Draw Block Code Review
Farhan Rahman
November 28, 2011

Block being reviewed: Draw Block
Reviewer: Farhan Rahman (fr909)
Reviewee: Jonathan Ely (jonathan.ely09)

The code has been analysed using the following criteria as stated in the Code Review Guideline.

Compilation:

The code compiles well in ModelSim however there are some warnings when the code is synthesised.

Synthesis, Code Correctness & Design correctness:

The following problems were identified in Draw block that relates to synthesis:

1) The code was not changed in ***draw_octant*** to support 6 bits for x and y. Moreover when the entity inside ***draw_any_octant*** was created for ***draw_octant***, no generic was used to make the length of x and y to be 6 bits, instead only 12 bits default was used. This might have caused some errors while communicating with the ***RCB***. This can be fixed in the following way:

FIX: Introduce generic in ***draw_octant***. Default value is 12 and this has to be overridden when creating the entity in ***db.vhd.***

Overriding "**vsize**" in **line 25** of ***draw_any_octant.vhd*** in the generic provided with "8" and passing it along to override that of ***draw_octant.vhd.***

2) One of the things that I have noticed is that in both ***draw_octant.vhd*** and ***draw_any_octant.vhd,*** in order to stall the draw block when the RCB has been delayed, a gated input with the *clock* and the *delaycmd* has been used. It can be seen in

1) line 129 of *draw_any_octant.vhd,* in process R1
```
"WAIT UNTIL clk'EVENT AND clk = '1' AND delay = '0';"
```
2) line 55 of *draw_octant.vhd,* in process R1
```
"WAIT UNTIL clk'EVENT AND clk = '1' AND delay = '0';"
```
3) line 191 of db.vhd in process FSM
```
"WAIT UNTIL clk'EVENT AND clk = '1' AND delay = '0';"
```

All the above code will give rise to difficult delta delays.

Possible latches:
3) In the process ***STATECOMB*** of ***db.vhd,*** from **line 128**, it can be seen that the whole process is wrapped in an "IF STATEMENT" followed by "CASE STATEMENTS". There is no part which shows the "ELSE" part of the IF statement and therefore creates a latch for all the variables that are inside the IF statement such as the signals "**y, x, pens, peny hdb_busy"** and so on.

FIX: This can be easily fixed by giving default values for the signals that are being assigned. Once synthesised, this error is flagged as "feedback MUX created" for the input.

Regarding Rules stated in lectures:

Processes with WAIT FOR statements: None of the processes have used this.
Processes with both positive and negative edges: None of the processes have used this.
FOR or WHILE loops with variable length: no processes have FOR or WHILE loops.
Badly formed processes: There are some instances where processes have been formed badly. This can be seen at the top where I have mentioned about latches and so on.
Signals driven from two different processes: I didn't find any places where such a rule was broken.

FSM:

Regarding the Finite State Machine, the following problems might arise.

1) Problem might arise in state "listen". The draw block is in the " listen" state when "***reset, moveop, flush***" and "***clear***" commands are true. This might not be a problem for "***reset***" or "***moveop***" because they can happen in one clock cycle. **NOTE** that in this state, the **hdb_busy** signal is set high, which makes the host processor wait. However the problem arises when:
Flush command is sent to **RCB**: When the flush command is sent, the ***RCB*** spends some cycles to flush out to the **VRAM**. However the ***draw_block*** is in the listen state and still sends command to the ***RCB***. I think that the ***delaycmd*** should somehow map to the hdb_busy with perhaps a gated (ORED) output.
Similar problem might arise if the ***RCB*** is in the clearscreen state. In this state the ***RCB*** is going to spend significantly more cycles and this is going to be much worse.

2)

```
-       WHEN draw_start =>
-               -- Send end position to draw_octant
-               draw_reset <= '0';
-               draw <= '1';
-               dxin <= xin;
-               dyin <= yin;
-               nstate <= draw_run;
```

The above is a snippet from ***db.vhd*** from ***line 161 to 167.*** In the state diagram given with the Draw block report, it is shown that when the block is in *draw_start* state, it goes to *draw_run* state if "**draw = 1 && draw_reset = 0**", but however in the FSM, it does not check for this condition, it changes these signals and then goes to *draw_run* state, which contradicts with the state diagram.

3) Another issue that might arise with the FSM design is that the draw_block drives *hdb_busy* high but it should also be a gated output with *delaycmd*. None of the states take into account *delaycmd*.

Reset

1) It is stated that the blocks should have common reset inputs. However, it can be seen in db.vhd that **draw_any_octant** entity has a resetx that is mapped to a signal called **draw_reset.** This should have been mapped to reset that is provided by the host processor i.e. common reset.

Code Style

1) In ***draw_octant.vhd*** in process R1 (**from line 52 to line 84**), the code can be optimised in the following way:

resetx = '0' is include in the else part of resetx = '1' and this can be removed because the else part of resetx = '1' implies that already. The last else part where ((resetx = '0') and (draw = '0') and (done1 = '1')), can be removed as it is a NULL statement.

Problem with report:

1) One problem with the report is that there is no state transition table. This will help the reader identify what the states outputs are, which hasn't been included in the states as well.
2) A diagram for the Draw block with internal connections and inputs and outputs has not been provided.

CODE FOR DRAW BLOCK:

db.vhd:

```
--DRAW BLOCK takes input from Host processor and outputs to rcb,
implemented as FSM
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE WORK.ALL;

ENTITY draw_block IS
      GENERIC(
            xsize             : INTEGER := 6;
            ysize             : INTEGER := 6
            );
      PORT(
            -- HOST INTERFACE
            clk,reset,hdb_dav : IN std_logic;
            hdb                     : IN
std_logic_vector(3+xsize+ysize DOWNTO 0);
            hdb_busy                : OUT std_logic;

            -- DB/RCB Interface
            delaycmd          : IN std_logic;
            x                       : OUT std_logic_vector(xsize-1
DOWNTO 0);
            y                       : OUT std_logic_vector(ysize-1
DOWNTO 0);
            rcbcmd                  : OUT std_logic_vector(2 DOWNTO
0);
            startcmd          : OUT std_logic
      );
END ENTITY draw_block;

ARCHITECTURE behav OF draw_block IS
-- FSM Signals
TYPE   state_t                IS (draw_run, draw_start, listen);
SIGNAL state, nstate          : state_t;

-- General Signals
SIGNAL op, pen                : std_logic_vector(1 DOWNTO 0);
SIGNAL xin                    : std_logic_vector(xsize-1 DOWNTO 0);
SIGNAL yin                    : std_logic_vector(ysize-1 DOWNTO 0);
SIGNAL penx                   : std_logic_vector(xsize-1 DOWNTO 0);
SIGNAL peny                   : std_logic_vector(ysize-1 DOWNTO 0);

-- draw_octant signals
```

```
SIGNAL swapxy, negx, negy           : std_logic;
SIGNAL draw_reset, draw_done, draw, xbias : std_logic;
SIGNAL draw_x                               : std_logic_vector(xsize-1
DOWNTO 0);
SIGNAL draw_y                               : std_logic_vector(ysize-1
DOWNTO 0);
SIGNAL dxin                       : std_logic_vector(xsize-1 DOWNTO
0);
SIGNAL dyin                     : std_logic_vector(ysize-1 DOWNTO 0);

BEGIN

-- wrapper for draw_any_octant

-- swapxy negx  negy  octant
--  0      0      0     ENE
--  1      0      0     NNE
--  1      1      0     NNW
--  0      1      0     WNW
--  0      1      1     WSW
--  1      1      1     SSW
--  1      0      1     SSE
--  0      0      1     ESE

draw_block_i      : ENTITY draw_any_octant
      PORT MAP(
            -- IN
            clk    => clk,
            resetx => draw_reset,
            delay  => delaycmd,
            draw   => draw,
            xbias  => xbias,
            xin    => dxin,
            yin    => dyin,
            swapxy => swapxy,
            negx   => negx,
            negy   => negy,
            -- OUT
            done   => draw_done,
            x      => draw_x,
            y      => draw_y
            );

-- Set useful signals
SIGS: PROCESS(hdb)
BEGIN
      op  <= hdb(3+xsize+ysize DOWNTO 2+xsize+ysize);
      pen <= hdb(1 DOWNTO 0);
      xin <= hdb(1+xsize+ysize DOWNTO 2+ysize);
      yin <= hdb(1+ysize DOWNTO 2);
END PROCESS SIGS;

-- Configure draw octant - Combinational
OCT: PROCESS(xin, yin, penx, peny)
BEGIN
      xbias <= '1';

      -- Shall we swap xy? reflects on x=y
      IF (abs(signed(xin) - signed(penx)) < abs(signed(yin) -
signed(peny))) THEN
            swapxy <= '1';
```

```
        ELSE
              swapxy <= '0';
        END IF;

        -- Is x negative?
        IF (penx > xin) THEN
              negx <= '1';
        ELSE
              negx <= '0';
        END IF;

        -- Is y negative?
        IF (peny > yin) THEN
              negy <= '1';
        ELSE
              negy <= '0';
        END IF;

END PROCESS OCT;


-- State Combinational Logic
STATECOMB: PROCESS(state, hdb_dav, xin, yin, penx, peny, pen, draw_x,
draw_y, op, draw_done)
BEGIN
        -- defaults
        nstate <= state;

        -- First, is the input valid?
        IF (hdb_dav = '1') THEN

        CASE state IS
              WHEN listen =>
                    hdb_busy <= '0';
                    startcmd <= '0';

                    -- check op and deal with it
                    CASE op IS
                          WHEN "00" => -- Move pen
                                penx <= xin;
                                peny <= yin;

                          WHEN "01" => -- Draw
                                -- Send start postion to draw_octant
                                hdb_busy <= '1';
                                draw_reset <= '1';
                                dxin <= penx;
                                dyin <= peny;
                                nstate <= draw_start;

                          WHEN "10" => -- Clear
                                x <= xin;
                                y <= yin;
                                rcbcmd <= "1" & pen;
                                startcmd <= '1';

                          WHEN "11" => -- Flush
                                rcbcmd <= "000";
                                startcmd <= '1';

                          WHEN others =>
```

```
                        null;

                END CASE;

            WHEN draw_start =>
                -- Send end position to draw_octant
                draw_reset <= '0';
                draw <= '1';
                dxin <= xin;
                dyin <= yin;
                nstate <= draw_run;

            WHEN draw_run =>
                -- Run draw sending result to RCB
                rcbcmd <= "0" & pen;
                x <= draw_x;
                y <= draw_y;
                startcmd <= '1';

                -- Check if finished
                IF (draw_done = '1') THEN
                    nstate <= listen;
                END IF;

        END CASE;

        END IF;

    END PROCESS STATECOMB;


    -- State change clocked
    FSM: PROCESS
    BEGIN
    WAIT UNTIL clk'EVENT AND clk = '1' AND delaycmd ='0';
        state <= nstate;
        IF reset = '1' THEN
            state <= listen; -- sychronous reset
        END IF;
    END PROCESS FSM;

    END ARCHITECTURE behav;
```

draw_any_octant.vhd:

```
    LIBRARY IEEE;

    USE IEEE.std_logic_1164.ALL;
    USE IEEE.numeric_std.ALL;
    USE work.ALL;

    ENTITY draw_any_octant IS

      -- swapxy negx  negy  octant
      --  0      0      0     ENE
      --  1      0      0     NNE
      --  1      1      0     NNW
      --  0      1      0     WNW
      --  0      1      1     WSW
```

```
  --  1       1       1      SSW
  --  1       0       1      SSE
  --  0       0       1      ESE

  -- swapxy: x & y swap round on inputs & outputs
  -- negx:   invert bits of x on inputs & outputs
  -- negy:   invert bits of y on inputs & outputs

  -- xbias always give nias in x axis direction, so swapxy must
invert xbias
  GENERIC(
    vsize: INTEGER := 12
  );

  PORT(
    clk, resetx, delay, draw, xbias : IN  std_logic;
    xin, yin                 : IN  std_logic_vector(vsize-1 DOWNTO
0);
    done                     : OUT std_logic;
    x, y                     : OUT std_logic_vector(vsize-1 DOWNTO
0);
    swapxy, negx, negy       : IN  std_logic
    );
END ENTITY draw_any_octant;

ARCHITECTURE comb OF draw_any_octant IS
  -- you may find the following signals useful, if not delete them
  SIGNAL xin1, yin1, xin1temp, yin1temp, x1, y1, xtemp, ytemp
      : std_logic_vector(xin'range);
  SIGNAL xbias1, negx1, negy1, swapxy1
                 : std_logic;

BEGIN

      -- wrapper draw_octant
      wrapper : ENTITY draw_octant
    PORT MAP (
      clk    => clk,
      resetx => resetx,
      delay  => delay,
        draw   => draw,
      done   => done,
      x      => x1,
      y      => y1,
      xin    => xin1,
      yin    => yin1,
      xbias  => xbias1);

      C1 : PROCESS(xin, yin, yin1temp, xin1temp, xbias, negx, negy,
swapxy) -- combinational process for cycle x

      BEGIN

            -- defaults
            xin1temp <= xin;
            yin1temp <= yin;

            -- negx
            IF (negx = '1') THEN
                  xin1temp <= NOT xin;
            END IF;
```

```
		-- negy
		IF (negy = '1') THEN
			yin1temp <= NOT yin;
		END IF;

		-- swapxy & final xyin
		IF (swapxy = '1') THEN
			xin1 <= yin1temp;
			yin1 <= xin1temp;
		ELSE
			xin1 <= xin1temp;
			yin1 <= yin1temp;
		END IF;

		-- xbias
		IF (swapxy = '1') THEN
			xbias1 <= NOT xbias;
		ELSE
			xbias1 <= xbias;
		END IF;

	END PROCESS C1;


	C2 : PROCESS(x1, y1, negx1, negy1, swapxy1, xtemp, ytemp) --
combinational process for cycle x+1

	BEGIN

		-- defaults
		xtemp <= x1;
		ytemp <= y1;

		-- negx
		IF (negx1 = '1') THEN
			xtemp <= NOT x1;
		END IF;

		-- negy
		IF (negy1 = '1') THEN
			ytemp <= NOT y1;
		END IF;

		-- swapxy & final xyin
		IF (swapxy1 = '1') THEN
			x <= ytemp;
			y <= xtemp;
		ELSE
			x <= xtemp;
			y <= ytemp;
		END IF;


	END PROCESS C2;


	R1 : PROCESS -- registered process

	BEGIN
		WAIT UNTIL clk'EVENT AND clk = '1' AND delay = '0';
```

```
                    -- n+1 cycle signals
                    negx1 <= negx;
                    negy1 <= negy;
                    swapxy1 <= swapxy;

        END PROCESS R1;

END ARCHITECTURE comb;
```

draw_octant.vhd:

```
LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY draw_octant IS
  PORT(
    clk, resetx, delay, draw, xbias : IN  std_logic;
    xin, yin                        : IN  std_logic_vector(11 DOWNTO
0);
    done                            : OUT std_logic;
    x, y                            : OUT std_logic_vector(11 DOWNTO
0)
    );
END ENTITY draw_octant;

ARCHITECTURE comb OF draw_octant IS

  SIGNAL done1                    : std_logic;
  SIGNAL x1, y1                   : std_logic_vector(11 DOWNTO 0);
  SIGNAL xincr, yincr, xnew, ynew : std_logic_vector(11 DOWNTO 0);
  SIGNAL error                    : std_logic_vector(11 DOWNTO 0);
  SIGNAL err1, err2               : std_logic_vector(12 DOWNTO 0);

  ALIAS slv IS std_logic_vector;
  ALIAS sg  IS signed;

BEGIN


  C1 : PROCESS(error, xincr, yincr, x1, y1, xnew, ynew, resetx, draw)


  BEGIN
            -- err1 = | error + yincr |
            err1 <= slv(resize(abs(sg(error) + sg(yincr)), 13));

            -- err2 = | error + yincr - xincr |
            err2 <= slv(resize(abs(sg(error) + sg(yincr) -
sg(xincr)), 13));

            -- done =  x = xnew and y = ynew
            IF ((x1 = xnew) and (y1 = ynew) and (resetx = '0') and
(draw = '0')) THEN
```

```
            done1 <= '1';
          ELSE
            done1 <= '0';
          END IF;


  END PROCESS C1;

  R1 : PROCESS

  BEGIN
          WAIT UNTIL clk'EVENT AND clk = '1' AND delay = '0';
          IF (resetx = '1') THEN -- RESET
                error <= "000000000000";
                x1 <= xin;
                y1 <= yin;
                xincr <= "000000000000";
                yincr <= "000000000000";
                xnew <= xin;
                ynew <= yin;

          ELSIF ((resetx = '0') and (draw = '1')) THEN -- DRAW
                xincr <= slv(sg(xin) - sg(x1));
                yincr <= slv(sg(yin) - sg(y1));
                xnew <= xin;
                ynew <= yin;

          ELSIF ((resetx = '0') and (draw = '0') and (done1 = '0'))
THEN
                x1 <= slv(sg(x1) + 1);

                IF ((sg(err1) > sg(err2)) or ((err1 = err2) and
(xbias = '0'))) THEN
                      y1 <= slv(sg(y1) + 1);
                      error <= slv(sg(error) + sg(yincr) -
sg(xincr));
                ELSIF ((sg(err1) < sg(err2)) or ((err1 = err2) and
(xbias = '1'))) THEN
                      error <= slv(sg(error) + sg(yincr));
                END IF;

          ELSIF ((resetx = '0') and (draw = '0') and (done1 = '1'))
THEN
                NULL; -- do nothing

          END IF;


  END PROCESS R1;

      done <= done1;

          -- drive x and y from x1 and y1 (otherwise we cannot read
output)
      x <= x1;
      y <= y1;
END ARCHITECTURE comb;
```

Draw Block Documentation

Jonathan Ely

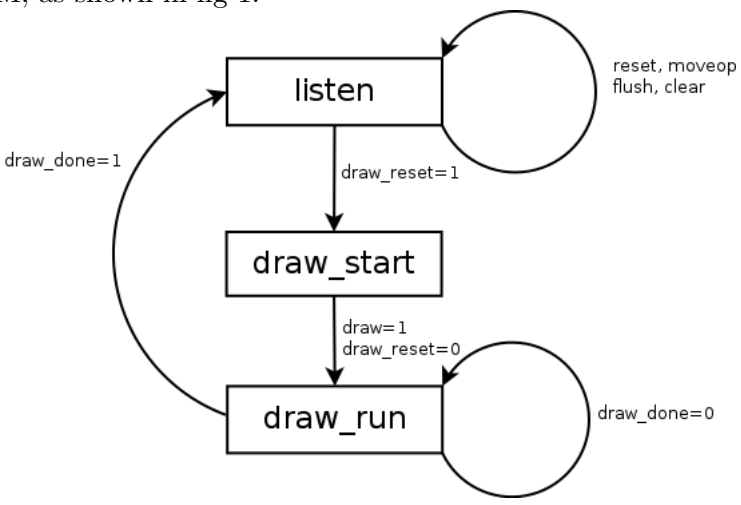
November 25, 2011

1 Introduction

The Draw Block is responsible for taking input commands from the host processor, decoding them and passing the pixel operations to the RAM Control Block. It is comprised of three VHDL entities: *draw_block*, *draw_any_octant* and *draw_octant*.

2 draw_block

This is the top entity for the draw block and supports interfaces for the host and the RCB. The main job of this block is to decode the host commands and provide pixel operations to the RCB. It is implemented as a 3 state FSM, as shown in fig 1.



In order to calculate the draw operations, the draw block creates a wrapper for the *draw_any_octant* entity from exercise 4.

2.1 SIGS: PROCESS

This combinational process decodes the *hdb* signal from the host processor into its more useful component parts: *xin*, *yin*, *pen* and *op*.

2.2 OCT: PROCESS

This process compares the x and y input positions to the pen position to work out which octant we want to draw in. It is used by the *draw_any_octant* entity.

2.3 STATECOMB: PROCESS

STATECOMB manages the state of the FSM. It also implements the Move Pen, Clear and Flush operations as they only take 1 cycle. For the draw command, it sets up the *draw_any_octant module* with the start position (by holding the *draw_reset* signal high) and passes to the *draw_start* state.

2.4 FSM: PROCESS

This clocked process increments the state with the *nstate* signal on the rising edge of the clock. There are two cases when this does not happen: when *delaycmd* is high or when *reset* is high. The *delaycmd* signal is set by the RCB and stops state progression. The *reset* signal returns the state to *listen*.

3 draw_any_octant - Exercise 4

This the *draw_any_octant* entity from exercise 4. It's job is to translate the input/output to *draw_octant* so that it can draw in any of the 8 octants: NNE, ENE, ESE, SSE, SSW, WSW, WNW and NNW.

It has been modified slightly to allow halting whilst the RCB is busy. This is achieved by stopping the clocked process whilst *delay* is high.

4 draw_octant - Exercise 1

This is the *draw_octant* entity from exercise 1. It has been modified slightly to allow halting whilst the RCB is busy. This is achieved by stopping the clocked process whilst *delay* is high.