DB Code

Jake Humphrey

March 2014

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.project_pack.all;
use work.draw_any_octant;

entity db is
  generic(vsize : integer := 6);
  port(
    clk         : in  std_logic;
    reset       : in  std_logic;

    -- host processor connections
    hdb         : in  std_logic_vector(2*vsize+3 downto 0);
    dav         : in  std_logic;
    hdb_busy    : out std_logic;

    -- rcb connections
    dbb_bus     : out db_2_rcb;
    dbb_delaycmd : in std_logic;
    dbb_rcbclear : in std_logic;

    -- vdp connection
    db_finish   : out std_logic
    );
end db;

architecture rtl of db is
  signal dao_draw, dao_xbias, dao_done, dao_swap, dao_negx, dao_negy,
      dao_disable, dao_reset : std_logic;
  signal dao_xin, dao_yin, dao_xout, dao_yout: std_logic_vector(vsize
      downto 0);
  signal pen_x, pen_y: std_logic_vector(vsize-1 downto 0);
  signal previous_command : std_logic_vector(2*vsize+3 downto 0);

  type state_t is (idle, draw_reset, draw_start, send_command);
  signal state, nstate : state_t;

  type opcode_t is array (1 downto 0) of std_logic;
  constant movepen_op    : opcode_t := "00";
  constant drawline_op   : opcode_t := "01";
  constant clearscreen_op : opcode_t := "10";

  type pentype_t is array (1 downto 0) of std_logic;
  constant white : pentype_t := "01";
  constant black : pentype_t := "10";
  constant invert : pentype_t := "11";

  type command_t is record
    op   : opcode_t;
    x, y : std_logic_vector(vsize-1 downto 0);
    pen  : pentype_t;
  end record;
  signal command_in, command, prev_command : command_t;
begin
  -- decoding command
  command_in.op <= opcode_t(hdb(2*vsize+3 downto 2*vsize+2));
  command_in.x  <= hdb(2*vsize+1 downto vsize+2);
  command_in.y  <= hdb(vsize+1 downto 2);
  command_in.pen <= pentype_t(hdb(1 downto 0));

  -- disable dao when rcb not ready
  dao_disable <= dbb_delaycmd;

  dao: entity draw_any_octant generic map(vsize) port map(
    clk => clk,
    resetx => dao_reset,
    draw => dao_draw,
    xbias => dao_xbias,
    xin => dao_xin,
    yin => dao_yin,
    done => dao_done,
    x => dao_xout,
    y => dao_yout,
    swapxy => dao_swap,
    negx => dao_negx,
    negy => dao_negy,
    disable => dao_disable
    );

  read_new_command: process
  begin
    wait until clk'event and clk='1';
    if state = idle and dav = '1' then
      command.op <= opcode_t(hdb(2*vsize+3 downto 2*vsize+2));
      command.x  <= hdb(2*vsize+1 downto vsize+2);
      command.y  <= hdb(vsize+1 downto 2);
      command.pen <= pentype_t(hdb(1 downto 0));
      prev_command <= command;
    end if;
  end process read_new_command;

  set_dao_inputs: process(command, prev_command) -- drives negx, negy,
      swapxy,
                                                 -- xin, yin, xbias
    variable dx: signed(vsize downto 0);
    variable dy: signed(vsize downto 0);
    --variable zero : std_logic_vector(vsize-1 downto 0) := (others
        =>'0');
  begin
    dx := signed(resize(unsigned(command.x), vsize+1)) -
        signed(resize(unsigned(prev_command.x), vsize+1));
    dy := signed(resize(unsigned(command.y), vsize+1)) -
        signed(resize(unsigned(prev_command.y), vsize+1));
    -- set negx if dx is negative
    if dx >= 0 then
      dao_negx <= '0';
    else
      dao_negx <= '1';
    end if;
    -- set negy if dy is negative
    if dy >= 0 then
      dao_negy <= '0';
    else
      dao_negy <= '1';
    end if;
    -- set swapxy if dx is closer to 0 than dy
    if abs(dx) < abs(dy) then
      dao_swap <= '1';
    else
      dao_swap <= '0';
    end if;
    --
    if state = draw_reset then
      dao_xin <= prev_command.x;
      dao_yin <= prev_command.y;
    else
      dao_xin <= command.x;
      dao_yin <= command.y;
    end if;
    dao_xbias <= 'X'; --God knows
  end process set_dao_inputs;

  db_fsm_clocked: process
  begin
    wait until clk'event and clk='1';
    -- go to next state
```

```
    state <= nstate;
  end process db_fsm_clocked;

  db_fsm_comb: process(state, command, dav, dao_done, dbb_delaycmd) --
      drives nstate, hdb_busy, dao_draw, dao_reset
  begin
    nstate <= state; --default, stay in current state
    case state is
      when idle =>
        -- outputs for idle state
        hdb_busy <= '0';
        dao_draw <= '0';
        dao_reset <= '0';
        --compute next state
        if dav = '1' then
          --read command and decide which state to go to.
          case command_in.op is
            when movepen_op => nstate <= send_command;
            when drawline_op => nstate <= draw_reset;
            when clearscreen_op => nstate <= send_command;
            when others => null;
          end case;
        end if;
      when draw_reset =>
        --outputs for draw_reset state
        hdb_busy <= '1';
        dao_draw <= '0';
        dao_reset <= '1';
        --compute next state
        nstate <= draw_start;
      when draw_start =>
        --outputs for draw_start state
        hdb_busy <= '1';
        dao_draw <= '1';
        dao_reset <= '0';
        --compute next state
        nstate <= send_command;
      when send_command =>
        --outputs for send_command state
        hdb_busy <= '1';
        dao_draw <= '0';
        dao_reset <= '0';
        --compute next state
        if (command.op = drawline_op and dao_done = '0') or dbb_delaycmd
            = '1' then nstate <= send_command;
        else nstate <= idle;
        end if;
      when others => nstate <= idle; -- reset undefined states to idle
          state
    end case;
  end process db_fsm_comb;

  send_rcb_inputs: process(state, prev_command, command, dao_xout, dao_yout) --drives dbb_bus
    variable undefined : std_logic_vector(vsize-1 downto 0) := (others =>'X');
  begin
    if state = send_command then
      if command.op = drawline_op then
        dbb_bus.x <= dao_xout;
        dbb_bus.y <= dao_yout;
        dbb_bus.startcmd <= '1';
      else
        dbb_bus.X <= command.x;
        dbb_bus.Y <= command.y;
        dbb_bus.startcmd <= '1';
      end if;
    else
      dbb_bus.X <= undefined;
      dbb_bus.Y <= undefined;
      dbb_bus.startcmd <= '0';
    end if;

    -- encode operation
    case command.pen is
      when white => dbb_bus.rcb_cmd(1 downto 0) <= "01";
      when black => dbb_bus.rcb_cmd(1 downto 0) <= "10";
      when invert => dbb_bus.rcb_cmd(1 downto 0) <= "11";
      when others => dbb_bus.rcb_cmd <= "100"; -- invalid command
    end case;
    case command.op is
      when movepen_op => dbb_bus.rcb_cmd <= "000";
      when drawline_op => dbb_bus.rcb_cmd(2) <= '0';
      when clearscreen_op => dbb_bus.rcb_cmd(2) <= '1';
      when others => dbb_bus.rcb_cmd <= "100"; -- invalid command
    end case;
  end process send_rcb_inputs;

  finished: process(state, dav) -- drives db_finish
  begin
    if state = idle and dav = '0' then db_finish <= '1';
    else db_finish <= '0';
    end if;
  end process finished;
end rtl;
```