ECE 385 – Digital Systems Laboratory

Lecture 15 – VGA Continued and Sprite Drawing

Zuofu Cheng

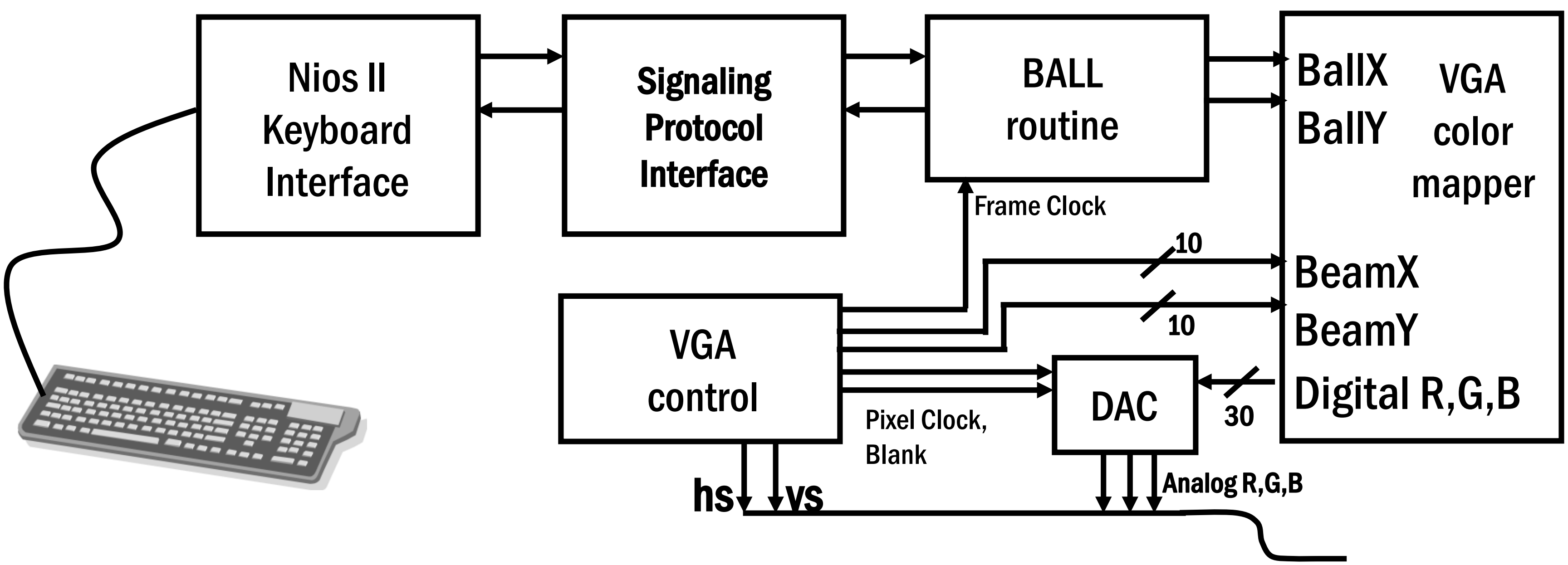
Spring 2018

Link to Course Website

Experiment 8 Goals

- Create low-level interface between NIOS II and USB chip (CY7C67200 "EZ-OTG")
- Connect USB keyboard to "USB Host" port on DE2-115 and be able to enumerate & read key-codes
- Display bouncing ball using VGA controller on monitor (connect to VGA port)
- Use key-codes to control bouncing ball

Experiment 8 Overall Block Diagram



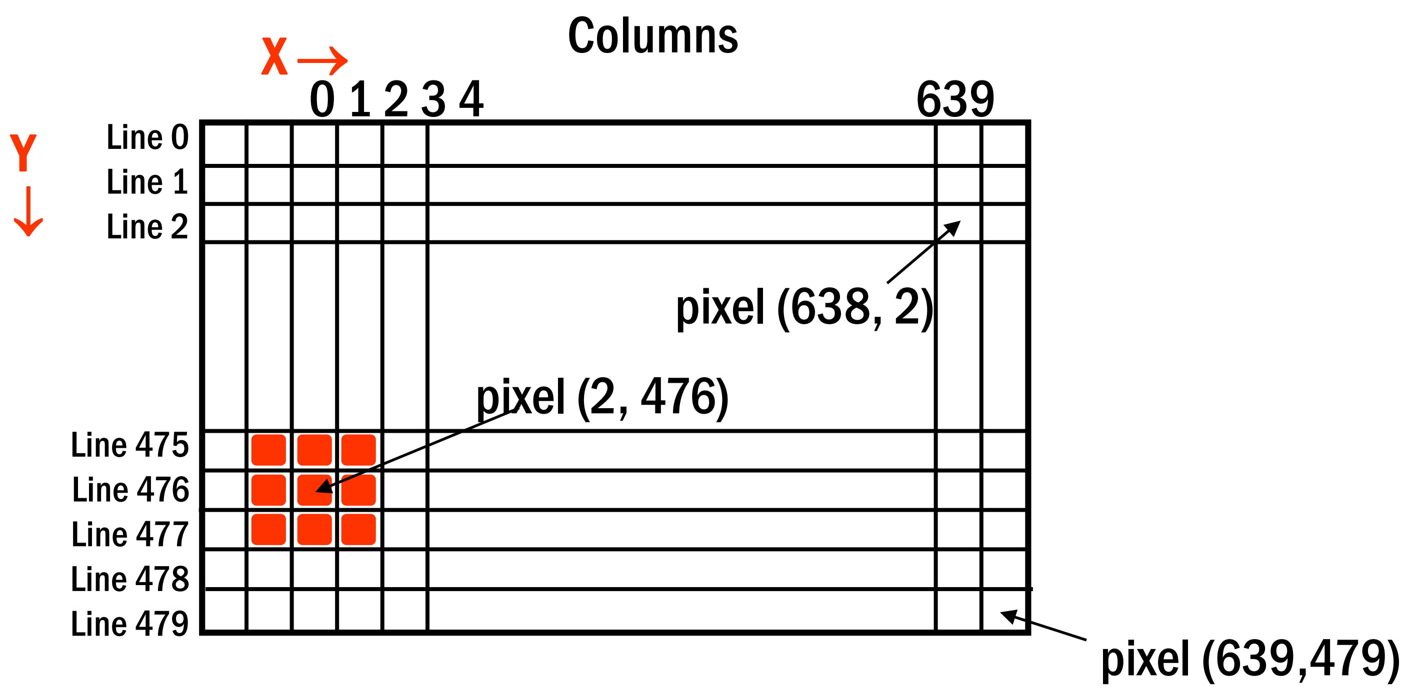
Ball routine: partially given

Color mapper: given

VGA controller: given

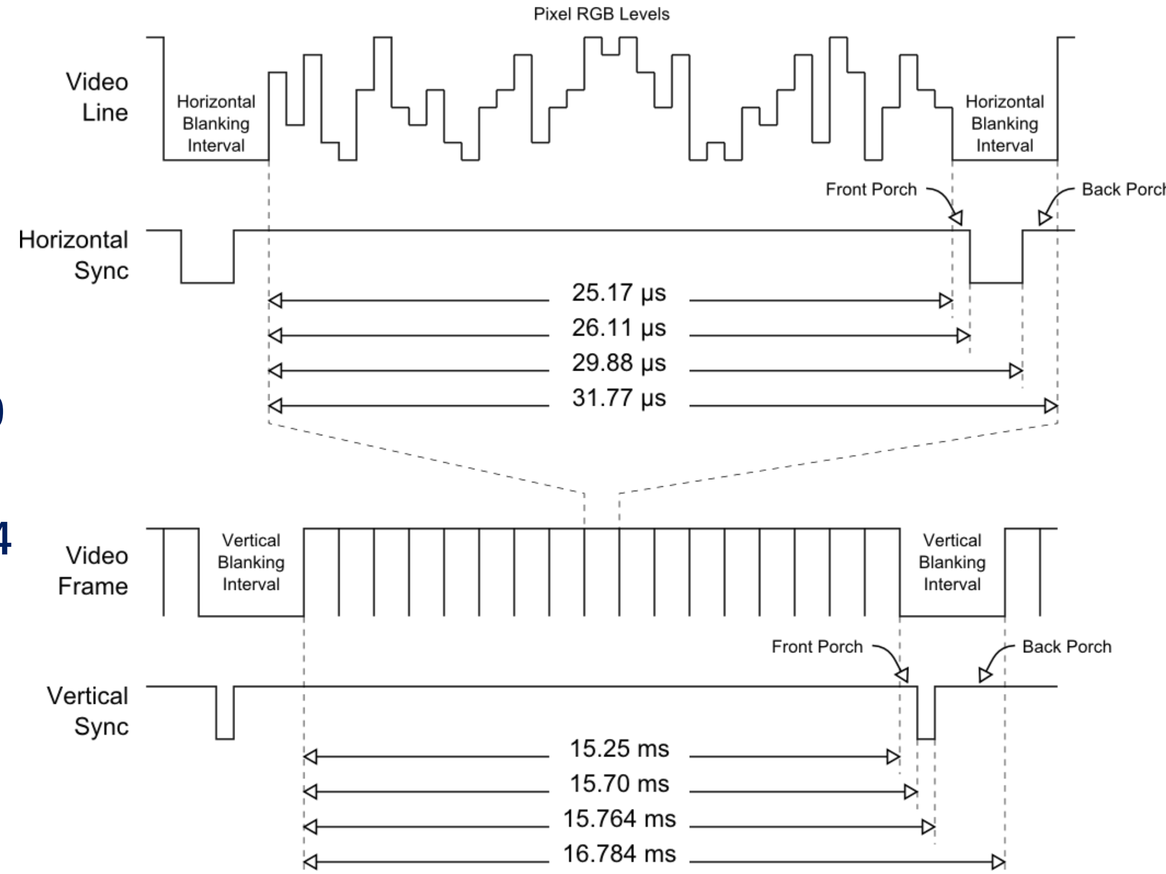
VGA Monitor Operation

- 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



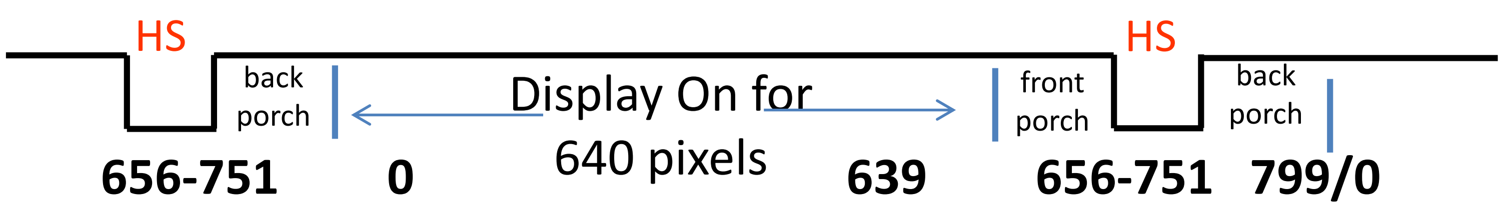
VGA Timing (continued)

- 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)
- Generate by PLL (why is this better?)



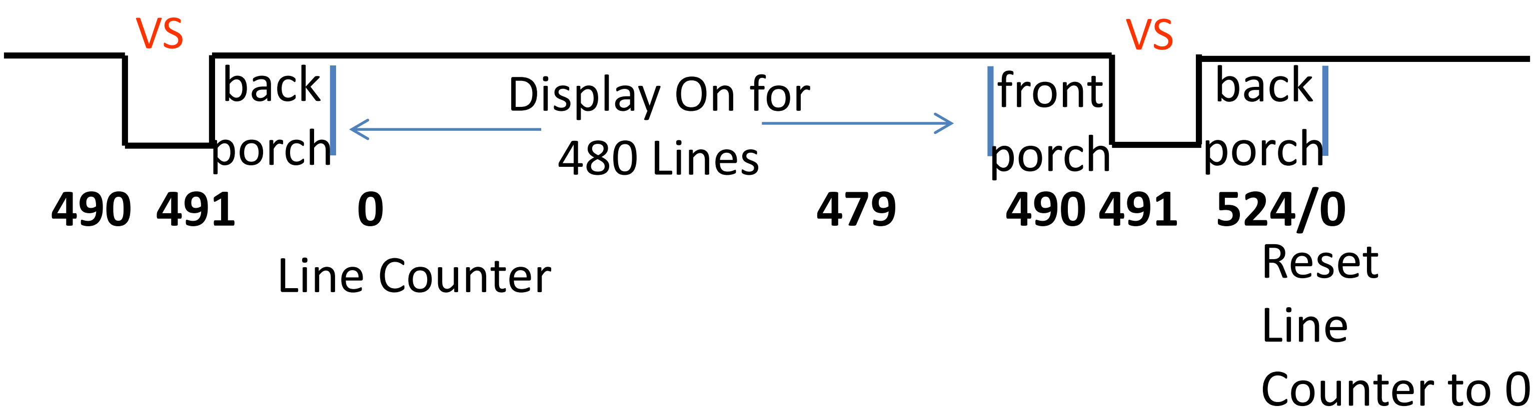
VGA Horizontal Timing

- To generate a Horizontal Sync Pulse, use a 10-bit pixel counter modulo-800
  - Counter increments with a 25MHz clock (pixel clock)
  - Pixel Counts <0 thru 639>: Display On
  - Pixel Counts <640 thru 799>: Display Off
  - Pixel Count <656 thru 751>: HS Pulse Active for 96 pixels
  - Pixel Count 799: Reset pixel counter
  - HS Pulse is <u>Active Low</u> for most monitors



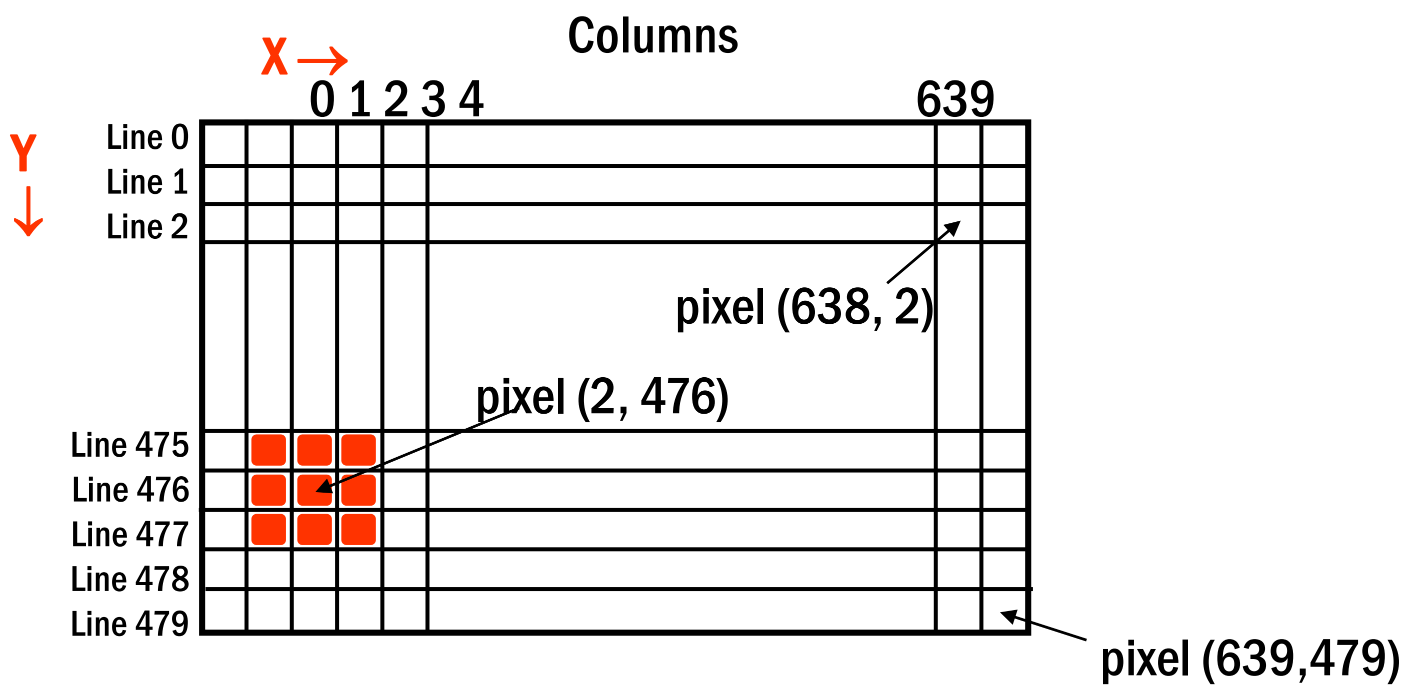
VGA Vertical Timing

- To generate Vertical Sync Pulse (VS), start a 10-bit Line Counter modulo-525
  - Counter increments every 800 pixels
  - Line Counts <0 thru 479> : Display On
  - Line Counts <480 thru 524>: Display Off
  - Line Counts 490 and 491: VS Pulse Active
  - Line Count 524: Reset Line Counter
  - VS Pulse is Active Low for most monitors



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 satisified

Limitations of Simple Approach

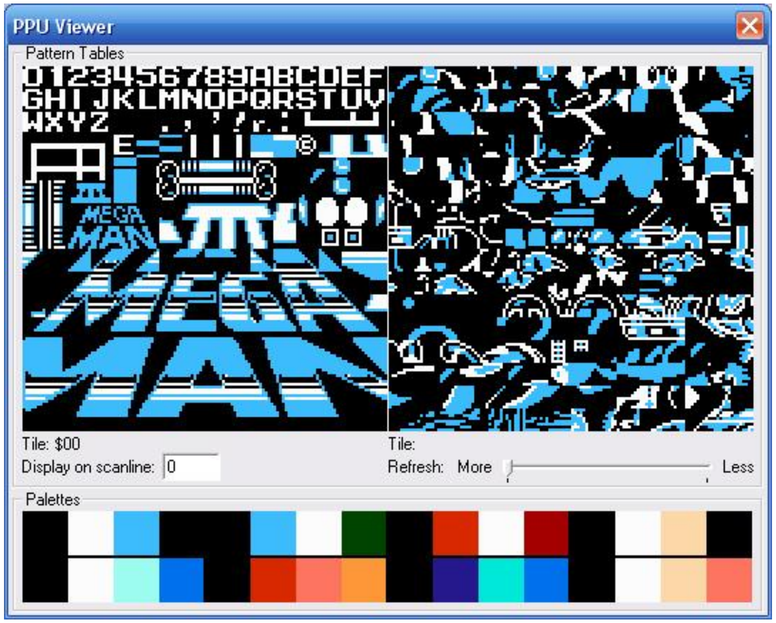
- If we strictly draw based on H and V pixel positions, we can only draw boxes
- What if we want to draw more sophisticated graphics (circles, fonts, spaceships, Mario?)
- We do not want to instantiate logic which describes everything we would ever want to draw – want instead to make *generalized* hardware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

Case Study: Ricoh RP2C02

- Ricoh RP2C02 (NES PPU)
- Break up background and foreground graphics into 8x8 or 8x16 tiles
  - Example tiles are shown on right
  - To further save memory, tiles saved in a paletted format
  - A palette is a look up table which maps an index (e.g. 5 bit for 32 colors) to a full range color (e.g. 24 bit for true color)
  - Can do graphical effects by swapping palettes (fade screen, animated water or fire)
- PPU keeps track of horizontal and vertical position and draws background tiles according to a *nametable*, which maps tile position to tile index
  - Background tiles are drawn strictly on boundaries
  - Reduces memory required to keep track of background tiles



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

Case Study: Ricoh RP2C02 (continued)

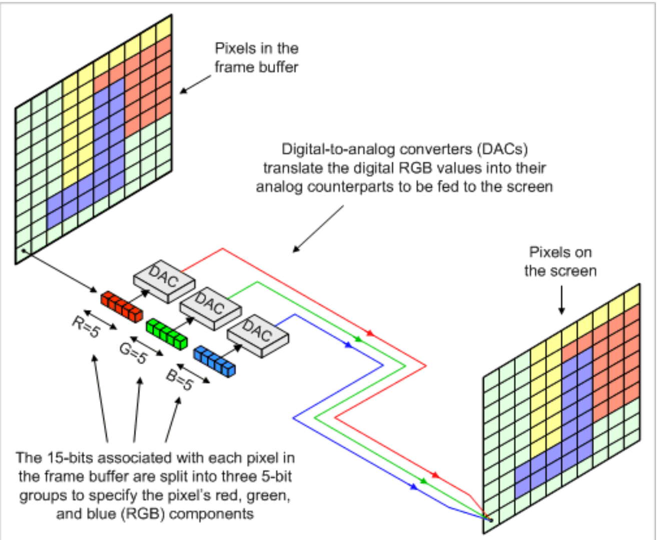
- Foreground drawn using limited number of hardware sprites (hardware graphic)
  - PPU supports 64 total sprites (8 per line max)
  - Each sprite is drawn from *object attribute memory*
  - Each OAM is a set of registers which keeps track of each hardware sprite
  - Data includes:
    - Index of tile to draw
    - Which palette to use
    - Position of sprite (sprites are not quantized to tile boundaries)
    - Optional bits to flip sprite vertically or horizontally
    - Priority (whether sprite is in front/behind background)
  - Each hardware sprite has dedicated hardware to detect whether it should draw, MUX selects between different sprites according to priority and sprite #

PPU Raster Process

- Each hardware sprite keeps track of position range of itself based on position in OAM and system wide X and Y counter
- Background unit also keeps track of which tile it should be drawing based on X and Y counter
- When a hardware sprite has detected that it should currently be drawing it asserts an select pin to a wide MUX which gives it control over the color mapper
- Color mapper then draws according to sprite tile and palette until X and Y counter have moved out of draw range for given hardware sprite
- If no sprites assert select, select defaults to background unit to draw background tiles
- All tile data may be stored in ROM, total RAM requirement is low (background tables, OAM, palettes, etc…)

Frame Buffer Approach

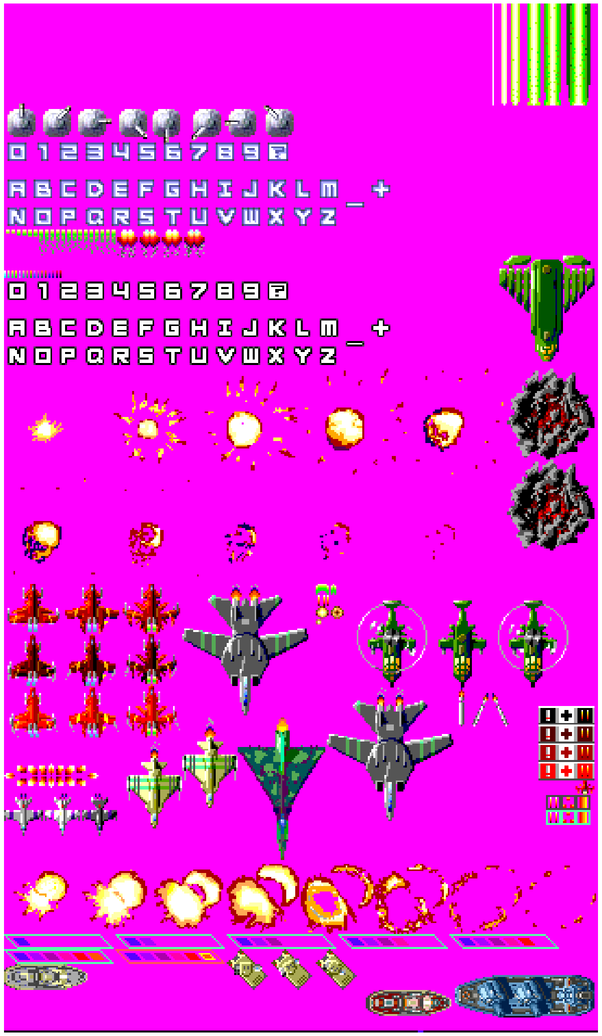
- Find enough RAM which can hold all pixels in frame - for 640*480*24 bit, how much RAM is required?
- RAM must be at least twice as fast as pixel clock to be able to do full frame animation – why?
- One side of RAM -> Video DAC
- Other side of RAM -> Drawing engine
- Any suitable RAM spaces on the DE2?



Graphical representation of 15-bit color

Frame Buffer Approach (continued)

- 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 blitter 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