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:

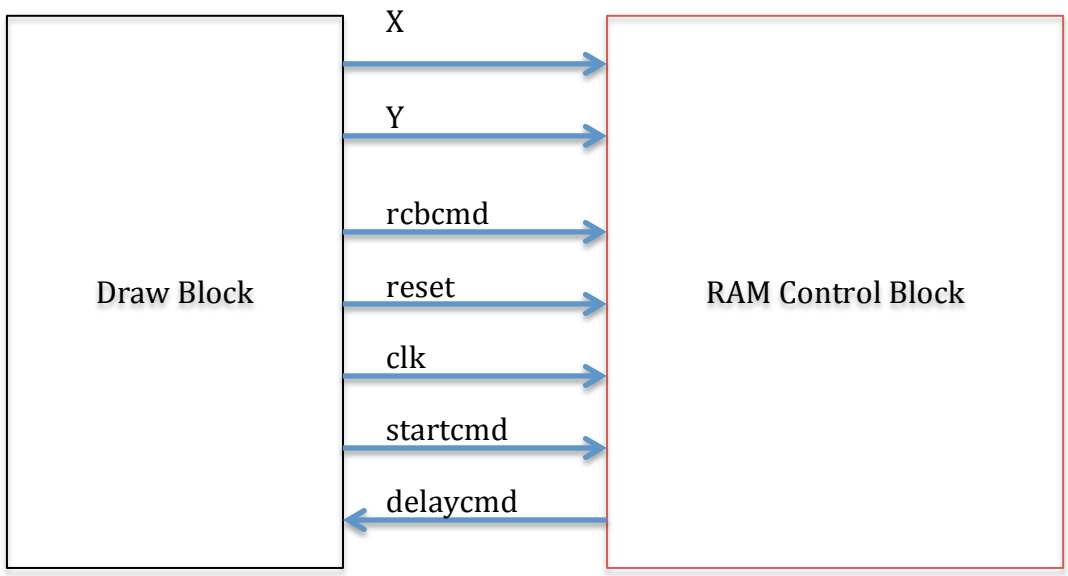
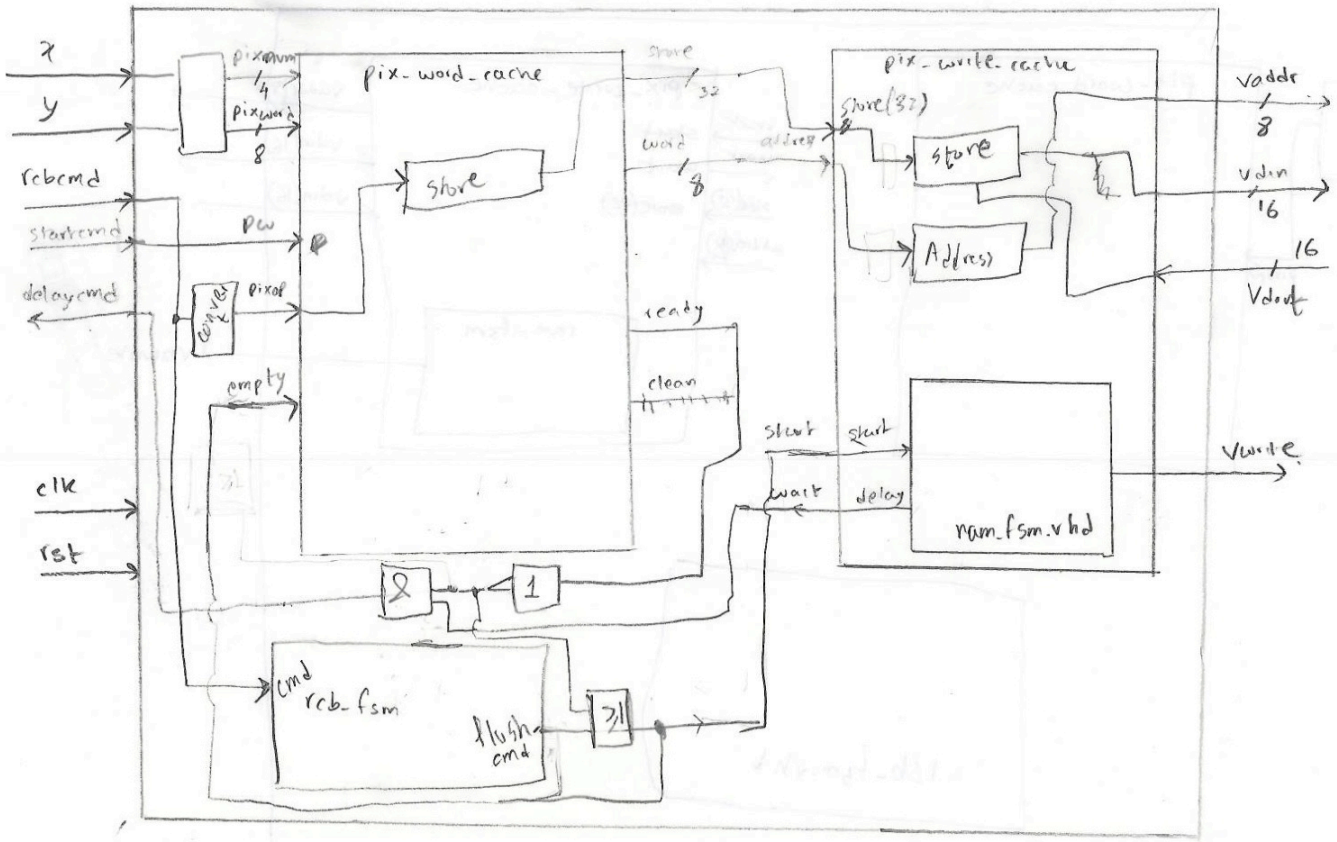


Figure 1

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 → read 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):

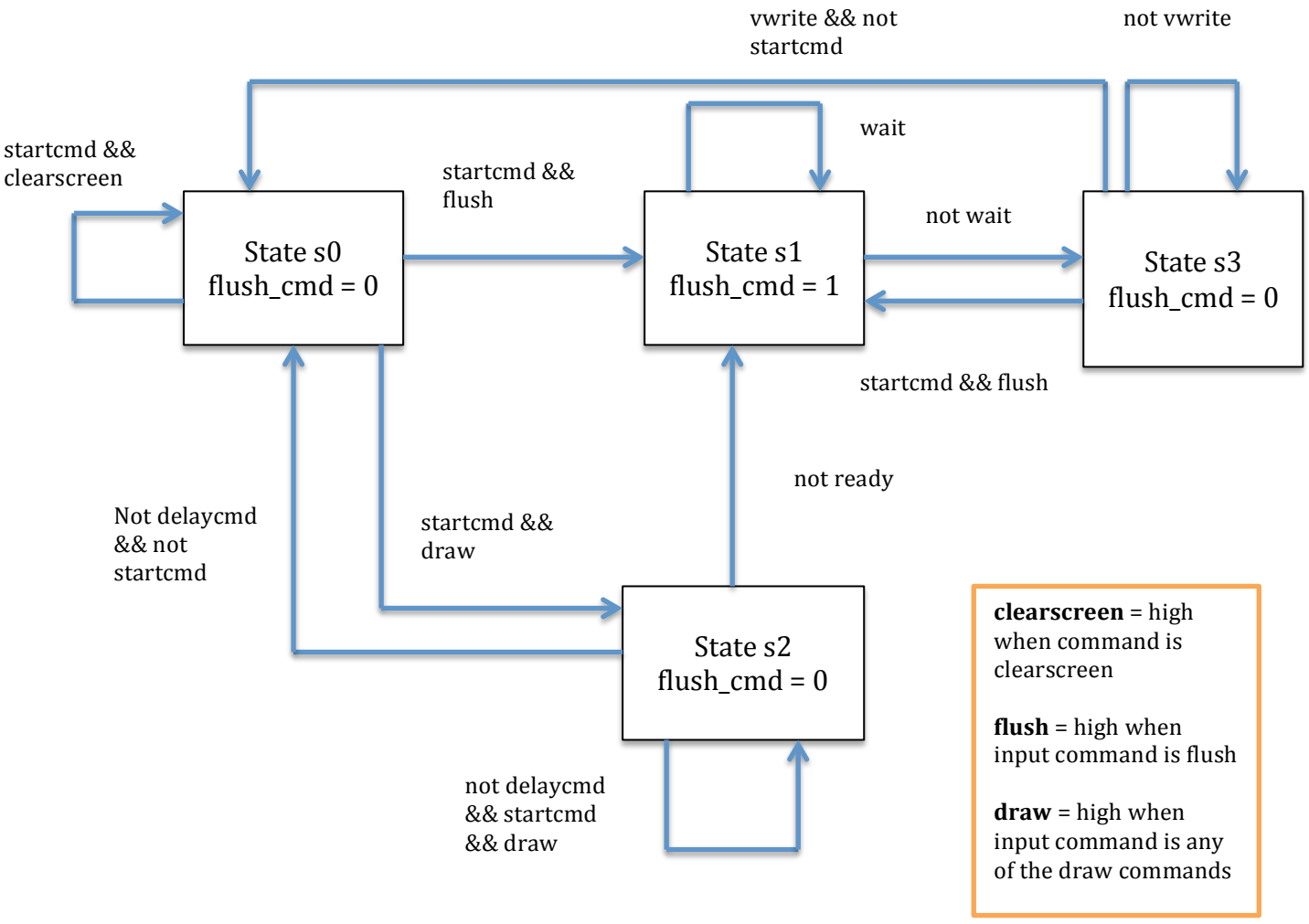


Figure 3

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.