LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE IEEE.NUMERIC\_STD.ALL ;

ENTITY Tape\_Counter IS

PORT(

Clk : IN STD\_LOGIC;

Increment : IN STD\_LOGIC;

Decrement : IN STD\_LOGIC;

Reset : IN STD\_LOGIC;

TimeOut : OUT UNSIGNED(19 downto 0)

);

END;

ARCHITECTURE BEHAVIORAL OF Tape\_Counter IS

Signal tempTime : UNSIGNED(19 downto 0);

begin

COUNT : Process(Reset,Clk,Increment, Decrement)

Begin

if(rising\_edge(clk)) THEN

if(Reset = '0') THEN

if Increment = '1' THEN

tempTime <= tempTime + 1;

elsif Decrement = '1' THEN

temptime <= tempTime - 1;

end if;

else

tempTime <= X"00000";

end if;

end if;

if(temptime > 14400 and increment = '1') then

temptime <= X"03840";

end if;

if temptime = X"00000" and decrement = '1' then

temptime <= X"00000";

end if;

END PROCESS COUNT;

TimeOut <= temptime;

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY HexDisplayDriver IS

PORT(

Inputs\_DCBA : in std\_logic\_vector(3 downto 0) ; -- inputs A,B,C,D = slider switches SW[0], SW[1], SW[2], SW[3] respectively

Segments\_g\_to\_a : out std\_logic\_vector(6 downto 0) -- outputs a,b,c,d,e,f,g = Hex display HEX0 (right most 7 segement display on the Altera DE Board

);

END;

ARCHITECTURE Behavioural OF HexDisplayDriver IS

BEGIN

PROCESS(Inputs\_DCBA) -- process sensitive to changes in these signals

BEGIN

-- describe how the values of ABCD in range {0000 – 1111}, i.e. 0 - F in Hex

-- map to abcdefg using if-else statements etc

if(Inputs\_DCBA = "0000") then

Segments\_g\_to\_a <= "0000001"; -- fill in the required 7 bit value between the quotes, currently all 7 segments turned ON when DCBA = 0000

elsif(Inputs\_DCBA = "0001") then

Segments\_g\_to\_a <= "1001111";

elsif(Inputs\_DCBA = "0010") then

Segments\_g\_to\_a <= "0010010";

elsif(Inputs\_DCBA = "0011") then

Segments\_g\_to\_a <= "0000110";

elsif(Inputs\_DCBA = "0100") then

Segments\_g\_to\_a <= "1001100";

elsif(Inputs\_DCBA = "0101") then

Segments\_g\_to\_a <= "0100100";

elsif(Inputs\_DCBA = "0110") then

Segments\_g\_to\_a <= "0100000";

elsif(Inputs\_DCBA = "0111") then

Segments\_g\_to\_a <= "0001111";

elsif(Inputs\_DCBA = "1000") then

Segments\_g\_to\_a <= "0000000";

elsif(Inputs\_DCBA = "1001") then

Segments\_g\_to\_a <= "0000100";

elsif(Inputs\_DCBA = "1010") then

Segments\_g\_to\_a <= "0001000";

elsif(Inputs\_DCBA = "1011") then

Segments\_g\_to\_a <= "1100000";

elsif(Inputs\_DCBA = "1100") then

Segments\_g\_to\_a <= "0110001";

elsif(Inputs\_DCBA = "1101") then

Segments\_g\_to\_a <= "1000010";

elsif(Inputs\_DCBA = "1110") then

Segments\_g\_to\_a <= "0110000";

elsif(Inputs\_DCBA = "1111") then

Segments\_g\_to\_a <= "0111000";

else -- if inputs = 1111

Segments\_g\_to\_a <= "1111111"; -- fill in the required 7 bit value between the quotes, currently all 7 segments turned OFF when DCBA NOT EQUAL to 0000

end if ;

END PROCESS ; -- end of the process

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE IEEE.NUMERIC\_STD.ALL ;

ENTITY TIME\_FORMATTER IS

PORT(

TimeIN : IN UNSIGNED(19 downto 0);

S , M , H : OUT UNSIGNED(7 downto 0)

);

END;

ARCHITECTURE BEHAVIORAL OF TIME\_FORMATTER IS

SIGNAL seconds, minutes, hours : UNSIGNED (19 downto 0);

begin

Process(TimeIN)

Variable temptime : UNSIGNED(19 downto 0);

begin

temptime := TimeIN;

seconds <= temptime mod 60;

temptime := (temptime - seconds) / 60;

minutes <= temptime mod 60;

hours <= (temptime - minutes) / 60; -- this math guarantees that hours, seconds and minutes are within the correct constraints.

End Process;

S <= seconds(7 downto 0);

M <= minutes(7 downto 0);

H <= hours(7 downto 0);

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE IEEE.NUMERIC\_STD.ALL ;

ENTITY Frequency\_Adjuster IS

PORT(

Clk : IN STD\_LOGIC;

speed : IN STD\_LOGIC\_VECTOR(1 downto 0);

Done :out STD\_LOGIC

);

END;

ARCHITECTURE BEHAVIORAL OF Frequency\_Adjuster IS

SIGNAL count : UNSIGNED(24 downto 0);

SIGNAL ClkTemp :STD\_LOGIC;

begin

Process(Clk, speed)

Variable Conversion : integer range 0 to 25000000;

Begin

if speed = "00" then

conversion := 25000000;

elsif speed = "01" then

conversion := 5000000;

elsif speed = "10" then

conversion := 250000;

end if;

if(rising\_edge(clk) and speed /= "11") THEN

if(count = conversion) THEN

ClkTemp <= not ClkTemp;

count <= count - count;

else

count <= count + 1;

end if;

end if;

END PROCESS;

Done <= ClkTemp;

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE IEEE.NUMERIC\_STD.ALL ;

ENTITY BIN\_TO\_BCD IS

PORT(

BIN : IN UNSIGNED(7 downto 0);

Top, Bottom : OUT UNSIGNED(3 downto 0)

);

END;

ARCHITECTURE BEHAVIORAL OF BIN\_TO\_BCD IS

begin

Process(BIN)

variable z :UNSIGNED (17 downto 0);

begin

for i in 0 to 17 loop

z(i) := '0';

end loop;

z(10 downto 3) := BIN;

for i in 0 to 4 loop

if z(11 downto 8) > 4 then

z(11 downto 8) := z(11 downto 8) +3;

end if;

z(17 downto 1) := z(16 downto 0);

end loop;

Top <= z(15 downto 12);

Bottom <= z(11 downto 8);

End Process;

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE IEEE.NUMERIC\_STD.ALL ;

ENTITY VCR\_CONTROLLER IS

PORT(

Clk , ResetBTN , StopBTN , PlayBTN , PauseBTN , RecBTN , FFBTN , RwdBTN , Tape\_load, ejectBTN , Start\_tape , end\_tape , protected , LCD\_ready : IN STD\_logic;

Message : OUT STD\_LOGIC\_VECTOR(3 downto 0);

speed : out STD\_LOGIC\_VECTOR(1 downto 0);

Count\_reset , Decrement , Increment , Write\_msg , msg\_reset : out STD\_logic;

state\_debug : out std\_LOGIC\_VECTOR (1 downto 0)

);

END;

ARCHITECTURE Mealy OF VCR\_CONTROLLER IS

CONSTANT Play : STD\_LOGIC\_VECTOR(1 downto 0) := "11";

CONSTANT Stop : STD\_LOGIC\_VECTOR(1 downto 0) := "01";

CONSTANT Eject : STD\_LOGIC\_VECTOR(1 downto 0) := "10";

SIGNAL Next\_State , Current\_State : STD\_LOGIC\_VECTOR(1 downto 0);

begin

Process(clk,resetBTN)

Begin

if(resetBTN = '0') then

current\_State <= eject;

count\_reset <= '1';

elsif(rising\_edge(clk) and resetBTN = '1') then

current\_State <= next\_State;

count\_reset <= '0';

end if;

END PROCESS;

--real world inputs

process(current\_State,StopBTN , PlayBTN , PauseBTN , RecBTN , FFBTN , RwdBTN , Tape\_load , Start\_tape , end\_tape, ejectBTN)

begin

next\_State <= current\_State;

message <= "1110";

write\_msg <= '0';

msg\_reset <= '1';

speed <= "11";

decrement <= '0';

increment <= '0';

--current\_State play

if current\_state = play then

if end\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '1' and recBTN = '0' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--ff

next\_State <= play;

message <= "0010";

speed <= "01";

write\_msg <= '0';

increment <= '1';

elsif start\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '1' and FFBTN = '0' and recBTN = '0' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--rwd

next\_State <= play;

message <= "0011";

speed <= "01";

write\_msg <= '0';

decrement <= '1';

elsif ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '0' and stopBTN = '0' and playBTN = '0' and pauseBTN = '1' then

--pause

next\_State <= play;

message <= "0110";

write\_msg <= '0';

elsif end\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '1' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--record

next\_State <= play;

if (protected = '1') then

message <= "0111";

else

message <= "0001";

end if;

speed <= "01";

increment <= '1';

write\_msg <= '0';

elsif ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '0' and stopBTN = '1' and playBTN = '0' and pauseBTN = '0' then

--stop

next\_State <= stop;

elsif tape\_load = '0' then

--eject

next\_State <= eject;

elsif end\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '0' and stopBTN = '0' and pauseBTN = '0' then

--play

next\_State <= play;

message <= "0000";

speed <= "00";

write\_msg <= '0';

increment <= '1';

end if;

--current\_state stop

elsif current\_State = stop then

if tape\_load = '1' and playBTN = '1' then

--play

next\_State <= play;

elsif start\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '1' and FFBTN = '0' and recBTN = '0' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--rewind

next\_State <= stop;

message <= "0011";

speed <= "10";

write\_msg <= '0';

decrement <= '1';

elsif end\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '1' and recBTN = '0' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--ff

next\_State <= stop;

message <= "0010";

speed <= "10";

write\_msg <= '0';

increment <= '1';

elsif end\_tape = '0' and ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '1' and stopBTN = '0' and playBTN = '0' and pauseBTN = '0' then

--record

next\_State <= stop;

if (protected = '1') then

message <= "0111";

speed <= "11";

else

message <= "0001";

speed <= "00";

increment <= '1';

end if;

write\_msg <= '0';

elsif tape\_load = '0' then

--eject

next\_State <= eject;

elsif ejectBTN = '0' and tape\_load = '1' and rwdBTN = '0' and FFBTN = '0' and recBTN = '0' and playBTN = '0' then

--stop

next\_State <= stop;

message <= "0101";

write\_msg <= '0';

end if;

--current\_State eject

elsif current\_State = eject then

if tape\_load = '1' then

next\_State <= stop;

else

message <= "0100";

write\_msg <= '0';

end if;

end if;

end process;

state\_debug <= next\_State;

END;

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.numeric\_std.all ;

-- 5 by 7 pixel ASCii character generator rom

entity MessageRom is

Port (

Address : in std\_logic\_Vector(9 downto 0); -- 10 bits at least 544 locations (17 strings of 32 chars each)

DataOut : out std\_logic\_vector(7 downto 0) -- a character

);

end ;

architecture bhvr of MessageRom is

type CharRom5x7 is array ( 0 to 1023 ) of std\_logic\_vector(7 downto 0);

constant MyRom : CharRom5x7 :=

(

-- message strings 0 - 15 for displaying on LCD

-- Hex 80 or Hex C0 must be first character as it tells LCD to go to beginning of line 0 or line 1

-- then you can have upto 30 characters or text (remember LCD only displays 16 on a line)

-- then you MUST have a hex 00 byte at the end

-- do not delete ANY of these characters even if you do not use them, although you can replace the text

-- they are all needed.

X"80",X"50",X"6c",X"61",X"79",X"69",X"6e",X"67",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --Playing

X"80",X"52",X"65",X"63",X"6f",X"72",X"64",X"69",X"6e",X"67",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --Recording

X"80",X"46",X"46",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --FF

X"80",X"52",X"57",X"44",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --RWD

X"80",X"45",X"4a",X"45",X"43",X"54",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --EJECT

X"80",X"53",X"54",X"4f",X"50",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --STOP

X"80",X"50",X"41",X"55",X"53",X"45",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --PAUSE

X"80",X"50",X"72",X"6f",X"74",X"65",X"63",X"74",X"65",X"64",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", --Protected

X"80",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"80",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"C0",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"C0",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"80",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"80",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"C0",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

X"C0",X"21",X"21",X"21",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00", X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

-- final (17th) init string DO NOT CHANGE or DELETE - it contains characters to initialise the LCD display to use 2 line display, 8 bit data interface, cursor display on, display on and then clear display

X"38",X"0e",X"01",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",

others => X"00"

);

Begin

process(Address)

variable index : integer range 0 to 1023 ;

begin

index := to\_integer(unsigned(Address)) ;

DataOut <= MyRom(index);

end process ;

END ;