```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: David Paquette
--
-- Create Date: 11/19/15
-- Design Name:
-- Module Name:
-- Project Name:
-- Target Device:
-- Tool versions:
-- Description:

-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
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 MemoryWriter is
      Port ( clk_i : in std_logic;
          rst_i : in std_logic;
                adr_o : out std_logic_vector(31 downto 0);
          dat_i : in std_logic_vector(31 downto 0);
          dat_o : out std_logic_vector(31 downto 0);
          ack_i : in std_logic;
          cyc_o : out std_logic;
          stb_o : out std_logic;
          we_o  : out std_logic;
                   currentTemperature: in integer range 0 to 100;
                   desiredTemperature : in integer range 0 to 100;
                   fanSpeedPercent : in integer range 0 to 100
                   );
end MemoryWriter;

architecture Behavioral of MemoryWriter is
      signal ascii : std_logic_vector(7 downto 0);
      signal pixelnum : integer range 0 to 7;

      signal buffer_base : std_logic_vector(31 downto 0):=(others=>'0');
      signal rst_p : std_logic;
      signal txtcolor : std_logic_vector(3 downto 0):="1111"; -- white
      signal bgcolor : std_logic_vector(3 downto 0):="0001"; -- blue
      signal pixels : std_logic_vector(7 downto 0);
```

```
	type StateType is (initialzeMemoryState, writeIntialMemoryState,
		startState,
		writePixelToMemory, getPixelData, waitState);
	signal state : StateType := initialzeMemoryState;
	signal memoryInitilizationComplete : std_logic:='0';
	signal row, column, line : integer range 0 to 100:=0;
	signal cletter : std_logic_vector(7 downto 0):=x"63"; --"c"
	signal dletter : std_logic_vector(7 downto 0):=x"64"; --"d"
	signal fletter : std_logic_vector(7 downto 0):=x"66"; --"f"
	signal Sletter : std_logic_vector(7 downto 0):=x"53"; --"S"
	signal Tletter : std_logic_vector(7 downto 0):=x"54"; --"T"
	signal equalletter : std_logic_vector(7 downto 0):=x"3d"; --"="
	signal twoDigitAscii : std_logic_vector(15 downto 0);
	signal MVLetters : std_logic_vector(15 downto 0):=x"4d56"; --MV
	signal number : integer range 0 to 100:=0;
begin

	rst_p <= not rst_i;

--	 the lookup table maps the ascii code to the pixels for that particular
	character.
--	 The line input determines which of the 12 lines of the character we want.
	The
--	 lookup table is implemented with the builtin registered BRAM, so the
	output is
--	 available only at the next clock cycle
	lut : entity work.char8x12_lookup_table
		port map( clk => clk_i, reset => rst_p, ascii => ascii, line => line,
			pixels => pixels );

	-- two digit int to ascii
	process(number)
	begin
		if(number < 100) then
			twoDigitAscii(15 downto 8)<= x"30" +
				std_logic_vector(to_unsigned((number-(number mod 10))/
				10, 8));
			twoDigitAscii(7 downto 0)<= x"30" +
				std_logic_vector(to_unsigned((number mod 10), 8));
		else
			twoDigitAscii<=MVLetters;
		end if;
	end process;


	process( clk_i, rst_i, state)
		variable textCounter : std_logic_vector(15 downto 0):=(others=>'0');
	begin
		if ( rst_i = '0' ) then
			state <= initialzeMemoryState;
			column <= 0;
			line <= 0;
			row <= 0;
			stb_o<='0';
			cyc_o<='0';
			we_o <='0';
```

```
elsif ( clk_i'event and clk_i='1' ) then
      case state is
            when initialzeMemoryState=>
                  cyc_o<='1';
                  stb_o<='0';
                  we_o <='0';
                  state<= writeIntialMemoryState;
                  line <= line + 1;
                  if(line = 11) then
                        line <= 0;
                        column <= column + 1;
                        if(column = 79) then
                              column <= 0;
                              row<=row + 1;
                              if(row = 39) then
                                    row <= 0;
                                    state<=startState;
                                    column <= 0;
                                    line <= 0;
                                    row <= 0;
                              end if;
                        end if;
                  end if;
            when writeIntialMemoryState=>
                  stb_o<='1';
                  we_o <='1';
                  adr_o <= buffer_base + (row*80*12 + column +
                        80*line)*4;
                  dat_o <=
                        bgcolor&bgcolor&bgcolor&bgcolor&bgcolor&bgco
                        lor&bgcolor&bgcolor;
                  if(ack_i='1') then
                        state<= initialzeMemoryState;
                  end if;
            when startState=>
                  column <= 39;
                  line <= 0;
                  row <= 19;
                  stb_o<='0';
                  cyc_o<='0';
                  state<=getPixelData;
            when getPixelData=>
                  line <= line + 1;
                  if(textCounter=0) then
                  --print c
                        ascii <= cletter;
                        number <= currentTemperature; -- queue up
                              current temp for ascii conversion
                  elsif(textCounter=1) then
                  --print T
                        ascii <= Tletter;
                  elsif(textCounter=2) then
                  --print =
                        ascii <= equalletter;
                  elsif(textCounter=3) then
                  --print first digit of current temp
```

```
      ascii <= twoDigitAscii(15 downto 8);
elsif(textCounter=4) then
--print second digit of current temp
      ascii <= twoDigitAscii(7 downto 0);
elsif(textCounter=5) then
--print d
      ascii <= dletter;
      number <= desiredTemperature; --queue up
            desired temp for ascii conversion
elsif(textCounter=6) then
--print T
      ascii <= Tletter;
elsif(textCounter=7) then
--print =
      ascii <= equalletter;
elsif(textCounter=8) then
--print first digit of desired temp
      ascii <= twoDigitAscii(15 downto 8);
elsif(textCounter=9) then
      ascii <= twoDigitAscii(7 downto 0);
elsif(textCounter=10) then
--print f
      ascii <= fletter;
      number <= fanSpeedPercent; --queue up fan
            speed for ascii conversion
elsif(textCounter=11) then
--print S
      ascii <= Sletter;
elsif(textCounter=12) then
--print =
      ascii <= equalletter;
elsif(textCounter=13) then
--print first digit of fan speed
      ascii <= twoDigitAscii(15 downto 8);
elsif(textCounter=14) then
      ascii <= twoDigitAscii(7 downto 0);
end if;
if(line >= 11) then
      column <= column + 1;
      line <= 0;
      textCounter := textCounter + 1;
      if(textCounter = 0 )then
      elsif(textCounter = 1) then
            column <= column + 1;
      elsif(textCounter = 2) then
            column <= column + 1;
      elsif(textCounter = 3) then
            column <= column + 1;
      elsif(textCounter = 4) then
            column <= column + 1;
      elsif(textCounter = 5) then
            column <= 39;
            row <= row + 1;
      elsif(textCounter = 6) then
            column <= column + 1;
      elsif(textCounter = 7) then
```

```
                        column <= column + 1;
                    elsif(textCounter = 8) then
                        column <= column + 1;
                    elsif(textCounter = 9) then
                        column <= column + 1;
                    elsif(textCounter = 10) then
                        row <=row + 1;
                        column <= 39;
                    elsif(textCounter=11) then
                        column <= column + 1;
                    elsif(textCounter=12) then
                        column <= column + 1;
                    elsif(textCounter=13) then
                        column <= column + 1;
                    elsif(textCounter=14) then
                        column <= column + 1;
                    elsif(textCounter > 14) then
                        column <= 39;
                        row <=19;
                        textCounter := (others=>'0');
                    end if;
                else
                    state<=writePixelToMemory;
                end if;
            when writePixelToMemory=>
                stb_o<='1';
                we_o <='1';
                cyc_o<='1';
                adr_o <= buffer_base + (row*80*12 + column +
                    80*line)*4;
                if(pixels(0)='1')then dat_o(3 downto 0) <=
                    txtcolor;else dat_o(3 downto 0) <=
                    bgcolor;end if;
                if(pixels(1)='1')then dat_o(7 downto 4) <=
                    txtcolor;else dat_o(7 downto 4) <=
                    bgcolor;end if;
                if(pixels(2)='1')then dat_o(11 downto 8) <=
                    txtcolor;else dat_o(11 downto 8) <=
                    bgcolor;end if;
                if(pixels(3)='1')then dat_o(15 downto 12) <=
                    txtcolor;else dat_o(15 downto 12) <=
                    bgcolor;end if;
                if(pixels(4)='1')then dat_o(19 downto 16) <=
                    txtcolor;else dat_o(19 downto 16) <=
                    bgcolor;end if;
                if(pixels(5)='1')then dat_o(23 downto 20) <=
                    txtcolor;else dat_o(23 downto 20) <=
                    bgcolor;end if;
                if(pixels(6)='1')then dat_o(27 downto 24) <=
                    txtcolor;else dat_o(27 downto 24) <=
                    bgcolor;end if;
                if(pixels(7)='1')then dat_o(31 downto 28) <=
                    txtcolor;else dat_o(31 downto 28) <=
                    bgcolor;end if;
                if(ack_i='1') then
                    stb_o<='0';
```

```
                                        cyc_o<='0';
                                        we_o <='0';
                                        state<=getPixelData;
                                    end if;
                                when waitState=>

                            end case;
                    end if;
            end process;

end Behavioral;
```