behavioral;

Immagine che contiene schermata, linea

Descrizione generata automaticamente

Sintesi su board di sviluppo

Per fare la sintesi su board di sviluppo, è necessario sviluppare un ulteriore file VHDL denominato **molt\_booth\_on\_board** all’interno del quale si istanzia il moltiplicatore di Booth, nonché il modulo **display\_seven\_segments**, il quale consente la visualizzazione del risultato in formato esadecimale sui display, e il **ButtonDebouncer** applicato al bottone di **start**. Il ButtonDebouncer è una componente cruciale che svolge la funzione di mitigare gli effetti indesiderati causati da eventuali rimbalzi del segnale di input proveniente dal pulsante di avvio.

Gli input di molt\_booth\_on\_board sono **clk**, ovvero il segnale di tempificazione il cui periodo è di 10ns, **reset** collegato al bottone in posizione N17, **start** collegato al pulsante P17, **insert** collegato a P18, il vettore di 8 bit **X** associato agli switch dal pin **R13** al pin **J15** e **Y** associato agli switch da **V10** a **T8**. Gli outputsono, invece, **anodes\_out** e **cathodes\_out**, i quali sono direttamente connessi agli anodi e ai catodi del display.

Premendo **insert**, vengono acquisiti, dagli switch, i valori degli operandi del prodotto (in forma binaria e con segno); successivamente, premendo **start** si avvia l’elaborazione alla fine della quale il risultato viene stampato sulle prime 4 cifre del display. Prima di iniziare una nuova moltiplicazione è necessario premere il pulsante di **reset**.

**molt\_booth\_on\_board.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity molt\_booth\_on\_board is

    Port (clk : in std\_logic;

        reset : in std\_logic; -- da premere prima di calcolare il prossimo prodotto

        start : in std\_logic;

        insert : in std\_logic;

        X : in std\_logic\_vector(7 downto 0);

        Y : in std\_logic\_vector(7 downto 0);

        anodes\_out : out  std\_logic\_vector (7 downto 0);

        cathodes\_out : out  std\_logic\_vector (7 downto 0)

        );

end molt\_booth\_on\_board;

architecture Behavioral of molt\_booth\_on\_board is

    component molt\_booth is

     port( clock, reset, start: in std\_logic;

           X, Y: in std\_logic\_vector(7 downto 0);

           P: out std\_logic\_vector(15 downto 0);

           stop\_cu: out std\_logic);

    end component;

    component ButtonDebouncer is

    generic (

        CLK\_period: integer := 10;

        btn\_noise\_time: integer := 10000000

    );

    Port ( RST : in STD\_LOGIC;

           CLK : in STD\_LOGIC;

           BTN : in STD\_LOGIC;

           CLEARED\_BTN : out STD\_LOGIC);

    end component;

    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(31 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;

    signal x\_in, y\_in : std\_logic\_vector(7 downto 0);

    signal value : std\_logic\_vector(31 downto 0);

    signal P : std\_logic\_vector(15 downto 0);

    signal stop\_cu : std\_logic;

    signal start\_debounced : std\_logic;

begin

    process(clk)

    begin

        if(insert='1') then

            x\_in <= X;

            y\_in <= Y;

        end if;

    end process;

    debouncer : ButtonDebouncer generic map (CLK\_period => 10, btn\_noise\_time => 10000000) port map (RST => reset, CLK => clk, BTN => start, CLEARED\_BTN => start\_debounced);

    molticatore\_booth : molt\_booth port map(clock => clk, reset => reset, start => start\_debounced, X => x\_in, Y => y\_in, P => P, stop\_cu => stop\_cu);

    process(clk)

    begin

        if(reset='1') then

           value <= (others=>'0');

        elsif(stop\_cu='1') then

           value <= "0000000000000000" & P;

        end if;

    end process;

    seven\_segment\_array: display\_seven\_segments GENERIC MAP(

    CLKIN\_freq => 100000000,

    CLKOUT\_freq => 500

    )

    PORT MAP(

        CLK => clk,

        RST => reset,

        value => value,

        enable => "00001111",

        dots => "00000000",

        anodes => anodes\_out,

        cathodes => cathodes\_out

    );

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 { X[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

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

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

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

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

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

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { X[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { X[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { Y[0] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { Y[1] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { Y[2] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { Y[3] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { Y[4] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { Y[5] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { Y[6] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { Y[7] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

##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 { reset }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

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

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