ECE 385 – Digital Systems Laboratory

Lecture 19 – Memory, Sprites, and Applications

Zuofu Cheng

Fall 2018

Link to Course Website

ECE ILLINOIS

ILLINOIS

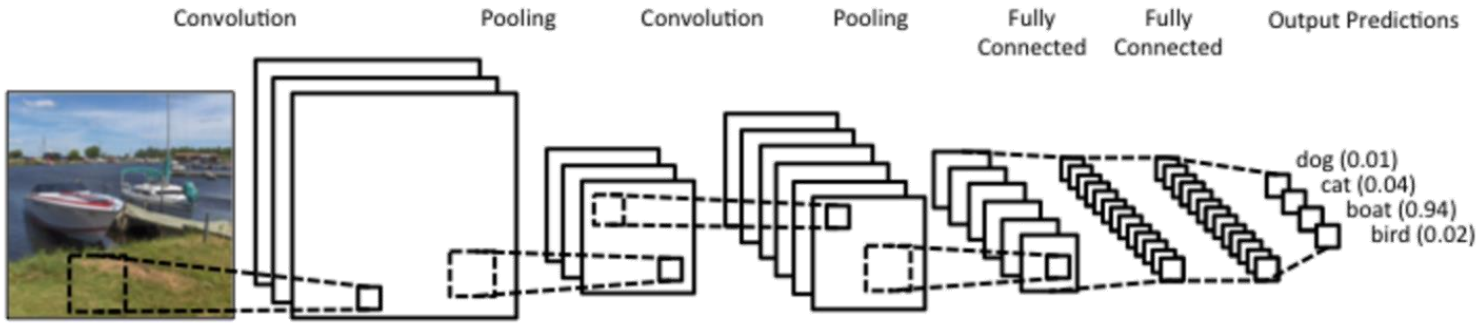
FPGA Applications

- FPGA's strength lies in flexibility
- Gives design engineer ability to balance between memory, compute, cost, and power

Technology	Performance (FLOPs)	Memory/Caches	Design Complexity	Power	Cost
FPGA	Medium	Medium	High	Scalable	Scalable
CPU (x86)	Medium	High	Low	Medium	Medium
CPU (ARM)	Low	Medium	Low	Low	Low
GPU	High	Medium	Medium	High	High

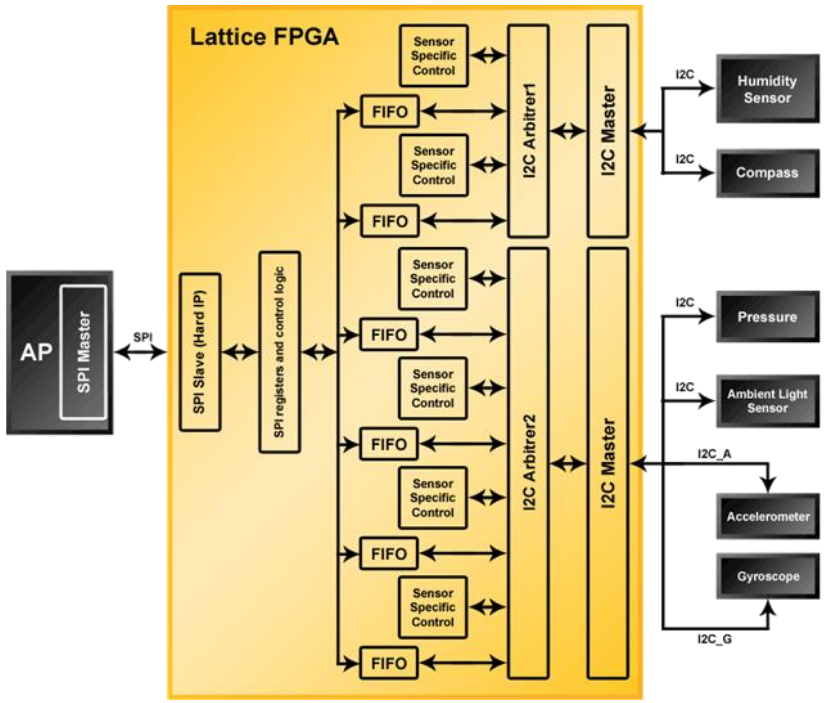
FPGA Applications (Continued)

- Typical usage cases for FPGAs:
  - Accelerator (encryption/decryption, cryptocurrency, deep learning)
  - Interface is handled through SoC CPU (NIOS II/ARM, etc)
  - Accelerate specific functions which take the bulk of the computational time and are parallelizable
  - For example, in deep learning application, lowest task might be image convolution



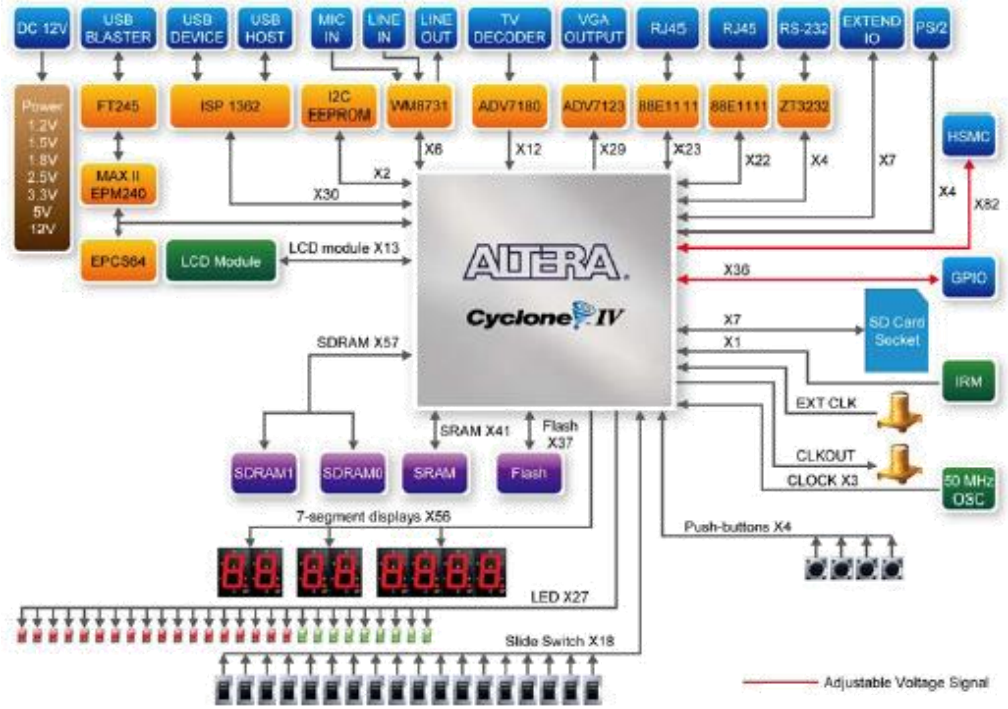
FPGA Applications (Continued)

- Alternatively FPGAs may be used as a "glue logic" or sensor hub
- Typically use for smaller FPGAs (fewer compute resources, more routing resources)
- Example, SDRAM controller for DSP processor or sensor hub for smartwatch, video controller for ATM
- Allows for flexibility in board design and firmware upgrades to fix "hardware" bugs



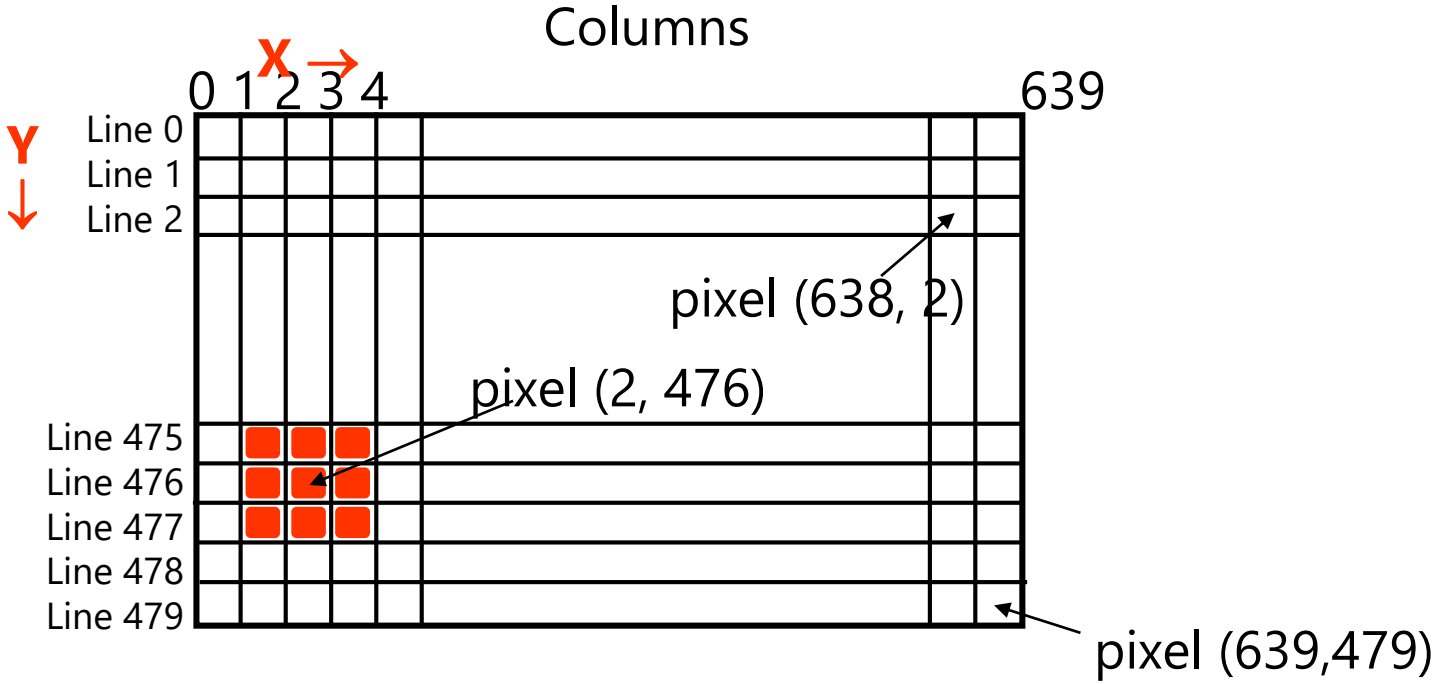
FPGA Applications (Continued)

- For final project, make sure you propose design which makes sense for the FPGA board
- Don't use a $200 FPGA part for an application where a 80 cent microcontroller would suffice!
- Good projects typically involve sufficient compute intensive tasks: machine learning, machine vision, DSP, graphics, video, hardware emulation



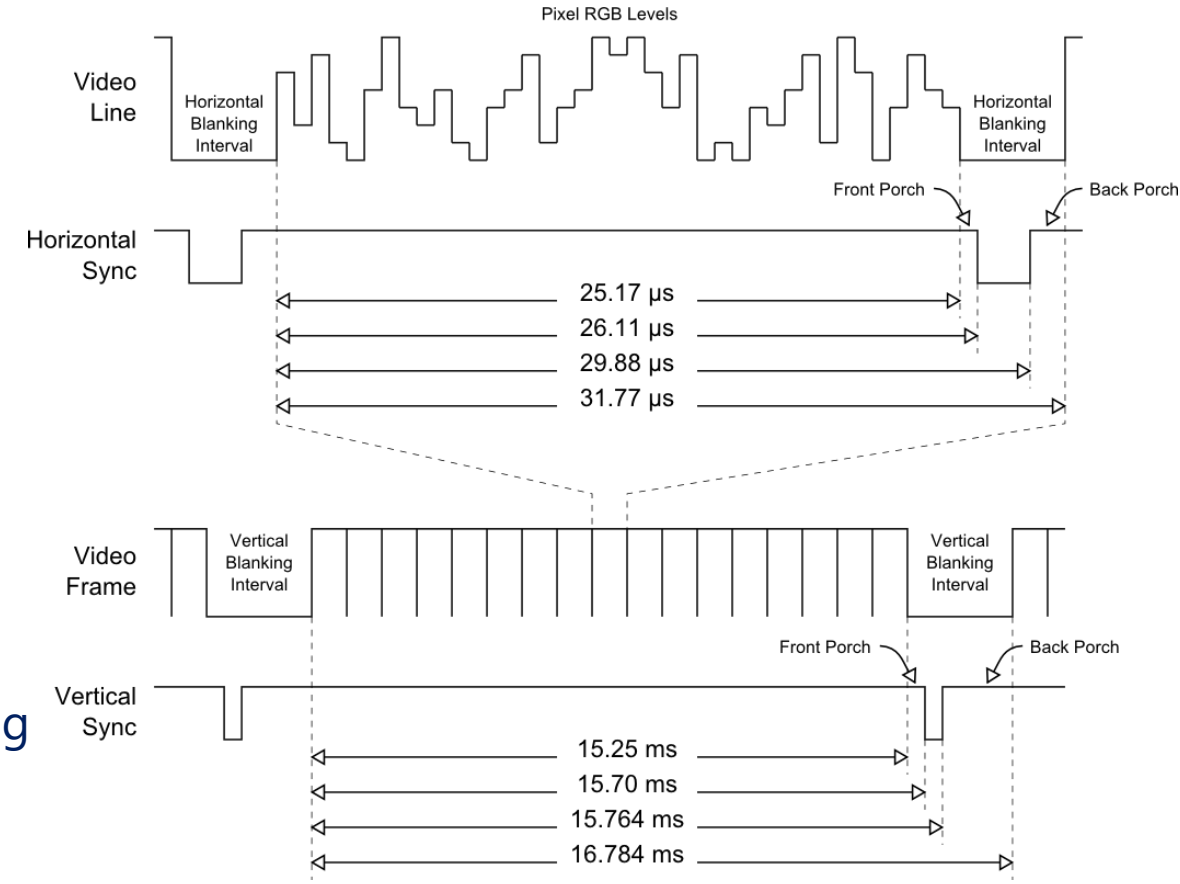
VGA Monitor Operation

- VGA (Video Graphics Array) Standard
  - The screen is organized as a matrix of pixels
    - 640 horizontal pixels x 480 vertical lines
  - An Electron Beam "paints" each pixel from left to right in each row, and each row from top to bottom



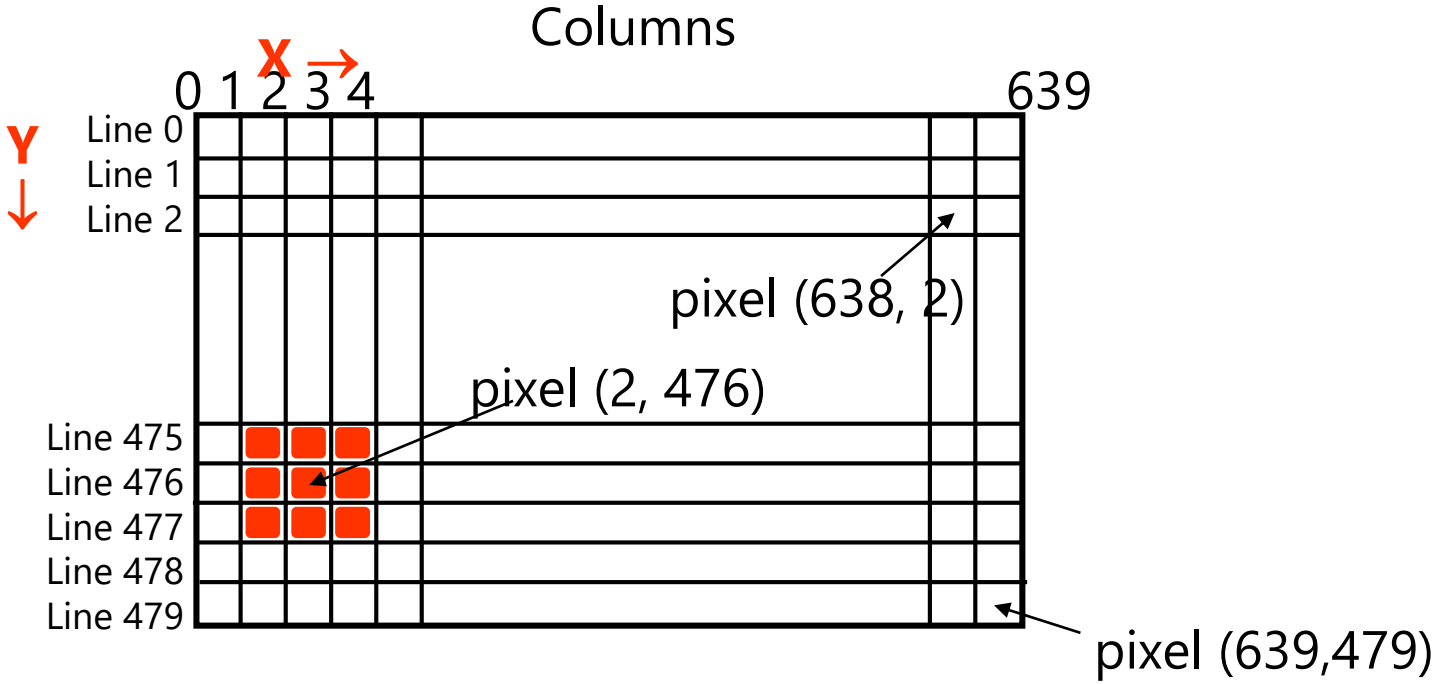
VGA Timing (continued)

- Screen refresh rate = 60 Hz
  - Note: this doesn't mean your game must run at 60 Hz
  - But you must generate VGA signal 60 times a second!
  - One frame = 16.67 ms
- Overall pixel frequency = 25.175 MHz
- Can approximate by using 25.000 MHz (50 MHz / 2 using flip flop)
  - Makes frame time longer, now 16.784 ms
- Note: VGA communicates via analog voltages (DE2-115 has DAC to generate these)
- Easy to create clock divider by 2 (how?)



VGA Monitor Operation

- In lab 8, we used a simple color mapper combined with the VGA controller to draw simple shapes
- Color mapper needs to have as inputs the horizontal and vertical position counters, and maps output color either to foreground color (e.g. red) or background color (e.g. white)



Drawing Shapes (Simplest Approach)

- A shape can be defined by specifying a boundary around a center. In the previous example, the center is (2, 476) and the box is defined *Center ± Size*. For Size=1 all pixels in the box satisfy:

(X≥2-1) AND (X≤2+1) AND
(Y≥476-1) AND (Y≤476+1)

- Color mapper detects the condition and maps output color (combinationally) to foreground color if condition is satisfied, background color if condition is not satisfied

Limitations of Simple Approach

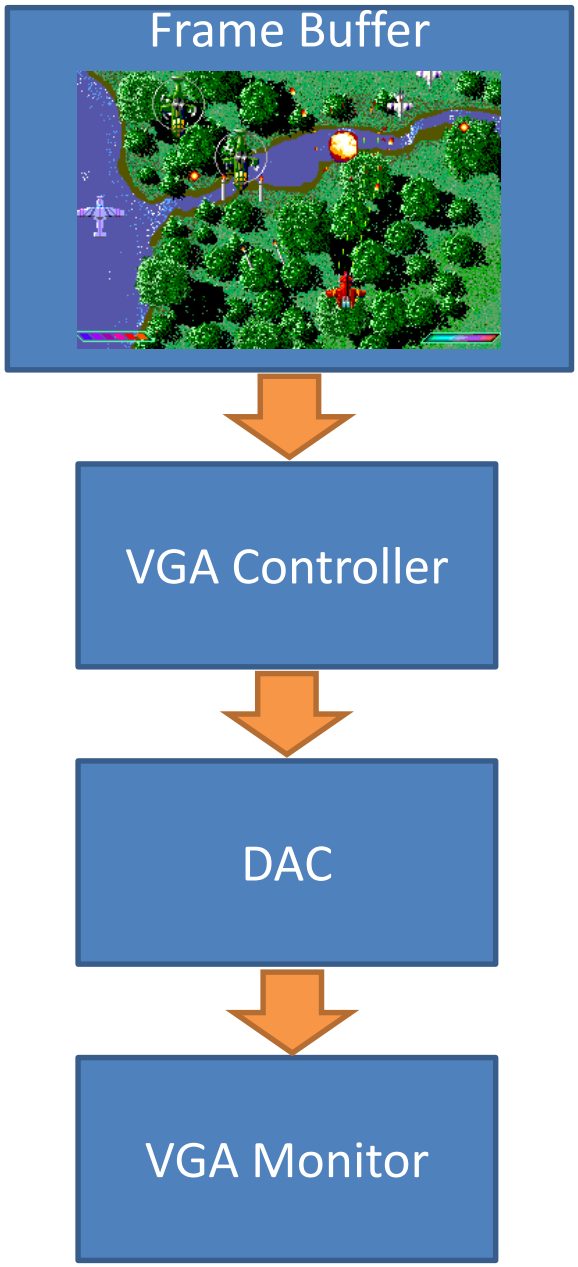
- If we strictly draw based on H and V pixel positions, we can only draw boxes
- What if we want to draw more sophisticated graphics (circles, fonts, spaceships, Mario?)
- We do not want to instantiate logic which describes everything we would ever want to draw – want instead to make *generalized* hardware to draw whatever *software* can describe
- In general, we want design to be *data driven* and not *logic driven*

Two Fundamental Approaches to Drawing

- Fixed Function
  - Fixed function hardware has limited number of primitives which may be on-screen at once
  - Breaks down graphics into smaller sprites and tiles
  - Plus: doesn't need memory to hold screen
  - Minus: limited by number of sprites, sprite size, hard to do effects such as transparency, scrolling, unless specialized hardware is designed
- Frame Buffer
  - Stores entire contents of screen to be drawn into high speed memory (frame buffer)
  - One side of memory goes to video controller, other side goes to blitter (memory copy unit)
  - Plus: limited only by performance of memory and blitter, can do effects easily
  - Minus: needs enough memory to hold the screen (possibly twice over)

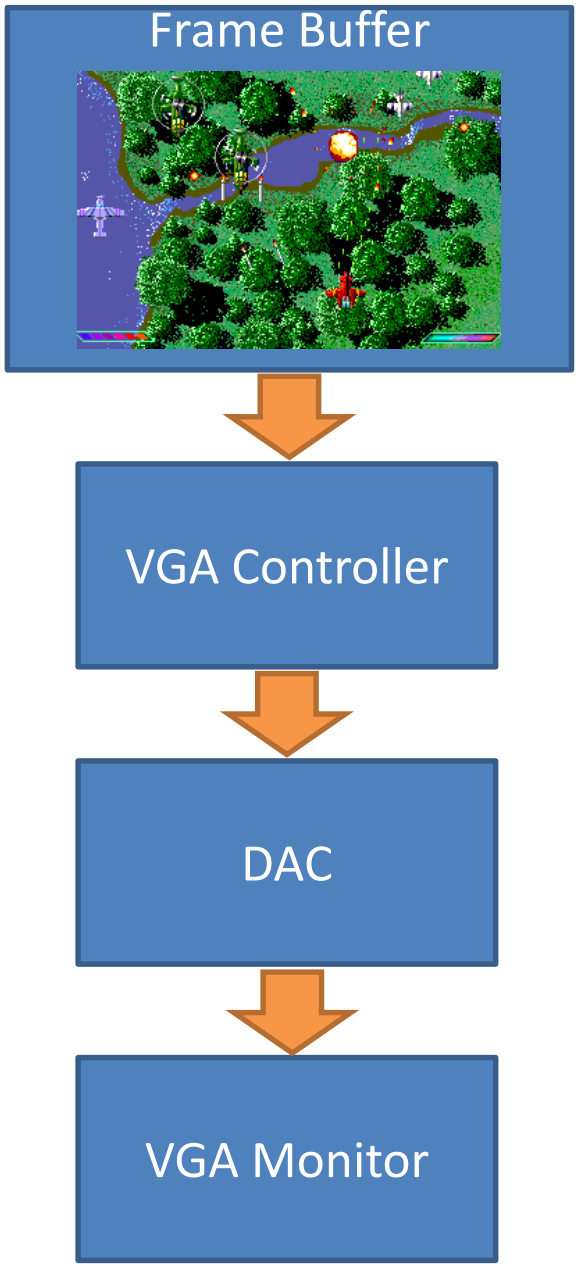
Frame Buffer

- Last time we talked about fixed function drawing, what about frame buffer?
- Frame buffer: a RAM which holds the contents of the screen to be drawn (every frame)
- Need memory large enough to hold the entire screen
- Every frame, VGA controller reads out contents of frame buffer and displays
- *Important concept*: frame buffer is erased and regenerated every frame!



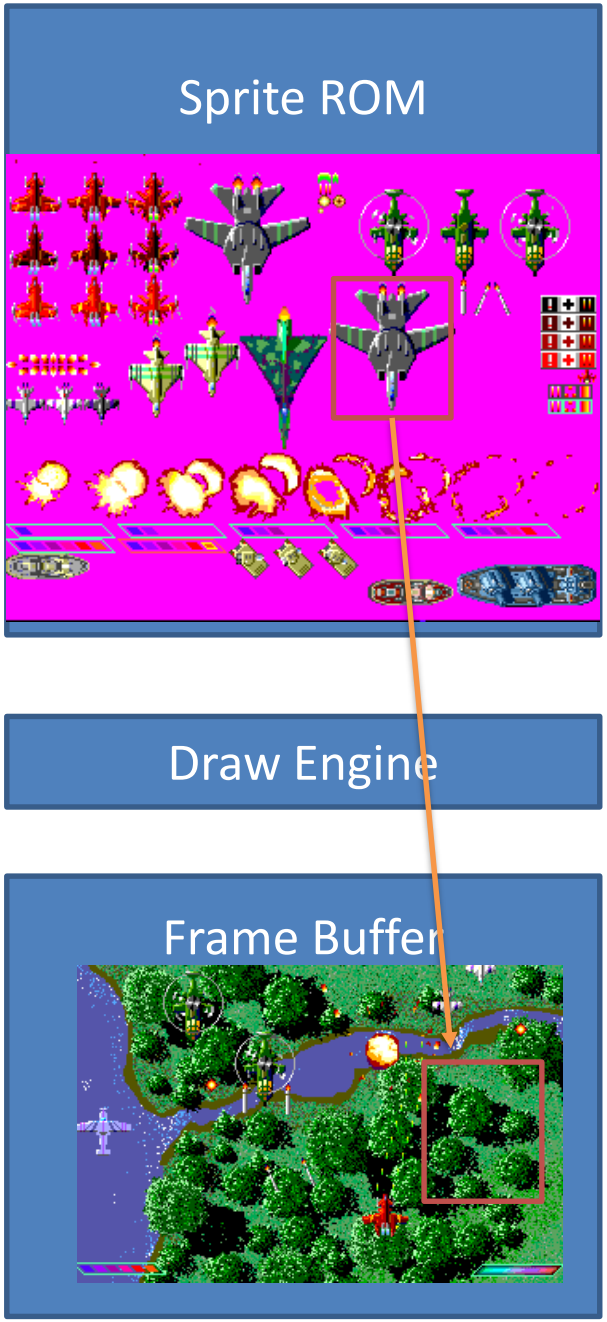
Frame Buffer (continued)

- VGA controller generates H and V sync signals (already provided in Lab 8 material)
- Output (data) of frame buffer memory goes to VGA_R/VGA_G/VGA_B signals (how you store in memory is up to you)
- Data should be read out in raster order (horizontal, and then down)
- Note that no data has to be read out during blanking intervals
- How fast do we need to read out of frame buffer RAM?



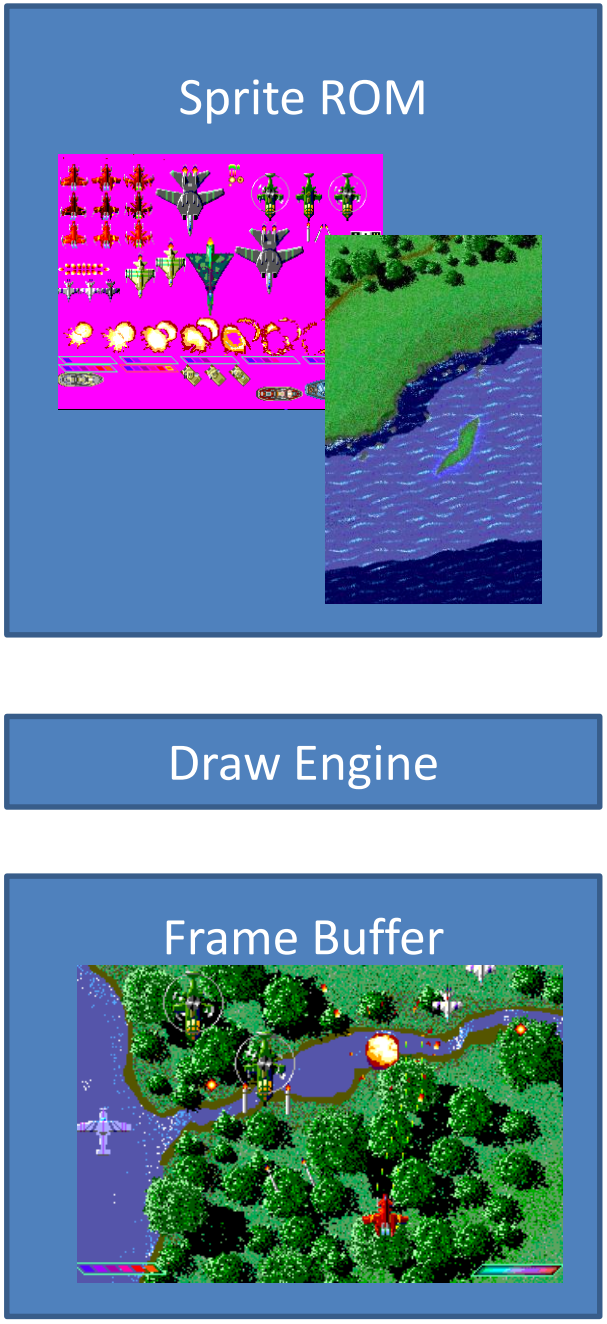
Frame Buffer Drawing

- Remember that frame buffer has to be regenerated every frame (60 times a second)
- Drawing engine is 2-D memory copy unit
- Copies 2-D region of memory from ROM into video RAM based on commands from CPU
  - Example command: copy 64x64 region starting from address 0x100005020 to location 230x110
  - Drawing engine (blitter) must be aware of screen and ROM layout to correctly handle clipping
  - Sprites may be arbitrary sizes and not constrained in number
- Performance is measured in megapixels/s drawn
- Can draw over the same region multiple times (why would we want to do this?)



Frame Buffer Approach (continued)

- Typically draw background once (e.g. draw 640x480 region from current background ROM address to screen position 0,0)
- Then draw sprites using same engine
- Sprites need to have some sort of "transparent color" which tells draw engine to ignore specific pixel (otherwise sprites will be drawn with bounding boxes)
- Must do drawing before VGA controller access pixel (otherwise will have flickering artifacts)
- Can either do all drawing in vertical / horizontal blank interval or use double buffering
  - Using blank intervals limits performance (can only write when VGA controller is idle)
  - Using double buffering limits memory, need to hold two screens



Memory Technology Comparison

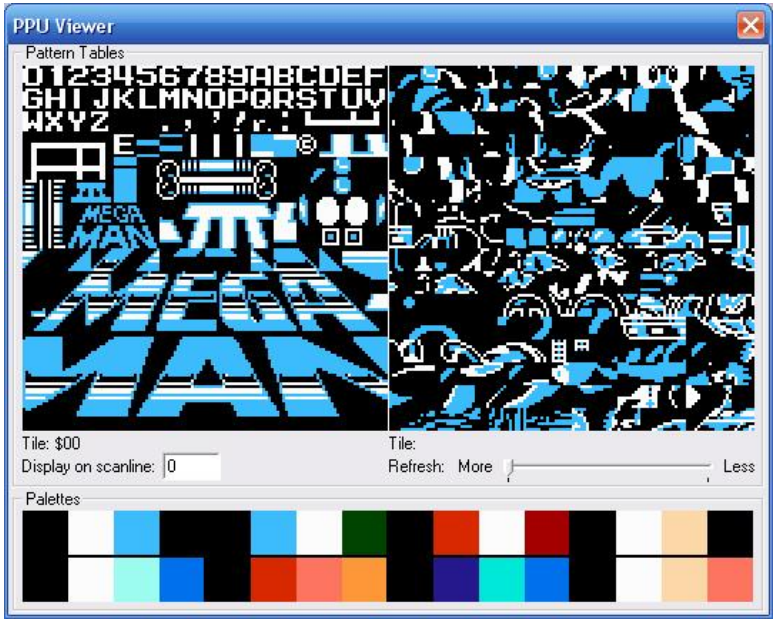
Technology	R/W	Size	Speed	Complexity	Initialization
On-Chip Memory (M9K)	R/W	9216 bits per block 432 blocks (3.888 Mbit)	Very Fast	Low	Directly in SV
SRAM	R/W	1M x 16 (16 Mbit)	Fast	Medium	Control Panel
SDRAM	R/W	32M x 16 x 2 (1 Gbit)	Variable	High	Control Panel
Flash	R (mostly)	8M x 8 (64 Mbit)	Fast (to read)	Medium	Control Panel

Synchronization and Usage of Memory Space

- Frame buffer typically uses dual port memory
- Only OCM (M9K) blocks are truly dual port (but not enough RAM for 24-bit color VGA frame)
- Need to create some kind of dual-port arbiter module
- This can be as simple as giving each port a even/odd clock cycle
  - Works well for SRAM since SRAM is agnostic to access patterns
  - Potentially very bad for Flash and SDRAM since they have a row/column architecture
- May need to run SRAM controller at higher frequency (e.g. 100 MHz to take advantage of 10ns access time)

Saving Memory (for OCM or SRAM)

- Can use palette to save memory
- Palette is just look-up-table for each color
- Normally each pixel requires 24-bits (8 red, 8 green, 8 blue)
- Palette maps smaller index (e.g. 8-bit number) to 24 bit color
- Many tools available online to generate palette (finds closest colors, quantizes colors, generates a table)
- Can use for animation as well (palette cycling)



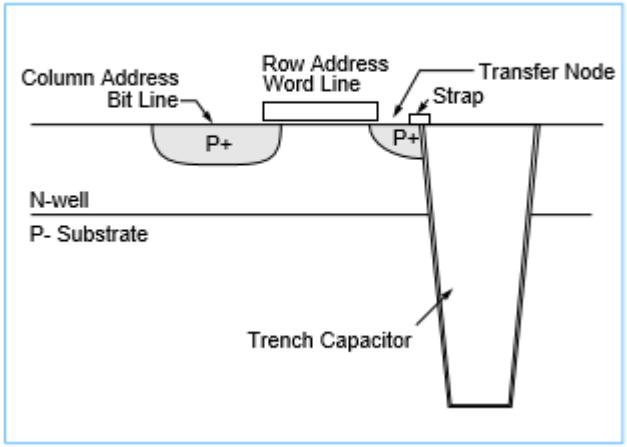
0	1	1	0
1	2	2	1
1	2	2	1
0	1	1	0

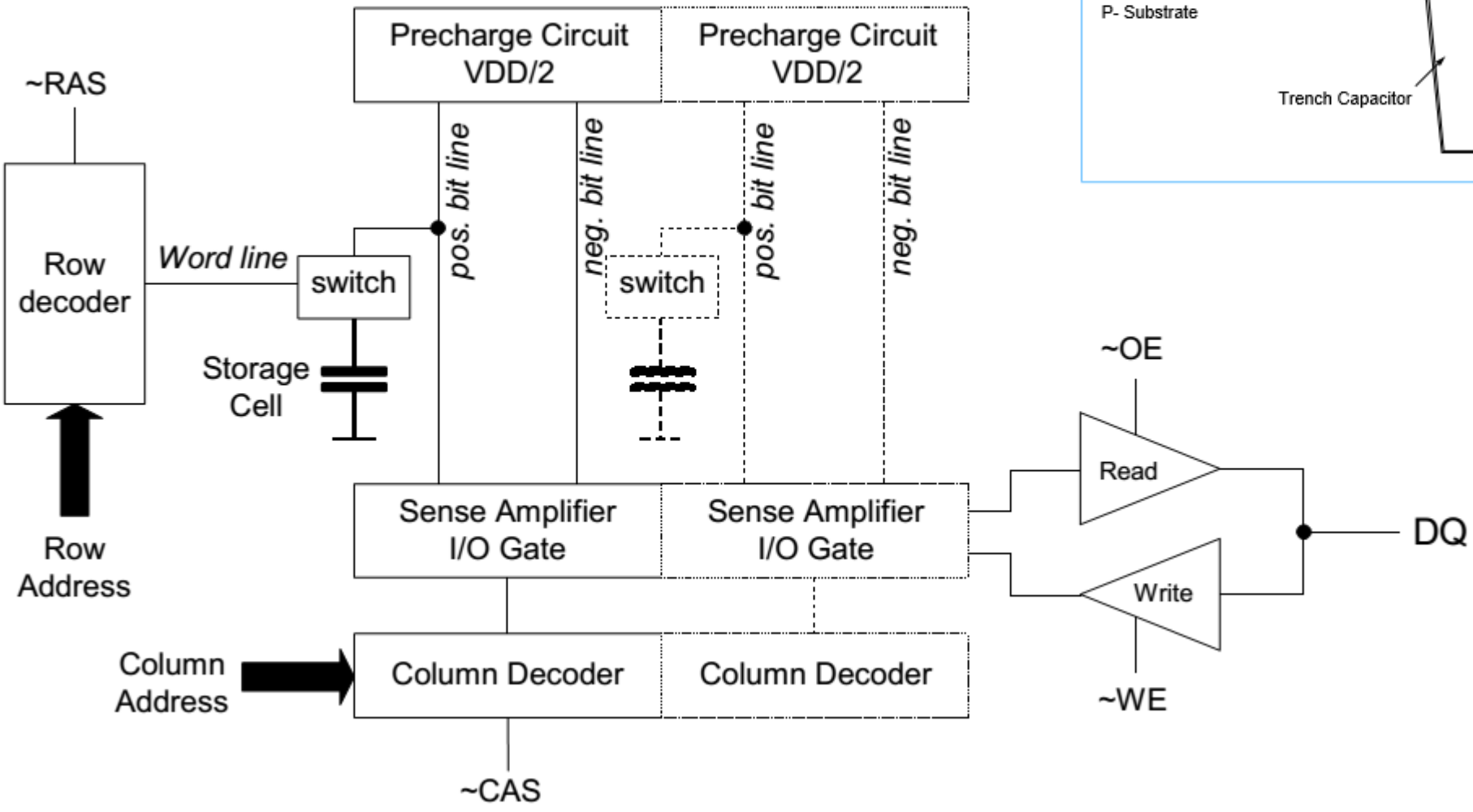
Introduction to SDRAM

- SDRAM stands for ***synchronous*** *dynamic random access memory*
- Dynamic memory = made from small capacitors instead of logic gates
  - Dynamic memory has much higher density than static memory
  - Must be refreshed periodically to hold value (typically once every couple of milliseconds)
  - Traditionally addressed with multiplexed address lines (in a row/column fashion due to high density)
- Synchronous = clocked
  - Responds to commands as signaled by CLE/CAS/RAS/WE
  - Typically driven by a state machine which is called the *SDRAM controller*
  - This abstracts the commands so that SDRAM looks like standard memory

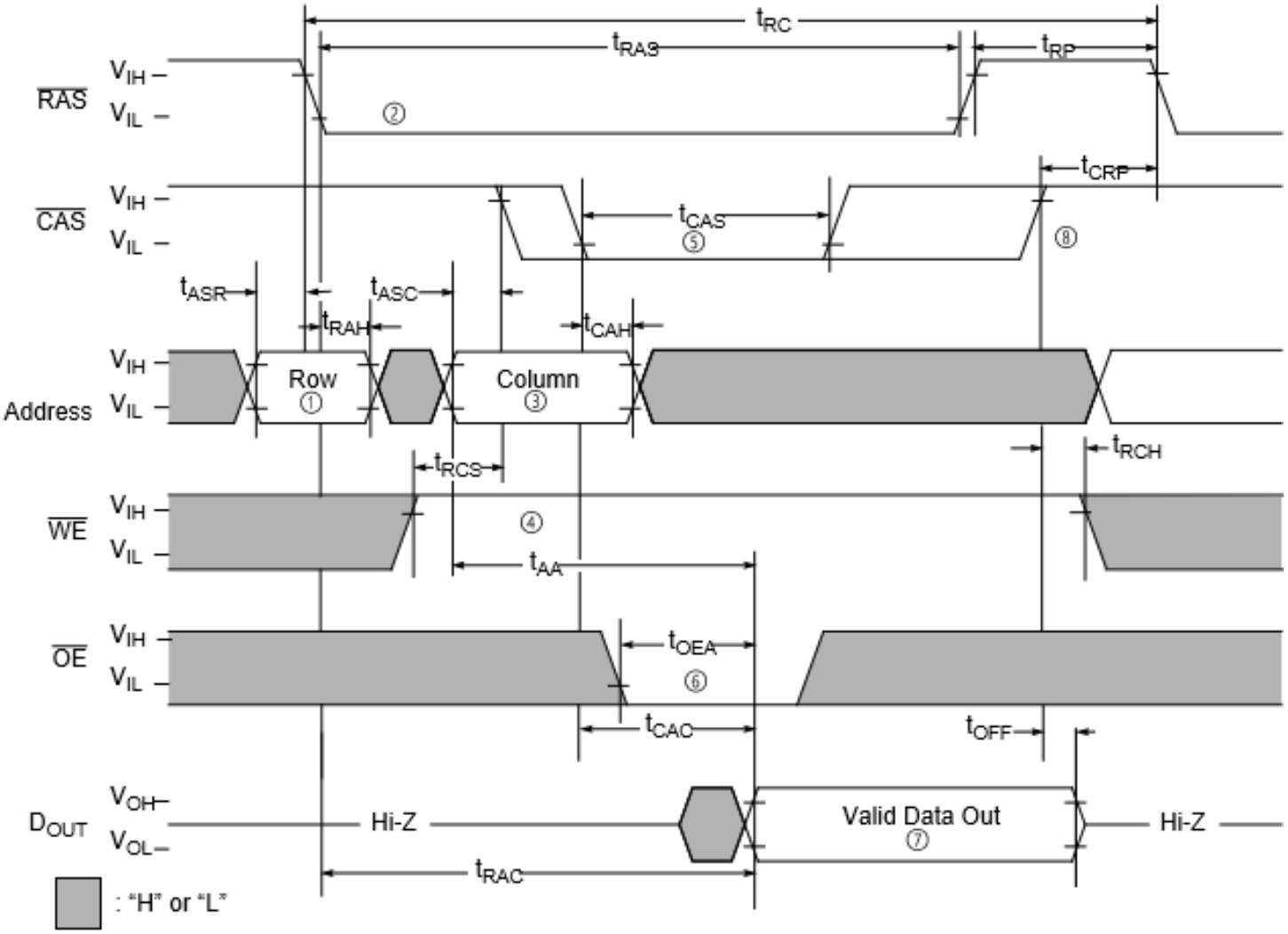
DRAM Basic Cell

Figure 1: IBM Trench Capacitor Memory Cell

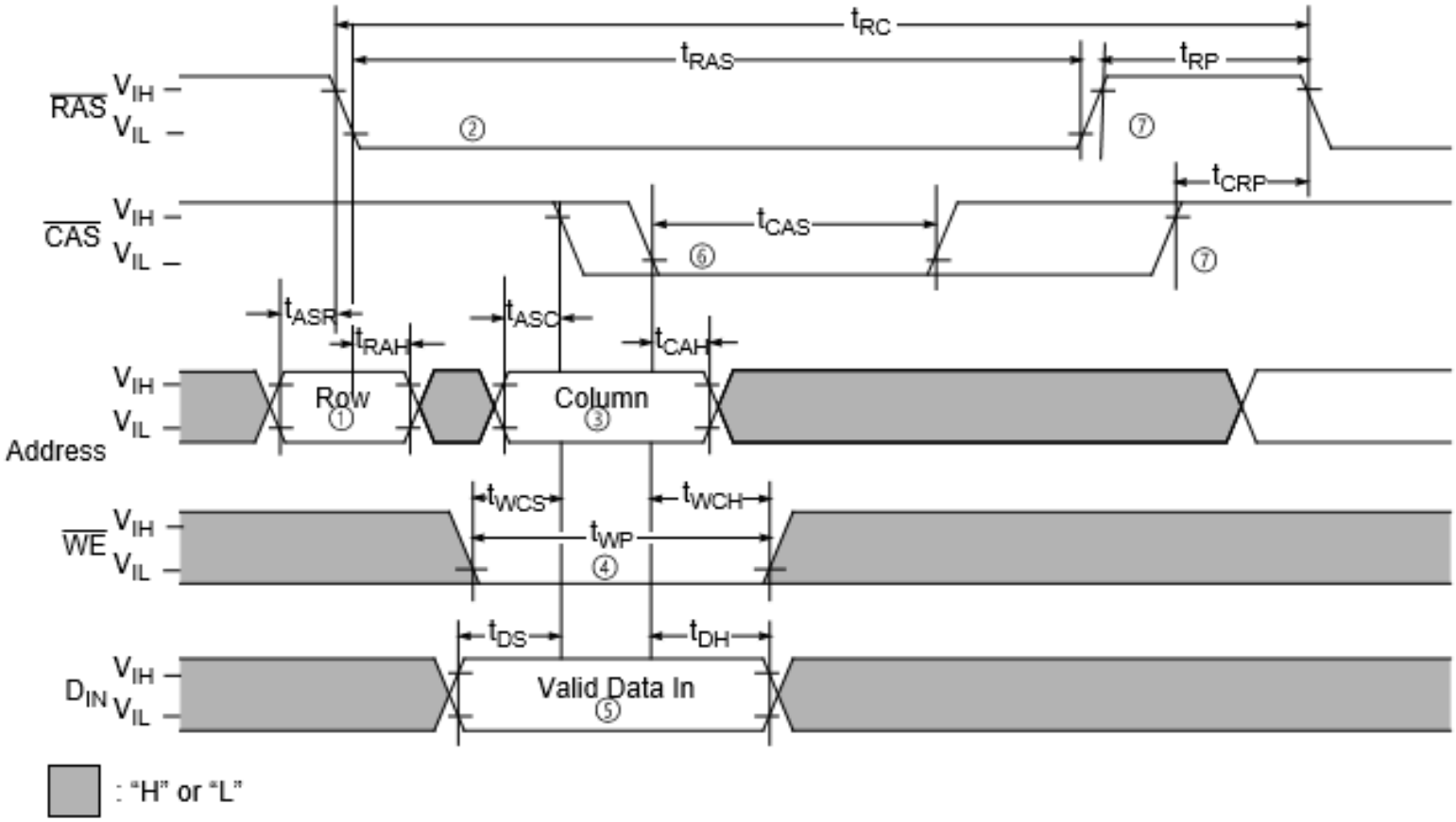




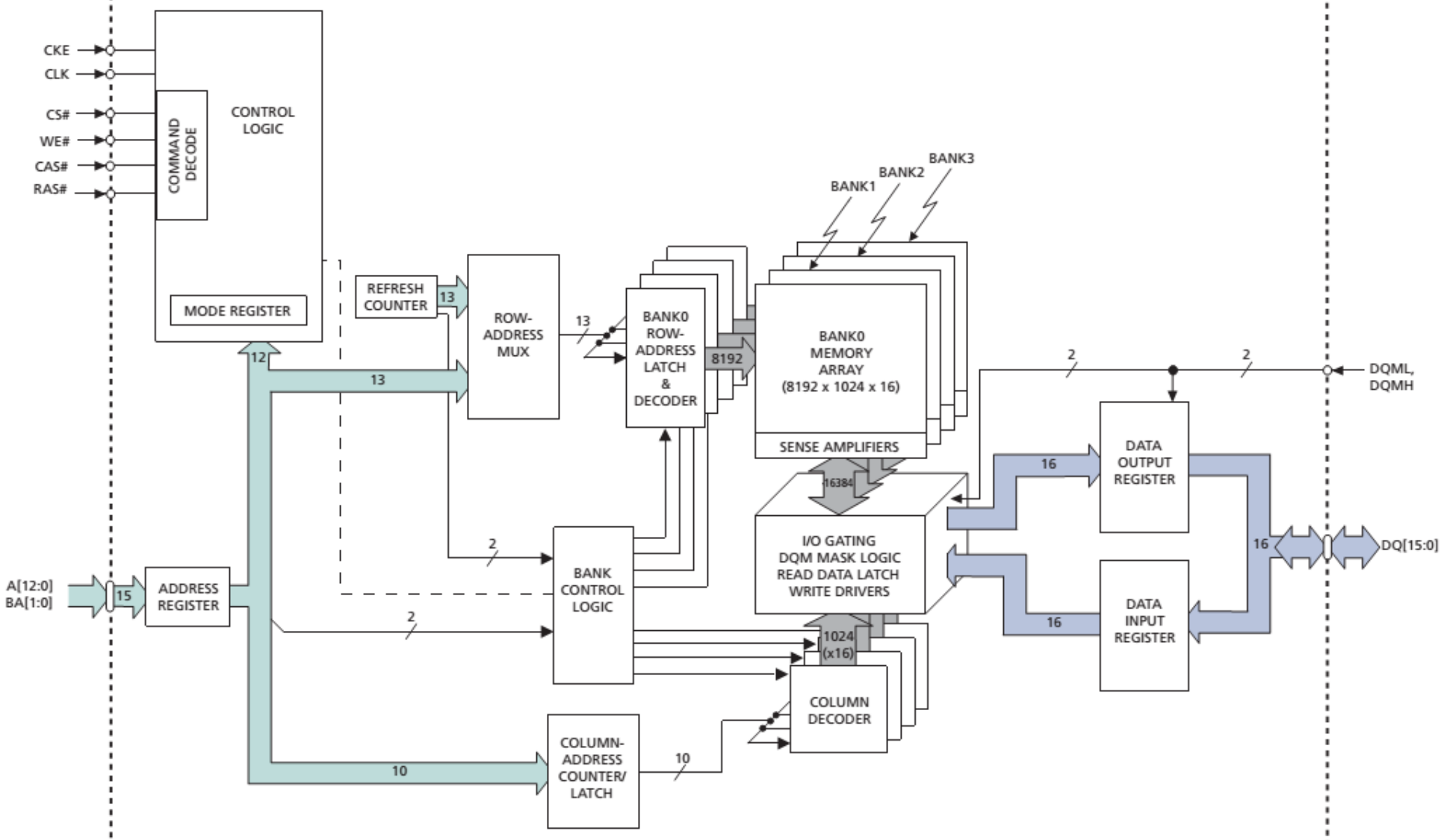
DRAM Read Cycle



DRAM Write Cycle



SDRAM Structure



SDRAM Commands

- Commands are combinations of CLE/CAS/RAS/WE
- Note that SDRAM addresses are multiplex, typically rows are selected before columns using the same address pins

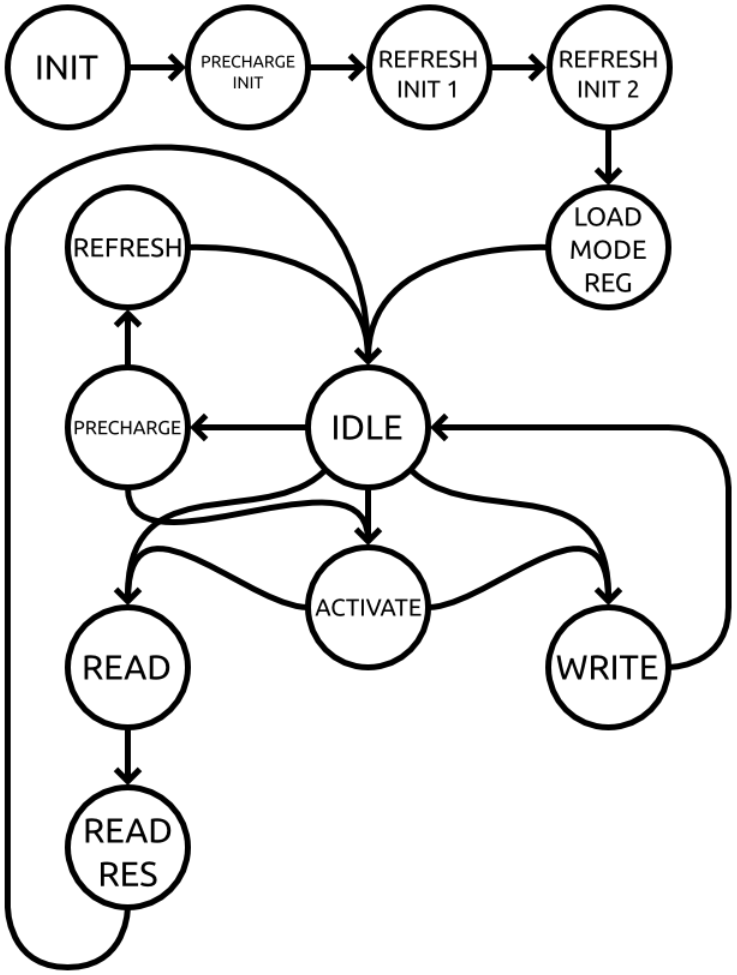
Table 13: Truth Table – Commands and DQM Operation

Note 1 applies to all parameters and conditions

Name (Function)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQ	Notes
COMMAND INHIBIT (NOP)	H	X	X	X	X	X	X	
NO OPERATION (NOP)	L	H	H	H	X	X	X	
ACTIVE (select bank and activate row)	L	L	H	H	X	Bank/row	X	2
READ (select bank and column, and start READ burst)	L	H	L	H	L/H	Bank/col	X	3
WRITE (select bank and column, and start WRITE burst)	L	H	L	L	L/H	Bank/col	Valid	3
BURST TERMINATE	L	H	H	L	X	X	Active	4
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	X	Code	X	5
AUTO REFRESH or SELF REFRESH (enter self refresh mode)	L	L	L	H	X	X	X	6, 7
LOAD MODE REGISTER	L	L	L	L	X	Op-code	X	8
Write enable/output enable	X	X	X	X	L	X	Active	9
Write inhibit/output High-Z	X	X	X	X	H	X	High-Z	9

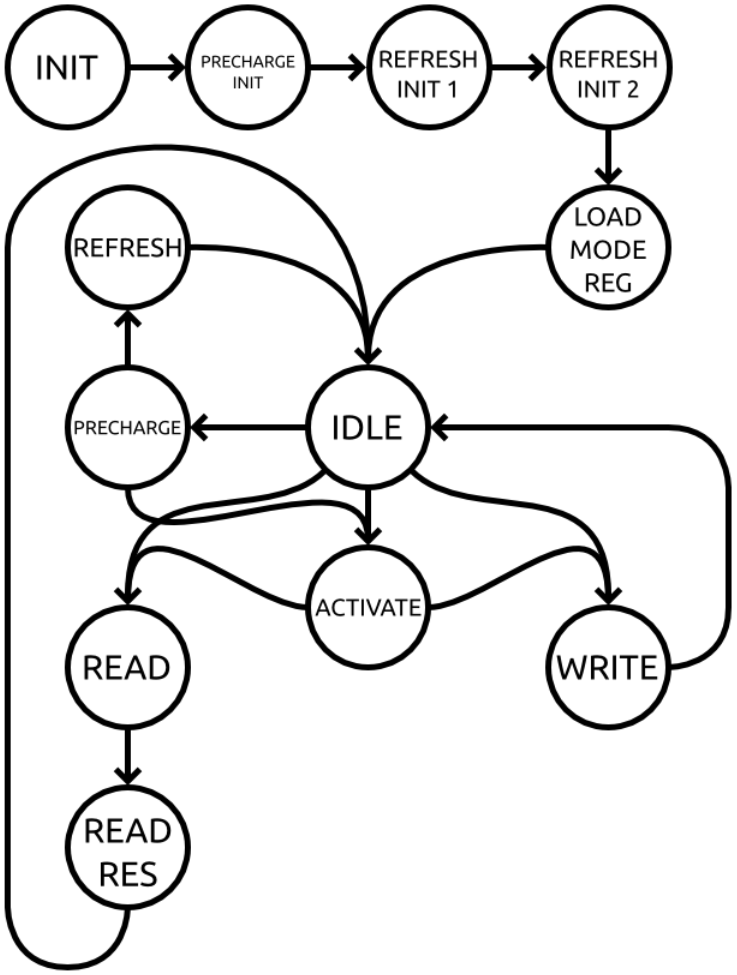
SDRAM Controller State Machine

- SDRAM operates using commands
- Controller needs internal timer to know when to send refresh operation
- SDRAM chip needs to be initialized to get into the IDLE state
- All SDRAM operations happen into a single row
- ACTIVE command initializes row for R/W



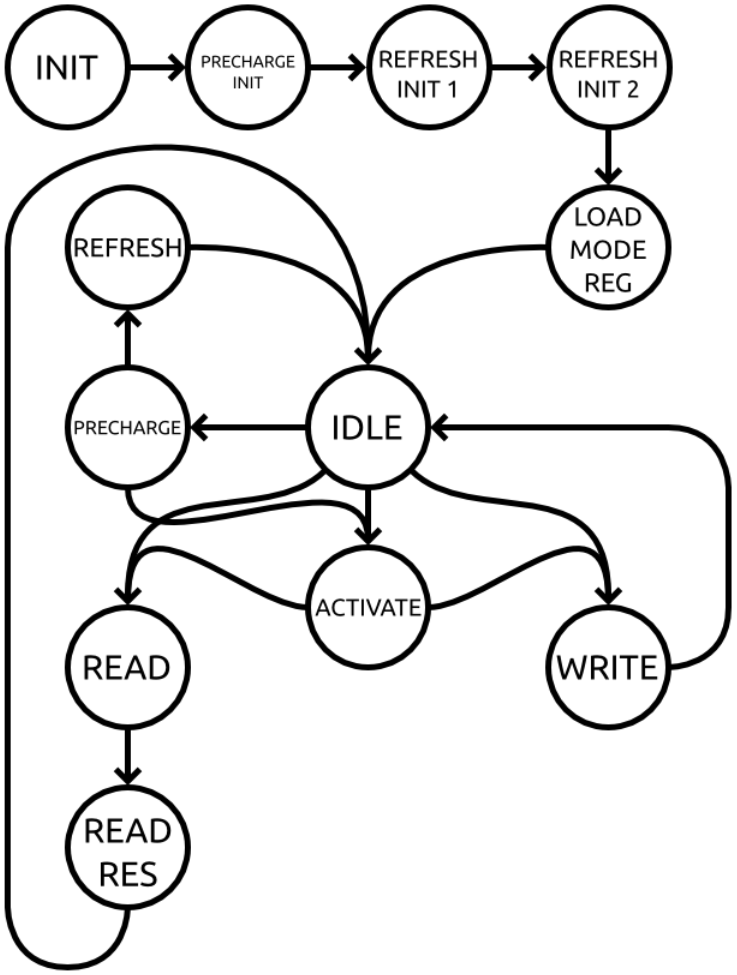
SDRAM Controller State Machine

- SDRAM can read/write words (columns) into a open row quickly
- SDRAM can also automatically read/write successive words without the need for a column address (burst operation)
- In order to read into another closed row, currently open row must be pre-charged to close



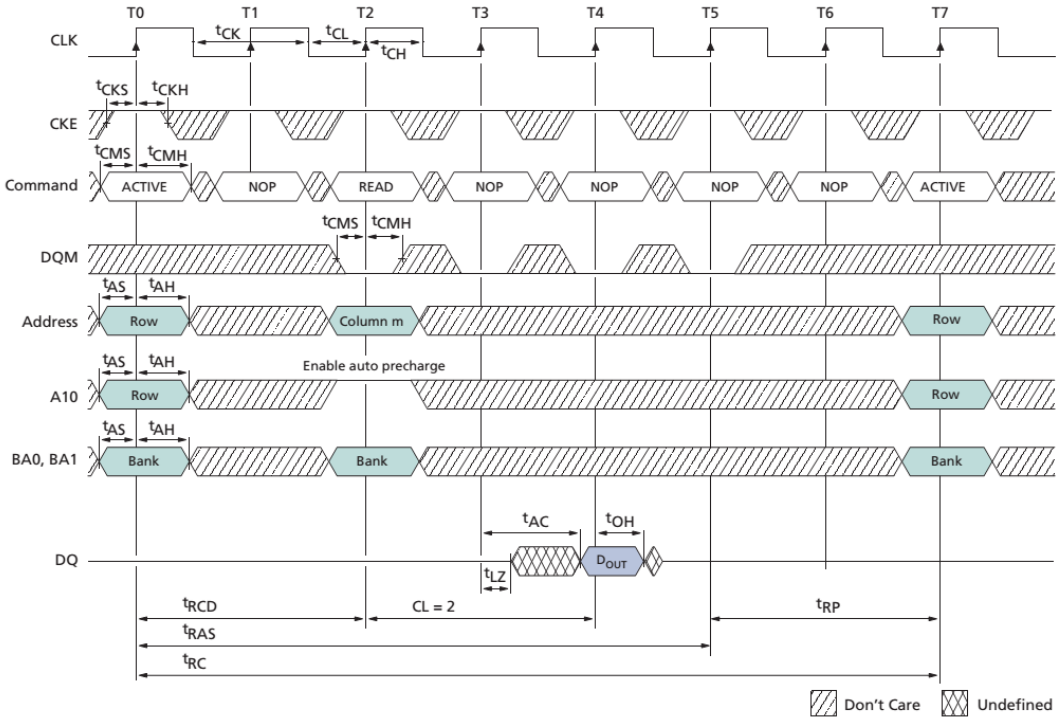
SDRAM Controller State Machine

- Controller must also manage refresh, which is handled per row
- Typically, each row needs to be refreshed every couple of milliseconds <100ms
- This means controller must keep track of when row was last refreshed and which row to refresh next



SDRAM Timing

- Because the SDRAM is synchronous, it requires a certain relationship between block and data to not violate setup and hold times
- That relationship is difficult to maintain when the SDRAM chip is physically off chip from FPGA (differences in PCB routing, FPGA I/O buffer delays, etc.)
- Use PLL (phase locked loop) to shift phase of the SDRAM's clock relative to data (settings from Experiment 7 tutorial were provided by the manufacturer)



Using SDRAM in FPGA Designs

- DE2-115 has 2*32Mx16 SDRAM chips wired in parallel
- Behaves like single 32Mx32 SDRAM
- You can use Altera's SDRAM controller if you export the Avalon MM port from Qsys
- Will need to place NIOS II memory somewhere else if this is the case, simple to use on-chip memory
  - Remember to make OCM sufficient size for your ELF file (~64KB should be enough)
  - Relocate the reset vector so your code executes out of OCM
  - Performance should be higher as well than execution from SDRAM
- Check out Avalon Interface Specifications to do this, you will need to create a Avalon MM master for your FPGA logic to use SDRAM