Spring 12

Post Synthesis Report: Ram Control Block, Clearscreen (fr909)

Farhan Rahman

The report contains description about the post-synthesis of the VDP block with detailed description of the ram control and clearscreen block. The report analyses the design choices made for the blocks and the performance analysis of the vdp in two different hardware. It emphasises on the differences and optimisations in the hardware made since the pre-synthesis.

Group 33 (Jonathan Ely & Farhan Rahman)

ABSTRACT OVERVIEW

The Ram Control Block follows through right after the Draw Block. It receives commands that are being sent from the Draw Block. The design was changed a bit compared to the given specifications in the following way: a *'clearscreen'* entity was created that sits just between the *rcb (ram control block)* and the *drawblock*. The top-level design can be explained with the following diagram.

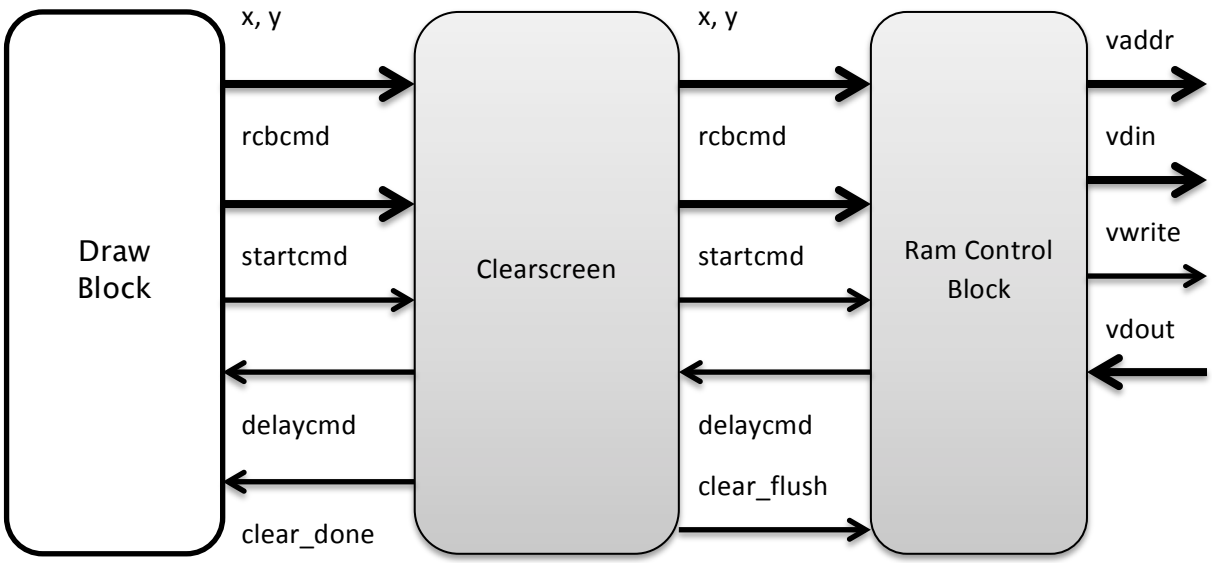

FIGURE 1

'Clearscreen' entity operates in two modes which are when it is *idle* and when it is doing the clears operations. This helps when passing draw commands from *'clearscreen'* entity but however there is a tradeoff between increased complexities in the critical path. The *'clearscreen'* entity has been improved since the pre-synthesis. More will be explained in the section describing the entity in later sections.

Both *'clearscreen'* and *'ram control block'* has been optimised since the pre-synthesis submission and this resulted into increased clock frequency and throughput.

In the next sections the important features of the two entities (clearscreen and rcb) will be explained.

ANALYSIS AND DESIGN

The following section will go through how the internal parts of the hardware were designed, discuss about several VHDL constructs used and analyse how the algorithm has been improved since the pre-synthesis.

CLEARSCREEN

The clearscreen algorithm can operate in two basic modes, which are clearing by one pixel at a time or clearing one word at a time. Clearing one pixel at a time is very inefficient and this was the algorithm that was used in the pre-synthesis tests. However now the algorithm has been optimised and improved in the sense that it uses a combination of both clearing a word at a time (when all the pixels fall in the given rectangle) and clearing one pixel at a time (when some of the pixels in a word falls in the rectangle that needs to be cleared).

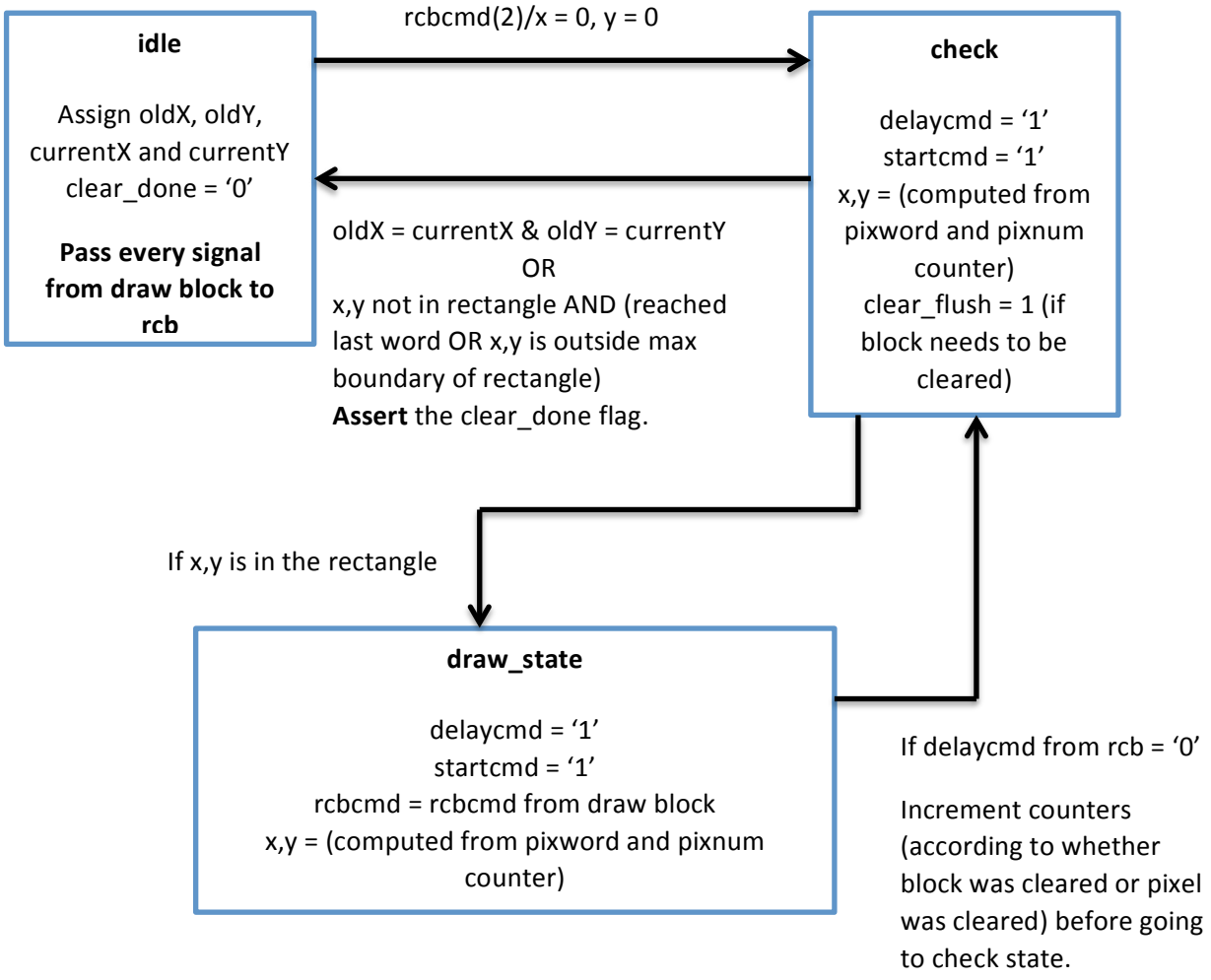

FIGURE 2

Figure 2 shows the state machine diagram that *'clearscreen'* entity uses. The way this bit of hardware works is that it initially always remains in the idle state. In this state it allows all the signals coming from the draw block to go to the *'ram control block'*. The entity keeps track of old coordinates in signals called 'oldX' and 'oldY' and new coordinates in signals called 'currentX' and 'currentY' and it assigns them accordingly in the idle state every cycle just in case it has to go into clearscreen modes which are the states 'check' and 'draw_state'. Two more counters include 'pixnum' and 'pixword' which are internal signals that help it to iterate through the pixels or words.

The first thing that 'clearscreen' does is visualise a rectangle formed from the values 'oldX', 'oldY', 'currentX' and 'currentY'. It initialises the 'pixnum' and 'pixword' counters to '0' and starts the 'check' state where it does one of 3 things:

1) Go to idle state if all the current and old coordinates are the same OR
2) Go to draw state if current pixel counter falls in the rectangle OR
3) Go to idle state if we have reached the last pixel of the last word or the current pixel counter is greater than the max coordinates of the rectangle being visualised.

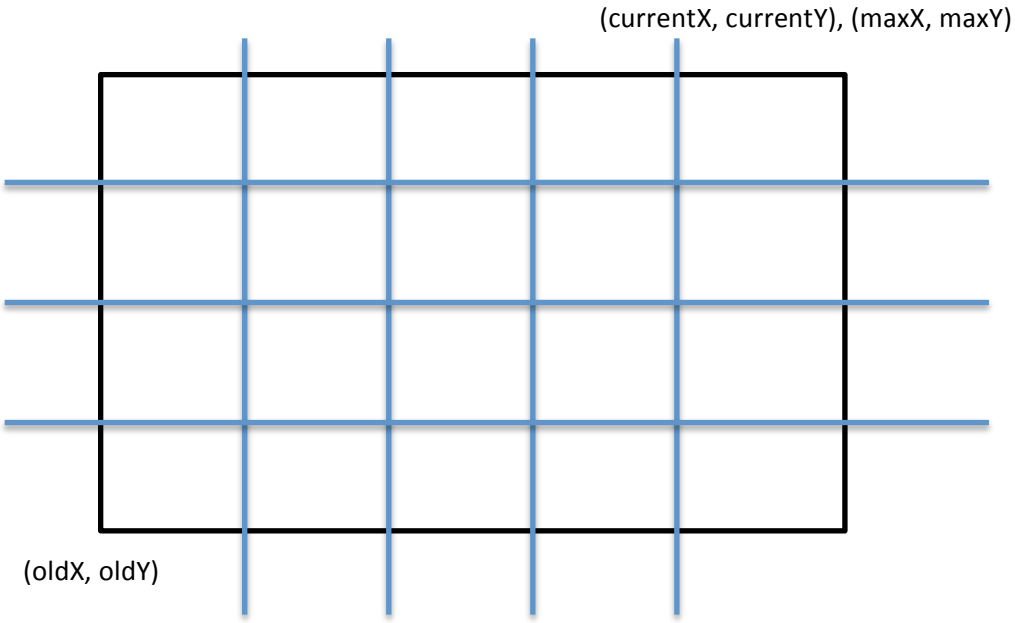

FIGURE 3

Figure 3 shows an example of when the clearscreen operation may start. The cells above represent words on the screen.

Once in the 'draw_state' the 'clearscreen' executes two of the following draw commands:

1) Sends word flush command by setting 'clear_flush' signal high: This was done by making some changes to the 'pix_word_cache' inside the 'ram control block'. This is one of the optimisations that has been added which increases the throughput.
2) Sends a draw command to the 'ram control block' to draw a single pixel.

The algorithm follows a set of rules for incrementing 'pixword' and 'pixnum' after returning from 'draw_state' as follows:

1) Increment pixel number by '1' if a single pixel was cleared
2) Increment pixword to the next row if there is no need to go to any more words to the right of the visualised rectangle: This is another optimisation which stops clearscreen to iterate through words which do not fall in the visualised rectangle.
3) Increment pixword and set pixnum to '0' if a whole word was cleared.

Functions used by clearscreen: 'Clearscreen' uses functions as defined in the file 'cl_pack.vhd'. The funtions are as follows:

1) **is_in_rect (returns BOOLEAN):** This function computes whether the current pixel the clearscreen is in falls in the visualised rectangle. If it does then it returns 'TRUE' else it returns 'FALSE'. The algorithm has been made quite efficient in the sense that it uses '=' operators to do the comparison rather than '<= or >= or < or >' operators. This reduces the critical path in clearscreen as using the latter operators creates nested adders as shown in 'Synplify'.

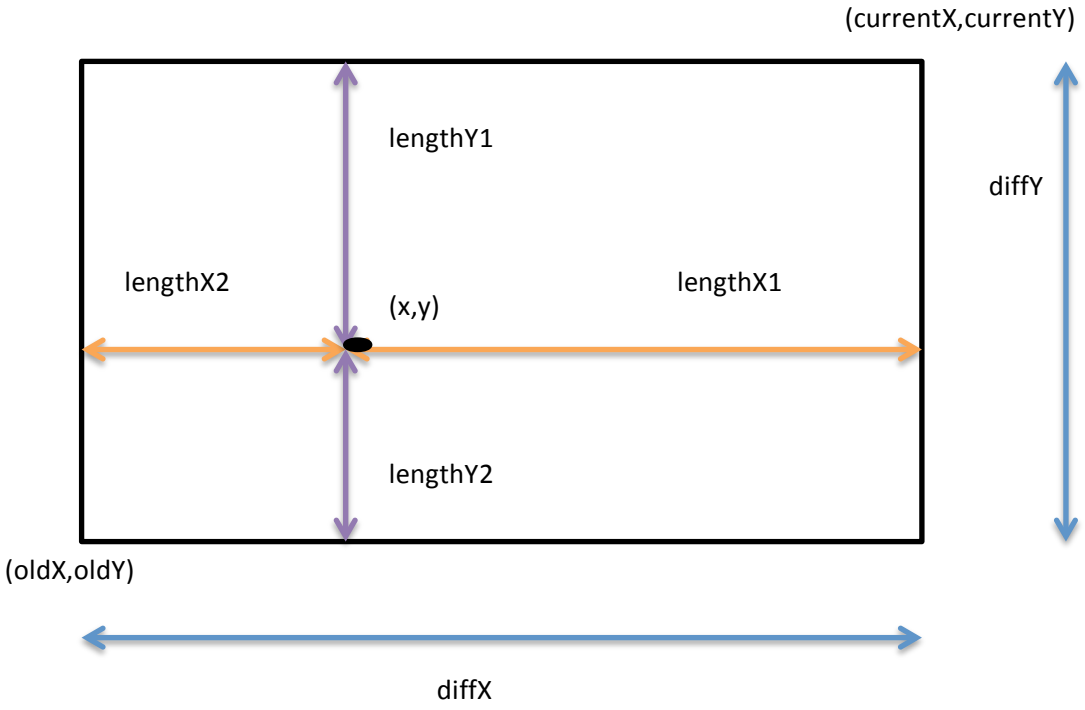

FIGURE 4

lengthX1 = absolute(currentX – x). lengthX2 = absolute(x – oldX).

lengthY1 = absolute(currentY – y). lengthY2 = absolute(y – oldY).

diffX = absolute(currentX – oldX). diffY = absolute(currentY – oldY).

The function returns true if the following condition is met:

(lengthX1 + lengthX2 = diffX) AND (lengthY1 + lengthY2 = diffY)

2) **is_greaterthan_maxX (returns BOOLEAN):** Returns 'TRUE' if current pixel is greater than the maximum 'x' coordinate value of the visualised rectangle.

3) **is_outside_max_points (returns BOOLEAN):** Returns 'TRUE' if clearscreen has exceeded maxX and maxY points.

4) **is_block_in_rect (returns BOOLEAN):** Returns 'TRUE' if block needs to be cleared.

Initially the function operations were coded into the clearscreen entity but later in order to improve readability of code and to reuse the same hardware for computation, functions were created. These functions can be further studied in detail in the 'cl_pack.vhd' file.

RAM CONTROL BLOCK

In this section the 'Ram Control Block' will be explained briefly mainly including the updated design and optimisation techniques used. The block consists of the internal blocks 'ram_fsm' and 'pix_word_cache' both of which have been modified compared to the original specifications in the exercises in order to suit the design that has been done. The 'pix_word_cache' (clocked in positive edge) output is connected to the 'ram_fsm' (clocked in negative edge of clock). There is a state machine of the rcb which mainly deals with assigning the start command for ram_fsm, the empty command for pix_word_cache to initiate a flush and set the delaycmd that goes to the clearscrean block.

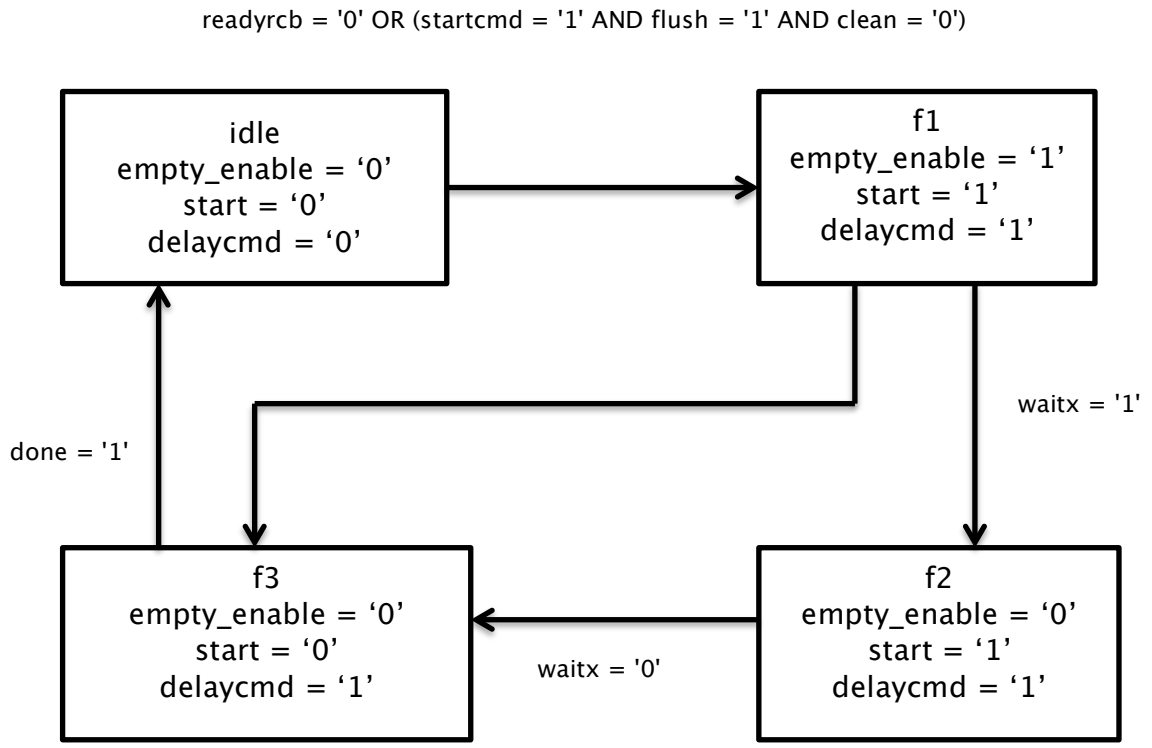

FIGURE 5

Figure 5 shows the state machine for the ram control block. Other small optimisations inside the rcb file include code minimisations and some logic simplification.

ram_fsm: The ram_fsm has been optimised in the sense that the 'done' flag now goes high in state 'm2' rather than state 'm3' (the last state of ram_fsm) which reduces the total number of cycles required to finish the testbench.

pix_word_cache: The pix_word_cache has an added feature, which allows it to draw a single word with black, white or invert commands. This is a required feature for clearscreen when it has to clear a single word block.

Generics and functions have been used in the design, which improves code readability and makes the hardware more flexible.

Regarding changes made to 'vdp.vhd': There has been a change made to the hdb_busy signal. It now takes into account the clear_done flag from clearscreen to as it is a required feature in the design.

FEATURES

The hardware passes the test-bench given for the project. There were some problems faced in the pre-synthesis testing and they were fixed. Notably commands such as Draw commands without a Move pen, Clearscreen command without a Move Pen before it and so on. Moreover the throughput and clock-frequency of the hardware has been removed. More on that will be discussed in the performance section.

Pre-Synthesis: Pre-Synthesis passes the test-bench.

Post-Synthesis: The post-synthesis also passes the test-bench.

There is an issue with the clearscreen. It does not prefer a certain combination of clearscreen after another. However that can be worked around by rearranging the clearscreen commands.

PERFORMANCE

The performance data was extracted from Synplify by analysing or synthesising on two of the hardware which are **Actel 3200DX** and **Altera Cyclone II.** The synthesis in the following sections were done for vdp block.

SYNTHESIS ON ACTEL 3200DX

The code was synthesised using the options 'Resource Sharing' checked in the Implementation Options and that gave a overall clock frequency of operation of 8.6 MHz and slack of 0.765.

```
Performance Summary
*******************


Worst slack in design: 0.765

                   Requested     Estimated     Requested     Estimated               Clock         Clock
Starting Clock     Frequency     Frequency     Period        Period        Slack     Type          Group
-------------------------------------------------------------------------------------------------------------
vdp|clk            8.6 MHz       8.7 MHz       116.279       114.750       0.765     inferred      Inferred_clkgroup_0
=============================================================================================================
```

FIGURE 6

The actual post-synthesis test was done using 1MHz and in both the synthesis the tests passes with a total cycle time of **10925 ns.** These tests were run on a clock period of **10ns** and last vram check of **100 cycles.**

It is worth mentioning that the post-synthesis will not pass if any other optimisation options are clicked such as FSM compiler and so on.

ALTERA CYCLONE II FPGA

The code synthesised for the Altera Cyclone II FPGA gave an overall clock frequency of 85.6 MHz with a slack of 0.060 as shown in Figure 7 below.

```
Performance Summary
*******************


Worst slack in design: 0.060

                   Requested     Estimated     Requested     Estimated               Clock         Clock
Starting Clock     Frequency     Frequency     Period        Period        Slack     Type          Group
-------------------------------------------------------------------------------------------------------------
vdp|clk            85.6 MHz      86.5 MHz      11.682        11.563        0.060     inferred      Inferred_clkgroup_0
=============================================================================================================
```

FIGURE 7

SYNTHESISING RAM CONTROL BLOCK AND CLEARSCREEN ON CYCLONE II

The ram control block along with clearscreen was synthesised on the Altera Cyclone II FPGA and it clocked in at about 92.0 MHz.

All synthesis passes without any warnings.

FURTHER IMPROVEMENTS

Further improvements from the rcb side would be to include fifos to store draw commands and therefore not result into stalling the whole hardware.

The state machine in rcb can be removed and a concurrent process can be created which increases throughput and clock frequency.