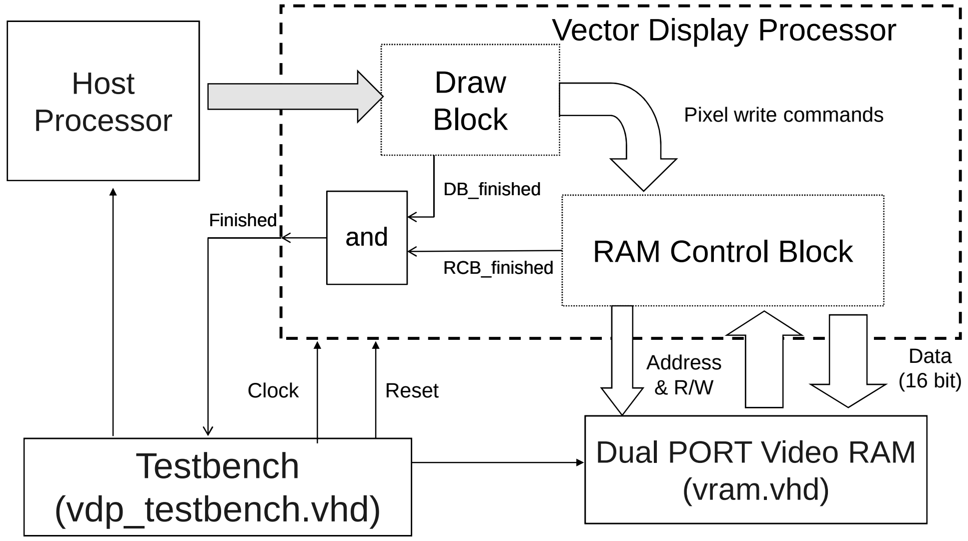
VHDL Team Project Introduction

- Objective
  - Design, Test & Synthesise a Vector Display Processor
    - » Input - sequence of line drawing commands from a host processor
    - » Output - read/write interface to Video RAM
  - Conducted in teams of 2 – but with individual mark components
  - Make use of VHDL already designed in exercises 1-4

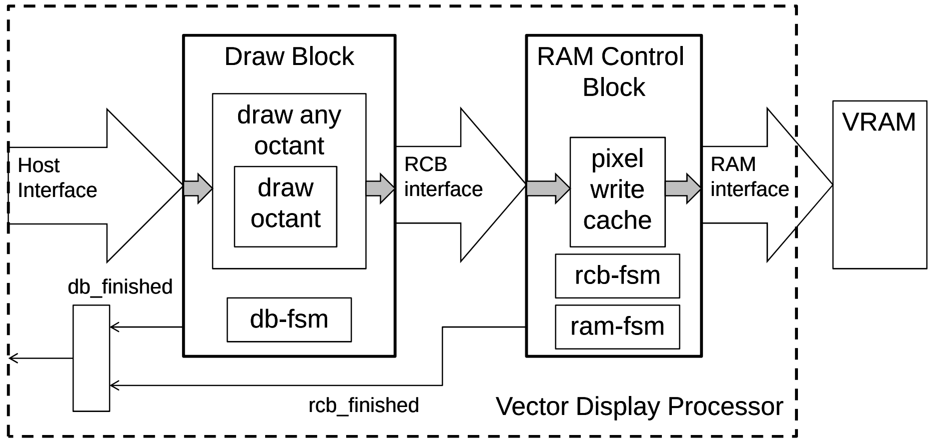
Project Deliverables

1. **Code review**
   - Group mark
   - test quality and effectiveness of review, not quality of code that is reviewed.
2. **Complete design**
   - Passes testbench pre- and post-synthesis
   - Properly tested
   - Group mark with individual components
3. **Final report**
   - Performance (group and individual)
   - Code quality, scalability & modularity (individual)
   - design analysis (individual)

Overview of Vector Display Processor & Testbench



Hardware Block Diagram



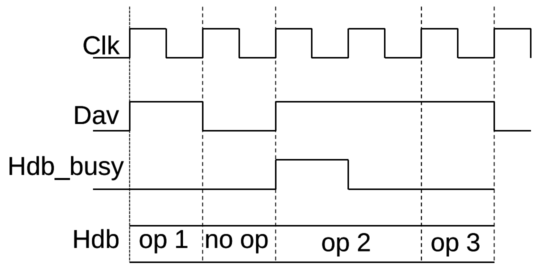
Dataflow through 2 wire handshakes

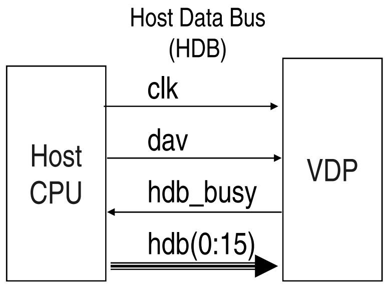
Sender signal	Receiver signal	Data transfer cycles	Data
hdb_dav	hdb_busy	hdb_dav and not(hdb_busy)	commands from host to VHD
startcmd	delaycmd	startcmd and not(delaycmd)	commands from DB to RCB
start	delay	start and not delay	input to ram_fsm

- ◆ Think about design as propagation of data through hardware blocks. Internal interfaces are all controlled by clock
  - ❖ Data is presented by sender when ready
  - ❖ Data is received by receiver when ready
  - ❖ maximum data rate is one item per cycle
- ◆ Send & receive signals must both be in correct state to transfer data. Note that receive signal state when transmitter signal is off does not matter.

Host Interface

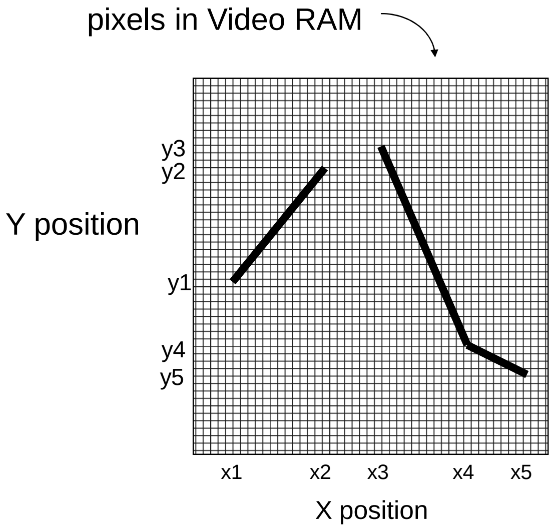
type	name	description
In	dav	high when new command data is available
Out	hdb_busy	handshake for new data
In	hdb(0:15)	command word (see next slides)





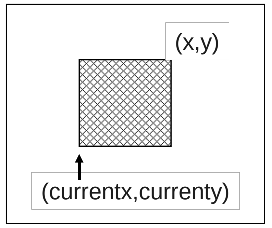
Line drawing commands

- ◆ *Single line:*
  - MovePen x1, y1
  - DrawLine x2, y2
- ◆ *Two segment line:*
  - MovePen x3, y3
  - DrawLine x4, y4
  - DrawLine x5, y5

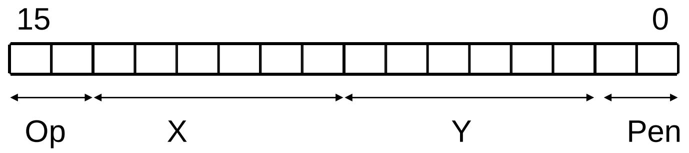


ClearScreen Command

- ◆ Clearscreen(x, y, pencol)
  - ❖ Colors a rectangle on the screen **and its interior** with pen type pencol.
- ◆ Rectangle corners:
  - ❖ Current pen position
  - ❖ Position given in **ClearScreen** command word (x,y)



Host command Coding

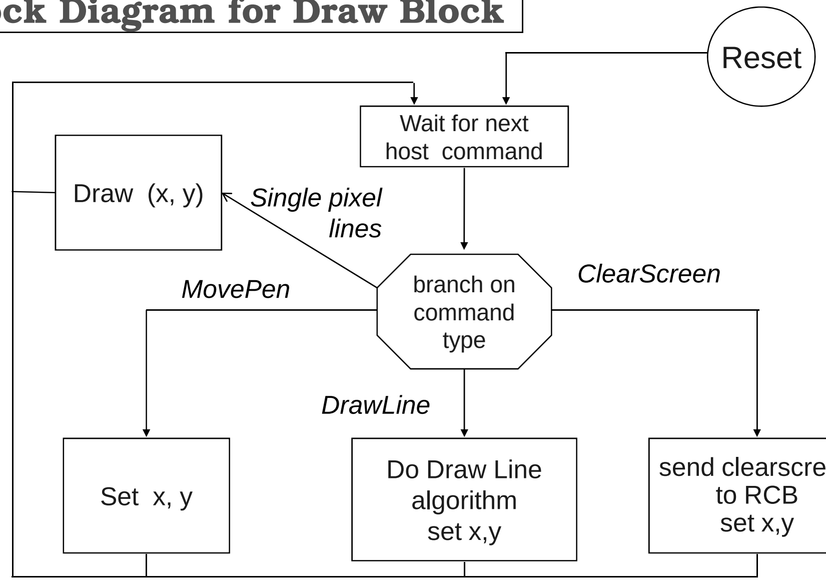


Command	Op
MovePen	00
DrawLine	01
ClearScreen	10
Not used	11

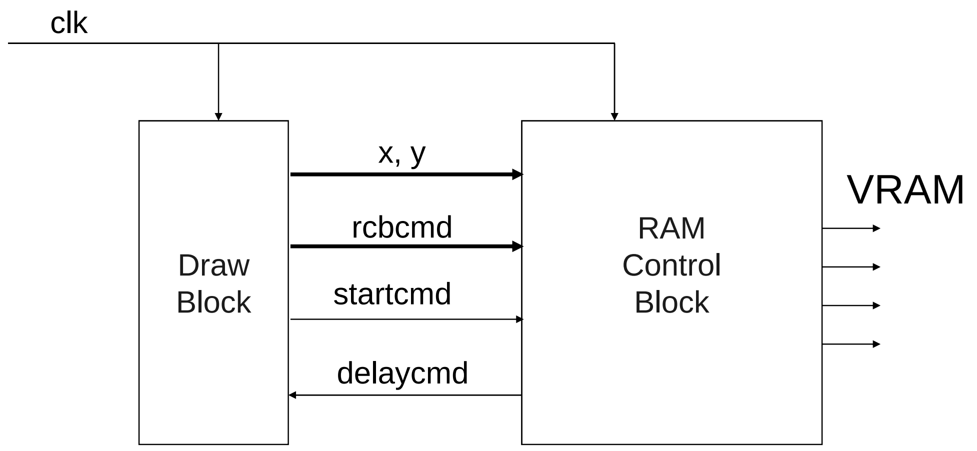
Pen Type	Pen
Not used	00
White	01
Black	10
Invert	11

- Every command from the host is coded as a 16 bit word.
- The command is split into separate fields for operation, X and Y pixel position, and pen colour.
- MovePen sets the current pen position (X,Y)
- DrawLine commands draw a line from the current pen position to (X,Y), and then set the current pen position to (X,Y).

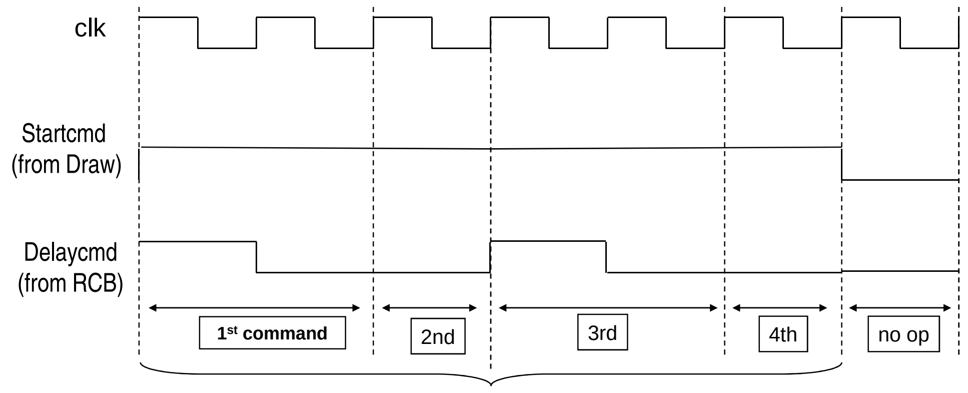
Block Diagram for Draw Block



DB/RCB Interface



DB/RCB Interface Timing



Startcmd: commands can be sent in consecutive clock cycles if Delaycmd=0, otherwise each command stretches until the first cycle in which Delaycmd is 0. This allows the RCB to hold a new command from the VDP until after it has finished processing the previous command. Startcmd goes to 1 from the first cycle until the final one (when Delaycmd is 0).

***x**, **y**, **rcbcmd** are valid when **startcmd** is '1'*

DB/RCD Commands

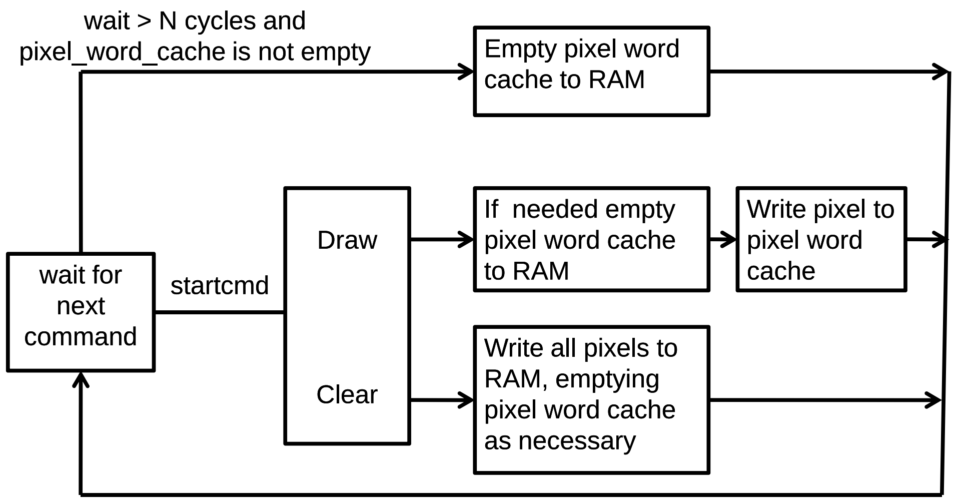
- X (6 bits)
- Y (6 bits)
- **rcbcmd** - see table
- Note that **clearscreen** could with less efficiency be implemented in DB, in which case the **clear** and **move** commands are not used

RCB CMD		
000	move	Define first corner of rectangle for clearscreen
001	draw white	Draw performs op on specified pixel
010	draw black	
011	draw invert	
100	not used	
101	clear white	Clearscreen performs op on rectangular area of screen with corners defined by current X,Y, and X,Y of previous RCB command.
110	clear black	
111	clear invert	

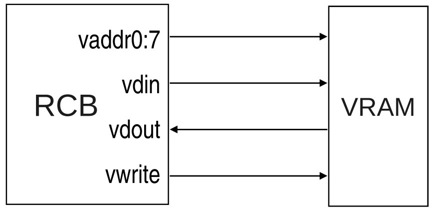
RAM Control Block

- RCB stores pixel operations in a pixel_word_cache unit
  - It will be emptied into RAM whenever necessary using a single RMW cycle to operate on existing RAM contents
- Pixel_word_cache is emptied whenever the next pixel operation is in a different RAM word from the previous one.
  - RAM **words** are 16 pixels, arranged as a 4X4 square.
  - Typically a line will have 4 pixels in each word before an RMW operation is needed.
- Pixel_word_cache will also be emptied if some number of cycles pass without a new pixel write request

RAM Control Block Block Diagram

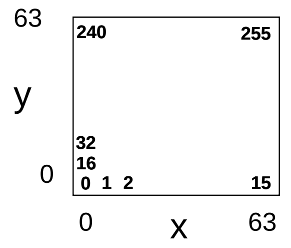

wait > N cycles and
pixel_word_cache is not empty
Empty pixel word cache to RAM
wait for next command
startcmd
Draw
Clear
If needed empty pixel word cache to RAM
Write pixel to pixel word cache
Write all pixels to RAM, emptying pixel word cache as necessary

Video RAM interface



vdin/vdout:
black = 1
white = 0

- *VRAM is a static RAM*
- *Separate I/O*
- *RAM words map to screen as below:*



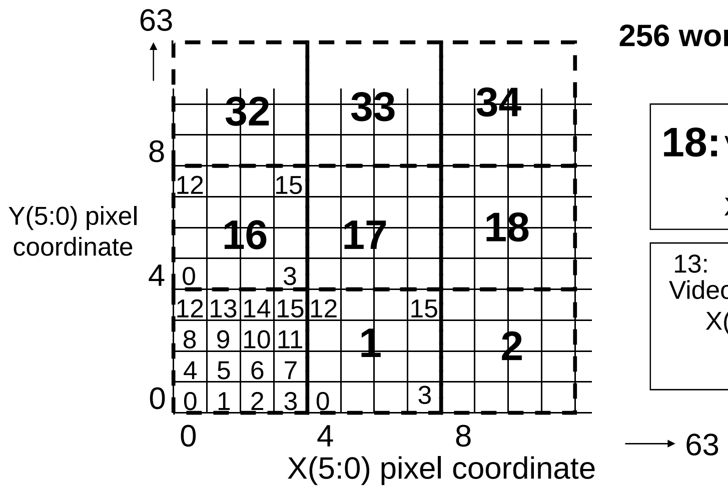
Interface from RCB to VRAM

vaddr0:7	Address to VRAM
vdin(0:15)	VRAM data input
vdout(0:15)	VRAM data output
vwrite	VRAM write strobe

See next slide for pixel mapping into RAM words

RAM pixel packing

256 word * 16 bit RAM



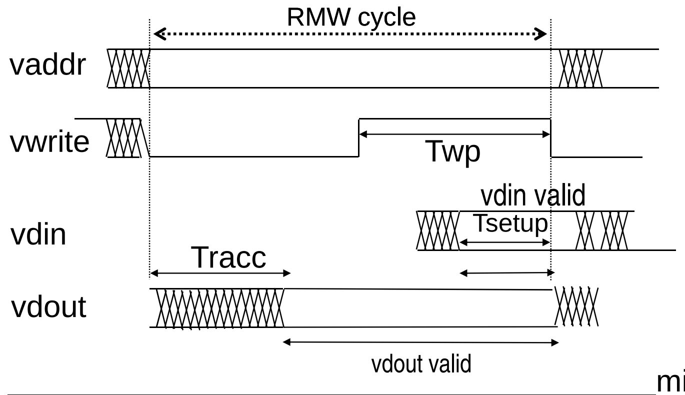
18: Video RAM word Address X(5:2)+16*Y(5:2)

13: Video ram bit number: X(1:0)+4*Y(1:0)

Vram word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Read-Modify-Write Cycle Timing



		min/ns	max/ns
Taccess	*time, vaddr to vdout*		Tracc
Twrite	*vwrite high time*	Twp	
Tsetup	*write data setup time*	Twds	

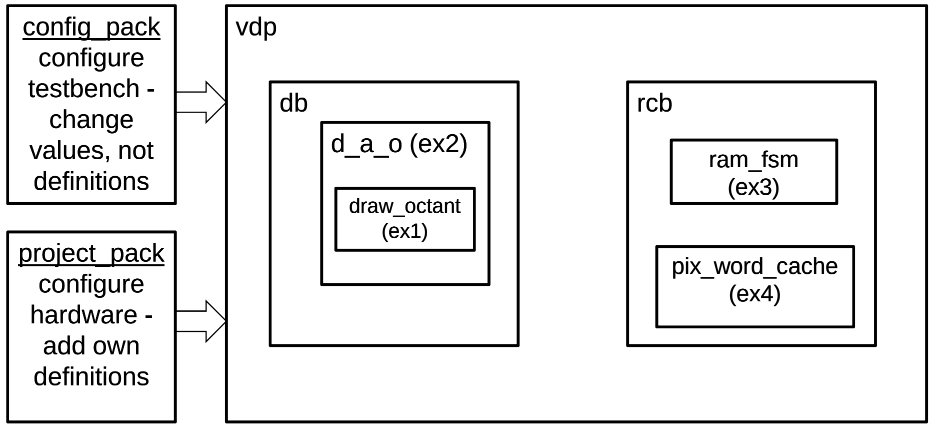
RAM characteristics

- You may assume (initial deliverable) that **twp** & **tracc** are defined so that **ram_fsm** gives correct RAM timing (see **vdp_pack** and **config_pack**).
- For better RCB you should redesign **ram_fsm** such that state m2 (for **tracc**) and m3 (for **twp**) can be stretched the correct number of cycles to match these values. You will need to add one extra state to distinguish the last cycle of the stretched m3.
- Since these times are constants the necessary arithmetic is done at compile time
  - **ram_fsm** could be redesigned with two integer generics indicating lengths of these two states.

Finish

- DB_Finish
  - assert whenever block is inactive
  - no host command is requested
  - no host command is being executed
- RCB_Finish
  - assert whenever block is inactive
  - no pixel write is outstanding
  - pixel_ram_cache is empty
  - ram is inactive
- Finish = DB_finish and RCB_finish

VHDL implementation


config_pack
configure testbench - change values, not definitions
project_pack
configure hardware - add own definitions
vdp
db
d_a_o (ex2)
draw_octant (ex1)
rcb
ram_fsm (ex3)
pix_word_cache (ex4)

NB - ex1,ex2,ex3,ex4 design files contain entities draw_octant, draw_any_octant, ram_fsm, pixel_word_ram