VHDL Final Report

Group number: 15

Name: Hao Ding

Partner: Liyangyi Yu

Content

1. Introduction
2. Design and Analysis
   2.1 Functions and Principles
   2.2 Features and Performance
   2.3 Possible Improvements
3. Experience and Summary
4. Appendix
   4.1 Appendix I, submitted fast-version RCB
   4.2 Appendix II, slow version RCB
5. Teamwork statement

1. Introduction

RCB block is written by me in this team project. Before complete design deadline, I managed to accomplish a slow-version RCB with clear screen function and a fast-version RCB without clear-screen. The latter one is tested with corner cases, pre and post synthesis simulation and is considered as a fully working code, but still have space to improve if given more time. This report explains basic working principles of RCB block, features and possible enhancement to implement in the future.

2. Design and Analysis

2.1 Functions and principles

Thanks to the sample RCB block design provided by Dr Tom Clarke, which is shown in Figure 1, my design is mostly based on it. To simplify design process, I added some blocks which detect command and signal change. These will be explained in detail later. Also, since clear-screen block is not included in my submitted code, so MUX and other command designed for clear-screen is eliminated, but skeleton is remained since my submitted code is modified from my slow-version code with clear-screen function.

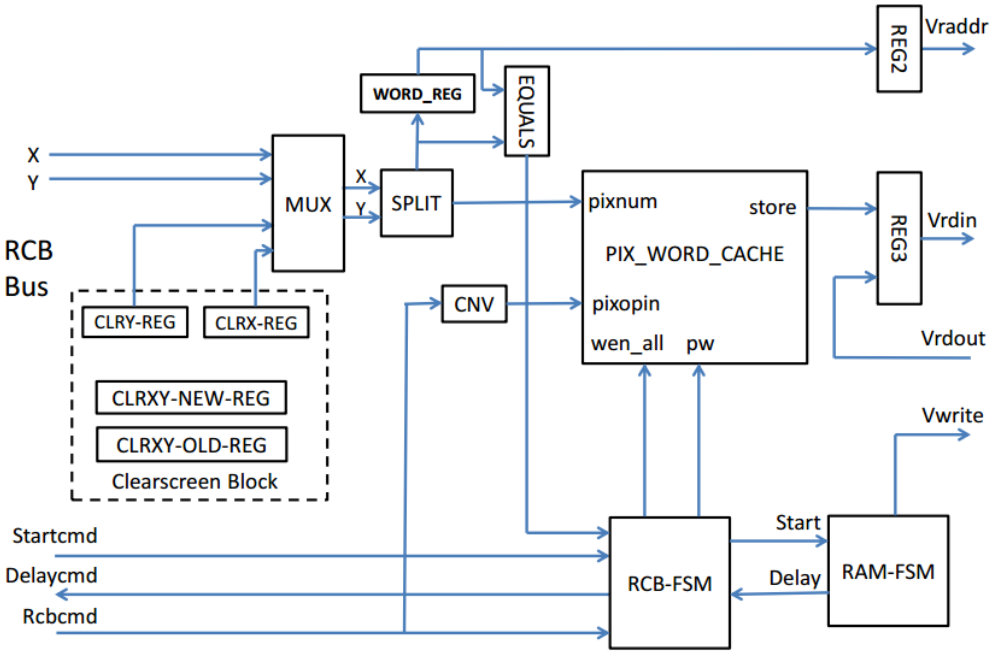

Figure 1. Sample RCB block diagram

With respect to the slow-version, which is included in Appendix II for you interest, working principle is not efficient enough so will only introduced briefly here.

Each pixel is processed individually, that is to say, for each pixel, finite state machine goes through idle, read, modify and write states. The advantage is this algorithm is that it is easy to implement and debug since every pixel is following exactly same process. The inefficient clear-screen function which output is pixel by pixel is also included. However, this RCB block is inevitably slow. So I move on to the next algorithm.

The fast-version is efficient since output is written in word, rather than in pixel. This requires RAM finite state machine (RAM_FSM) to loop inside modify state until pixel work number is changed. RCB finite state machine (RCB_FSM), hence, is designed to looping in the state which giving command to pix_word_cache in order to cooperate with RAM_FSM.

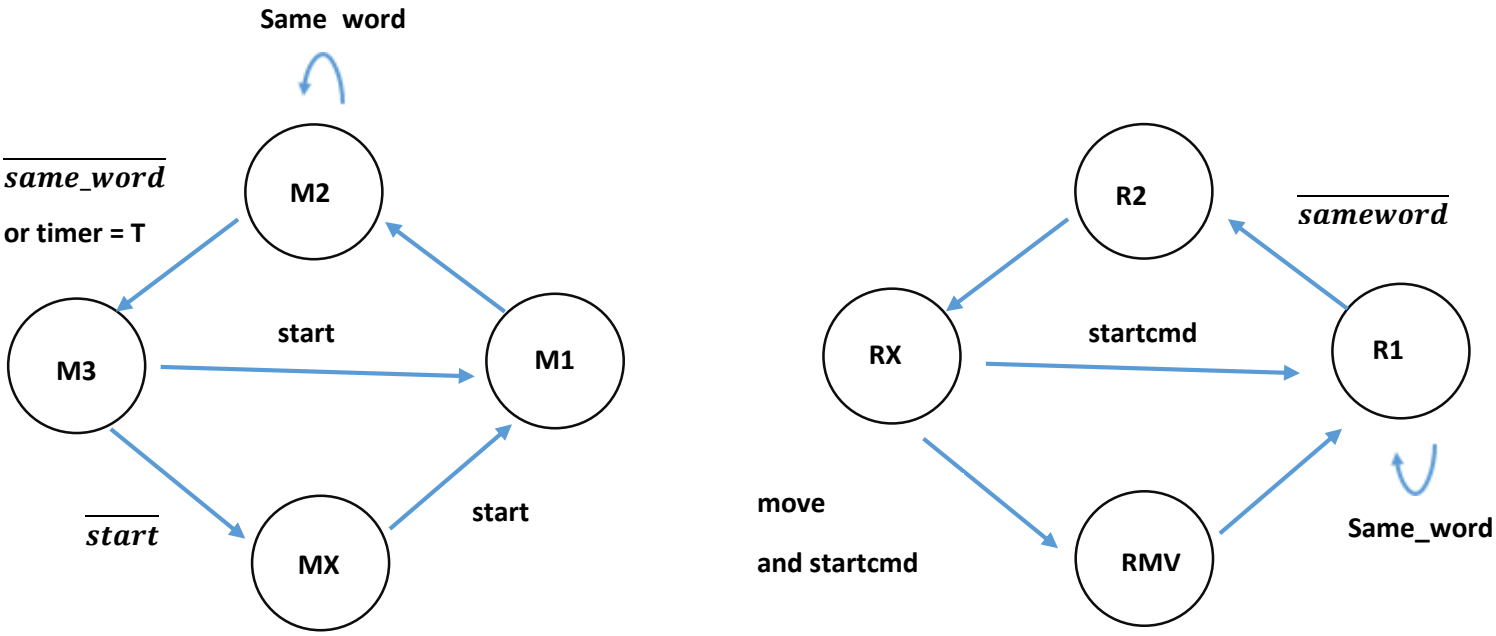

Figure 2. (LEFT) RAM_FSM, (RIGHT) RCB_FSM

State diagrams of RCB_RSM and RAM_FSM are shown above. In RAM_FSM, design is similar to original design which follows the RMW pattern, except branches and conditions using same_word, which detects whether word number changes. RAM_FSM controls the reading and writing time of RAM and pix_word_cache. Writing to RAM is executed in M3.

RCB_FSM is completely designed by myself, which cooperate with RAM_FSM, generate command for pix_word_cache and produce one cycle delay (RMV is specifically designed for clear_screen, so it can be removed in this RCB without clear_screen). In R1, commands to pix_word_cache is always updated to keep generating the write cache output.

Both finite state machine contributes to the interface with db block. Each of them has a delay signal to show system is busy or not.

Detailed function explanation is listed in the following (excluding RAM_FSM and RCB_FSM):

pix_word_cache: This part is consist of change, store_ram and some signal assignement. This is similar to the one in coursework. The difference is that I only allow cache to be written when not in M3 (write state) and startcmd is high. It makes sense since otherwise some useless command will be written and cause wrong result.

split: It splits input vector into pixel number and word number and send them to corresponding blocks.

word_reg: It tracks the previous word number for detect whether word number changes.

equals: It follows word_reg and produce same_word, which is an indicator to show whether word number changes.

cmd_reg_i: It tracks the previous command.

cmd_same_i: It generates an indicator to show whether command type changes. This signal is particular designed for interface with my partner's db block since signal come out from his db block is slightly different from behaviour file.

cmd_type: It detects type of command. This is useful when clear_screen implemented, but here it is only used to detect move signal.

cnv: It converts incoming command to the recognisable command to pix_word_cache

Delay_generation: It is designed specifically for my partner's db, since he use my delay signal in a different way from behaviour file. The specific part required is when startcmd is low, delaycmd must be low, otherwise no signal would come in. But my code works fine with behaviour file without this statement.

reg3_pro: It uses some functions to generate right ram output depending on pix_word_cache command and RAM current pixel colours.

rcbclear: It tells db whether rcb has clear-screen function

port mapping: signal flow to select input from dbb bus.

2.2 Features and Performance

In the rcb block, generic vsize is kept. However, it is not specifically useful since input and output length of RAM is fixed.

In reg3_pro, I use some functions written by me in project_pack to convert data type between pix_word_cache output command and standard logic vector. Then the standard logic vector is compared with RAM input vector and generate the right output using logic gates. Functions I used here are pixop2slv and store2slv. There are some functions written by me but not used in our final design.

Regarding speed performance, my partner's db block is sufficiently fast, which takes around same time as behaviour block. My rcb block use around twice the running time of rcb behaviour block, depending on command type since the clear screen function is not optimised. If we put out code together, we require 1045 clock cycles to finish drawing the sample test code. My algorithm has maximised the efficiency when writing in the same word, which takes one cycle for each pixel to be modified.it can be improved in the following way.

2.3 Possible Improvements

The first possible improvement is about clear-screen function. Due to the very limited time, clear-screen function is not implemented in an inefficient way in db block. However, in rcb block, clear-screen can be done block by block, word number and command can be generated by calculating two sets of coordinates. The challenge should be FSM design and the sub-blocks then clear-screen area covers part of a single word, which consists of 16 pixels.

The second possible improvement is about RMV. In our design, RMV state is not useful since move command is designed for clear-screen function.

The third possible improvement is using array of drawing units and pix_word_cache blocks.

3. Experience and Summary

Thanks to the help from Dr Tom Clarke and my partner, I completed this coursework in time. This is an interesting coursework. Apart from the VHDL knowledge, I also practice debugging skills and logic reasoning skills. There are many ways to finish this coursework and even many methods to fix a bug. However, there are also good methods and bad methods, sometimes I can fix bugs by guessing or using some strange method I cannot understand. But it feels disappointing even the issue is solved. After that, I try to keep my mind clear about the practical meaning of each signal and each block during debugging. It helps understanding the general working mechanism of the block. Focusing too much on tiny details sometimes is not a good idea.

For this project to be accomplished, teamwork is inevitable. Our code never works first time when we put them together since the command coming from db is not exactly same as behaviour file. Sometimes my partner thinks in a different way as me. For example, in the very beginning my partner always send startcmd in the next cycle after draw command, which causes many trouble for my block. And in the complete design submission, command from db still changes when startcmd is low. But we communicate well to design a interface and find a way to compromise both his db block and db_behav_new.

4. Appendix

4.1 Appendix I, submitted fast-version RCB

```
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.project_pack.ALL;

ENTITY rcb IS
	GENERIC(vsize : INTEGER := 6);
	PORT(
		clk		: IN  std_logic;
		reset		: IN  std_logic;

		-- db connections
		dbb		: IN db_2_rcb;
		dbb_delaycmd : OUT STD_LOGIC;
		dbb_rcbclear : OUT STD_LOGIC;

		-- vram connections
		vdout		: IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
		vdin		: OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
		vwrite		: OUT STD_LOGIC;
		vaddr		: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);

		-- vdp connection
		rcb_finish	: OUT STD_LOGIC
	);
END rcb;

ARCHITECTURE rtl1 OF rcb IS
--port match
		SIGNAL x		:  std_logic_vector(VSIZE-1 DOWNTO 0);
		SIGNAL y		:  std_logic_vector(VSIZE-1 DOWNTO 0);
		SIGNAL rcbcmd		:  std_logic_vector(2 DOWNTO 0);
		SIGNAL startcmd		:  std_logic;
--pix_word_cache
	SIGNAL pw, wen_all				:  std_logic;
	SIGNAL pixnum			:  std_logic_vector(3 DOWNTO 0);
	SIGNAL pixopin			:  pixop_t;
	SIGNAL opout,opram		:     pixop_t;
		SIGNAL rdout_par		:      store_t;
--ram_fsm
	SIGNAL start				:     std_logic;
		SIGNAL delay			:     std_logic;
		TYPE   state_t IS (m3, m2, m1, mx); --FSM states
		SIGNAL state, nstate : state_t;
--split
		SIGNAL xin,yin				:
std_logic_vector(VSIZE-1 DOWNTO 0);
		SIGNAL word_num_new			:      std_logic_vector(7
DOWNTO 0);
--word_reg
		SIGNAL word_num_old			:      std_logic_vector(7
DOWNTO 0);
--equals
```

```
            SIGNAL same_word                        :       std_logic;  --check
whether word number is same
--rcb_fsm
            TYPE   rstate_t IS (r2, r1,rx,rmv); --rcb_fsm states
            SIGNAL rstate,nrstate                   :           rstate_t;
            SIGNAL timer                            :
      std_logic_vector(4 DOWNTO 0);--terminate the process at the end of
program
            SIGNAL cmd_slv                                  :
      std_logic_vector(31 DOWNTO 0);--slv form of commands from pix_word_cache
            SIGNAL ram_in_slv                       :
      std_logic_vector(15 DOWNTO 0);--correct value of ram after drawing
            SIGNAL p1,p2                            :
      std_logic_vector(15 DOWNTO 0);--intermediate signal for generating
correct value of ram
            SIGNAL delaycmd_i                       :           std_logic;--
delay command request for clearing screen
--detect cmd is about draw or clear '1' means draw, '0' means clear
            SIGNAL move                                     :
      std_logic;

--indicator
            SIGNAL cmd_same                                 :
      std_logic;
            SIGNAL cmd_reg                                  :
      std_logic_vector(1 DOWNTO 0);
            SIGNAL delaycmd                                 :       std_logic;
BEGIN

--pix_word_cache design starts here, similar to previous exercises, but is_same
signal is removed. For details please see CW4---------------------------------
--------------------------------------------------------------------
change: PROCESS(pixopin,opram,pw,wen_all)
BEGIN
        opout<=pixopin;
        IF pixopin=pinvert OR pixopin = psame THEN
                IF wen_all='0' THEN
                        IF pixopin=psame THEN
                        opout<=opram;
                        ELSIF  opram=pblack THEN
                        opout<=pwhite;
                        ELSIF  opram=pwhite THEN
                        opout<=pblack;
                        ELSIF  opram=pinvert THEN
                        opout<=psame;
                        END IF;
                ELSIF pw='0' THEN
                        opout<=psame;
                END IF;
        END IF;
END PROCESS change;

--ram command generator of pix_word_cache
store_ram: PROCESS
BEGIN
        WAIT UNTIL clk'EVENT AND clk='1';
        IF reset='1'THEN
        rdout_par<=(OTHERS=>psame);
```

```
        ELSIF state=m3 or startcmd='0' THEN--If next cycle is m3 or startcmd is
low, no need to write result.
        ELSIF wen_all='1'AND  pw='1' THEN
        rdout_par<=(OTHERS=>psame);
        rdout_par(to_integer(unsigned(pixnum)))<=opout;
        ELSIF wen_all='1'THEN
        rdout_par<=(OTHERS=>psame);
        ELSIF pw='1' THEN
        rdout_par(to_integer(unsigned(pixnum)))<=opout;
        END IF;
        IF same_word='0'THEN
             rdout_par<=(OTHERS=>psame);
        END IF;
END PROCESS store_ram;
opram<=rdout_par(to_integer(unsigned(pixnum)));      --tracks the latest command
cache

--pix_word_cache ends here-----------------------------------------------------
-------------------------------------------------------------------

--ram_fsm design. Same as exercises. Details explained in CW3------------------
-------------------------------------------------------------------------------
-----
--The only difference is commented in the following
ram_fsm: PROCESS(state, start,same_word,timer)
       BEGIN
              nstate<=state;
              delay<='0';
              vwrite<='0';
              CASE state IS
              WHEN mx=>
                     IF start='1' THEN
                            nstate<=m1;
                     END IF;
              WHEN m1=>
                     nstate<=m2;--delay in m1 is no longer needed
              WHEN m2=>
                     nstate<=m3;
                     IF start='1' THEN
                            delay<='1';
                     END IF;
                     IF same_word='1'THEN
                            nstate<=m2;
                     END IF;
                     IF timer="00100"THEN--output the final value before program
terminates
                            nstate<=m3;
                     END IF;
              WHEN m3=>
                     vwrite<='1';
                     IF start='1' THEN
                            nstate<=m1;
                     ELSE
                            nstate<=mx;
                     END IF;
              END CASE;
END PROCESS ram_fsm;

--Positively triggered process to change state and reset
```

```
clk_pos:     PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='1';
       state<=nstate;
       IF reset='1' THEN
              state<=mx;
       END IF;
END PROCESS clk_pos;

--Negatively triggered process to pass address and data
clk_neg:     PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='0';
       IF state/=m3 THEN
       vdin<=ram_in_slv;
       END IF;
       IF state=mx THEN--start new word reading
       vaddr<=word_num_new;
       END IF;
END PROCESS clk_neg;
cmd_slv<=store2slv(rdout_par);
--RAM_FAM ends here-----------------------------------------------------------
-------------------------------------------------------------

--This block splits x&y into pixel number and word number
split: BLOCK
BEGIN
       word_num_new(3 DOWNTO 0)<=xin(5 DOWNTO 2);--First four digits are word
number. (64 * 64 words in total)
       word_num_new(7 DOWNTO 4)<=yin(5 DOWNTO 2);
       pixnum(1 DOWNTO 0)<=xin(1 DOWNTO 0);--Last two digits are pixel number.
(4*4 pixel in each word)
       pixnum(3 DOWNTO 2)<=yin(1 DOWNTO 0);
END BLOCK split;

--This block stores the previous word number for comparison
word_reg: PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='1';
       word_num_old<=word_num_new;
END PROCESS word_reg;

--Following word_reg, this block checks whether word location changes, and set
same_word to 0 if so.
equals: PROCESS(word_num_old,word_num_new,startcmd,cmd_same)
BEGIN
       same_word<='1';
       IF word_num_old /= word_num_new and startcmd='1' THEN      --This is only
valid when there is command coming in. This is necessary in our code design,
              same_word<='0';
       -- since command coming from db block changes even when startcmd is low,
which causes confusion of my rcb block.
       END IF;
       IF cmd_same='0' THEN
              same_word<='0';
       END IF;
END PROCESS equals;

--This block stores the previous command for comparison
```

```
cmd_reg_i: PROCESS
BEGIN
	WAIT UNTIL CLK'EVENT and CLK='1';
	cmd_reg<=rcbcmd(1 DOWNTO 0);
END PROCESS cmd_reg_i;

--Following cmd_reg_i, this block assigns value to the indicator showing whether
command changes.
cmd_same_i:PROCESS(rcbcmd,cmd_reg)
BEGIN
	cmd_same<='1';
	IF rcbcmd (1 DOWNTO 0)/=cmd_reg THEN
		cmd_same<='0';
	END IF;
END PROCESS cmd_same_i;

--Originally here is a multiplexer to choose coordinates(either from db or from
clear screen block).
--In this version, clear_block is implemented in db_block, so MUX block is no
longer needed.
xin<=x;
yin<=y;

--detect type of command. Either move or not. In the beginning this process is
designed to check clear_screen command. Here this process is simplified
cmd_type:PROCESS
BEGIN
	WAIT UNTIL clk'EVENT and clk='1';
		move<='0';
		IF rcbcmd="000"THEN
			move<='1';
		END IF;
END PROCESS cmd_type;

--generate colour input for pix_word_cache, which detect colour of output from
rcbcmd. Only the last two digits determines colour.
cnv: PROCESS(rcbcmd)
BEGIN
	IF rcbcmd(1)='1' THEN
		pixopin<=pblack;
		IF rcbcmd(0)='1'THEN
			pixopin<=pinvert;
		END IF;
	ELSE
		pixopin<=psame;
		IF rcbcmd(0) ='1'THEN
			pixopin<=pwhite;
		END IF;
	END IF;
END PROCESS cnv;

--FSM for rcb controller, which is similar to RAM_FSM but with extra outputs----
-------------------------------------------------------------------------------
--------------------
rcb_fsm: Process(rstate,move,startcmd,same_word)
BEGIN
	nrstate<=rstate;
	delaycmd_i<='0';
	start<='0';
```

```
        wen_all<='0';
        pw<='1';
        CASE rstate IS
        WHEN rmv=>
                start<='1';
                nrstate<=r1;
                wen_all<='0';
                pw<='1';
        WHEN rx=>
                pw<='0';
                IF startcmd='1' THEN
                        nrstate<=r1;
                        start<='1';
                        delaycmd_i<='1';
                END IF;
                IF move='1'and startcmd='1' THEN
                        nrstate<=rmv;
                END IF;
        WHEN r1=>
                wen_all<='0';
                pw<='1';
                nrstate<=r2;
                delaycmd_i<='1';
                IF same_word ='1'THEN
                        nrstate<=r1;
                        delaycmd_i<='0';
                END IF;
        WHEN r2=>
                nrstate<=rx;
                delaycmd_i<='1';
        END CASE;
END PROCESS rcb_fsm;

--interface specifically designed for our db block. When system is busy and
there is command coming in, set delaycmd to busy
Delay_Generation : PROCESS(startcmd, delaycmd_i, delay)
BEGIN
IF startcmd = '0' THEN
  delaycmd <='0';
ELSE
  delaycmd<=delaycmd_i or delay;--split delaycmd to delaycmd_i (for potential
clear_screen command) and delay (for RAM FSM)
END IF;
END PROCESS Delay_Generation;

rcb_clk_pos: PROCESS
BEGIN
        WAIT UNTIL clk'EVENT and CLK='1';
        rcb_finish<='0';
        rstate<=nrstate;
        IF reset='1' THEN
                rstate<=rx;
        END IF;
        IF timer="00000"THEN
                rcb_finish<='1';
        END IF;
        IF rstate=r2 THEN --reset timer to MAX if write executed.
                timer<="11111";
        ELSIf timer/="00000" THEN
```

```
            timer<=std_logic_vector(unsigned(timer)-1);--timer will stop is
write is not executed within 32 clock cycles.
      END IF;
END PROCESS rcb_clk_pos;
dbb_delaycmd<=delaycmd;
--RAM_FSM ENDS HERE-----------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------

--this process compare command with value stored in RAM to produce output value.
The logics come from a true table about command, input pixel colour and output
pixel colour.
reg3_pro: PROCESS(cmd_slv,vdout)
BEGIN
      FOR i in 0 to 15 LOOP
            p1(i)<=cmd_slv(2*i+1) and (not vdout(i)) ;
            p2(i)<=vdout(i) and (not cmd_slv(2*i)) ;
      END LOOP;
END PROCESS reg3_pro;
ram_in_slv<= p1 or p2;

--rcbclear function setting
dbb_rcbclear<='0';

--port matching
x<=dbb.x;
y<=dbb.y;
startcmd<=dbb.startcmd;
rcbcmd<=dbb.rcb_cmd;

END rtl1;
```

4.2 Appendix II, slow version RCB

Note that port is required to be modified slightly to connect dbb

```
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.project_pack.ALL;

ENTITY rcb IS
      GENERIC(vsize : INTEGER := 6);
      PORT(
            clk         : IN  std_logic;
            reset       : IN  std_logic;
```

```
            -- db connections
            x            : IN std_logic_vector(VSIZE-1 DOWNTO 0);
            y            : IN std_logic_vector(VSIZE-1 DOWNTO 0);
            rcbcmd       : IN std_logic_vector(2 DOWNTO 0);
            startcmd     : IN std_logic;
            dbb_delaycmd : OUT STD_LOGIC;
            dbb_rcbclear : OUT STD_LOGIC;

            -- vram connections
            vdout        : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
            vdin         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
            vwrite       : OUT STD_LOGIC;
            vaddr        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);

            -- vdp connection
            rcb_finish   : OUT STD_LOGIC
      );
END rcb;

ARCHITECTURE rtl1 OF rcb IS
--pix_word_cache
        SIGNAL pw, wen_all                          :  std_logic;
        SIGNAL pixnum                  :  std_logic_vector(3 DOWNTO 0);
        SIGNAL pixopin                 :  pixop_t;
        --SIGNAL store                      :  store_t;
        SIGNAL is_same                 :  std_logic;

        SIGNAL opout,opram             :     pixop_t;
             SIGNAL rdout_par               :      store_t;
             CONSTANT all_same              :      store_t :=(OTHERS=>psame);
             SIGNAL vdout_par                      :     store_t;
--ram_fsm
        SIGNAL start                               :     std_logic;
        --SIGNAL data                              :
      std_logic_vector;--Vrdin and Vraddr
             SIGNAL delay                          :     std_logic;
             --SIGNAL addr,addr_del,data_del       :     std_logic_vector;

             TYPE   state_t IS (m3, m2, m1, mx);
             SIGNAL state, nstate : state_t;

--split
             SIGNAL xin,yin                                :
std_logic_vector(VSIZE-1 DOWNTO 0);
             SIGNAL word_num_new                   :      std_logic_vector(7
DOWNTO 0);

--word_reg
             SIGNAL word_num_old                   :      std_logic_vector(7
DOWNTO 0);

--equals
             SIGNAL same_word                      :      std_logic;

--mux


--clear
```

```
            SIGNAL clrx_reg,clry_reg         :       std_logic_vector(VSIZE-1
DOWNTO 0);
            SIGNAL clrxy_new_reg,clrxy_old_reg            :
std_logic_vector(2*VSIZE-1 DOWNTO 0);
            SIGNAL delaycmd                               :      std_logic;
            SIGNAL scan_done                        :             std_logic;

--rcb_fsm
            TYPE   rstate_t IS (r3,r2, r1,rx);
            SIGNAL rstate,nrstate                   :             rstate_t;
            SIGNAL timer                            :
      std_logic_vector(3 DOWNTO 0);
            SIGNAL cmd_slv                                :
      std_logic_vector(31 DOWNTO 0);
            SIGNAL ram_in_slv                       :
      std_logic_vector(15 DOWNTO 0);
            SIGNAL p1,p2                            :
      std_logic_vector(15 DOWNTO 0);
            SIGNAL delaycmd_i                       :             std_logic;

--detect cmd is about draw or clear '1' means draw, '0' means clear
            SIGNAL draw_or_clear                    :             std_logic;
            SIGNAL draw_or_clear_i                  :             std_logic;
--ram_clear
            SIGNAL is_clear                               :
      std_logic;
BEGIN




--pix_word_cache:

--module named change
change: PROCESS(pixopin,opram,pw,wen_all)
BEGIN
        opout<=pixopin;
        IF pixopin=pinvert OR pixopin = psame THEN
                IF wen_all='0' THEN
                        IF pixopin=psame THEN
                        opout<=opram;
                        ELSIF  opram=pblack THEN
                        opout<=pwhite;
                        ELSIF  opram=pwhite THEN
                        opout<=pblack;
                        ELSIF  opram=pinvert THEN
                        opout<=psame;
                        END IF;
                        ELSIF pw='0' THEN
                        opout<=psame;
                END IF;
        END IF;
END PROCESS change;

--ram implementation
store_ram: PROCESS
BEGIN
        WAIT UNTIL clk'EVENT AND clk='1';
```

```
        IF reset='1'THEN
        rdout_par<=(OTHERS=>psame);
        ELSIF wen_all='1'AND  pw='1' THEN
        rdout_par<=(OTHERS=>psame);
        rdout_par(to_integer(unsigned(pixnum)))<=opout;
        ELSIF wen_all='1'THEN
        rdout_par<=(OTHERS=>psame);
        ELSIF pw='1' THEN
                rdout_par(to_integer(unsigned(pixnum)))<=opout;
                IF opout=psame THEN
                END IF;
        END IF;
        IF same_word='0'THEN
             rdout_par<=(OTHERS=>psame);
        END IF;
END PROCESS store_ram;

--combinational logic for is_same, opram and store
same: PROCESS(rdout_par)
BEGIN
        is_same<='0';
        IF rdout_par = all_same THEN
        is_same<='1';
END IF;
END PROCESS same;

opram<=rdout_par(to_integer(unsigned(pixnum)));


--ram_fsm:

ram_fsm: PROCESS(state, start)
       BEGIN
              nstate<=state;
              delay<='0';
              vwrite<='0';
              CASE state IS
              WHEN mx=>
                     IF start='1' THEN
                            nstate<=m1;
                     END IF;
              WHEN m1=>
                     nstate<=m2;
                     IF start='1' THEN
                            delay<='1';
                     END IF;
              WHEN m2=>
                     nstate<=m3;
                     IF start='1' THEN
                            delay<='1';
                     END IF;
              WHEN m3=>
                     vwrite<='1';
                     IF start='1' THEN
                            nstate<=m1;
                     ELSE
                            nstate<=mx;
                     END IF;
              END CASE;
```

```
END PROCESS ram_fsm;

--Positively triggered process to change state and reset
clk_pos:     PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='1';
       state<=nstate;

       IF reset='1' THEN
              state<=mx;
       END IF;
END PROCESS clk_pos;

--Negatively triggered process to pass address and data
clk_neg:     PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='0';
       vaddr<=word_num_new;
       vdin<=ram_in_slv;
END PROCESS clk_neg;
cmd_slv<=store2slv(rdout_par);

split: BLOCK
BEGIN
       word_num_new(3 DOWNTO 0)<=xin(5 DOWNTO 2);
       word_num_new(7 DOWNTO 4)<=yin(5 DOWNTO 2);
       pixnum(1 DOWNTO 0)<=xin(1 DOWNTO 0);
       pixnum(3 DOWNTO 2)<=yin(1 DOWNTO 0);
END BLOCK split;


word_reg: PROCESS
BEGIN
       WAIT UNTIL clk'EVENT and CLK='1'; --positive clock triggered
       word_num_old<=word_num_new;
END PROCESS word_reg;


equals: PROCESS(word_num_old,word_num_new)
BEGIN
       same_word<='1';
       IF word_num_old /= word_num_new and startcmd='1' THEN
              same_word<='0';
       END IF;
END PROCESS equals;

clear: PROCESS--performs raster scan
BEGIN
       WAIT UNTIL clk'EVENT and CLK='1'; --positive clock triggered
       IF reset='1'THEN
              draw_or_clear_i<='1';
              scan_done<='1';
       END IF;
       clrx_reg<=clrx_reg;
       clry_reg<=clry_reg;
       scan_done<=scan_done;
```

```
	--swap to right orientation
	--IF clrxy_new_reg(2*VSIZE-1 DOWNTO VSIZE)<clrxy_old_reg(2*VSIZE-1 DOWNTO
VSIZE) AND clrxy_new_reg(VSIZE-1 DOWNTO 0) < clrxy_old_reg(VSIZE-1 DOWNTO 0)
THEN
	--	clrxy_old_reg<=clrxy_new_reg;
	--	clrxy_new_reg<=clrxy_old_reg;
	--END IF;

	IF delay='0' and rstate=rx THEN
		scan_done<='0';
		IF clrx_reg=clrxy_new_reg(2*VSIZE-1 DOWNTO VSIZE)THEN
			IF clrxy_new_reg(2*VSIZE-1 DOWNTO
VSIZE)<clrxy_old_reg(2*VSIZE-1 DOWNTO VSIZE) THEN
				clry_reg<=std_logic_vector(signed(clry_reg)-1);
			ELSE
				clry_reg<=std_logic_vector(signed(clry_reg)+1);
			END IF;
				clrx_reg<=clrxy_old_reg(2*VSIZE-1 DOWNTO VSIZE);
		ELSE
			IF clrxy_new_reg(VSIZE-1 DOWNTO 0) < clrxy_old_reg(VSIZE-1
DOWNTO 0) THEN
				clrx_reg<=std_logic_vector(signed(clrx_reg)-1);
			ELSE
				clrx_reg<=std_logic_vector(signed(clrx_reg)+1);
			END IF;
		END IF;
	END IF;

	IF delaycmd='0' THEN
		clrxy_old_reg<=clrxy_new_reg;
	END IF;

	IF clrx_reg=clrxy_new_reg(2*VSIZE-1 DOWNTO VSIZE) and
clry_reg=clrxy_new_reg(VSIZE-1 DOWNTO 0) THEN
		scan_done<='1';
	END IF;

	IF draw_or_clear='1' THEN
		scan_done<='1';
	END IF;

	IF draw_or_clear='0' and draw_or_clear_i='1' THEN
		clrx_reg<=clrxy_new_reg(2*VSIZE-1 DOWNTO VSIZE);
		clry_reg<=clrxy_new_reg(VSIZE-1 DOWNTO 0);
	END IF;

	draw_or_clear_i<=draw_or_clear;
END PROCESS clear;
--assuming when delaycmd is '1', no new command come in.
clrxy_new_reg(2*VSIZE-1 DOWNTO VSIZE)<=x;
clrxy_new_reg(VSIZE-1 DOWNTO 0)<=y;

mux: PROCESS(draw_or_clear, clrx_reg, clry_reg, x, y)
BEGIN
	xin<=x;
	yin<=y;
	IF draw_or_clear='0' THEN
		xin<=clrx_reg;
		yin<=clry_reg;
```

```
	END IF;
END PROCESS mux;

cmd_type:PROCESS(rcbcmd)
BEGIN
	draw_or_clear<='1';
	IF rcbcmd(2)='1' OR rcbcmd="000" THEN
		draw_or_clear<='0';
	END IF;
END PROCESS cmd_type;

--generate colour input for pix_word_cache
cnv: PROCESS(rcbcmd)
BEGIN
	IF rcbcmd(1)='1' THEN
		pixopin<=pblack;
		IF rcbcmd(0)='1'THEN
			pixopin<=pinvert;
		END IF;
	ELSE
		pixopin<=psame;
		IF rcbcmd(0) ='1'THEN
			pixopin<=pwhite;
		END IF;
	END IF;
END PROCESS cnv;

rcb_fsm:
Process(draw_or_clear,rstate,startcmd,delay,scan_done,same_word,is_same,rdout_pa
r,is_clear)
BEGIN
	nrstate<=rstate;
	delaycmd_i<='1';
	wen_all<=wen_all;
	pw<=pw;
	start<='0';
	IF is_clear='0'THEN
		start<='1';
	END IF;
	CASE rstate IS
	WHEN rx=>
		IF startcmd='1'THEN
			nrstate<=r1;
			start<='1';
		END IF;
		IF draw_or_clear_i='1' and draw_or_clear='0'THEN
			nrstate<=rx;
			delaycmd_i<='0';
		END IF;
	WHEN r1=>
		wen_all<='0';
		pw<='1';
		nrstate<=r2;
		IF draw_or_clear='0'THEN
			wen_all<='1';
		END IF;
	WHEN r2=>
		wen_all<='1';
		pw<='1';
```

```
                nrstate<=r3;
        WHEN r3=>
                nrstate<=rx;
                delaycmd_i<='0';
                IF draw_or_clear='0'THEN
                        delaycmd_i<='1';
                END IF;
                IF is_clear='1'THEN
                        delaycmd_i<='0';
                END IF;
        END CASE;

END PROCESS rcb_fsm;
delaycmd<=delaycmd_i or delay;--code in running slow, so delay is always low.

rcb_clk_pos: PROCESS
BEGIN
        WAIT UNTIL clk'EVENT and CLK='1';
        rcb_finish<='0';
        rstate<=nrstate;
        IF reset='1' THEN
                rstate<=r1;
        END IF;
        IF timer="0000"THEN
                rcb_finish<='1';
        END IF;
        IF rstate/=rx THEN
                timer<="1111";
        ELSE
                timer<=std_logic_vector(unsigned(timer)-1);
        END IF;
END PROCESS rcb_clk_pos;
dbb_delaycmd<=delaycmd;

--assume draw_or_clear does not change during delay
ram_clear: PROCESS(draw_or_clear, delay, scan_done)
BEGIN
        is_clear<='1';
        IF draw_or_clear='0' THEN
                is_clear<='0';
        END IF;
        IF scan_done='1'and delay='0'THEN
                is_clear<='1';
        END IF;
END PROCESS ram_clear;


reg3_pro: PROCESS(cmd_slv,vdout)
BEGIN
        FOR i in 0 to 15 LOOP
                p1(i)<=cmd_slv(2*i+1) and (not vdout(i)) ;
                p2(i)<=vdout(i) and (not cmd_slv(2*i)) ;
        END LOOP;
END PROCESS reg3_pro;
        ram_in_slv<= p1 or p2;

--rcbclear setting
dbb_rcbclear<='1';
```

```
END rtl1;
```

5. Teamwork statement

I am responsible for RCB block in this project. My partner and I help each other debugging code. We communicate well to complete the interface and I enjoy the team work with my partner.

Signature: