**VHDL Final Report**

**Content**

1. Introduction

2. Design and Analysis

2.1 Functions and principles

2.2 Features

2.3 Possible improvements

3. Appendix

3.1 Appendix I, submitted fast-version RCB

3.2 Appendix II, slow version RCB

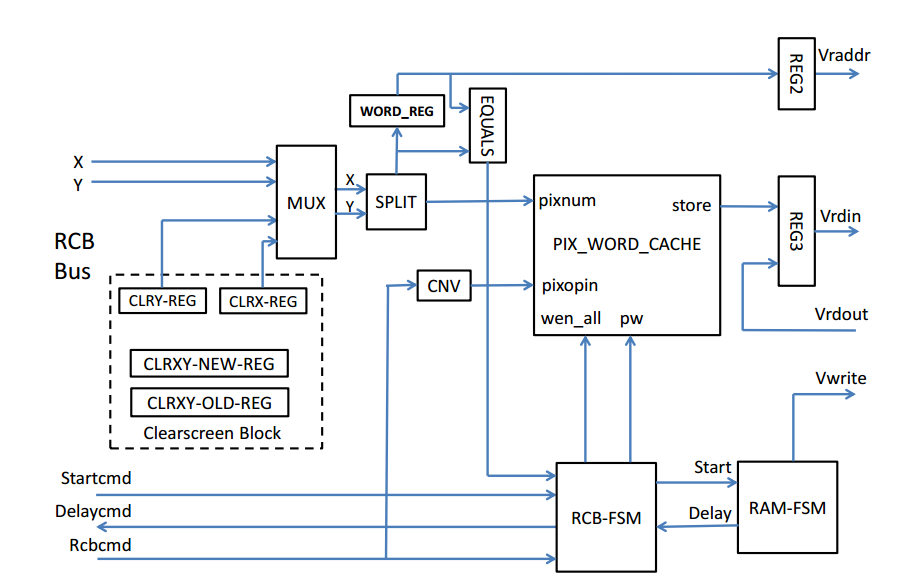
4. Teamwork statement

**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.

**Same\_word**



**or timer = T**

**M2**

**R2**



**startcmd**

**start**

**move**

**and startcmd**

**Figure 2.** (LEFT) RAM\_FSM, (RIGHT) RCB\_FSM

**start**

**RMV**

**M3**

**M1**

**RX**

**R1**

**MX**

**Same\_word**

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**

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.

**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. Appendix**

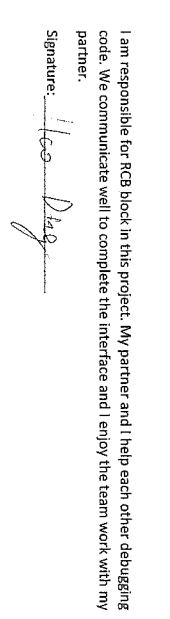
**3.1 Appendix I, submitted fast-version RCB**

|  |
| --- |
| **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; |

**3.2 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\_par,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; |

**4. Teamwork statement**