# VHDL Post Synthesis and Final Report

*Jonathan Ely – Draw Block – Group 33*

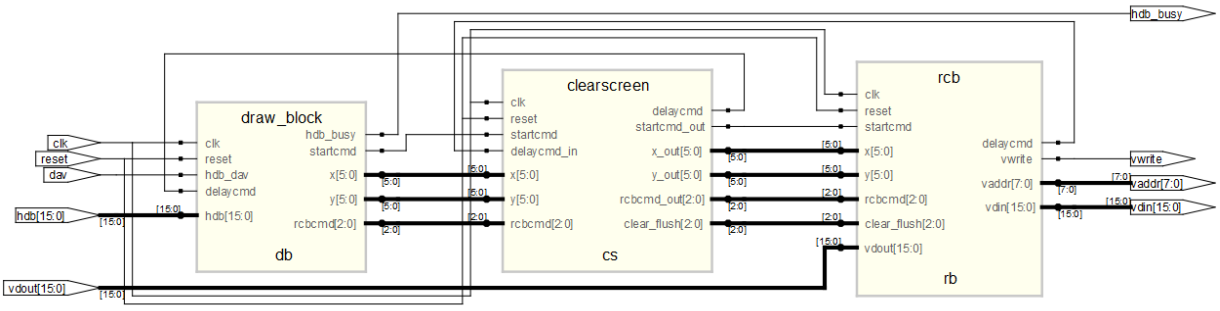


**Figure 1 - RTL Overview of VDP Project**

For this project I took responsibility for the Draw Block. Farhan took responsibility for the RCB. In implementing clear screen functionality, Farhan decided to create a dedicated block at the top level, hence the 3 blocks shown above.

## Performance

### Synthesis on Actel 3200DX

We achieve a maximum clock frequency of 8.6 MHz using the resource sharing option.

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


Worst slack in design: 0.765

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

```
Worst Path Information
View Worst Path in Analyst
***********************


Path information for path number 1:
    Requested Period:                      58.140
    - Setup time:                          0.375
    = Required time:                       57.765

    - Propagation time:                    57.000
    = Slack (critical) :                   0.765

    Number of logic level(s):              8
    Starting point:                        rb.empty_enable / q
    Ending point:                          rb.pwordcache.store1_1[0] / d
    The start point is clocked by          vdp|clk [falling] on pin clk
    The end   point is clocked by          vdp|clk [rising] on pin clk
```

Post synthesis simulation - with the same set up except target frequency set to 1 MHz - passes the original test bench with a completion time of 10925ns.

Using the FSM Compiler and FSM Explorer options gave some increase in clock speed but caused problems in simulation.

## Synthesis on Altera Cyclone II

We achieve a maximum clock frequency of 86.5 MHz using the resource sharing and pipelining options.

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


Worst slack in design: 0.014

                   Requested     Estimated     Requested     Estimated
Starting Clock     Frequency     Frequency     Period        Period        Slack
--------------------------------------------------------------------------------
vdp|clk            86.5 MHz      86.7 MHz      11.561        11.534        0.014
================================================================================
```

```
Worst Path Information
View Worst Path in Analyst
**************************


Path information for path number 1:
    Requested Period:                       5.780
    - Setup time:                           0.048
    = Required time:                        5.732

    - Propagation time:                     5.719
    = Slack (critical) :                    0.014

    Number of logic level(s):               10
    Starting point:                         rb.delaycmd1 / regout
    Ending point:                           db.draw_block_i.wrapper.error[5] / datain
    The start point is clocked by           vdp|clk [falling] on pin clk
    The end   point is clocked by           vdp|clk [rising] on pin clk
```

## Features

Our design passes, in post synthesis simulation, the original command file provided. It can also process and perform all the commands described in the specification. There are however, a few bugs with clear screen that mean certain corner case tests will not pass. In particular, tests that include multiple clear screen operations without any move pen operations or draw operations between them sometimes (although not always) fail. We tried to resolve this behaviour although were not able to in the within the given project time.

## Analysis and Design

In terms of the number of cycles (ie actual time taken) to complete a given piece of code our VDP is very much limited by the time taken to make stores (from draw) and to complete the clear screen.

The most intensive operation required of the draw block is the draw operation. Even this only takes a couple of cycles to set up and then one cycle to provide each pixel in the line. However, in each draw operation the draw block is spending multiple cycles waiting for stores to complete for every few pixels provided. It is for this reason that many of the theoretical improvements I might make to the draw block would not be worth the complexity (which would likely impede clock speed) unless the RCB was much faster. The clear screen, move pen and flush cache commands each take one cycle for the draw block to process after which the draw block will wait upon the RCB's busy signal before it will move on. That being said, there are still several design improvements that I would suggest.

One such improvement would be to add a FIFO to allow calculating and storing the pixels to be changed from a draw operation whilst *delay_cmd* is high. However since this design only takes one cycle to calculate the next pixel in a line this wouldn't reduce the time taken to complete a given program in with RCB setup as it would still take a cycle to read from the FIFO.

Another improvement that could be made would be to split the draw operation and use multiple *draw_any_octant* blocks in parallel to draw sections of the desired line. This alteration would add some complexity in calculating a suitable position in which to split the line however one block could start calculating from the beginning – allowing immediate output pixels – whist this was computed – once suitable breakpoints were calculated the multiple *draw_any_octants* could start calculating the remaining sections of the desired line.

It would be possible to make even more significant improvements to the time taken if you were to make the *draw_block* more joined up with the *RCB* – i.e. making the draw_block more memory aware. The simplest optimisation this would allow would be the converting of horizontal draws to word operations instead of single pixel operations. However, more complex optimisations could be implemented. For instance, I might design my *draw_block* to consider whether multiple draw or clear operations have overlapping memory locations. Where they do not I might choose to process the commands in parallel. Where they do I might consider whether one will overwrite the other – if so, I may choose to ignore one of the operations (such as drawing a line then clearing the area) or I may want to change the order of the operations.

When writing my VHDL I tried to adhere to certain good design principles that helped ensure good performance. For instance I tried to avoid unnecessary complexity from using too many nested conditionals. Instead I tried to workout distinct use cases with simple conditional statements. I also tried to use combinational logic where appropriate (thus being more efficient with area and logic required). For instance, the values for *swapxy, negx* and *negy* that are provided to the *draw_any_octant* block are continually assigned in the *OCT* combinational process as they will always be correct when needed and it doesn't matter at other times.

I decided to design my FSM (as described in the code review) to try and keep a minimum time between operations. It is for this reason that the single cycle operations: (from the view of the DB) move pen, clear screen, flush and the first draw setup cycle; are done in the listen state with extra states for the remaining draw operation cycles. It would have been slightly neater to have had a separate listen state and then individual states for each operation. I did not do this because it would have added a cycle delay in between each operation.