# Ram Control Block Code Review

Jonathan Ely

November 28, 2011

## 1 Introduction

This report is a code review by Jonathan Ely of the *RCB Block* which was developed by Farhan Rahman.

## 2 Compilation

Farhan's files compile with no warnings or errors in Modelsim, with the compilation set to 1993 VHDL and 'Check for Synthesis enabled'.

## 3 Synthesis

Despite Modelsim showing now complaints, Synplify did come up with a couple of errors and warnings.

### 3.1 Errors

The errors all related to the declaration of the *pix_write_cache* instance in *rcb.vhd*. Here, Farhan forgets to map the *w_size* and *p_size* generics causing width errors since the *w_size* and *p_size* generics declared in *pix_write_cache.vhd* are not the same size as those used in *rcd.vhd*.

### 3.2 Warnings

Synplify displays two warnings from the *rcb.vhd* file.

The first is regarding the OTHERS section of the nextstate CASE statement on line 198. Synplify states "Others clause is not synthesised", this is because there are no other states - all the states in the enumerated type have been accounted for.

The last warning is due to an undriven signal in *rcb.vhd*. *delaycmd1* is not driven at at any point, however it is read from in several IF statements. It seems the purpose of this signal is to be able to read the *delaycmd* signal since VHDL does not allow the reading of an output. To fix this warning line 211 should be changed to drive *delaycmd1*.

## 4 FSMs

Farhan's FSM appears to work and the selection of states is sensible.

## 5 Reset

All the RCB subentities use the same reset signal that is an input into the rcb entity. The FSM implements a scynchronous reset in the FSM process.

## 6 Code Correctness

**Processes with WAIT FOR:** There are none of these.

**Processes with both positive and negative edges:** None.

**FOR or WHILE loops with variable length:** There is one FOR loop in the *vdin_compute* process of *pix_write_cache*. The length of this loop is the range of the *store* array. The size of the array (16) is declared in *pix_cache_pak*. Although it is not variable sized (the size is set at compile time) it may be clear to both the user and compiler if a generic was used and included in the loop declaration.

**Badly formed processes:** All the sensitivity lists were correct. There was one signal *delay-cmd1* that was not correctly driven (as discussed earlier in this report)

**Signals driven from two different processes:** There are no signals that are driven in multiple processes.

## 7 Design Correctness

Except the one signal discussed earlier, all signals seem to be correctly driven and the specification is adhered to. I don't see any major problems with this design.

## 8 Code Style

On the whole Farhan's code is fairly clear and well writen. However it would be nice if there were more comments.

# A RCB Code

## A.1 rcb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.numeric_std.ALL;
4  USE work.pix_word_cache;
5  USE work.ram_fsm;
6  USE work.pix_cache_pak.ALL;
7  USE work.pix_write_cache;
8
9
10 ENTITY rcb IS
11   GENERIC(
12     w_size  : INTEGER := 8;
13     p_size   : INTEGER := 4
14   );
15
16   PORT(
17     clk, reset   : IN  std_logic;
18     x,y           : IN  std_logic_vector(5 DOWNTO 0);
19     rcbcmd       : IN  std_logic_vector(2 DOWNTO 0);
20     startcmd     : IN  std_logic;
21     delaycmd     : OUT std_logic;
22     vaddr        : OUT std_logic_vector(w_size - 1 DOWNTO 0);
23     vdin         : OUT std_logic_vector(w_size - 1 DOWNTO 0);
24     vdout        : IN  std_logic_vector(w_size - 1 DOWNTO 0);
25     vwrite       : OUT std_logic
26   );
27 END ENTITY rcb;
28
29
30 ARCHITECTURE behav OF rcb IS
31   SIGNAL delaycmd1 : std_logic;
32
33   SIGNAL pixword   : std_logic_vector(w_size - 1 DOWNTO 0);
34   SIGNAL pixnum     : std_logic_vector(p_size - 1 DOWNTO 0);
35   SIGNAL pixopin   : pixop_t;
36   SIGNAL clean     : std_logic;
37   SIGNAL ready     : std_logic;
38   SIGNAL empty     : std_logic;
39   SIGNAL store     : store_t;
40   SIGNAL word      : std_logic_vector(w_size - 1 DOWNTO 0);
41
42
43   SIGNAL start   : std_logic;
44   SIGNAL waitx   : std_logic;
45   SIGNAL vwrite1 : std_logic;
46
47   TYPE states IS (s0,s1,s2,s3);
48   SIGNAL state     : states;
49   SIGNAL nstate    : states;
50   SIGNAL flush_cmd : std_logic;
51
```

```
52    ALIAS slv  IS std_logic_vector;
53    ALIAS usg  IS unsigned;
54    ALIAS sg   IS signed;
55
56  BEGIN
57
58  pwordcache : ENTITY pix_word_cache
59    GENERIC MAP(
60       w_size => w_size,
61       p_size => p_size
62    )
63    PORT MAP(
64       clk      => clk,
65       reset    => reset,
66       pw       => startcmd,
67       empty    => empty,
68       pixnum   => pixnum,
69       pixopin  => pixopin,
70       pixword  => pixword,
71       store    => store,
72       word     => word,
73       clean    => clean,
74       ready    => ready
75    );
76
77
78  pwritecache : ENTITY pix_write_cache
79    PORT MAP(
80      clk      => clk,
81      reset    => reset,
82      start     => start,
83      store     => store,
84      address   => word,
85      waitx     => waitx,
86      vwrite   => vwrite1,
87      vdout     => vdout,
88      vdin     => vdin,
89      vaddr    => vaddr
90    );
91
92
93  P1 : PROCESS (x,y,rcbcmd)
94    VARIABLE temp_pixword    : std_logic_vector(w_size - 1 DOWNTO 0);
95    VARIABLE temp_pixnum     : std_logic_vector(p_size - 1 DOWNTO 0);
96    VARIABLE temp_y           : std_logic_vector(w_size - 1 DOWNTO 0);
97    VARIABLE temp_y_pix      : std_logic_vector(p_size - 1 DOWNTO 0);
98    VARIABLE pix_cmd        : std_logic_vector(1 DOWNTO 0);
99  BEGIN
100   temp_y       := slv(resize(sg(y(5 DOWNTO 2)), w_size) sll 4);
101   temp_y_pix   := slv(resize(sg(y(1 DOWNTO 0)), p_size) sll 2);
102
103   temp_pixword := slv(sg(resize(sg(x(5 DOWNTO 2)), w_size)) + sg(temp_y));
104   temp_pixnum  := slv(sg(resize(sg(x(1 DOWNTO 0)), p_size)) + sg(temp_y_pix
        ));
105
```

```
106     pix_cmd := rcbcmd(1) & rcbcmd(0);
107
108     CASE pix_cmd IS
109       WHEN "00" => pixopin <= same;
110       WHEN "01" => pixopin <= white;
111       WHEN "10" => pixopin <= black;
112       WHEN "11" => pixopin <= invert;
113       WHEN OTHERS => NULL;
114     END CASE;
115
116     pixword  <= temp_pixword;
117     pixnum   <= temp_pixnum;
118
119 END PROCESS P1;
120
121 FSM : PROCESS (reset, rcbcmd, ready, state, delaycmd1, vwrite1, startcmd,
        waitx)
122     VARIABLE flush : std_logic;
123     VARIABLE draw  : std_logic;
124     VARIABLE clear : std_logic;
125 BEGIN
126     flush := NOT rcbcmd(2) AND NOT rcbcmd(1) AND NOT rcbcmd(0);
127     clear := rcbcmd(2);
128     draw  := (NOT rcbcmd(2) AND NOT flush) OR (rcbcmd(2) AND NOT flush);
129     flush_cmd <= '0';
130     IF reset = '1' THEN
131       nstate <= s0;
132     ELSE
133       CASE state IS
134         WHEN s0 =>
135
136           IF startcmd = '1' THEN
137
138             IF clear = '1' THEN
139               nstate <= s0; --clearscreen not implemented yet
140             END IF; -- clear = '1'
141
142             IF flush = '1' THEN
143               nstate    <= s1;
144               flush_cmd <= '1';
145             END IF; --flush = '1'
146
147             IF draw = '1' THEN
148               nstate <= s2;
149             END IF; -- draw = '1'
150
151           END IF; --startcmd = '1'
152
153         WHEN s1 =>
154
155           IF waitx = '1' THEN
156             nstate <= s1;
157             flush_cmd <= '1';
158           ELSE -- waitx = '0'
159             nstate <= s3;
```

```
160           END IF; -- waitx = '1'
161
162         WHEN s3 =>
163
164           IF vwrite1 = '1' THEN
165             IF startcmd = '0' THEN
166               nstate <= s0;
167             END IF; --startcmd = '0'
168           ELSE --vwrite1 = '0'
169             nstate <= s3;
170           END IF; --vwrite1 = '1'
171
172           IF startcmd = '1' AND flush = '1' THEN
173             nstate    <= s1;
174             flush_cmd <= '1';
175           END IF; --startcmd = '1' AND flush = '1'
176
177         WHEN s2 =>
178
179           IF ready = '0' THEN
180             nstate <= s1;
181             flush_cmd <= '1';
182           END IF; --ready = '0'
183
184           IF startcmd = '1' THEN
185
186             IF delaycmd1 = '0'  AND draw = '1' THEN
187               nstate <= s2;
188             END IF; --delaycmd = '0'
189
190           ELSE --startcmd = '0'
191
192             IF delaycmd1 = '0' THEN
193               nstate <= s0;
194             END IF; --delaycmd = '0'
195
196           END IF; --startmcd = '1'
197
198         WHEN OTHERS => nstate <= s0;
199       END CASE;
200     END IF;
201 END PROCESS FSM;
202  vwrite    <= vwrite1;
203  delaycmd  <= delaycmd1;
204  C1 : PROCESS
205  BEGIN
206  WAIT UNTIL rising_edge(clk);
207     state <= nstate;
208 END PROCESS C1;
209
210 ----------DATAFLOW STATEMENTS----------
211  delaycmd <= NOT ready AND waitx;
212  empty <= NOT ready OR flush_cmd;
213  start <= NOT ready OR flush_cmd;
214
```

```
215 END ARCHITECTURE behav;
```

### A.2 ram_fsm.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  ENTITY ram_fsm IS
4  PORT(clk, reset, start: IN  std_logic; vwrite, delay: OUT std_logic );
5  END ram_fsm;
6  ARCHITECTURE synth OF ram_fsm IS
7  TYPE   state_t IS (m3, m2, m1, mx);
8  SIGNAL state, nstate  : state_t;
9  SIGNAL delay1         : std_logic;
10 SIGNAL vwrite1        : std_logic;
11 BEGIN
12 C: PROCESS(state, reset, start)
13 BEGIN
14 delay1 <= '0'; vwrite1 <= '0';
15   IF reset = '1' THEN
16     nstate <= mx;
17   ELSE
18
19   CASE state IS
20     WHEN mx =>
21       IF start = '1' THEN
22         nstate <= m1;
23       ELSE
24         nstate <= mx;
25       END IF;
26     WHEN m1 =>
27       IF start = '1' OR start='0' THEN
28         nstate <= m2;
29         delay1 <= start;
30       END IF;
31     WHEN m2 =>
32       IF start='1' OR start='0' THEN
33         nstate <= m3;
34         delay1 <= start;
35       END IF;
36     WHEN m3 =>
37       IF start='1' THEN
38         nstate <= m1;
39         vwrite1 <= '1';
40       ELSE
41         nstate <= mx;
42         vwrite1 <= '1';
43       END IF;
44   END CASE;
45   END IF;
46 END PROCESS C;
47
48 FSM: PROCESS
49 BEGIN
50   WAIT UNTIL clk'EVENT AND clk = '1';
51    state <= nstate;
52 END PROCESS FSM;
53 delay <= delay1; vwrite <= vwrite1;
54 END ARCHITECTURE synth;
```

### A.3 pix_word_cache.vhd

```
1  LIBRARY IEEE;
2  LIBRARY WORK;
3  USE ieee.std_logic_1164.ALL;
4  USE IEEE.numeric_std.ALL;                    -- add unsigned, signed
5  USE work.pix_cache_pak.ALL;
6
7  ENTITY pix_word_cache IS
8    GENERIC(
9     w_size : INTEGER := 4;
10    p_size : INTEGER := 4
11    );
12    PORT(
13       clk, reset, pw, empty : IN  std_logic;
14       pixnum                : IN  std_logic_vector(p_size - 1 DOWNTO 0);
15       pixopin               : IN  pixop_t;
16       pixword               : IN  std_logic_vector(w_size - 1 DOWNTO 0);
17       store                 : OUT store_t;
18       word                  : OUT std_logic_vector(w_size - 1 DOWNTO 0);
19       clean, ready          : OUT std_logic
20       );
21 END pix_word_cache;
22
23 ARCHITECTURE rtl OF pix_word_cache IS
24
25   CONSTANT init_store : store_t := (OTHERS=>same);
26   -- you may find these signals useful, feel free to delete them or add
         others
27   SIGNAL store1 : store_t;
28   SIGNAL clean1 : std_logic;
29   SIGNAL word1  : std_logic_vector(w_size - 1 DOWNTO 0);
30   SIGNAL ready1 : std_logic;
31 BEGIN
32 COMB : PROCESS (reset, store1, pixword, word1, empty, pw, clean1)
33 BEGIN
34 IF store1 = init_store THEN
35    clean1 <= '1';
36 ELSE -- store1 all element not same
37    clean1 <= '0';
38 END IF; -- if all element of store1 = same
39    ready1 <= '1';
40 IF reset = '1' THEN
41    ready1 <= '0';
42 ELSE -- reset = '0'
43
44 IF empty = '0' AND pw = '0' THEN
45
46 IF clean1 = '1' OR pixword = word1 THEN
47    ready1 <= '1';
48 END IF; -- clean1 = '1' OR pixword = word1
49
50 END IF; --empty = '0' AND pw = '0'
51
52 IF empty = '0' AND pw = '1' THEN
53
```

```
54  IF pixword = word1 THEN
55              ready1 <= '1';
56  ELSE -- pixword != word1
57      ready1 <= clean1;
58  END IF; -- pixword = word1
59
60  END IF; -- empty = '0' AND pw = '1'
61
62  IF empty = '1' AND pw = '0' THEN
63      ready1 <= '1';
64  END IF; -- empty = '1' AND pw = '0'
65
66  IF empty = '1' AND pw = '1' THEN
67      ready1 <= '1';
68  END IF; -- empty = '1' AND pw = '1'
69
70  END IF; -- reset = '1'
71  END PROCESS COMB;
72
73  REGISTERED : PROCESS
74  BEGIN
75    WAIT UNTIL clk'EVENT AND clk = '1';
76    IF reset = '1' THEN
77        store1 <= init_store;
78        word1 <= (OTHERS=>'0');
79    ELSE -- reset = '1'
80    IF (empty = '0' AND pw = '1' AND pixword = word1) OR (empty = '0' AND pw
        = '1' AND pixword /= word1 AND clean1 = '1') OR (empty = '1' AND pw =
        '1') THEN
81
82    IF empty = '1' AND pw = '1' THEN
83          store1 <= (OTHERS=>same);
84    END IF; -- empty = '1' AND pw = '1';
85
86    CASE pixopin IS
87    WHEN invert =>
88        CASE store1(to_integer(unsigned(pixnum))) IS
89          WHEN black =>
90            store1(to_integer(unsigned(pixnum))) <= white;
91          WHEN white =>
92            store1(to_integer(unsigned(pixnum))) <= black;
93          WHEN invert =>
94            store1(to_integer(unsigned(pixnum))) <= same;
95          WHEN same =>
96            store1(to_integer(unsigned(pixnum))) <= invert;
97          WHEN OTHERS => NULL;
98        END CASE; -- CASE store1(to_integer(unsigned(pixnum))) IS
99    WHEN same =>
100           store1(to_integer(unsigned(pixnum))) <= store1(to_integer(
                unsigned(pixnum)));
101   WHEN OTHERS =>
102           store1(to_integer(unsigned(pixnum))) <= pixopin;
103   END CASE; -- CASE pixopin IS
104   END IF; -- pixword = word1 OR (pixword /= word1 AND clean = '1')
105
```

```
106  IF empty = '0' AND pw = '1' THEN
107    IF pixword /= word1 THEN
108        IF clean1 = '1' THEN
109          word1 <= pixword;
110        END IF; -- clean1 - '1'
111    END IF; --pixword /= word1
112  END IF; -- empty = '0' AND pw = '1'
113
114  IF empty = '1' AND pw = '0' THEN
115        store1 <= init_store;
116        word1 <= pixword;
117  END IF; -- empty = '1' AND pw = '0'
118
119  IF empty = '1' AND pw = '1' THEN
120        word1 <= pixword;
121  END IF; -- empty = '1' AND pw = '1'
122
123  END IF; -- reset = '1'
124
125  END PROCESS REGISTERED;
126  -- output Assignments
127  store <= store1;
128  word <= word1;
129  clean <= clean1;
130  ready <= ready1;
131  END ARCHITECTURE rtl;
```

### A.4 pix_write_cache.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.numeric_std.ALL;
3  USE IEEE.std_logic_1164.ALL;
4  USE work.pix_cache_pak.ALL;
5  USE work.ram_fsm;
6
7  ENTITY pix_write_cache IS
8  GENERIC(
9     a_size : INTEGER := 8;
10    w_size : INTEGER := 16
11 );
12 PORT(
13    clk, reset, start  : IN   std_logic;
14    store              : IN   store_t;
15    address            : IN   std_logic_vector(a_size - 1 DOWNTO 0);
16    waitx              : OUT  std_logic;
17    vwrite             : OUT  std_logic;
18    vdout              : IN   std_logic_vector(w_size - 1 DOWNTO 0);
19    vdin               : OUT  std_logic_vector(w_size - 1 DOWNTO 0);
20    vaddr              : OUT  std_logic_vector(a_size - 1 DOWNTO 0)
21 );
22
23 END pix_write_cache;
24
25 ARCHITECTURE rtl OF pix_write_cache IS
26    SIGNAL add_temp      : std_logic_vector(a_size - 1 DOWNTO 0);
27    SIGNAL store_del     : std_logic_vector(w_size - 1 DOWNTO 0);
28 BEGIN
29
30 ramfsm : ENTITY work.ram_fsm
31    PORT MAP(
32       clk     => clk,
33       reset   => reset,
34       start   => start,
35       vwrite  => vwrite,
36       delay   => waitx
37    );
38
39 address_delay: PROCESS
40 BEGIN
41 WAIT UNTIL falling_edge(clk);
42    IF reset = '1' THEN
43       vaddr <= (OTHERS=>'0');
44    ELSE
45       vaddr <= address;
46    END IF;
47 END PROCESS address_delay;
48
49 vdin_compute : PROCESS
50    VARIABLE res  : pixop_t;
51 BEGIN
52    WAIT UNTIL falling_edge(clk);
53    IF reset = '1' THEN
54       vdin <= (OTHERS=>'0');
```

```
55    ELSE
56      FOR i IN store'RANGE LOOP
57        res := store(i);
58        CASE res IS
59          WHEN same    => vdin(i) <= vdout(i);
60          WHEN invert => vdin(i) <= NOT vdout(i);
61          WHEN black   => vdin(i) <= '1';
62          WHEN white   => vdin(i) <= '0';
63          WHEN OTHERS => NULL;
64        END CASE;
65      END LOOP;
66    END IF;
67  END PROCESS vdin_compute;
68
69  END ARCHITECTURE rtl;
```

### A.5 pix_cache_pak.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3
4  -- This package contains types and constants for use by the pix_word_cache
       block
5  -- pix_op_t is an array type used for the block ports, so this package must
        be
6  -- used by any architecture instantiated pix_word_cache.
7
8  -- Note that although the pixop_t array is similar to std_logic_vector(1
       DOWNTO
9  -- 0) the two cannot be directly assigned. In practice pixop_t will always
       be
10 -- used via the constants defined in this package, with CASE statements to
11 -- detect values or generate values as required.
12
13 -- store_t is the array type based on pixop_t that stores pixel operations.
14 -- Again it is used in  aport of pix_word_cache, so architectures
       instantiating
15 -- it will need to use this type.
16
17 PACKAGE pix_cache_pak IS
18   TYPE pixop_t IS ARRAY (1 DOWNTO 0) OF std_logic;
19
20   CONSTANT same   : pixop_t := "00";
21   CONSTANT black  : pixop_t := "10";
22   CONSTANT white  : pixop_t := "01";
23   CONSTANT invert : pixop_t := "11";
24
25   TYPE store_t IS ARRAY (0 TO 15) OF pixop_t;
26 END PACKAGE pix_cache_pak;
```

# Ram Control Block Report

## Interrim report for Code Review

Farhan Rahman

**ABSTRACT**

The document contains the initial design and logic for the RAM Control Block designed for the group assignment in the VHDL coursework. RAM Control Block sits between the “Draw Block” and the “VRAM” to act as an interface, manage Read-Modify-Write commands for pixels to be written onto the VRAM which is a 256 word * 16 bit external RAM.

The RAM control block is an entity that controls which pixel operations need to be stored in the “pixel_write_cache” (which will be discussed later in the report) and then store the pixels into the VRAM. The Draw Block will give draw, flush or clear-screen commands to the RAM Control Block.

NB. RCB uses the following files:

1) rcb.vhd
2) ram_fsm.vhd
3) pixel_write_cache.vhd
4) pixel_word_cache.vhd

The handshake signals between Draw Block and RCB is shown in the following diagram:

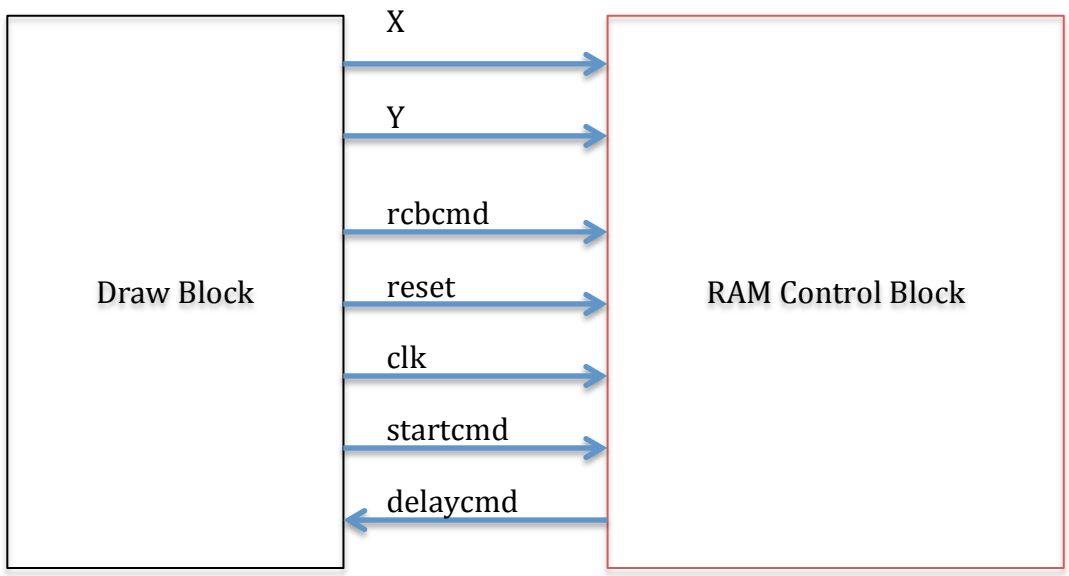
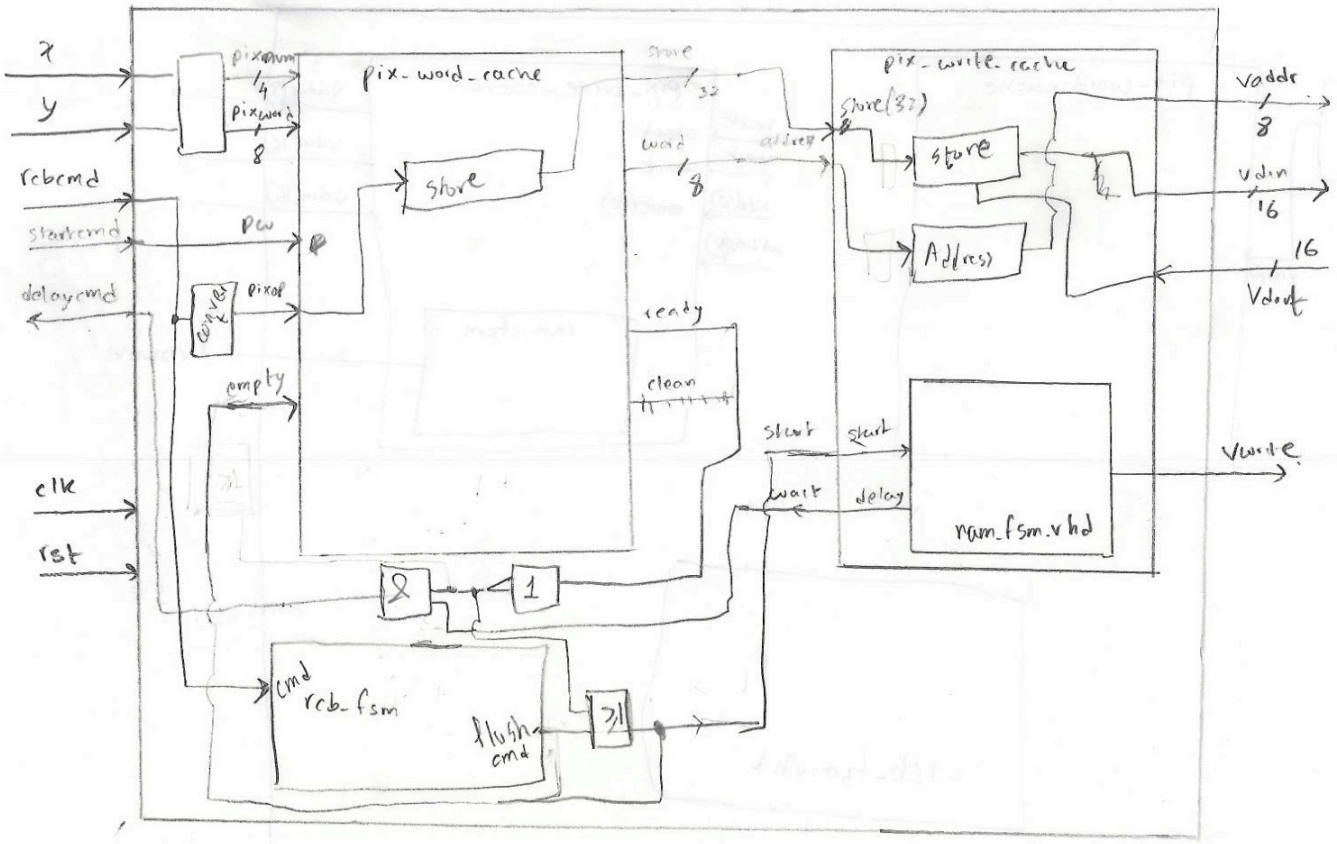



Figure 1

The top-level sketch of the design is shown in the next figure.




empty = 1 → flush out data to RAM (pix_write cache)

pw = 1 → new data is available

Figure 2

## Top-design:

There are three entities that go inside the RAM Control Block as follows:

1) pixel_word_cache
2) pixel_write_cache
3) ram_fsm

Among the top files, pixel_word_cache and ram_fsm have been already created and tested.

**Pixel_word_cache:**

The input signals “x” and “y” to the rcb are first converted to represent an 8-bit signal “word” and a 4-bit pixel number called “pixnum” of the pixel_word_cache. Then the “rcbcmd” from the draw block is converted to do draw commands if required in that cycle. The input signal to the rcb called “startcmd” is connected directly to the “pw” input of the pixel_word_cache.

One important thing to notice is that the “ready” signal from the output denotes that the pixel_word_cache needs its pixel operations to be flushed into the RAM (depending if the operations aren’t clean i.e. the dirty bit is high i.e. the data isn’t consistent with the RAM). Therefore in the same cycle the “empty” signal (which is an input to the entity) is set to high so that the flush operation can begin.

**Pixel_write_cache & Ram_fsm:**

The main purpose of having this is to reduce the number of cycles to address a large external RAM. This acts like a buffer between the “pixel_word_cache” and the RAM. The “pixel_word_cache” sees this as another generic RAM.

The “pixel_write_cache” includes the entity “ram_fsm” which does the timing for the RMW (Read-Modify-Write) operation, which takes 3 cycles to complete. Whenever “empty” is asserted, at the same time “start” signal is asserted so that the flushing and writing new operation (once the “ready” signal is set to high) can begin. The “word” output from the “pixel_word_cache” acts as the “address” input for this entity and it is stored in an address register, which is clocked at a negative clock edge to **delay** the **address** output by **half a clock cycle**. When the

"empty" signal is asserted, immediately the "store" output from the "pixel_word_cache" is stored inside the **"store" register** of the **pixel_write_cache.** Then the "vdin" output, which is the input to the VRAM is changed according to the stored pixel operations. This output is made sure to be delayed by **half a clock cycle**.

**Important to notice:** In the beginning it would not be quite intuitive to see that the delay outputs from pixel_word_cache (!ready) and pixel_write_cache (wait) are **ANDED** together to represent the **delaycmd** output of the rcb. However it makes sense because it can be seen that with this arrangement the "pixel_word_cache can still receive some inputs if the Ram_fsm is completing its RMW operation and so making things more efficient.

## Finite State Machine (RCB):

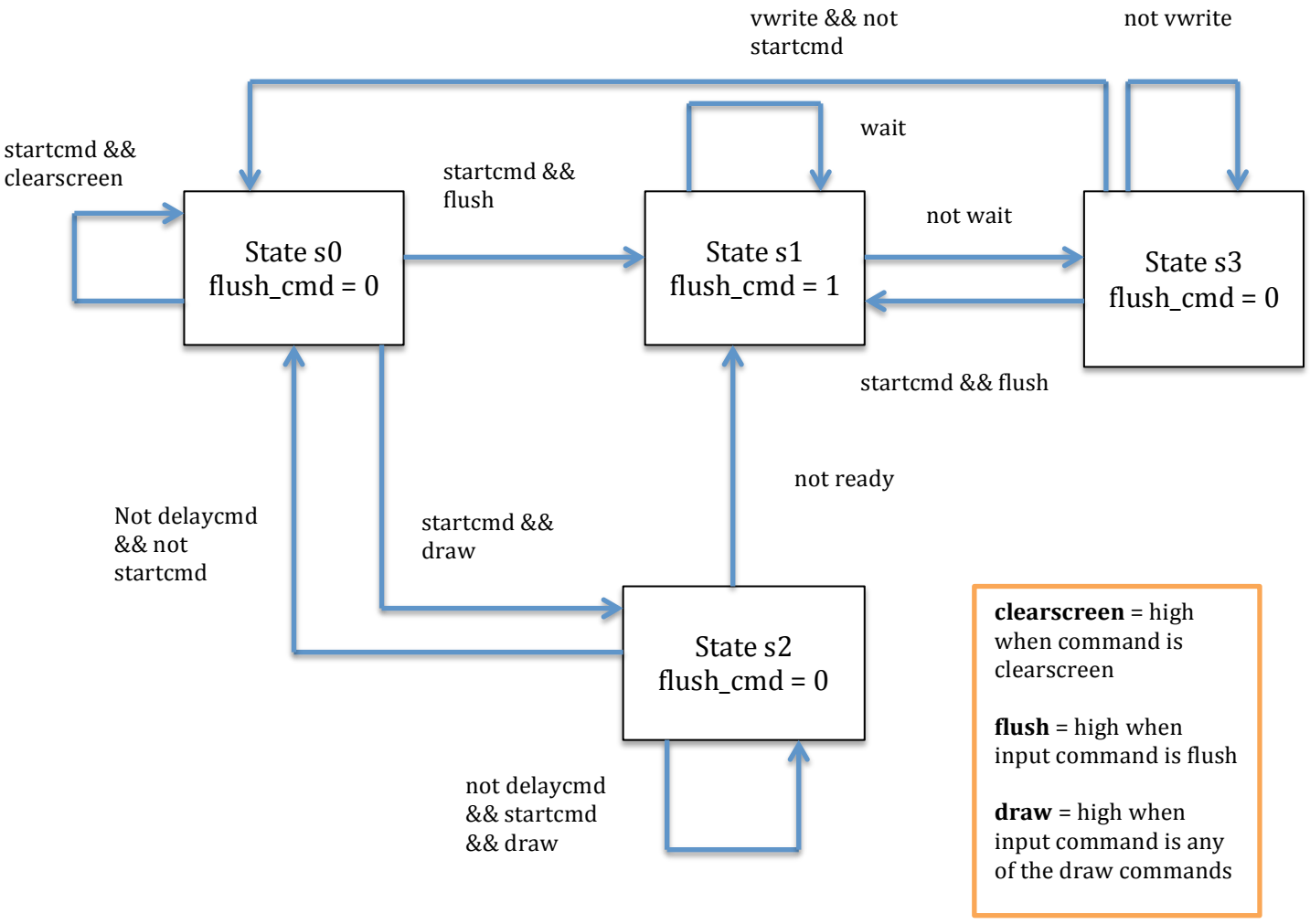



Figure 3

Figure 3 show the Finite State Machine for the Ram Control Block.

The following table outlines the truth table for the FSM shown in figure 3

| Current state | reset | startcmd | delaycmd | Ready | Flush | Draw | Clearscreen | vwrite | wait | nextstate | flush_cmd |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SX | 1 | X | X | X | X | X | X | X | X | S0 | 0 |
| S0 | 0 | 1 | X | X | 1 | X | X | X | X | S1 | 1 |
| S0 | 0 | 1 | X | X | X | X | 1 | X | X | S0 | 0 |
| S0 | 0 | 1 | X | X | X | 1 | X | X | X | S2 | 0 |
| S1 | 0 | X | X | X | X | X | X | X | 1 | S1 | 1 |
| S1 | 0 | X | X | X | X | X | X | X | 0 | S3 | 0 |
| S2 | 0 | 1 | 0 | X | X | 1 | X | X | X | S2 | 0 |
| S2 | 0 | X | X | 0 | X | X | X | X | X | S1 | 1 |
| S2 | 0 | 0 | 0 | X | X | X | X | X | X | S0 | 0 |
| S3 | 0 | 1 | X | X | 1 | X | X | X | X | S1 | 1 |
| S3 | 0 | 0 | X | X | X | X | X | 1 | X | S0 | 0 |
| S3 | 0 | X | X | X | X | X | X | 0 | X | S3 | 0 |

## Entities in terms of process blocks (Code):

**NB:** The functionalities for the already written entities (pix_word_cache and ram_fsm) are not included as they remain as they were (except additions for Generics).

**rcb.vhd:**

The rcb has the following process blocks:

P1: Process “P1” is a combinatorial process where the x,y and rcb command inputs are computed and then fed into the pixel_word_cache block.

Driven outputs:

1) pixword
2) pixnum

P2: Process “P2” is a combinatorial process where the next state is assigned depending on current inputs and current state. The outputs that are driven are as follows:

Driven outputs:

1) nstate

C1: Process “C1” is a clocked process and in this state, the current state is driven by whatever nexstate was form previous cycle.

Drive outputs:

1) state

**pix_write_cache.vhd:**

address_delay: Waits till falling edge of clock to ensure half a cycle delay and then assigns the output “vaddr” to the current address. This is a clocked process.

Driven outputs: 1) vaddr

din_compute: This is a clocked process. In half a cycle after vwrite, this assigns the value of din according to the current pixel operations.

Driven outputs:

1) vdin

**How to do clearscreen?**

At the moment the blocks aren’t designed to handle clearscreen commands. However I do have an idea of how to do the clearscreen command.

Firstly I have to create a FIFO for the instructions so that all the instructions get stored inside the FIFO and read off the other end at the same time. So when there is a clearscreen command, the previous values of x and y need to be taken and the new valus to of x1 and y1 to which the screen should be cleared. Then the delaycmd should be kept high. While the delaycmd is high, the RCB should create instructions

itself to write white or black depending on clearscreen instruction. These commands should be stored in the FIFO and the delaycmd should only be low when these instructions have been made and the system can recover back from stall.