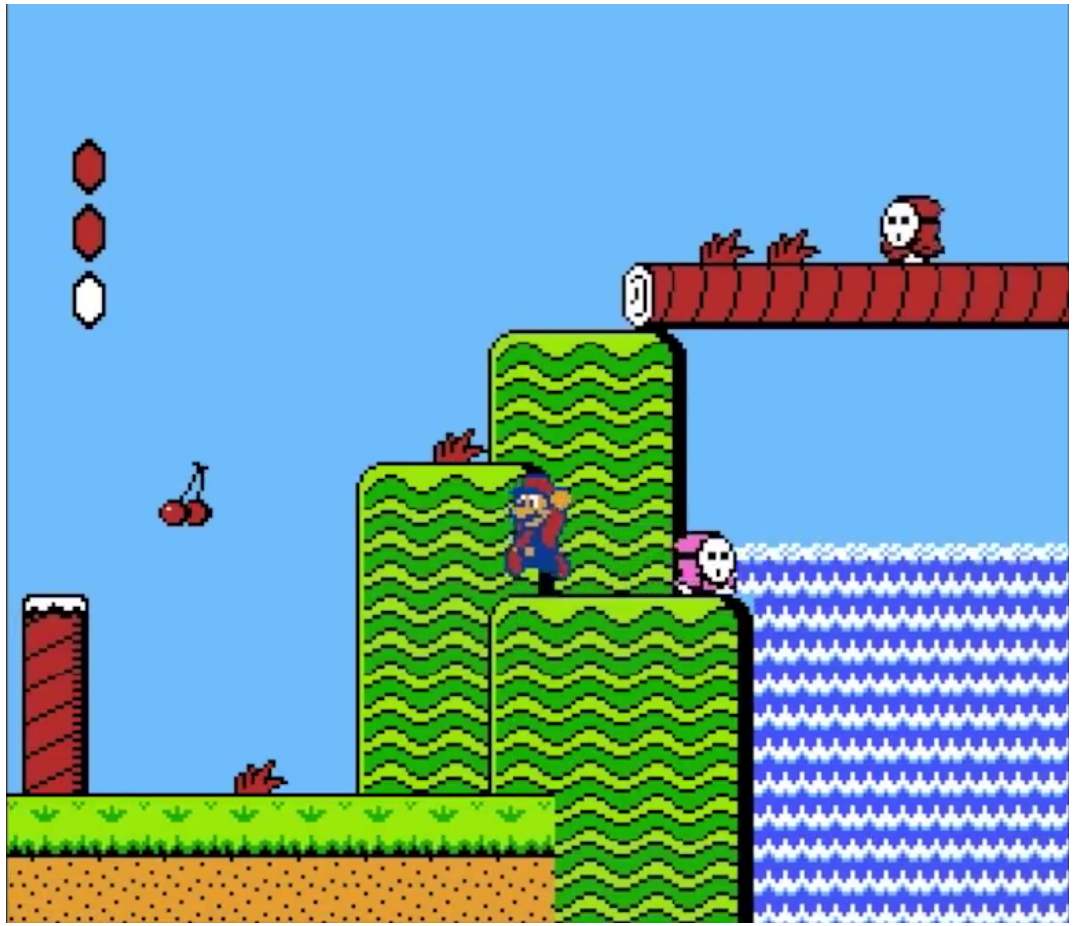
ESE519: Real-Time and Embedded Systems

Lecture 10: LCD Graphics and Touchscreens



Rahul Mangharam

Electrical and Systems Engineering

Computer and Information Science

University of Pennsylvania

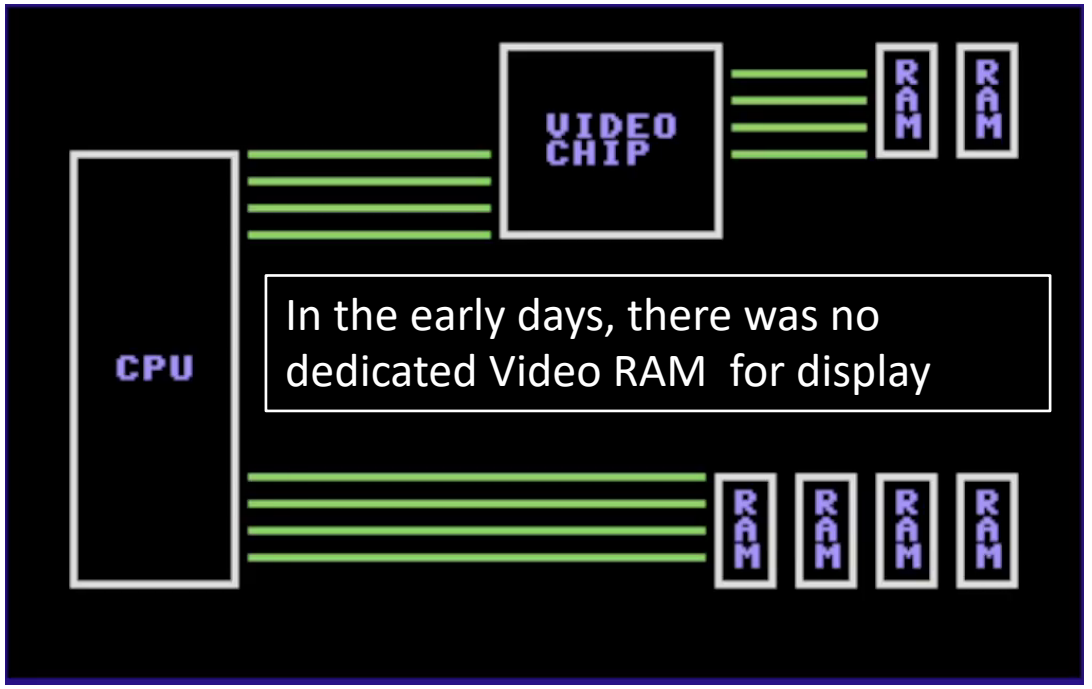
LCD Graphics and Touchscreens

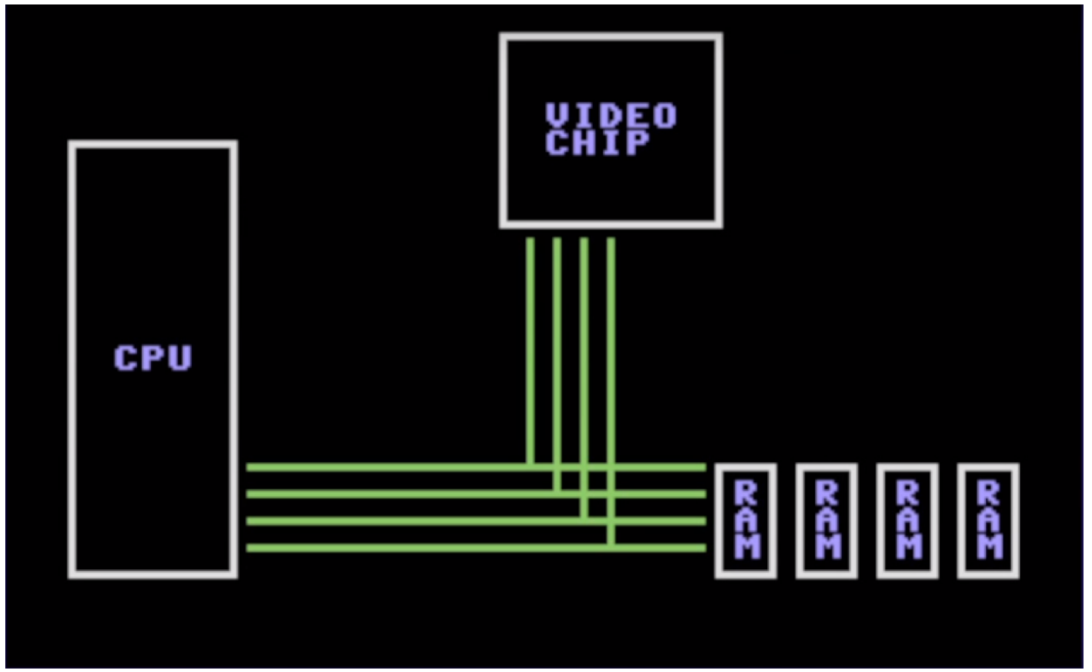
1. 8 Bit Graphics worked
2. Different ways to display
3. Example case – 128x64 LCD display
4. Understanding how LCD works
5. How to display text and graphics on LCD
6. Intro to Touchscreens
7. How to interface and calibrate touchscreen

History of 8-bit MCUs

- 8-bit MCUs in 1980s shared the RAM
- There was no separate memory for display
- Video chip shared screen memory with CPU

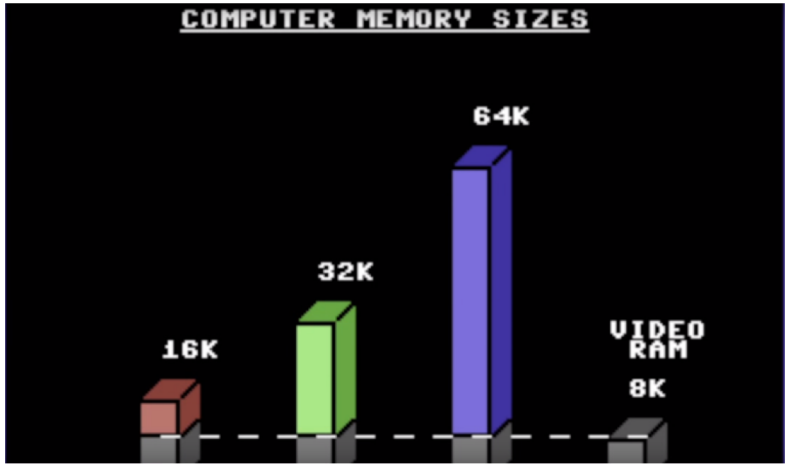
- Usually, RAM was 16KB or 32KB , best ones had 64KB
- Standard resolution of screens was 320 x 240 pixels

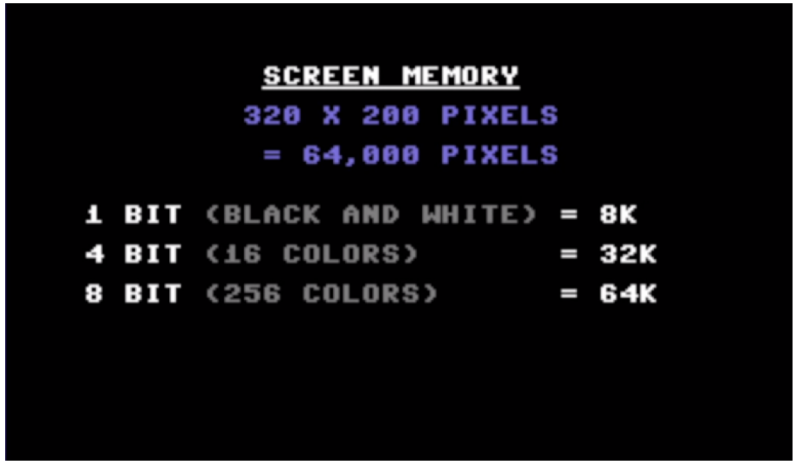


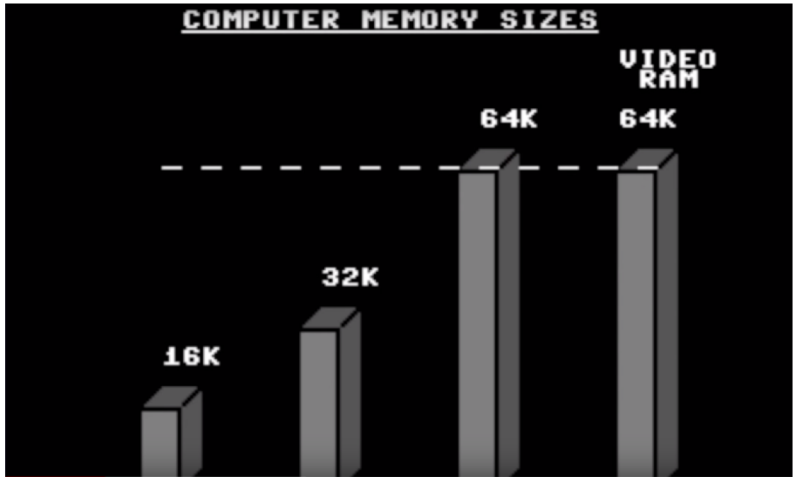


History of 8-bit MCUs

- For 320 x 240 pixels resolution , total no of pixel data will be 64000 pixels
- Considering 1pixel represented by 1 bit, total Video RAM needed to display screen alone will be 64Kb or 8KBytes.
- Essentially :
  - 1 Bit (black and white) = 8KBytes
  - 2 Bit (4 colors) = 16KBytes
  - 4 Bit (16 colors) = 32KBytes
  - 8 Bit (256 colors) = 64KBytes
  - 24 Bit (million colors) = 192KBytes ➔ 'impossible'
- 24 Bit colors like in modern displays was impossible to produce at that time







How were graphics like this generated then ?

How were graphics generated in 8-bit?

Three methods were popular :

1. Color cells
   - https://en.wikipedia.org/wiki/List_of_8-bit_computer_hardware_palettes
2. NTSC Artifact colorization
   - https://en.wikipedia.org/wiki/Composite_artifact_colors
3. CPU Driven graphics
   - https://en.wikipedia.org/wiki/Atari_8-bit_family_software-driven_graphics_modes

- A common theme between these different techniques was the balance between screen resolution or colors on the screen

+ A low resolution (graphics for games etc.) would be mixed with more colors

OR

+ A sharper display (for text editing etc.) would be monochrome mostly

1. Color Cells

There are three ways to do this :

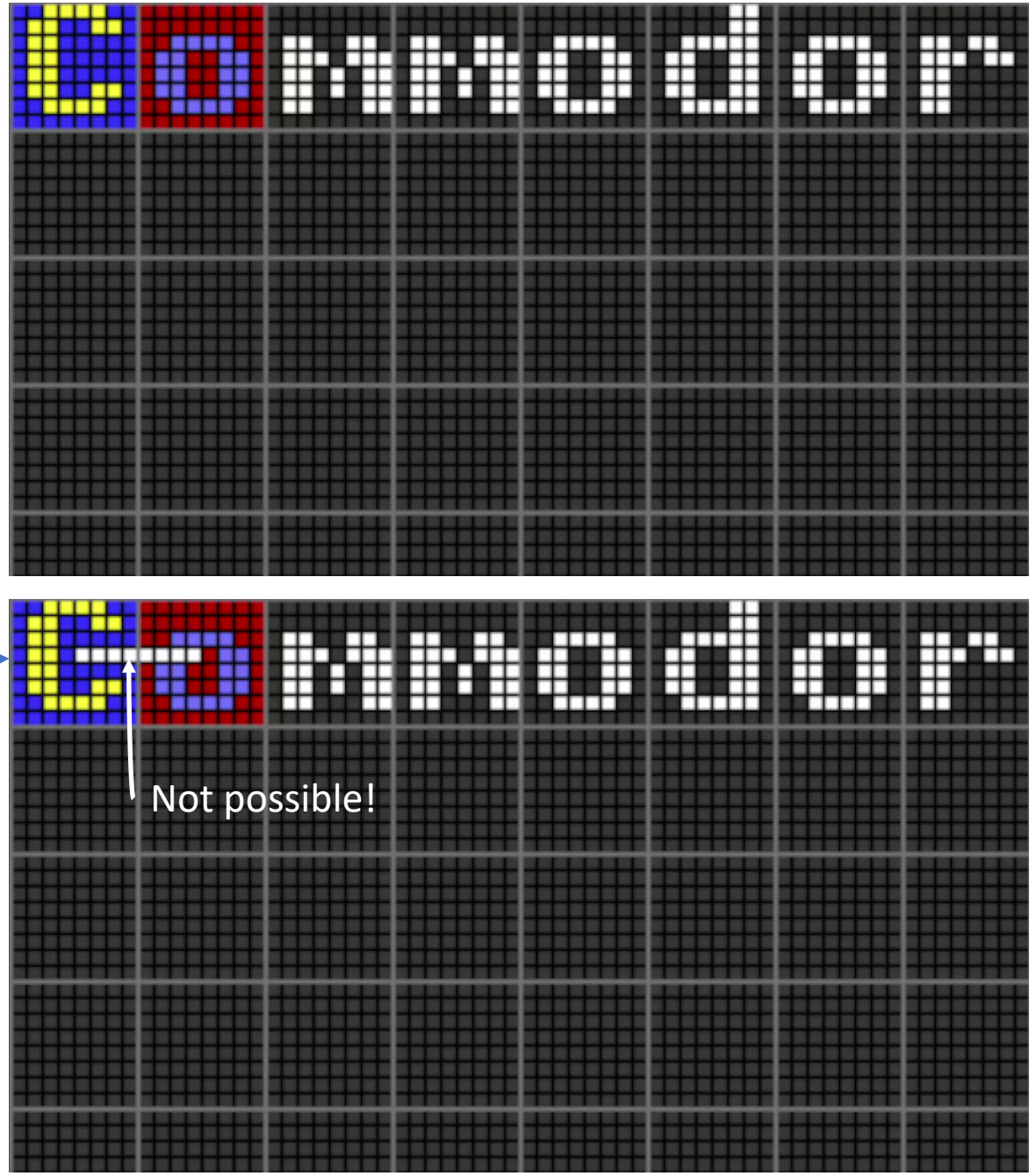
1. 2-Color mode :
   - Area of the screen divided into cells 8 pixels wide and 8pixels tall

2. Multicolor mode :
   - Lose resolution by doubling the pixel widths

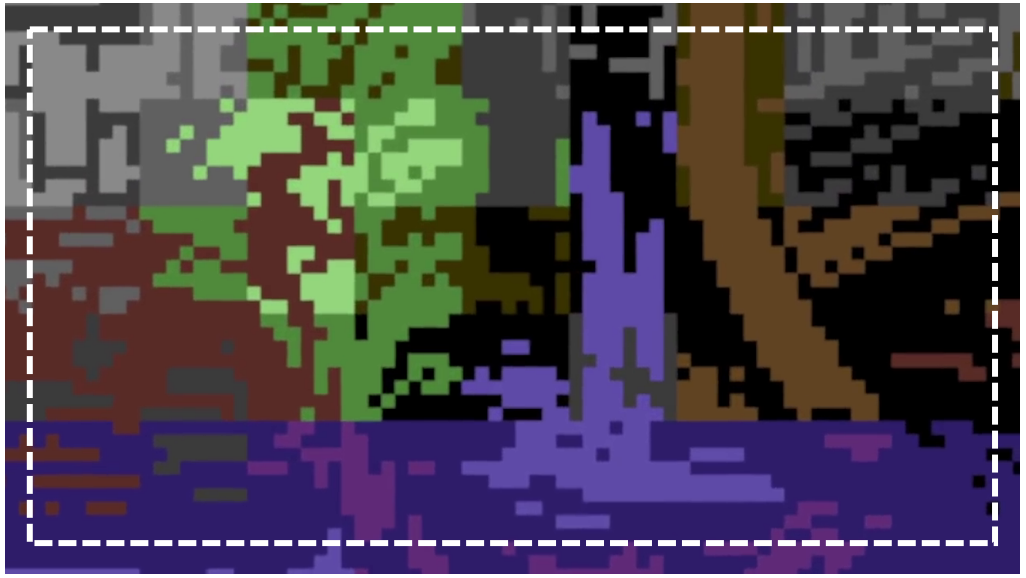
3. Hardware sprite generation :
   - Screen split into 3 sections of 8 pixels each
   - Used heavily in games

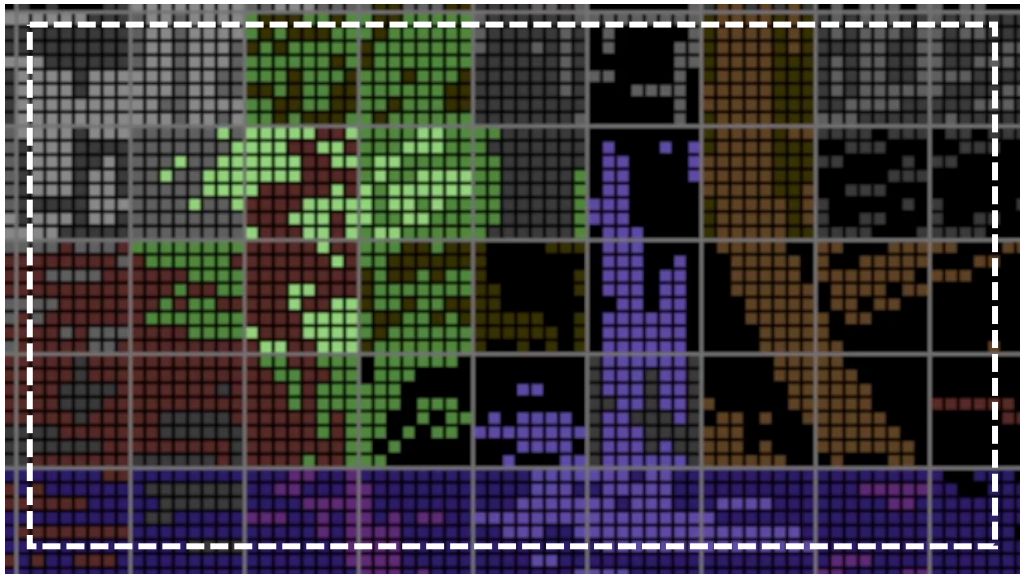
1.1 Color cells
2-color mode

- Screen divided into cells 8px wide to 8px tall
- Each color cell has additional 1B of memory for foreground/background colors
- Whole screen could have 16 colors using just 9KB memory as opposed to 8KB for monochrome
- Exceptions being, colors did not cross cells
- And no more 2 colors per cell

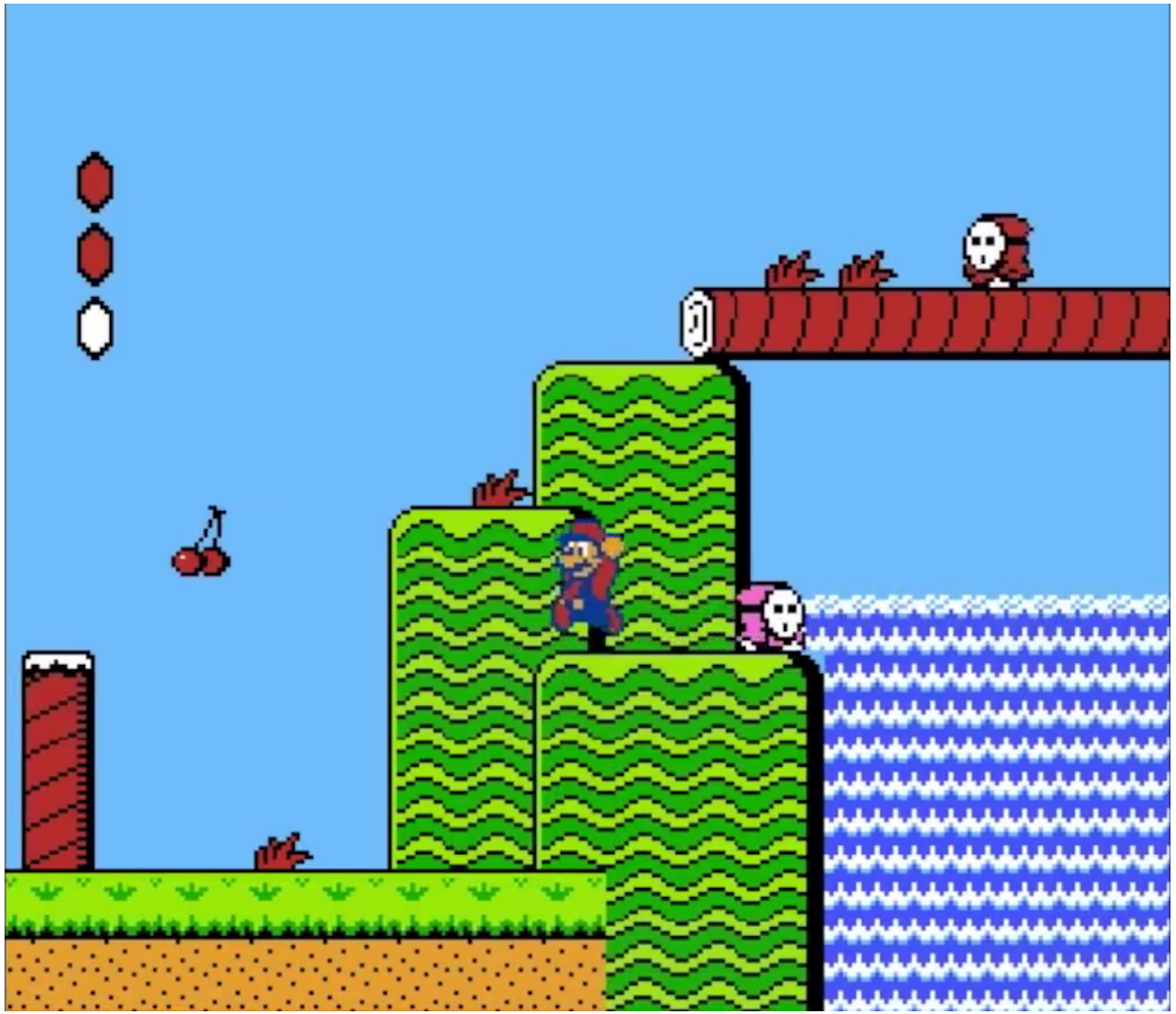


1.1 Color cells – **2-color mode**





1.1 Color cells – **2-color mode**

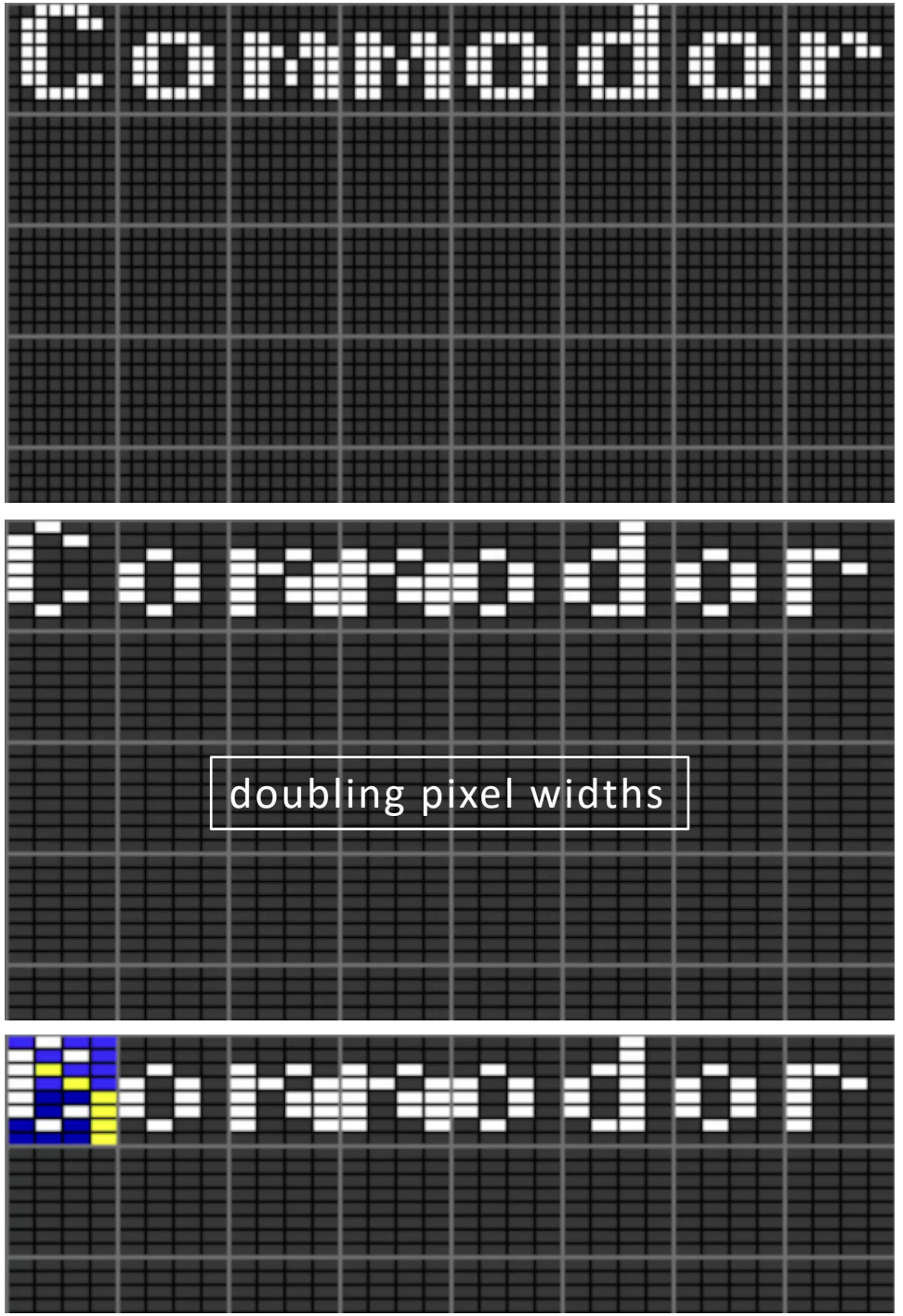


1.2 Color cells - multicolor mode

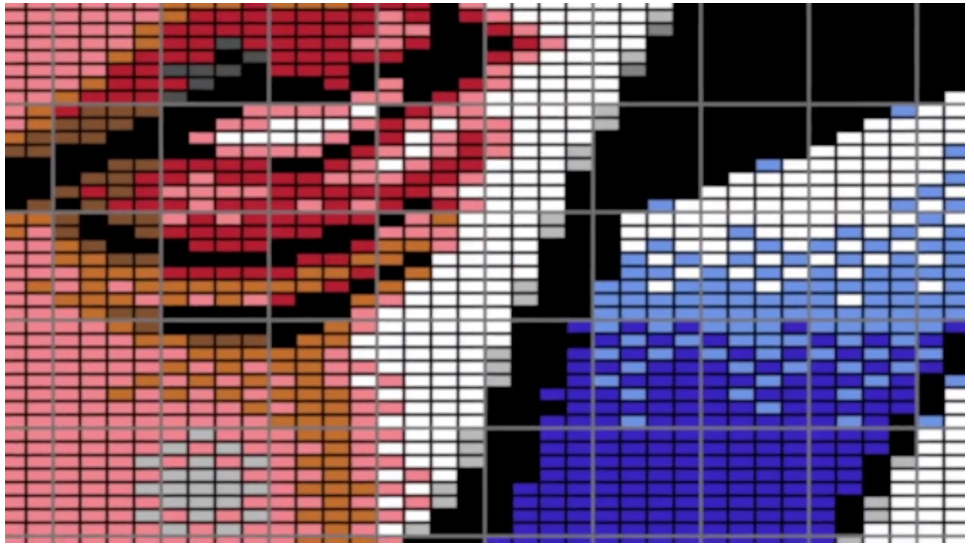
Multicolor mode:

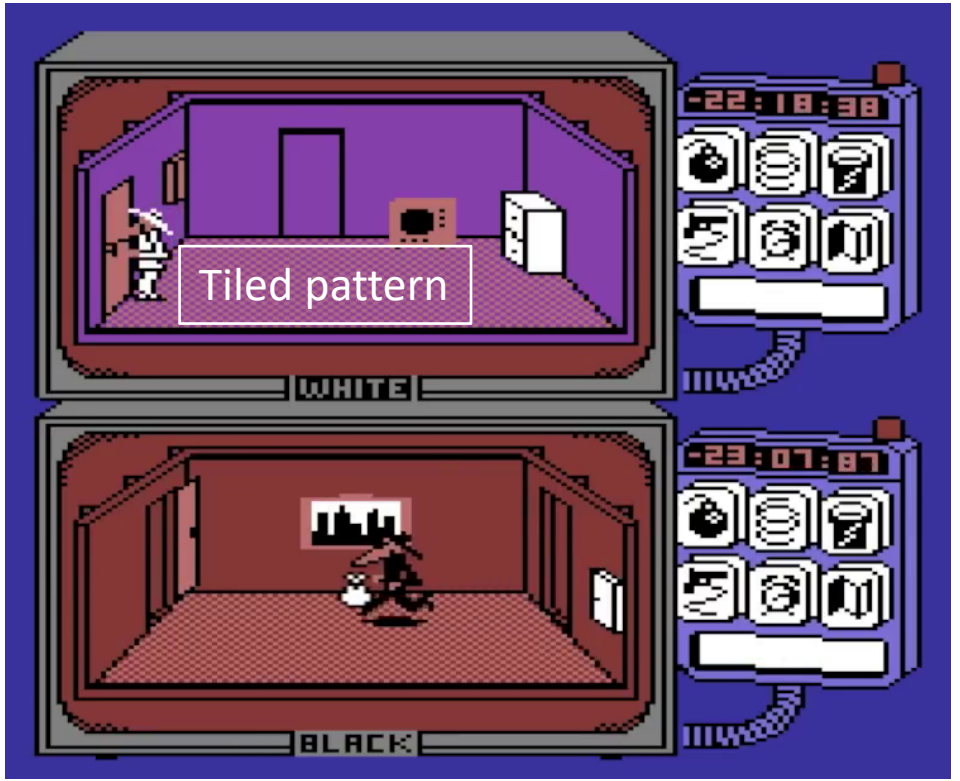
- Lose resolution by doubling pixel widths, so 8 pixels tall and 4 pixels wide
- This way, 4 colors per cell with 8KB of memory

- Higher memory usage vs lower resolution would result in more colors

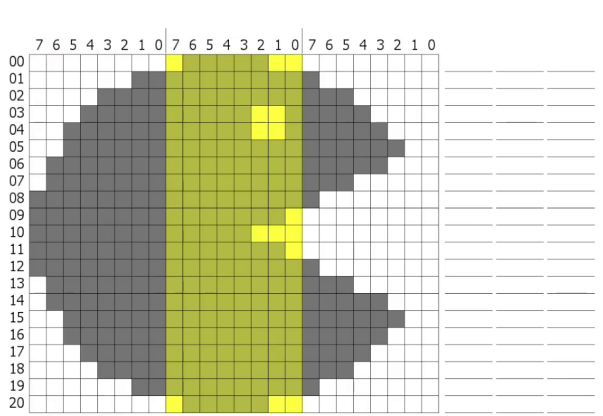


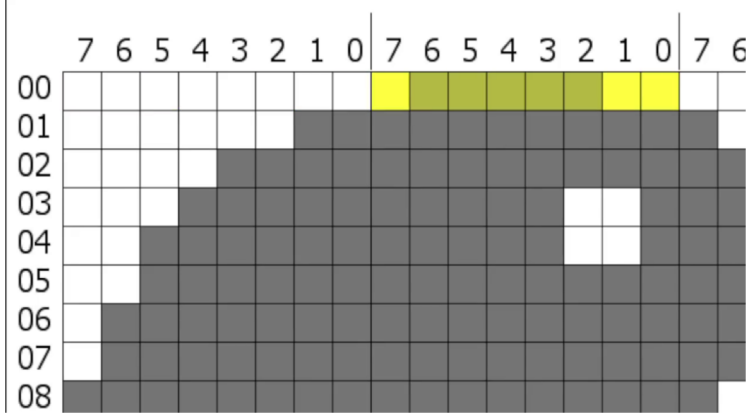
1.2 Color cells - **multicolor mode**





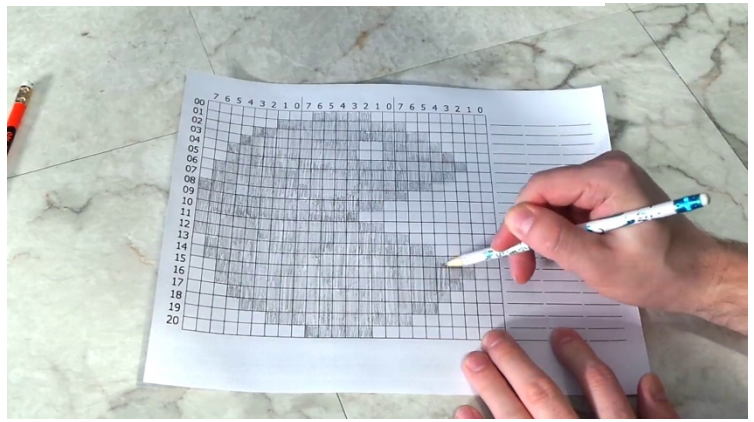
1.3 Color cells
hardware sprites

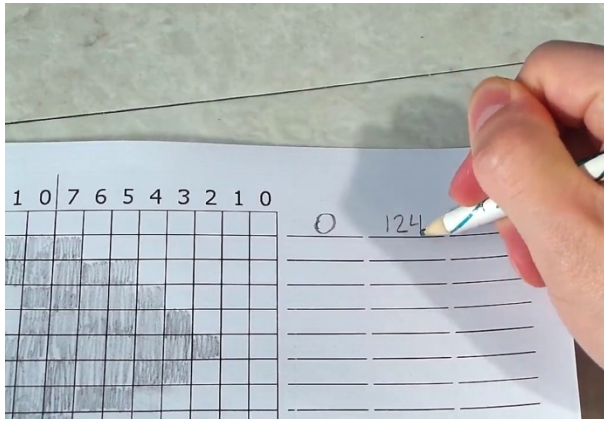


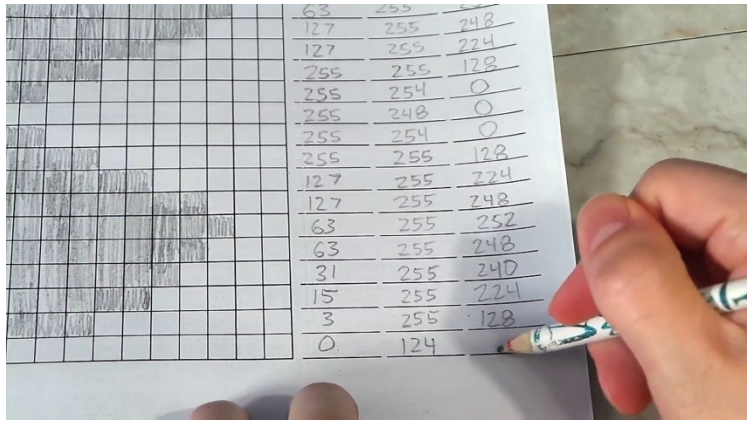


Hardware Sprite generation :

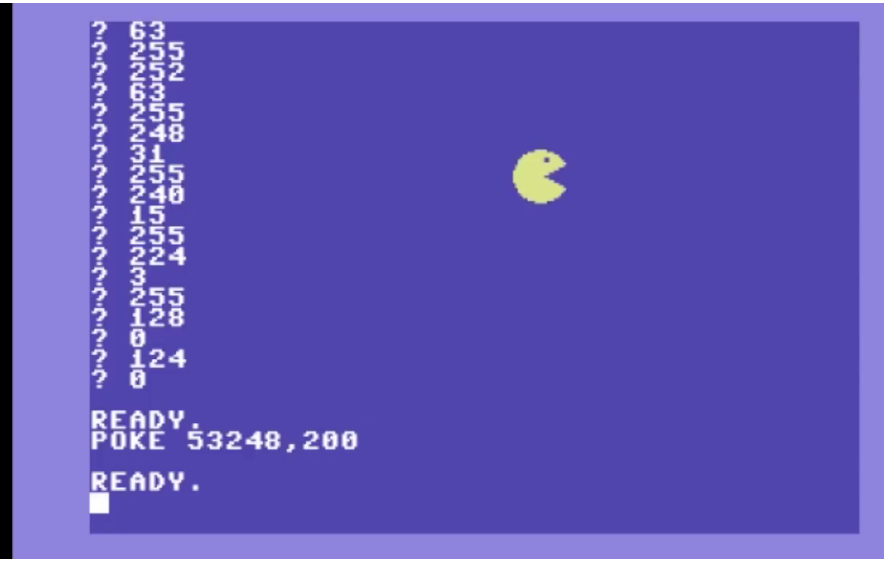
- Each pixel set as 0 or 1
- Screen split into 3 sections of 8 pixels
- Set binary values of sprites and fed to CPU's memory
- This way you could not erase or write over sprites or change text colors
- Used heavily in games like Nintendo Mario (64 sprites)

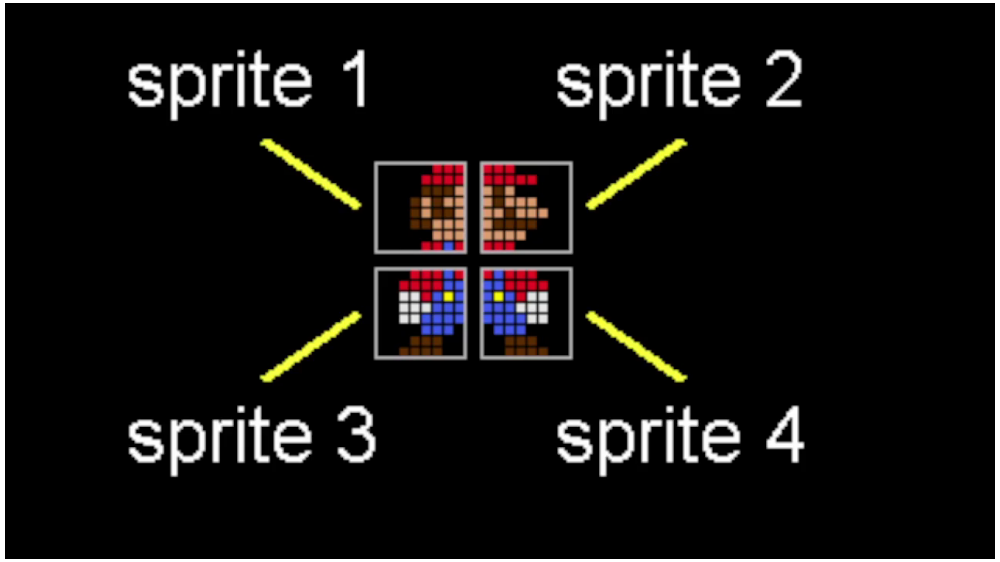






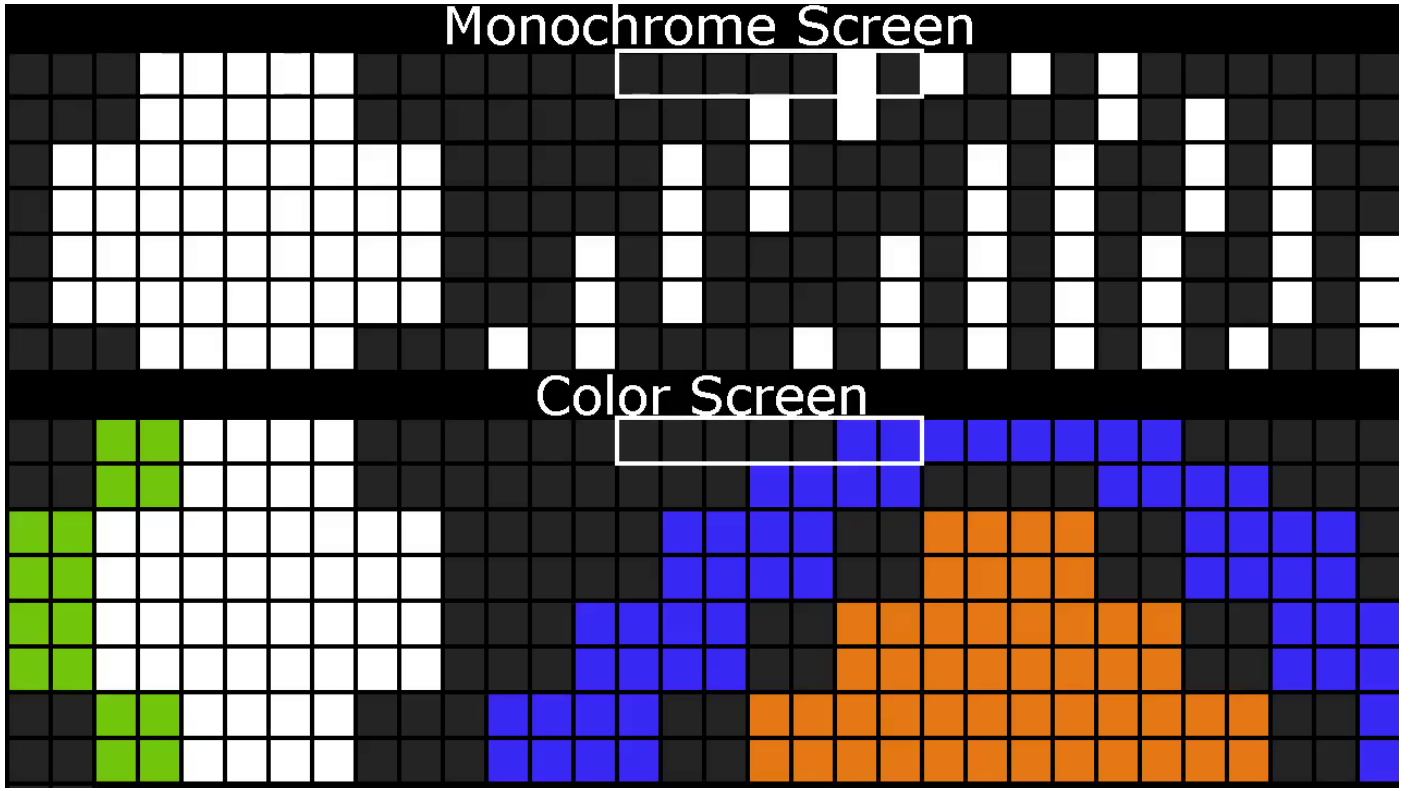
1.3 Color cells - **hardware sprites**



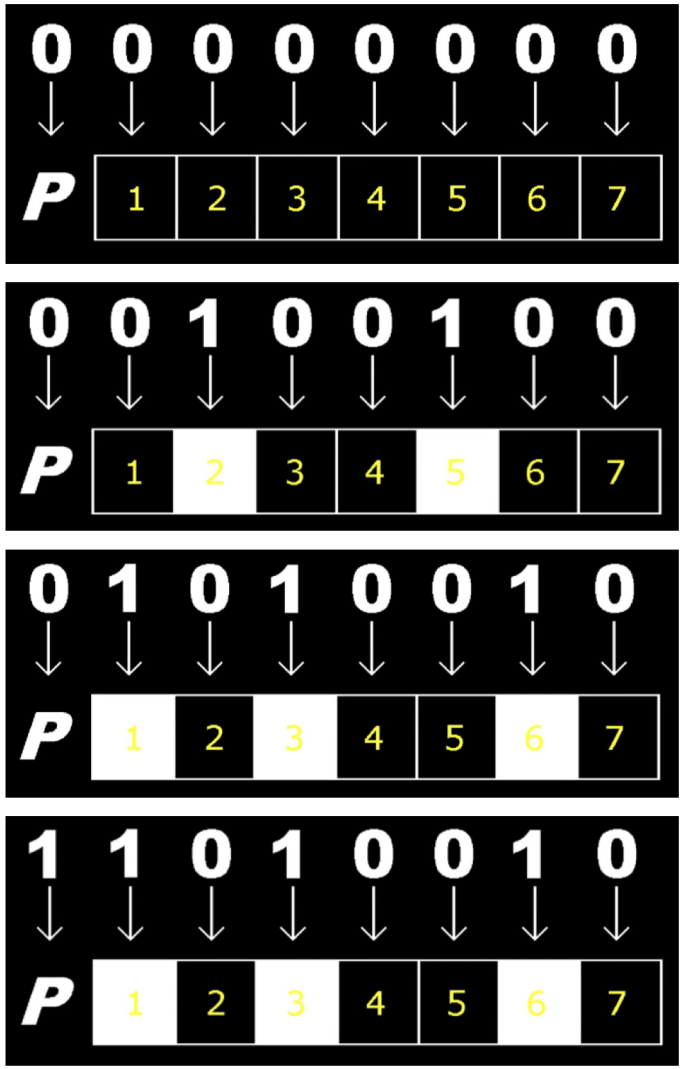


2. NTSC Artifact Colorization

- Apple 2 in 1977 used this to generate graphics – 5 years before Atari and ZX Spectrum!
- Two types of displays – Monochrome vs Color



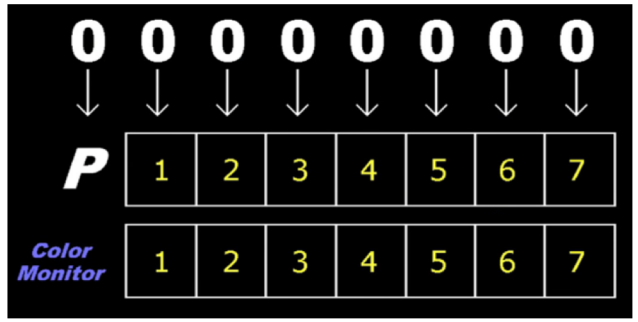
2. NTSC Artifact Colorization

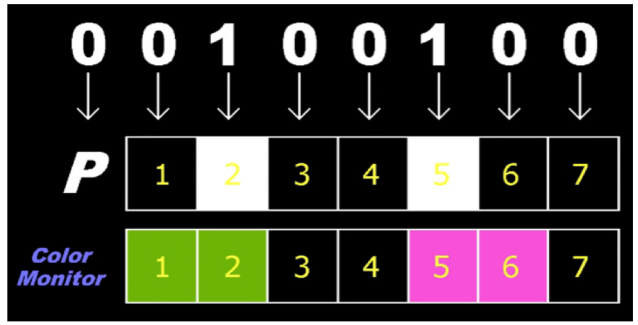


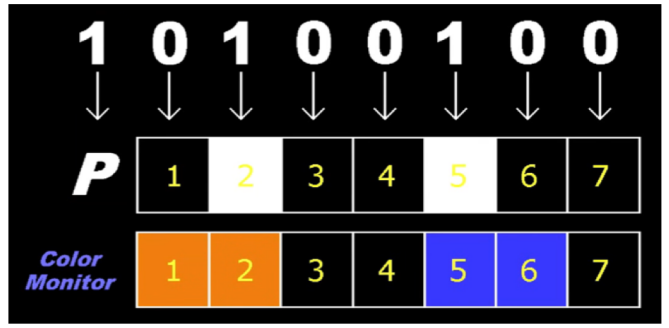
- Each 8 bit section was divided into 7 bits for pixel data and 1 bit(P) for toggling
- On a monochrome screen, toggling ‘P’(PAL) did not do anything
- In color screen, toggling bits will change color between two
- And now toggling ‘P’ additionally will toggle the set of colors too
- This way 6 different colors were generated (using 8KB memory)

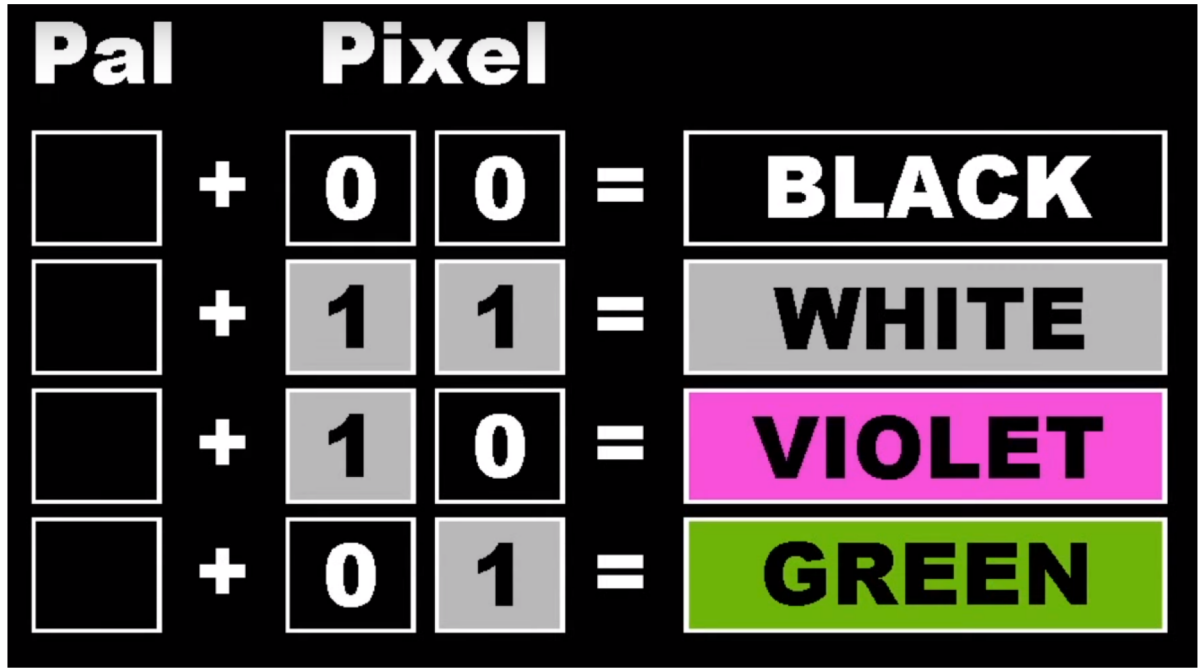
Monochrome screen →
Crisp graphics

2. NTSC Artifact Colorization - continued

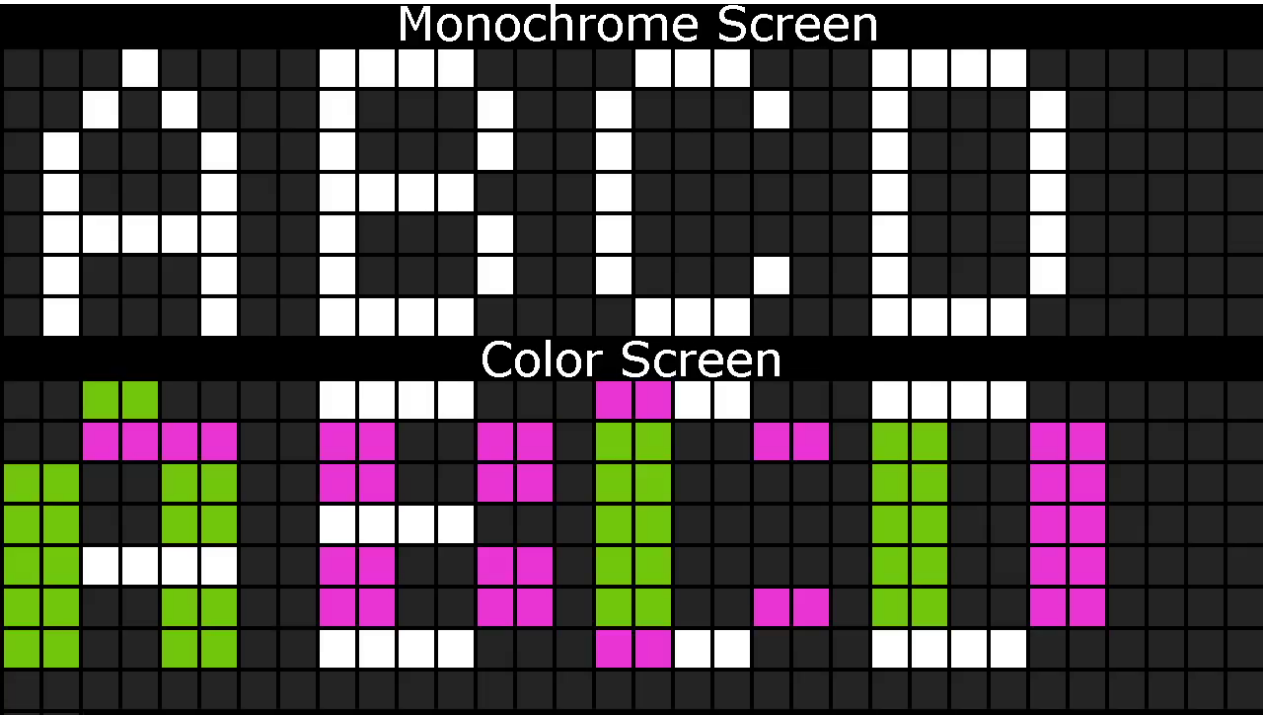




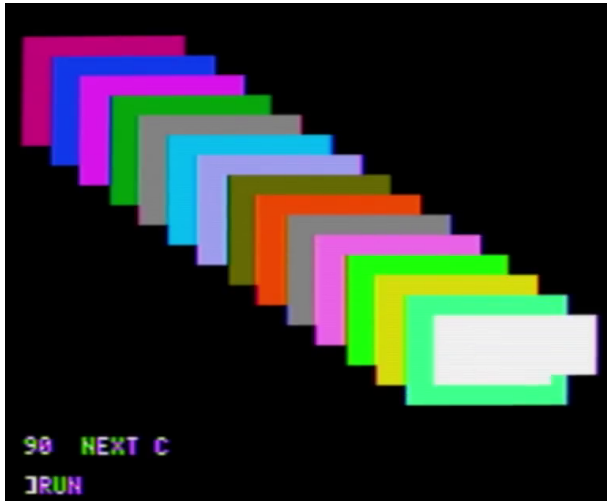




2. NTSC Artifact Colorization – Rainbow coloration

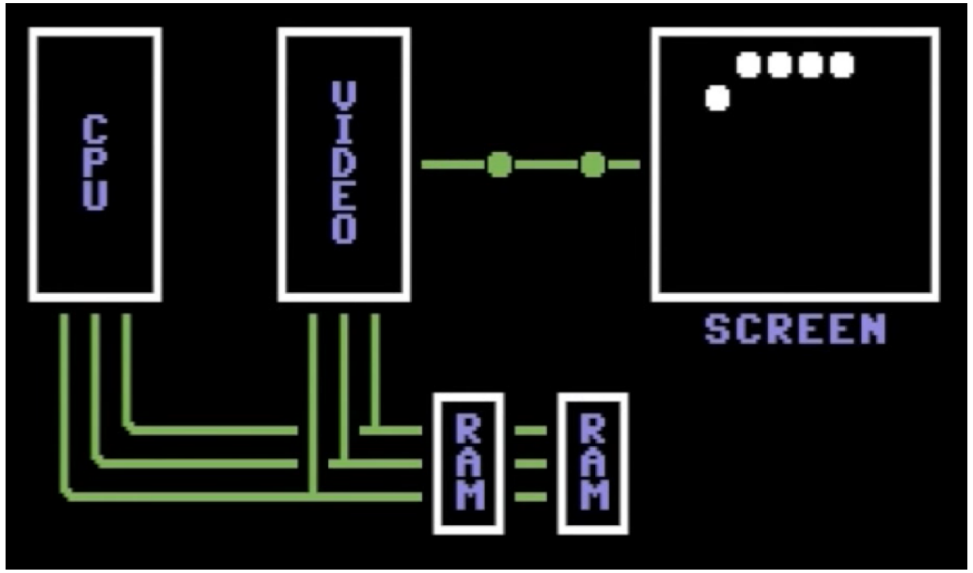


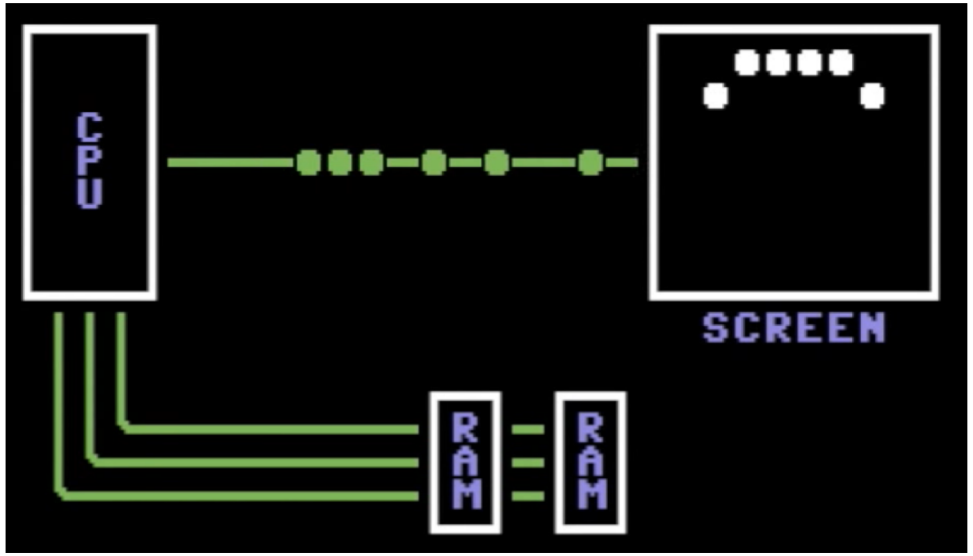
- So you get 16-bit colors ☺
- Just not everywhere you want them ☹



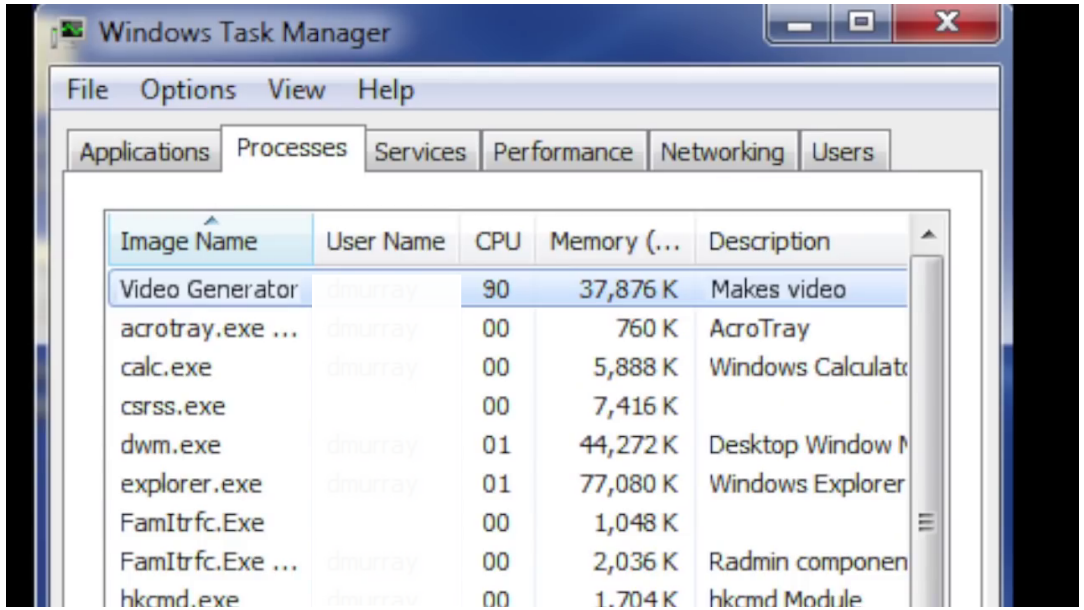
3. CPU Driven Graphics

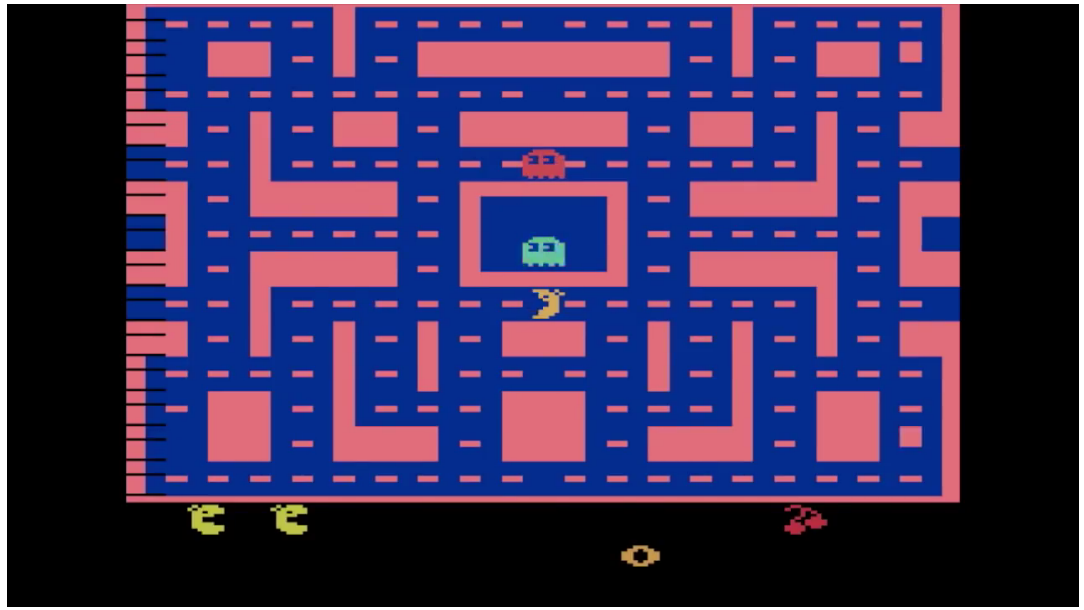
- Later systems started having dedicated video chip
- They sent pulses to monitor in fixed order
- Others without video chip had CPU send these pulses
- This worked but left very less CPU time to run game code





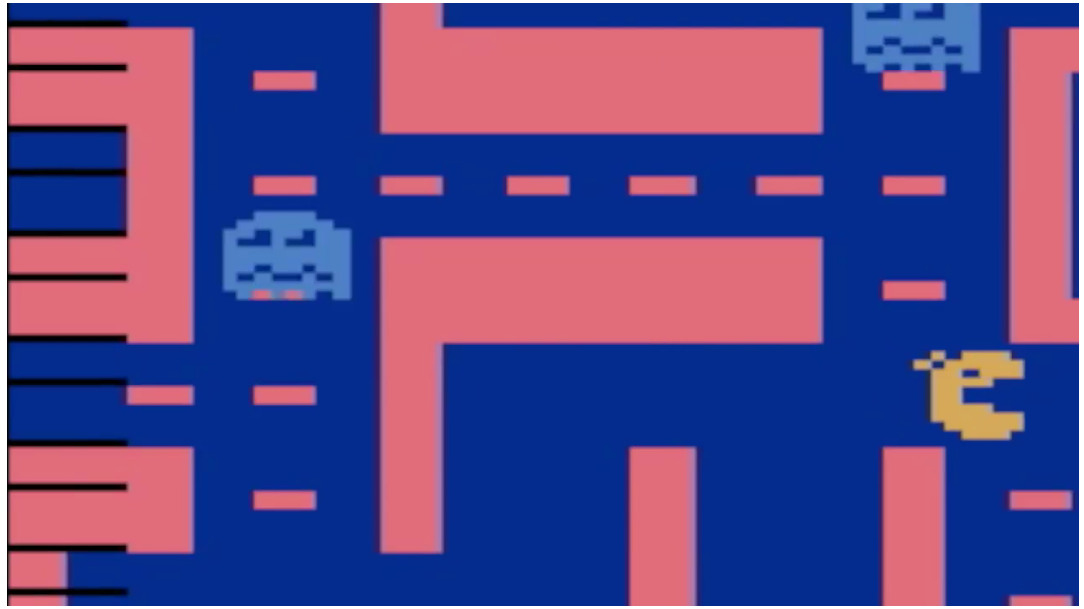
3. CPU Driven Graphics





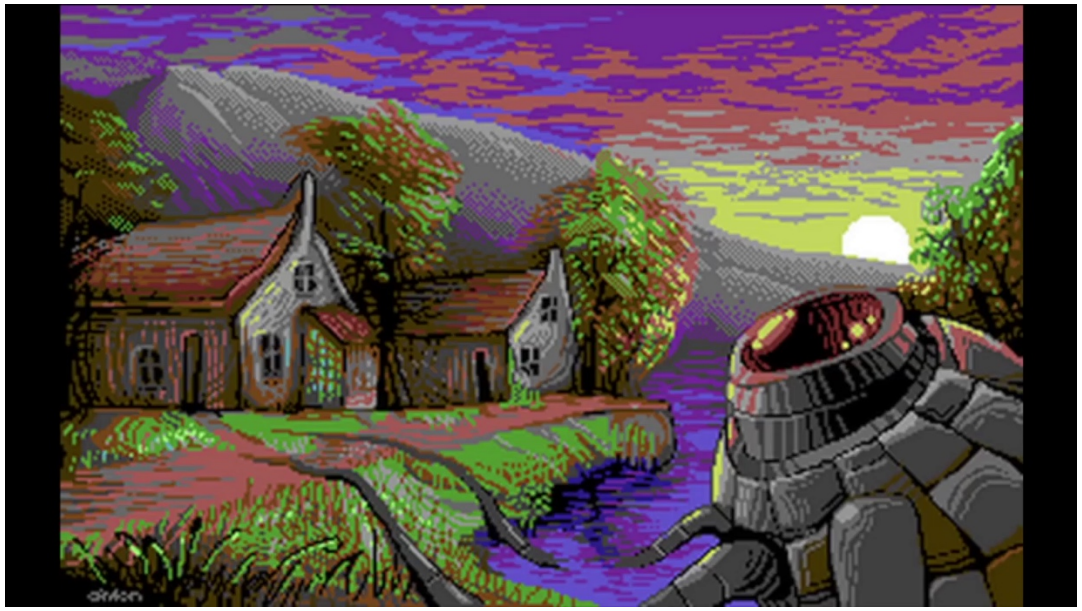
- Notice the black scan lines
- The CPU did not have enough time to draw them and also run the game

→



3. CPU Driven Graphics

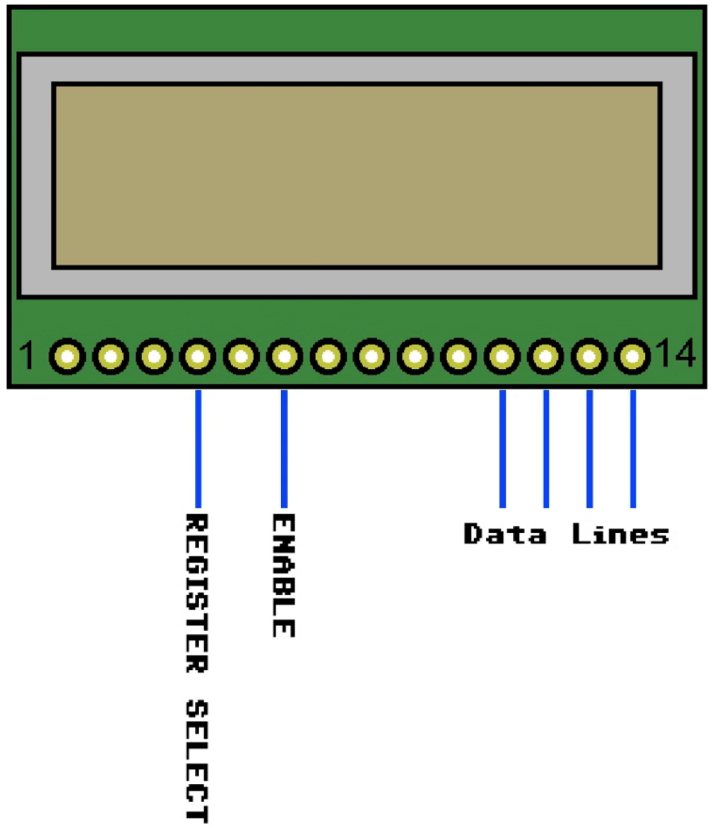
So, it was not that CPUs could not product lot of colors, but it was not able to run the game code simultaneously



How the LCD works

A typical LCD will have 14 or 16 pins

- 8 pins for 8 bit data
- GND and VCC for power
- Contrast pin for display contrast
- RS (Register select) to toggle between
  - Text select mode to write ASCII characters etc.
  - Instructions mode to turn on cursor, initialize etc.
- R/W (Read or Write): 1 for Read, 0 for Write
- EN (Enable pin)
  - Asserting it writes whatever is on Data lines to LCD
- Additional 2 pins for Backlight (VCC and GND)



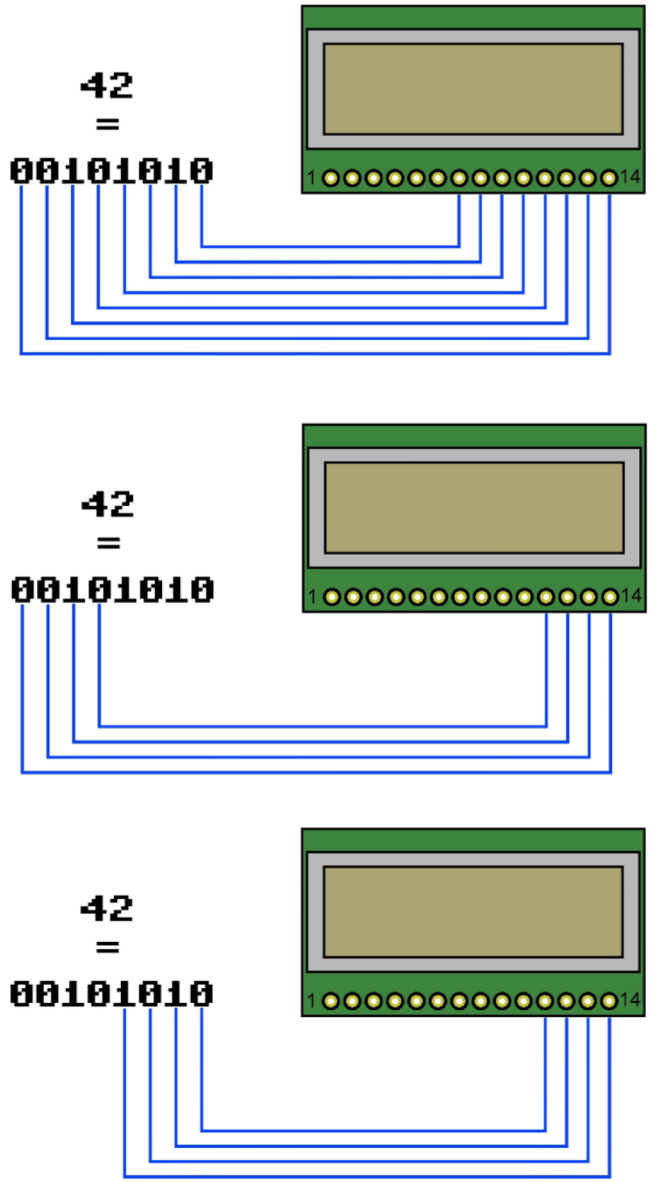
How the LCD works

- To display data in a LCD, 10 bits are needed (8 data lines + RS + EN)

- This can be restricted to using 8 bits by sending 4 bits of data at a time hence using 4+2 = 6 bits

- So, LCDs expect 2 bursts of data – First 4 bits and then next 4

- This in some terms determines the byte-orientation of how data is displayed on these LCDs



Example – 128x64 LCD used in Lab

- There are two generic types of LCDs available
- For the Lab, we use the 3.3V RGB Backlight monochrome LCD which uses serial interface (SPI).
- Serial and Parallel versions of LCD are not compatible with each other. Serial LCD use fewer pins
- Serial LCD interface requires level shifting from 5V down to 3.3V
- A disadvantage with Serial LCD is that you can't read, only write. MCU has to keep track of the display, as it will need to use 1024 bytes of RAM on display memory.
- For the lab we are using ATmega328, half of its RAM is used for display memory.

Voltage	5V	3.3V
Interface	Parallel	Serial
Data pins needed	14	4 or 5
Display size	128x64	128x64
Contrast adj.	requires potentiometer	internal, no extras!
Buffer needed?	No	Yes

128x64 LCD

- LCD has 11 exposed pins :
  1. CSn – Chip select
  2. RSTn – Reset
  3. A0/RS – Register Select
  4. SCLK – Serial Clock
  5. SID – Serial Input Data
  6. VDD – 3.3V Input power
  7. GND – Ground
  8. R – Red led cathode for backlight
  9. 3.3 – 3.3V input to Led Anode
  10. G – Green led cathode for backlight
  11. B – Blue led cathode for backlight

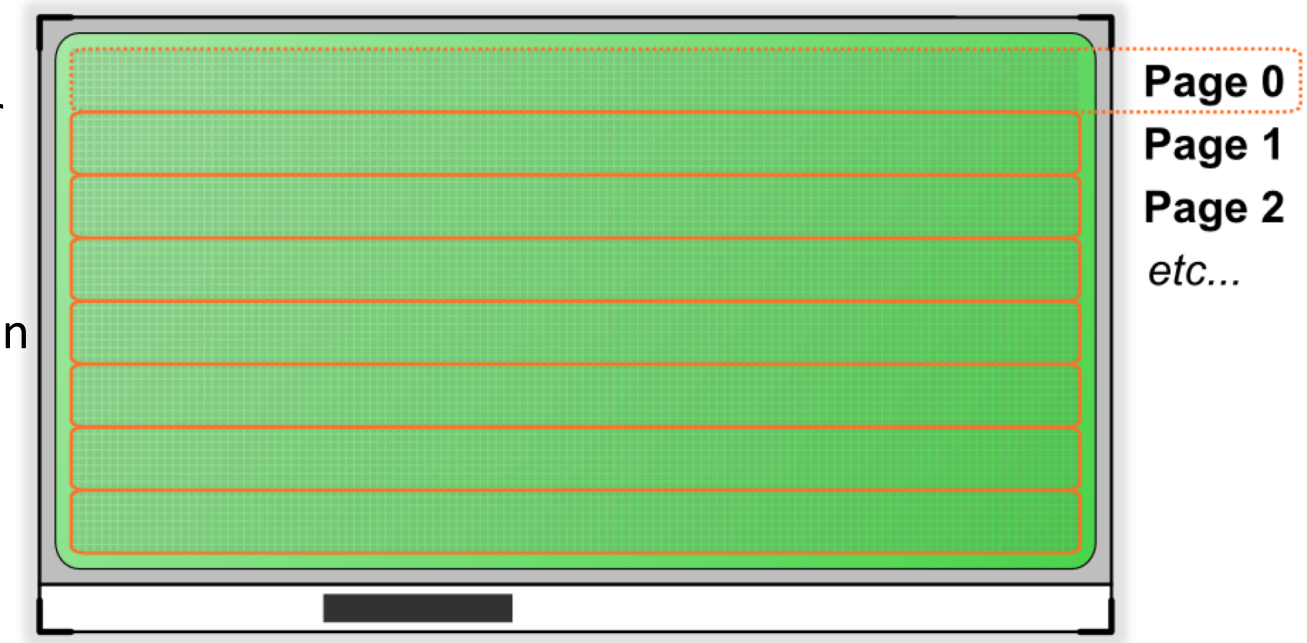
- To interface with ATmega328p, a level shifter or buffer IC can be used for 3.3V to 5V conversion.

Functioning of the LCD

- RESET pin from LCD is connected to microcontroller as initialization involves toggling this pin
- LCD also has an integrated voltage booster (charge pump) circuit that can provide high voltages needed.
- SPI mode provides simple interface to drive screen using fewer pins
- A single bit is written to data pin and clock is pulsed high then low.
- Onboard LCD controller implements the standard SPI bus protocol
- Since LCD is write only, it is necessary to keep a copy of current screen data in memory on MCU
- Size of this software buffer can be calculated easily. For example – 128 columns and 64 rows for a total of 8192 bits(1 bit per pixel), equivalent to 1024 bytes
- Also, with **software SPI (bit-banged)** it is possible to use screen at very fast clock speeds

Screen Layout

- 128x64 screen is made up of 8 pages
- Each page is 8 pixels high and 128 wide
- This maps nicely to software buffer required for caching the screen contents as each page is a byte high
- If top left of screen is (0,0) then bottom right can be imagined as (127,63).
- E.g. to write at (0,0) we select page 0 column 0 and set individual bit
- Similarly, (32,32) is in page 5



- Simplest method to write data to screen is to send whole page at a time

Sending commands to the screen

Issuing single command to screen is most basic building block of interface code:

- Set the A0/RS pin low to indicate that command is being sent (needs to be done every time a new command is sent)
- Set the chip select (CS) pin low
- Send each bit of command, starting with the most significant bit down to the least significant bit
- Set the chip select (CS) pin high, to free the bus
- E.g. to send “display on” command (0xAF or 0b10101111) should be sent as : 1,0,1,0,1,1,1,1

This command turns the display ON and OFF.

A0	E /RD	R/W /WR	D7	D6	D5	D4	D3	D2	D1	D0	Setting
0	1	0	1	0	1	0	1	1	1	1 0	Display ON Display OFF

More LCD Commands

Command	Command Code											Function
	A0	/RD	/WR	D7	D6	D5	D4	D3	D2	D1	D0	
(1) Display ON/OFF	0	1	0	1	0	1	0	1	1	1	0 1	LCD display ON/OFF 0: OFF, 1: ON
(2) Display start line set	0	1	0	0	1	Display start address						Sets the display RAM display start line address
(3) Page address set	0	1	0	1	0	1	1	Page address				Sets the display RAM page address
(4) Column address set upper bit	0	1	0	0	0	0	1	Most significant column address				Sets the most significant 4 bits of the display RAM column address.
Column address set lower bit				0	0	0	0	Least significant column address				Sets the least significant 4 bits of the display RAM column address.
(5) Status read	0	0	1	Status				0	0	0	0	Reads the status data
(6) Display data write	1	1	0	Write data								Writes to the display RAM
(7) Display data read	1	0	1	Read data								Reads from the display RAM
(8) ADC select	0	1	0	1	0	1	0	0	0	0	0 1	Sets the display RAM address SEG output correspondence 0: normal, 1: reverse
(9) Display normal/ reverse	0	1	0	1	0	1	0	0	1	1	0 1	Sets the LCD display normal/ reverse 0: normal, 1: reverse
(10) Display all points ON/OFF	0	1	0	1	0	1	0	0	1	0	0 1	Display all points 0: normal display 1: all points ON
(11) LCD bias set	0	1	0	1	0	1	0	0	0	1	0 1	Sets the LCD drive voltage bias ratio 0: 1/9 bias, 1: 1/7 bias (ST7565R)

More LCD commands - continued

(12) Read-modify-write	0	1	0	1	1	1	0	0	0	0	0	Column address increment At write: +1 At read: 0
(13) End	0	1	0	1	1	1	0	1	1	1	0	Clear read/modify/write
(14) Reset	0	1	0	1	1	1	0	0	0	1	0	Internal reset
(15) Common output mode select	0	1	0	1	1	0	0	0 1	*	*	*	Select COM output scan direction 0: normal direction 1: reverse direction
(16) Power control set	0	1	0	0	0	1	0	1	Operating mode			Select internal power supply operating mode
(17) V_0 voltage regulator internal resistor ratio set	0	1	0	0	0	1	0	0	Resistor ratio			Select internal resistor ratio(Rb/Ra) mode
(18) Electronic volume mode set Electronic volume register set	0	1	0	1 0	0 0	0	0	0 Electronic volume value	0	0	1	Set the V_0 output voltage electronic volume register
(19) Sleep mode set	0	1	0	1 * *	0 * *	1 * *	0 * *	1 * *	1 * *	0 0 0	0 1 0	0: Sleep mode, 1: Normal mode
(20) Booster ratio set	0	1	0	1 0	1 0	1 0	1 0	1 0	0 0	0 step-up value	0	select booster ratio 00: 2x,3x,4x 01: 5x 11: 6x
(21) NOP	0	1	0	1	1	1	0	0	0	1	1	Command for non-operation
(22) Test	0	1	0	1	1	1	1	*	*	*	*	Command for IC test. Do not use this command

Screen Initialization

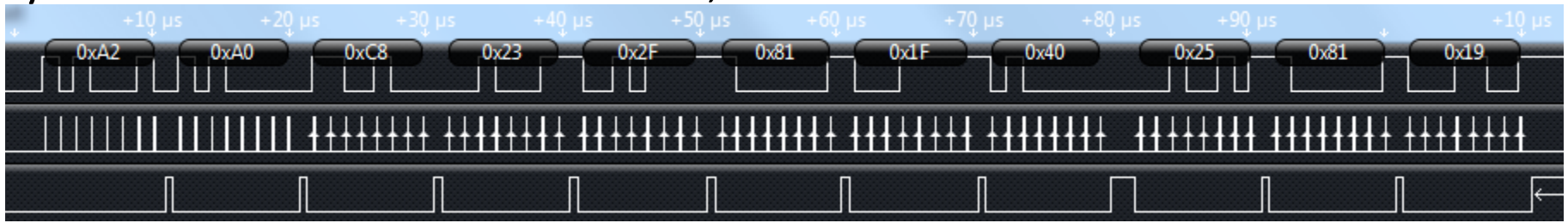
We need to follow a number of steps to setup the screen:

- Strobe the RESET pin low then high, to initiate a hardware reset
- Setup the duty cycle (either 1/7 or 1/9) depending on the physical LCD
- Set the horizontal and vertical orientation to a known state
- Configure the internal resistor divider which is used by the voltage regulator
- Turn on the internal voltage booster to provide power to the LCD glass
- Initialize the dynamic contrast to a default value
- Reset the current display position to the top left

- The commands for these are single byte except the dynamic contrast.
- Next slide explains the initialization code in more detail

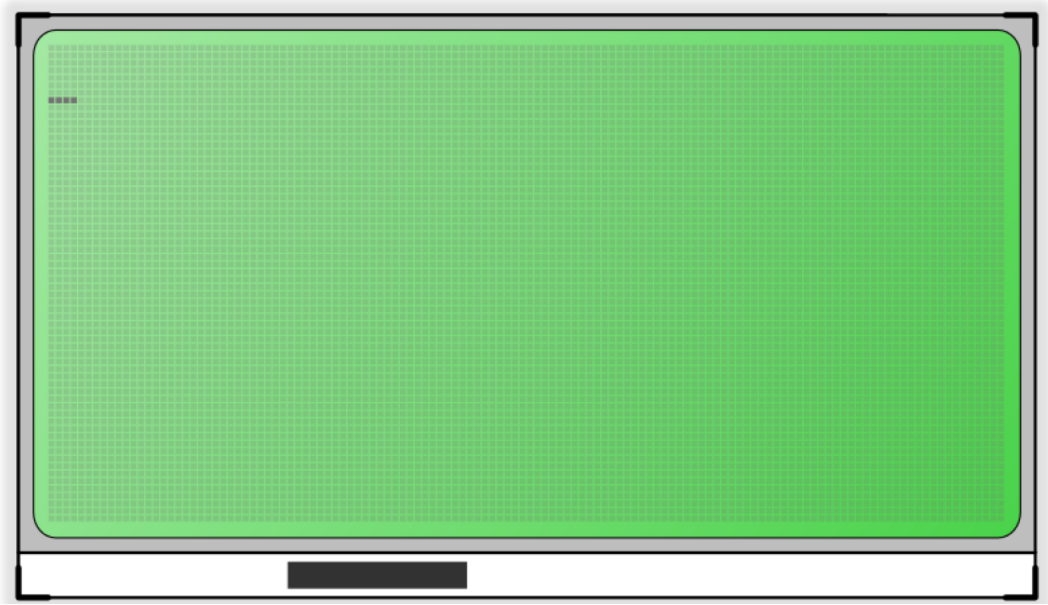
Screen Initialization - continued

- Below is an image capture from a logic analyzer while the initialization code runs
- In the trace , following things can be seen :
  - Command 0xA2 : set the LCD bias to 1/9th
  - Command 0xA0 : horizontally “normal”
  - Command 0xC8 : vertically “flipped”
  - Command 0x23 : internal resistor divider set to 3
  - Command 0x2F : power control, all internal blocks ON
  - Command 0x81 : enter dynamic contrast mode
  - Data for the dynamic contrast mode, set to 31
  - Command 0x40 : go back to top left of the display
- When power is first applied to screen, it is also important to clear onboard screen memory because it is a non-volatile RAM, it will start in an unknown state.



Sending data to Screen

- As we saw previously, A0/RS pin toggles between **command mode** and **data mode.**
- When sending data , A0/RS pin must be set ***high*** to differentiate the bus data from a command
- To start, we send a command to reset position to top left corner (0,0). Now, if we write data , It will display it on the screen. MSB writes to bottom of page working up to top of page
- Example below shows output of the byte 0x80 sent four time to the screen :



Writing to specific pixels

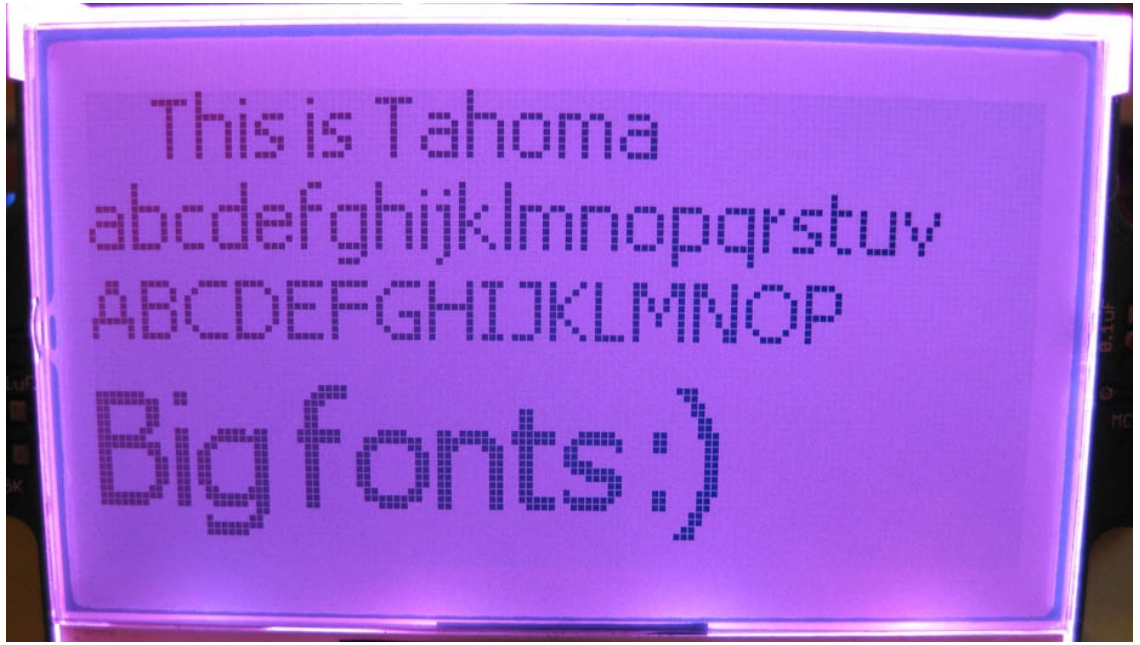
- Writing specific pixels allows to draw lines, circles, variable height fonts etc.
- In order to set a specific pixel, we break the process into a number of steps:
  - Find the corresponding byte in the software buffer
  - Calculate the bit within this byte
  - Use Logical OR (to set the bit) or AND (to clear the bit)
- Finding correct byte is straightforward. First position in our array is at the top left of the screen
- The 128th byte in the array (Array[127]) is on the first page at the right hand side of screen.
- Next bit “wraps” to the next page, at page 1 column 1.
- To calculate which byte is needed, we divide the Y position by 8 and multiply the result by 128. To this we add the X position and subtract 1 (index starts at 0). E.g. :
  - (1,1) on the screen would give 1 + (1/8 *128) - 1 = 0
  - (61,52) on the screen would give 61 + (52/8 *128) - 1 = 828

Writing to specific pixels

- To calculate the bit inside this byte, we remember that the 8th (MSB) appears at bottom of the page.
- So, to write to Y location ‘8’ , we need to set the 8th bit of a byte on first page. Similarly, to write to Y location ‘16’ , we need to write to 8th bit of corresponding byte on second page.
- This is simply Y%8

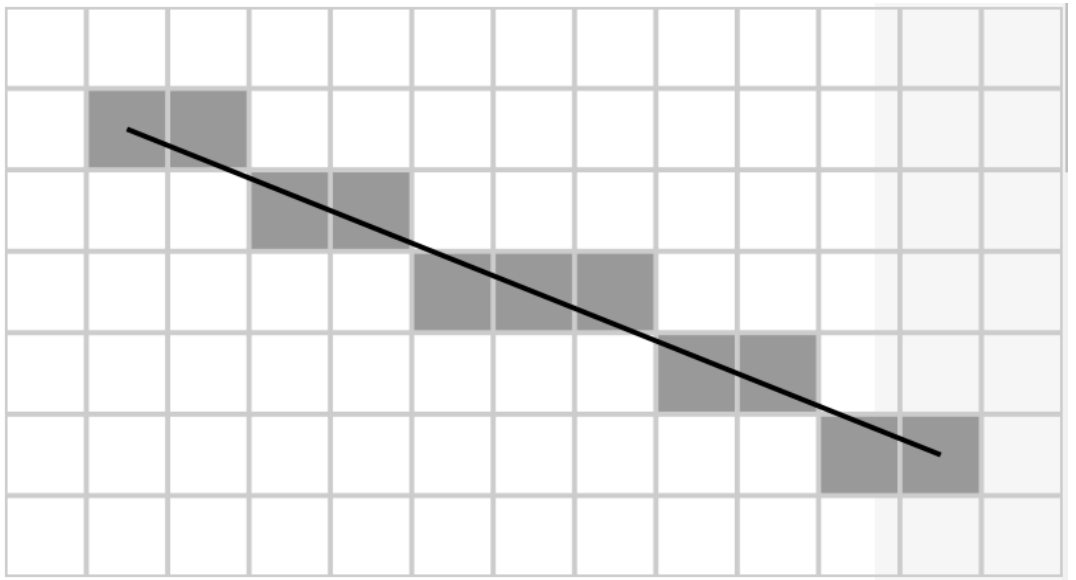
Displaying Text

- The LCD has no built-in font
- The quickest and easiest way to display text on screen is to define an 8-pixel high font
- This way it maps directly to one page of the display and therefore we use very fast instructions to copy the data from a predefined font to the right area of the screen.
- However, this limits us to a fixed height and only 8 lines of the screen



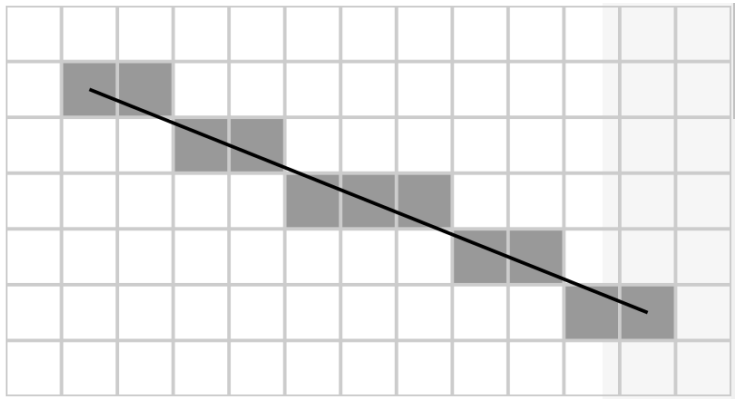
Draw Lines and Circles

- We have a limitation in hardware that we cannot handle floating point numbers
- This makes it difficult to draw accurate circles and lines using only integer addition, subtraction and bit shifting.
- Bresnham's Line Algorithm is an algorithm that determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points.
- An extension of this algorithm can be used for drawing circles too.
- It is an incremental error algorithm.



Bresenham's Line Algorithm

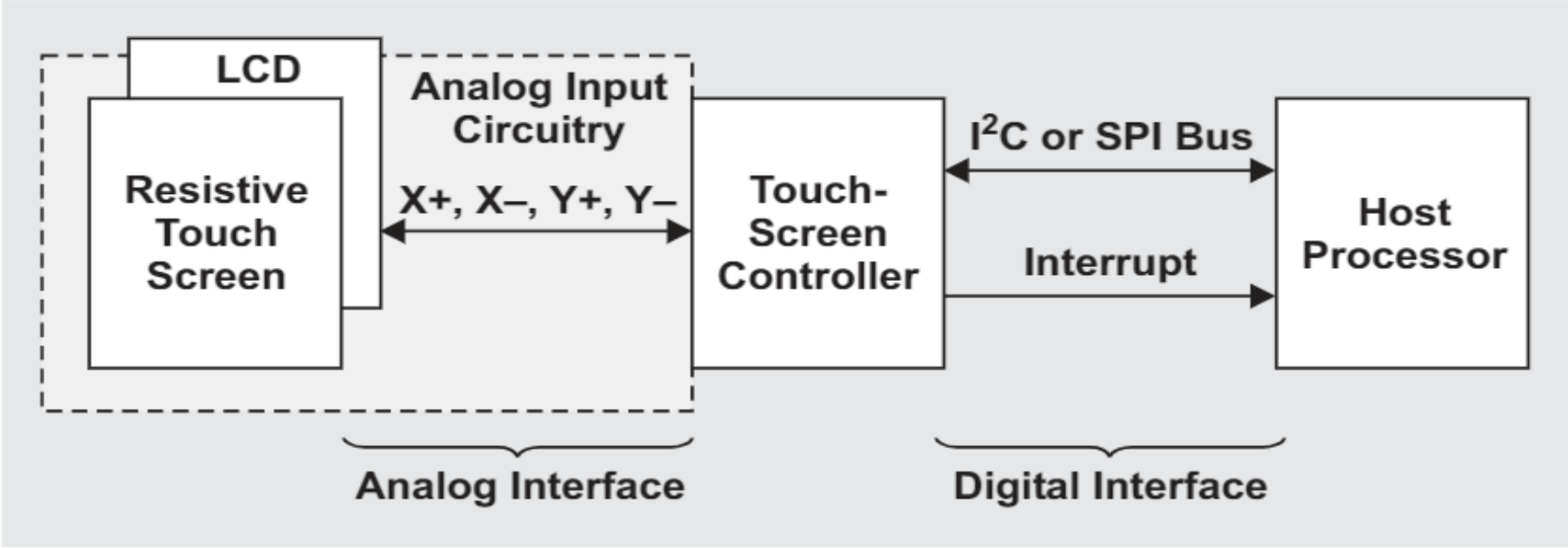
- Conventions used :
  - Top left is (0,0) such that pixel coordinates increase in right and down directions
  - Pixel centers have integer coordinates
- The endpoints of the line are the pixels at (x0,y0) and (x1, y1), where the first coordinate of the pair is the column and the second is the row.
- The algorithm will be initially presented only for octant in which the segment goes down and to the right, and its horizontal projection x1-x0 is longer than the vertical projection y1-y0.
- **Bresenham's algorithm chooses the integer y corresponding to the pixel center that is closest to the ideal y for the same x, on successive columns y can remain the same or increase by 1.**



Source :
https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/
https://www.geeksforgeeks.org/bresenhams-circle-drawing-algorithm/

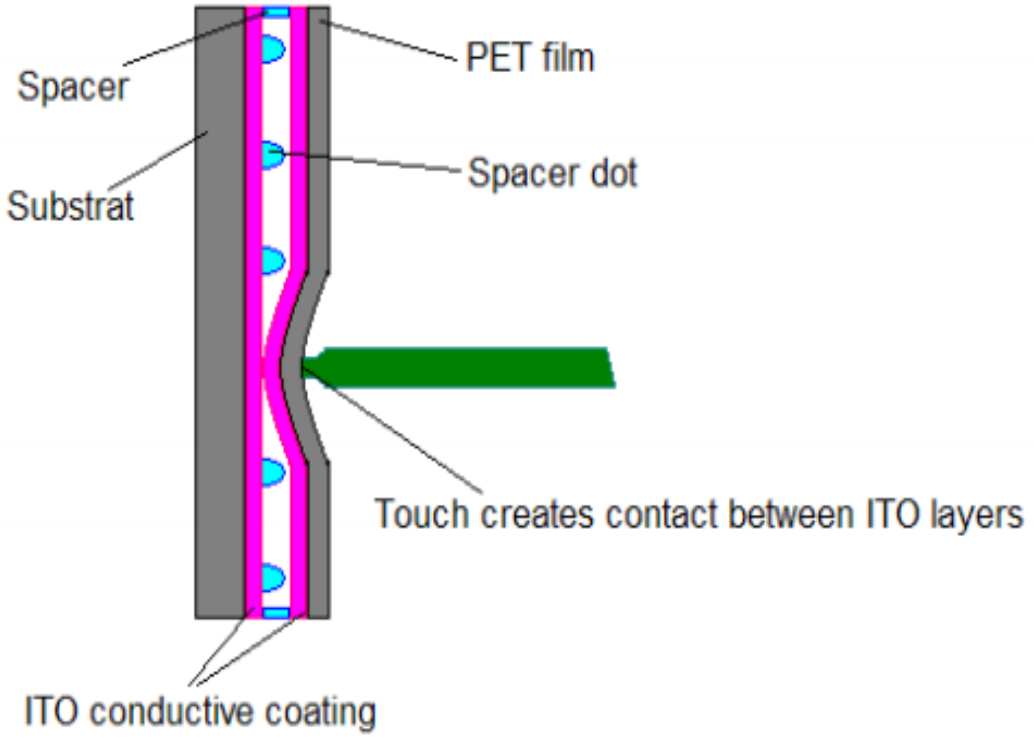
Touchscreens

- Touchscreen used for the lab is resistive, which can be used with a stylus or fingertip and quite easy to interface with microcontrollers.
- The Touchscreen has a 3.2” diagonal active area with a four wire resistive interface coming out on 0.5mm FPC connector.
- They can be interfaced with four of the ADC pins, 2 will be Analog input and 2 Digital at any time.



How analog resistive touch screens work

- Touchscreen has 600 ohms resistance across X pins and 300 ohms across Y pins.
- Usually resistive touch screen consists of at least three layers as shown in the figure.
- The conductive ITO layers are kept apart by an insulating spacer along the edges and by spacer dots on the inner surface of the two ITO layers.
- This way there will be no electrical connection unless pressure is applied to the top sheet.

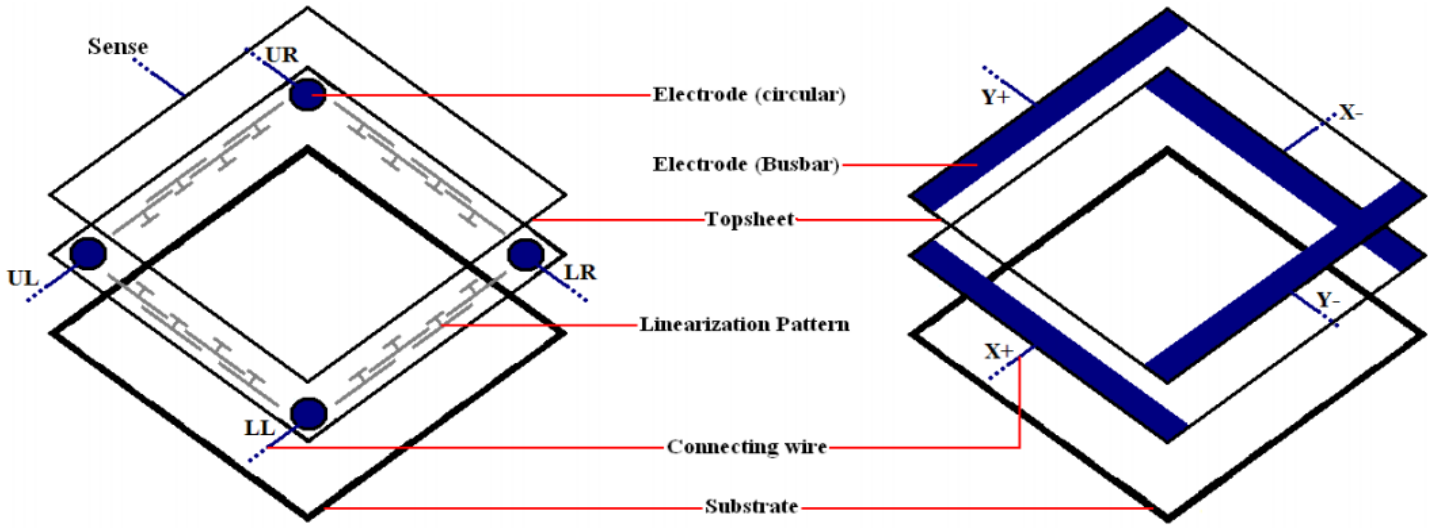


ITO → Indium Tin Oxide

1. High conductivity
2. Optically transparent
3. Easy to mass produce as thin films

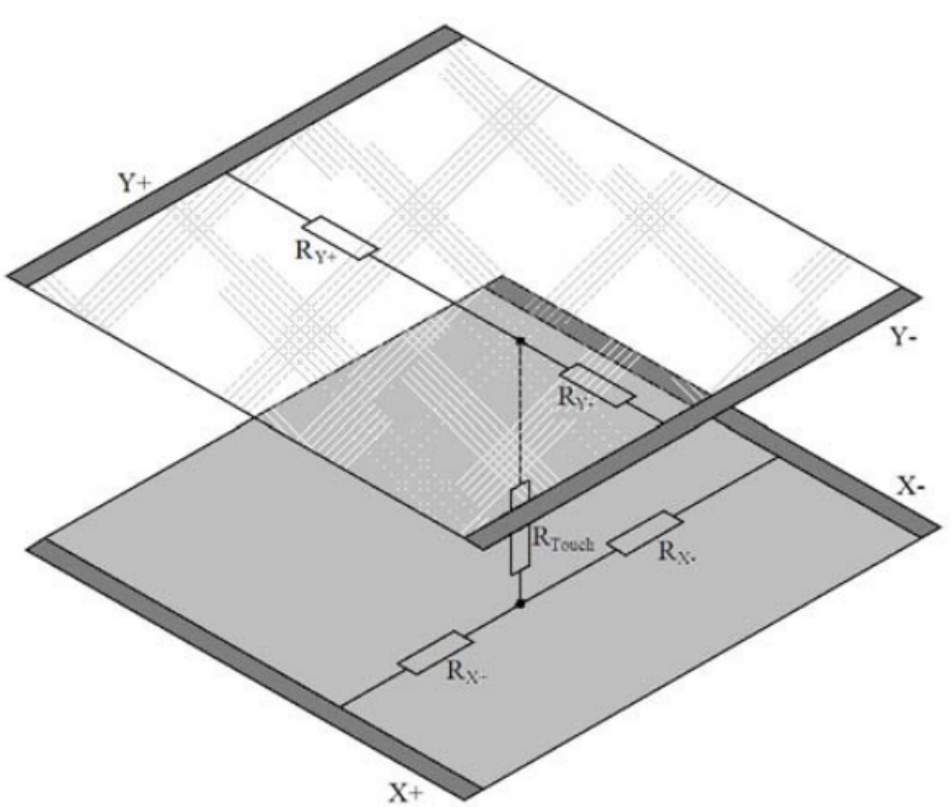
How Analog resistive touch screens work

- 4-wire touch screens use a single pair of electrodes (“Busbars”) on each ITO layer. The busbars in the topsheet and substrate are perpendicular to each other. The busbars are connected to the touch screen controller through a 4-wire flex cable. The 4 wires are referred as X+ (left), X- (right), Y+ (top) and Y- (bottom).
- An advantage of the 4-wire touch screens is that it is possible to determine the touch pressure by measuring the contact resistance (RTouch) between the two ITO layers. RTouch decreases as the touch pressure (or the size of the depressed area) increases. This characteristic can be useful in applications in which it is not only required to detect where the pressure is applied, but also the type of pressure (area and force).



Measuring Touch Point

- Method for measuring the pressure point is based upon a preferably homogenous resistive surface (ITO).
- When applying a voltage to electrode pair in resistive surface a uniform voltage gradient appears across the surface.
- A second ITO layer is necessary to do a high-resistance voltage measurement.
- A resistive touch screen can be seen as an electrical switch requiring a small amount of pressure to close.



Measuring Touch Point

- The point of contact “divides” each layer in a series resistor network with two resistors, and a connecting resistor between the two layers.
- By measuring the voltage at this point the user gets information about the position of the contact point orthogonal to the voltage gradient.
- To get a complete set of coordinates, the voltage gradient must be applied once in vertical and then in horizontal direction.

	X+excite	X-excite	Y+excite	Y-excite
Standby	Gnd	Hi-Z	Hi-Z	Pull up/Int
X-coordinate, Z1	Gnd	**Vcc**	Hi-Z	**Hi-Z/ADC**
Y-coordinate, Z2	Hi-Z	**Hi-Z/ADC**	Gnd	**Vcc**

4-wire touch screens scanning

Continued:

- So essentially, we need 4 digital IO pins and two analog inputs (ADC) but since the pins on the Atmega328p can be configured as both analog input and digital IO, 4 pins are sufficient for a four wire touch screen controller.
- So the two ADC input channels are connected to X- and Y- pins and the other two pins on the same port. Following on the scanning data from the table :

  1. X: To read X coordinate first, we will put X- and X+ in digital mode and set X- high and X+ low.

  2. X: Then we set Y- , Y+ to ADC input mode and read the Y- ADC value as X coordinate on Touchscreen.

  3. Y: Then we switch to read Y coordinate, for which we put Y- and Y+ to digital mode and set Y+ to low and Y- to high.

  4. Y: Finally, we set X- and X+ to ADC input mode and read the X- ADC value as Y coordinate on Touchscreen.

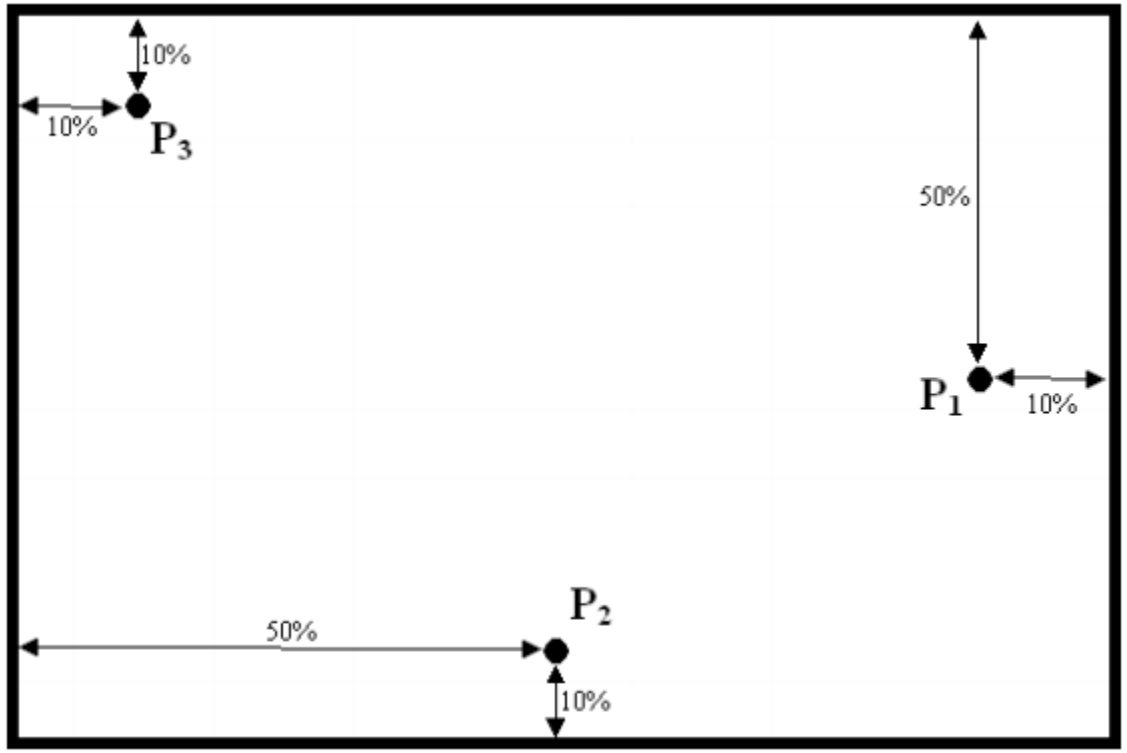
  - We continuously do above 4 steps in order to read for (x,y) coordinates on touchscreen.

Calibration of Touch Screens

- We need to calibrate the Touchscreen to translate the measured touch screen data into true screen coordinates of the LCD with which it is interfaced.
- The generic algorithm has to compensate for scaling errors, offset and rotation of the touch screen coordinates relative to display coordinates.
- To eliminate these error factors, six calibration coefficients (A,B,C,D,E,F) are required.
- Touch screen coordinates can be converted to display coordinates using following :
  - $Xd = A(Xt) + B(Yt) + C$
  - $Yd = D(Xt) + E(Yt) + F$
- Three sample points (P1, P2, P3) are needed to get the six calibration coefficient values.

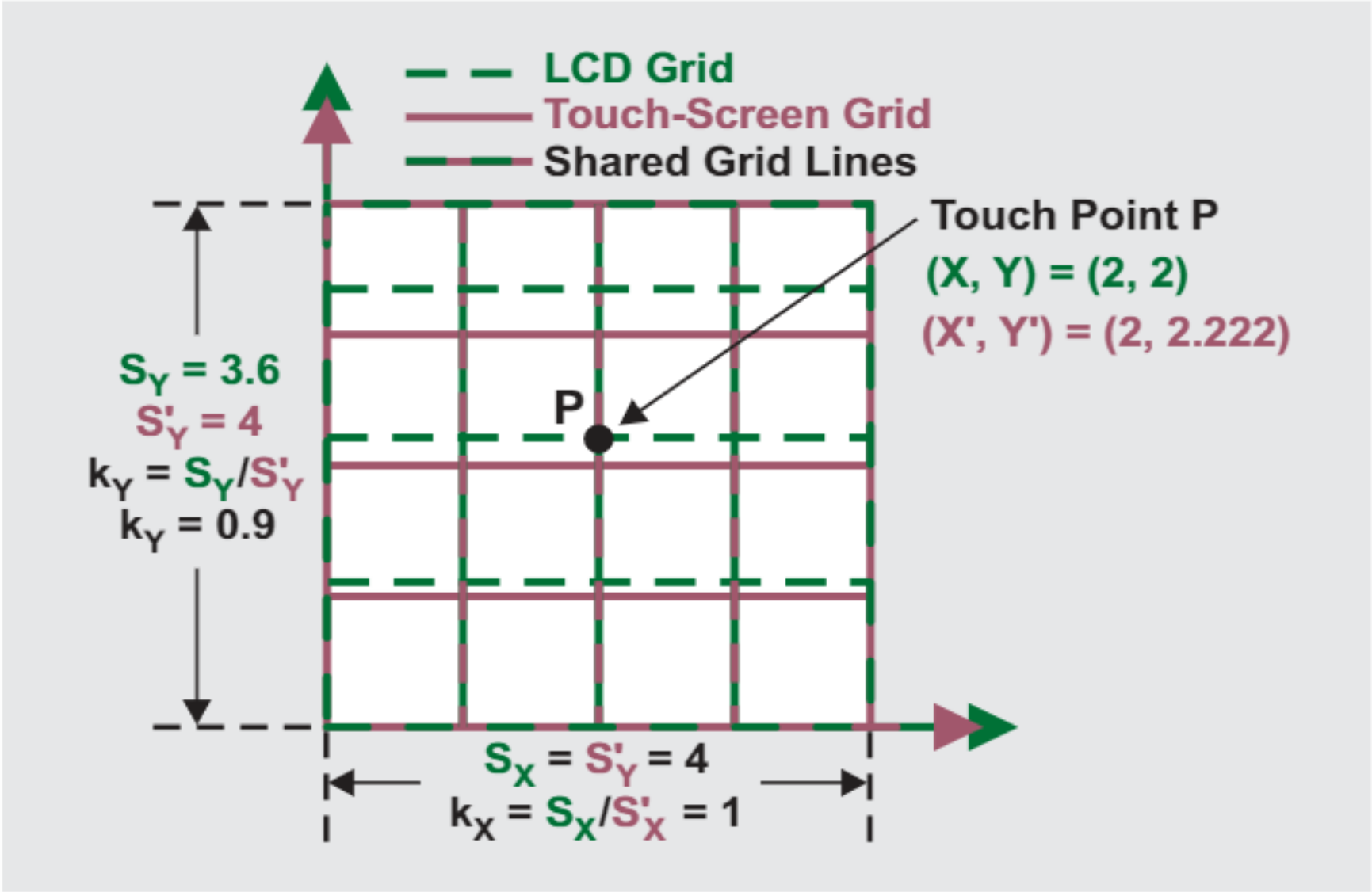
Continued..

- The sample points are used to create a set of linear equations.
  - Xd1 = A(Xt1) + B(Yt1) + C
  - Yd1 = D(Xt1) + E(Yt1) + F
  - Xd2 = A(Xt2) + B(Yt2) + C
  - Yd2 = D(Xt2) + E(Yt2) + F
  - Xd3 = A(Xt3) + B(Yt3) + C
  - Yd3 = D(Xt3) + E(Yt3) + F
- Solving these equations will give us values of A, B, C, D, E and F
- For most purposes, there is no rotation between LCD and Touchscreen and we can avoid this algorithm



Continued..

- Example, for the lab the touch screen controller (1024x1024, 10 bit ADC controller) and LCD display (128x64 pixels) do not have the same resolution, so simple scaling factors are needed to match their coordinates to each other.
- Consider a touch-screen system that uses an:
  - LCD with a resolution of 1024 (X coordinate) × 768 (Y coordinate)
  - Texas Instruments TSC2005 touch-screen controller with 12-bit (4096 × 4096) resolution
- The scaling factors to match them are kX = SX/S'X = 1024/4096 = 0.25 for the X-axis coordinate and kY= SY/S'Y= 768/4096 = 0.1875 for the Y-axis coordinate, where SX is the LCD's X-axis resolution, S'X is the touch-screen controller's X-axis resolution, SYis the LCD's Y-axis resolution, and S'Y is the touch-screen controller's Y-axis resolution.
- Thus, a touch-screen controller's X coordinate, X', should be understood by the LCD (the host) as X = kX × X'; and a touch-screen controller's Y coordinate, Y', should be understood by the LCD (the host) as Y = kY×Y' .



Scaling factors on the Y axes of LCD and Touch screen