1 Code Appendix

1.1 rcb.vhd

```
------------------------mux2to1----------------------------
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;

ENTITY mux2 IS
       PORT (i0,i1,sel: IN std_logic;
                       m: OUT std_logic);
END mux2;

ARCHITECTURE dflow OF mux2 IS
       SIGNAL x,y :std_logic;
BEGIN
       x <= i1 AND sel;
       y <= i0 AND NOT sel;
       m <= x or y;
END dflow;


-------------------------------main entity---------------
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.project_pack.ALL;
USE WORK.pix_cache_pak.ALL;
USE WORK.helper_funcs.ALL;
USE WORK.ram_fsm;
USE WORK.pix_word_cache;

ENTITY rcb IS
       GENERIC(      vsize : INTEGER := 6;
                            N: INTEGER := 10);
       PORT(
              clk         : IN  std_logic;
              reset       : IN  std_logic;

              -- db connections
              dbb_bus     : IN db_2_rcb; --this is what jake sends me
              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;

--id probably need to decode all the rcb commands
```

```
ARCHITECTURE rtl1 OF rcb IS
---------------Internal Wires--------------------


--For interfacing with ram block
    SIGNAL ram_start, ram_delay, ram_vwrite                                    :
        std_logic;
    SIGNAL ram_addr, ram_addr_del
                : std_logic_vector(7 DOWNTO 0);
    SIGNAL ram_data, ram_data_del
                : std_logic_vector(15 DOWNTO 0);

--For interfacing with pixel cache
        SIGNAL pxcache_wen_all, pxcache_pw
                    : std_logic;
        SIGNAL pxcache_is_same
                        : std_logic;
        SIGNAL pxcache_pixopin
                        : pixop_t;
        SIGNAL pxcache_pixnum
                        : std_logic_vector(3 DOWNTO 0);
        SIGNAL pxcache_store
                        : store_t;

--
--VSIZE is 6 ish...


--RCB state machine signals
        TYPE state_type IS (s_error,s_idle, s_rangecheck, s_draw, s_clear, s_flush,
            s_waitram,s_fetchdraw);
        SIGNAL state, next_state
                        : state_type;
        SIGNAL idle_counter_trig,draw_trig,move_trig                    : std_logic;
        SIGNAL fetch_draw_flag,fetch_draw_trig                                 :
            std_logic;
        --SIGNAL draw_done, move_done
                        : std_logic;

--draw_px process signals
        SIGNAL vram_waddr,curr_vram_word
                    : std_logic_vector(7 DOWNTO 0);
        SIGNAL change_curr_word
                        : std_logic;

--trigger the cache flush
        SIGNAL flush_trig
                                : std_logic;
--waiting for ram_fsm to complete
        SIGNAL vram_done
                                : std_logic;

        --hardcoded width to handle N up to 256
        SIGNAL idle_counter
                        : std_logic_vector(7 DOWNTO 0);
```

```
        SIGNAL one_vector
                  : std_logic_vector(7 DOWNTO 0);


BEGIN


--if clear then a massive square paint from current pen loc to specified location


----------------------state transition matrix-----------------------

        state_transition: PROCESS(state, dbb_bus.startcmd, curr_vram_word,
            draw_trig,vram_done,fetch_draw_flag, idle_counter)
        --idle counter variable declared in package

        BEGIN

                next_state <= s_error;  --default to error state

                --control signal default assignments
                idle_counter_trig <= '0';
                draw_trig <= '0';
                move_trig <= '0';
                flush_trig <= '0';
                fetch_draw_trig <= '0';

                dbb_delaycmd <='1'; --always busy unless in idle state

                --transitions
                CASE state IS
                       WHEN s_idle =>

                                       dbb_delaycmd <='0'; --im free, tell me to do stuff

                                       IF (dbb_bus.startcmd='1') THEN
                                              next_state <= s_rangecheck;

                                       ELSIF (to_integer(unsigned(idle_counter)) = N)
                                           THEN
                                              next_state <= s_flush;

                                       ELSIF (dbb_bus.startcmd ='0') THEN
                                              --increment loop counter
                                              idle_counter_trig <= '1';
                                              next_state <= s_idle;

                                       ELSE
                                              next_state <= s_error;
                                              assert false report "ERROR in rcb,
                                                  state_transition - when s_idle ";
                                       END IF;


                               WHEN s_rangecheck =>
                                       --check if command targets pixel in loaded word
                                       IF ( getRamWord(dbb_bus.X, dbb_bus.Y) =
                                           curr_vram_word ) THEN
```

```
                --instruction decode
                -- RCB CMD
                -- 000 move
                -- 001 draw white                    '-01'
                    if white
                -- 010 draw black                    '-10'
                    if black
                -- 011 draw invert                   '-11'
                    if invert
                -- 100 unused                        '0--'
                    if draw
                -- 101 clear white                   '1--'
                    if clear
                -- 110 clear black                   '000'
                    if move
                -- 111 clear invert

                IF (dbb_bus.rcb_cmd(2) = '0') THEN --draw
                    command issued (or move)
                       next_state <= s_draw;

                ELSIF (dbb_bus.rcb_cmd(2) = '1') THEN
                    --clear command issued
                       next_state <= s_clear;

                ELSE
                       next_state <= s_error;
                       assert false report "ERROR in rcb,
                           when s_rangecheck, instruction
                           decode ";

                END IF;

            ELSE
                   next_state <= s_flush;
            END IF;

WHEN s_draw =>
            --write to single pixel in cache

            IF (dbb_bus.rcb_cmd = "000") THEN
                   move_trig <='1';
          ELSE
              draw_trig <='1';
              END IF;

              IF (draw_trig='1') THEN --while drawing stay in
                  draw state
                     next_state <= s_draw;

              ELSIF (draw_trig='0') THEN --if finished draw go
                  to idle state
                     next_state <= s_idle;

              ELSE
                     next_state <= s_error;
```

```
                        assert false report "ERROR in rcb,
                            state_transition - when s_draw ";
                END IF;


WHEN s_clear => next_state <=s_idle; --to be implemented later

WHEN s_flush =>
                --write cache to vram
                flush_trig <= '1';

                IF vram_done = '1' THEN

                        IF (fetch_draw_flag ='1') THEN
                                next_state <= s_fetchdraw;
                        ELSE
                                next_state <= s_idle;
                        END IF;

                ELSIF vram_done ='0' THEN
                        next_state <= s_waitram;
                ELSE
                        next_state <=s_error;
                        assert false report "ERROR in rcb,
                            state_transition - when s_flush ";
                END IF;

WHEN s_waitram =>
                --stall the fsm
                IF vram_done = '1' THEN

                        IF (fetch_draw_flag ='1') THEN
                                next_state <= s_fetchdraw;
                        ELSE
                                next_state <= s_idle;
                        END IF;

                ELSIF vram_done ='0' THEN
                        next_state <= s_waitram;
                ELSE
                        next_state <= s_error;
                        assert false report "ERROR in rcb,
                            state_transition - when s_waitram ";
                END IF;

WHEN s_fetchdraw =>

                fetch_draw_trig <= '1';
                next_state <= s_idle;


WHEN s_error =>
                assert false report "Congrats, you managed to go
                    to the error state, fix me";
                next_state <= s_error; -- only reset moves state
                    to idle
```

```
                END CASE;
        END PROCESS state_transition;
-------------------------------------------------------------------------------
----------------------------fsm_clocked_process--------------------------------
        fsm_clocked_process: PROCESS
        BEGIN
                WAIT UNTIL clk'EVENT AND clk = '1';

                        --registering this signal
                        IF (state = s_rangecheck) THEN
                                fetch_draw_flag <= '1'; END IF;

                        IF (state = s_fetchdraw) THEN --deasserts when in fetch state
                                fetch_draw_flag <= '0'; END IF;

                        state <= next_state;

                        IF (next_state = s_idle) THEN
                                rcb_finish <= '1';
                        ELSE
                                rcb_finish <= '0';
                        END IF;


                        IF reset = '1' THEN
                                state <= s_idle;

                END IF;


        END PROCESS fsm_clocked_process;
-------------------------------------------------------------------------------
------------------------idle counter reset---------------------------------
idle_counter_proc: PROCESS
BEGIN
    WAIT UNTIL clk'EVENT AND clk= '1';
    one_vector <= "00000001";
    IF (idle_counter_trig ='1') THEN
        idle_counter <= std_logic_vector(unsigned(unsigned(one_vector)+
            unsigned(idle_counter)));
    END IF;

END PROCESS idle_counter_proc;


-------register storing current word to detect if out of range--------------
        current_word_register: PROCESS
        BEGIN
                WAIT UNTIL clk'EVENT AND clk ='1';
                IF (change_curr_word='1') THEN --enable for register
                        curr_vram_word <= vram_waddr; --join this to one of the
                            addresses
                END IF;
        END PROCESS current_word_register;
-------------------------------------------------------------------------------

--------combinatorial process handling the draw connecting to pxwordcache--------
        draw_px: PROCESS(draw_trig, move_trig, dbb_bus, fetch_draw_trig)
```

```
BEGIN

IF (draw_trig ='1' OR move_trig ='1') THEN

        --store vram word address of the current command
        vram_waddr <= getRamWord(dbb_bus.X, dbb_bus.Y);

        --if it was a draw command
        IF (draw_trig ='1') THEN
                --instruction decode
                IF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "01") THEN
                        pxcache_pixopin <= white;

                ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "10") THEN
                        pxcache_pixopin <= black;

                ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "11") THEN
                        pxcache_pixopin <= invert;
                ELSE
                        assert false report "ERROR in rcb, draw_px instruction
                            decode";
                END IF;

                --compute cache bit addresses and bit numbers from x,y
                pxcache_pixnum <= getRamBit(dbb_bus.X, dbb_bus.Y);
                pxcache_wen_all <= '0'; --for writing single px
                pxcache_pw <='1';     --enable the px cache for writing single px

        ELSIF (fetch_draw_trig='1') THEN

        --instruction decode
                IF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "01") THEN
                        pxcache_pixopin <= white;

                ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "10") THEN
                        pxcache_pixopin <= black;

                ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "11") THEN
                        pxcache_pixopin <= invert;
                ELSE
                        assert false report "ERROR in rcb, draw_px instruction
                            decode";
                END IF;

                --compute cache bit addresses and bit numbers from x,y
                pxcache_pixnum <= getRamBit(dbb_bus.X, dbb_bus.Y);
                pxcache_wen_all <= '1'; --for clear cache
                pxcache_pw <='1';     --enable the px cache for writing single px
                change_curr_word <='1'; --enable the register holding the
                    current word to update



        END IF;

        IF (move_trig ='1') THEN
                null;
                --just need to load the new word if that happens
```

```
                --upstream should be saving the previous x,y for
            END IF;

        END IF;

        END PROCESS draw_px;


-----------------fetch from vram and draw processs----------------------------
        --code duplication, temporary solutionn
        -- fetch_draw : PROCESS(fetch_draw_trig, dbb_bus)
        -- BEGIN
        -- IF (fetch_draw_trig = '1') THEN

        --      --draw instruction decode
        --      IF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "01") THEN
        --                      pxcache_pixopin <= white;

        --      ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "10") THEN
        --              pxcache_pixopin <= black;

        --      ELSIF (dbb_bus.rcb_cmd(1 DOWNTO 0) = "11") THEN
        --              pxcache_pixopin <= invert;
        --      ELSE
        --              assert false report "ERROR in rcb, draw_px instruction decode";
        --      END IF;

        --      --compute cache bit addresses and bit numbers from x,y
        --      pxcache_pixnum <= getRamBit(dbb_bus.X, dbb_bus.Y);
        --      pxcache_wen_all <= '1'; --to zero out the other cache bits
        --      pxcache_pw <='1';     --enable the px cache for writing single px
        --      change_curr_word <='1'; --enable the register holding the current word
            to update


        -- END IF;
        -- END PROCESS fetch_draw;

-------------------------------------------------------------------------------

------------------ flush cache out to vram-------------------------------------
        flush_cache : PROCESS(flush_trig,dbb_bus, curr_vram_word, ram_delay)
        BEGIN
        IF flush_trig = '1' THEN

                --if vram not busy
                IF (ram_delay = '0') THEN
                        ram_addr <= curr_vram_word;
                        ram_start <= '1'; --enable vram interfacing

                        --need to modify RAM fsm to take in second input frm cache, and
                        --merge changes with vdout to put on vdin by another truth table
                END IF;

    END IF;
    END PROCESS flush_cache;
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
---------------------------- structural ---------------------------------------
ram_state_machine: ENTITY ram_fsm PORT MAP(

        --inputs std_logic     "entity port => external signal"
        clk             => clk,
        reset   => reset,
        start   => ram_start,

        --input std_logic_vector
        addr    => ram_addr,
        data    => ram_data,
        cache_d => pxcache_store,


        --output std_logic
        delay   => ram_delay,
        vwrite  => ram_vwrite,

        --output std_logc_vector
        addr_del => ram_addr_del,
        data_del => ram_data_del
        );

px_cache: ENTITY pix_word_cache PORT MAP(

        --inputs std_logic
        clk           => clk,
        wen_all => pxcache_wen_all,
        reset  => reset,
        pw            => pxcache_pw,

        --inputs pixop_t
        pixopin => pxcache_pixopin,

        --inputs std_logic_vector(3 DOWNTO 0)
        pixnum => pxcache_pixnum,


        --outputs store_t
        store  => pxcache_store,

        --outputs std_logic
        is_same => pxcache_is_same

        );

------------------external connections and signal stuff----------------------
vram_done <= ram_delay;
vdin <= ram_data_del;
vaddr <= ram_addr; --joining external ram to the ram interface fsm
ram_data <= vdout; --joins vram output to ram data in
vwrite <= ram_vwrite;

--clear not implemented yet
dbb_rcbclear <= '0';
```

```
END rtl1;
```

1.2 helper funcs

```
-------------------------------------------------------------------------------------------------------------
--------------------------------functions package-----------------
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.project_pack.ALL;
USE WORK.pix_cache_pak.ALL;
USE WORK.config_pack.ALL;

PACKAGE helper_funcs IS
    FUNCTION getRamWord(x : std_logic_vector(vsize-1 DOWNTO 0); y
        :std_logic_vector(vsize-1 DOWNTO 0)) RETURN std_logic_vector;
    FUNCTION getRamBit( x : std_logic_vector(vsize-1 DOWNTO 0); y
        :std_logic_vector(vsize-1 DOWNTO 0)) RETURN std_logic_vector;
END;


PACKAGE BODY helper_funcs IS

    -- return the RamWord address as a 8 bit vector
    FUNCTION getRamWord( x : std_logic_vector(vsize-1 DOWNTO 0); y
        :std_logic_vector(vsize-1 DOWNTO 0)) RETURN std_logic_vector IS

      VARIABLE xVal, yVal     : integer;
      VARIABLE wordAddress    : std_logic_vector(7 DOWNTO 0);
    BEGIN

        xVal := to_integer(unsigned(x(VSIZE-1 DOWNTO 2)));
        yVal := to_integer(unsigned(y(VSIZE-1 DOWNTO 2)));

        wordAddress := std_logic_vector(to_unsigned(xVal+ 16*yVal,8));

        RETURN wordAddress;
    END;

    -- return the ramBit addr as a 4bit address vector
    FUNCTION getRamBit( x : std_logic_vector(vsize-1 DOWNTO 0); y
        :std_logic_vector(vsize-1 DOWNTO 0)) RETURN std_logic_vector IS

      VARIABLE xVal, yVal : integer;
      VARIABLE bitAddress : std_logic_vector(3 DOWNTO 0);

    BEGIN

        xVal := to_integer(unsigned(x(1 DOWNTO 0)));
        yVal := to_integer(unsigned(y(1 DOWNTO 0)));

        bitAddress := std_logic_vector(to_unsigned(xVal + 4*yVal,4));

        RETURN bitAddress;
    END;
```

```
END helper_funcs;
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
```

1.3 ram fsm

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.pix_cache_pak.ALL;

-- please insert appropriate entity (see tb instance for port names)

ENTITY ram_fsm IS

    PORT(
           clk, reset, start: IN std_logic;
           delay,vwrite: OUT std_logic;

           addr : IN std_logic_vector;
           data : IN std_logic_vector(15 DOWNTO 0); --changedd from unbounded to
               hardcoded size
           cache_d : IN store_t;
           addr_del, data_del: OUT std_logic_vector
    );
END ram_fsm;

ARCHITECTURE synth OF ram_fsm IS
  TYPE  state_t IS (m3, m2, m1, mx);    --possible states
  SIGNAL state, nstate : state_t;       --current state and next state
  SIGNAL data_merged: std_logic_vector(15 DOWNTO 0);
BEGIN

-----------------implements merge of cache changes and vram data---------

    MERGE_PROC:
    PROCESS(cache_d,data)
    BEGIN
    FOR i IN cache_d'RANGE LOOP
        CASE cache_d(i) IS
            WHEN same  => data_merged(i) <= data(i);
            WHEN white => data_merged(i) <= '0';
            WHEN black => data_merged(i) <= '1';
            WHEN invert => data_merged(i) <= NOT data(i);
            WHEN OTHERS => NULL;
        END CASE;
    END LOOP;
    END PROCESS MERGE_PROC;


-------------------------------------------------------------------------

-----------------implements the state transition matrix-------------------

    STATE_PROC:                 --combinatorial
    PROCESS(state, start)
```

```
    BEGIN
        nstate <= state;       --default value same as current state



                vwrite <='0';
                delay <='0';

        CASE state IS
            WHEN mx =>  IF start ='1' THEN nstate <= m1; END IF;

            WHEN m1 =>  IF start ='1' THEN delay <='1'; END IF;
                                    nstate <= m2;  --unconditional transition

            WHEN m2 =>  IF start ='1' THEN delay <='1'; END IF;
                                nstate <= m3;  --unconditioanl transition

            WHEN m3 => vwrite <= '1';
                                IF start = '1' THEN
                            nstate <= m1;
                        ELSIF start ='0' THEN
                            nstate <= mx; END IF;
        END CASE;
    END PROCESS STATE_PROC;
-------------------------------------------------------------------------------

----------------------------state register---------------------------------
    SS_PROC:
    PROCESS
    BEGIN
        WAIT UNTIL clk'EVENT AND clk = '1';
            state <= nstate;
                IF reset = '1' THEN    --reset signal
            state <= mx;
        END IF;

    END PROCESS SS_PROC;
-------------------------------------------------------------------------------

------------------addr & data delay registers-----------------------------

    DELAY_PROC:
    PROCESS
    BEGIN
        WAIT UNTIL CLK'EVENT AND clk ='0';
            addr_del <= addr;
            data_del <= data;

    END PROCESS DELAY_PROC;



END ARCHITECTURE synth;
```