

# Shield Speed Optimizations

## Overview

- LCD controller interface
- LCD bitmapped text
- Using SIMD for pixel data formatting

# LCD CONTROLLER INTERFACE

# LCD Controller IC – ST7789S

**Sitronix**

**ST7789S**

240RGB x 320 dot 262K Color with Frame Memory  
Single-Chip TFT Controller/Driver

**Datasheet**

Version 1.5  
2013/04

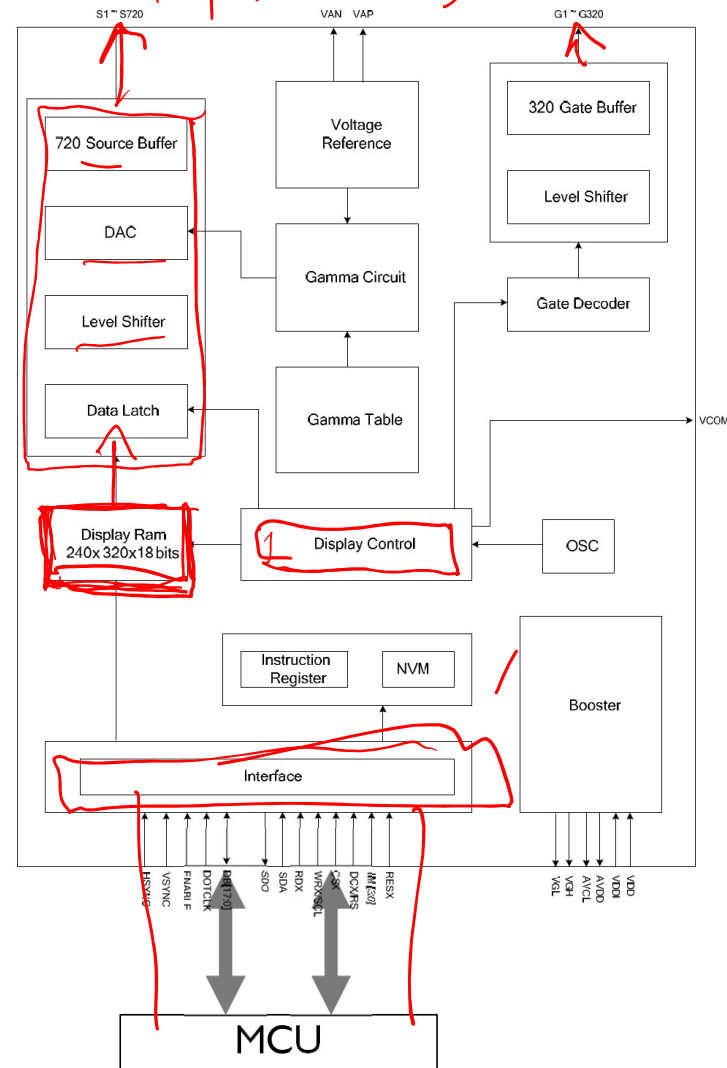
Sitronix Technology Corporation

Sitronix Technology Corp. reserves the right to change the contents in this document without prior notice.

240 cols, 3 pixels

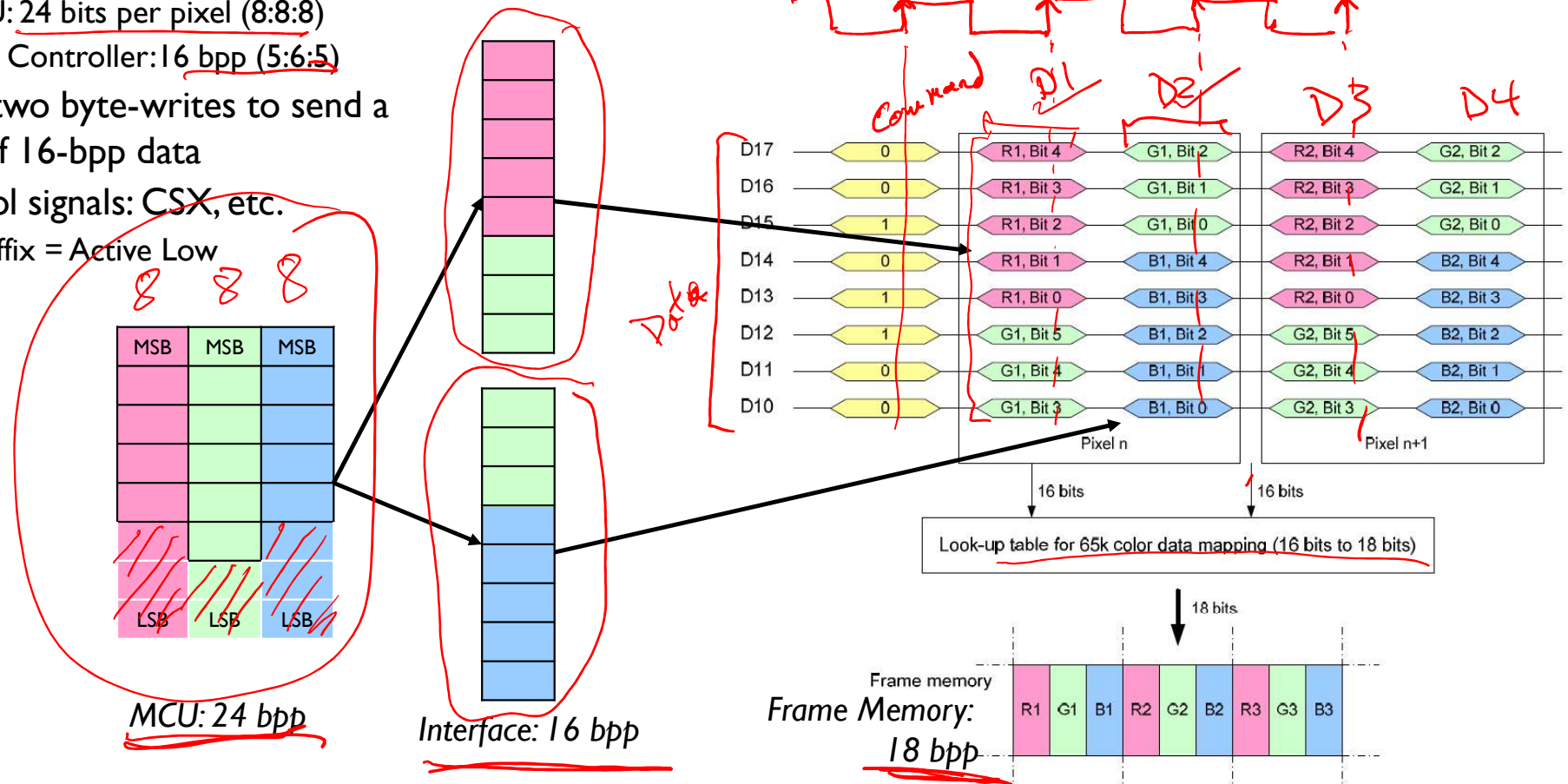
320 rows

NC STATE UNIVERSITY



# Pixel Data (8-Bit interface, 65k colors)

- Different pixel formats
  - MCU: 24 bits per pixel (8:8:8)
  - LCD Controller: 16 bpp (5:6:5)
- Takes two byte-writes to send a pixel of 16-bpp data
- Control signals: CSX, etc.
  - X suffix = Active Low



## LCD\_Plot\_Pixel

### Set-up operations

- Set column address: 5 writes
- Set page (row) address: 5 writes

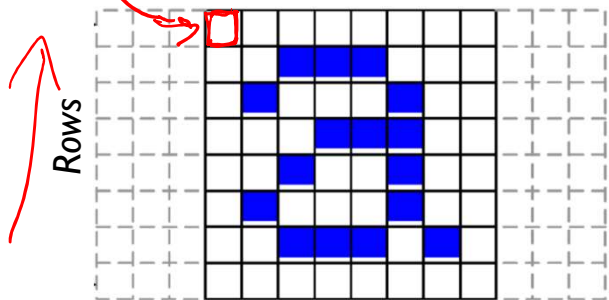
### Data write operation

- Write to memory command: 1 write
- MSB, LSB: 2 bytes, 2 writes

### Total LCD operations: 13/pixel

### Full screen update

- 240 x 320 pixels \* 13 write operations/pixel  
= ~1M write operations



```

/* Set the pixel at pos to the given color. */
void LCD_Plot_Pixel(PT_T * pos, COLOR_T * color) {
    uint8_t b1, b2;

    // Column address set 0x2a
    1 LCD_24S_Write_Command(0x002A); //column address set
    LCD_24S_Write_Data(0);

    4 LCD_24S_Write_Data(pos->X & 0xff); //start
    LCD_24S_Write_Data(0x0000);
    LCD_24S_Write_Data(0x00EF); //end 0x00EF

    // Page (row) address set 0x2b
    1 LCD_24S_Write_Command(0x002B); //page address set
    LCD_24S_Write_Data(pos->Y >> 8);
    4 LCD_24S_Write_Data(pos->Y & 0xff); //start
    LCD_24S_Write_Data(0x0001);
    LCD_24S_Write_Data(0x003F); //end 0x013F

    // Memory Write 0x2c
    // 16 bpp, 5-6-5. Assume color channel data is left-aligned
    b1 = (color->R&0xf8) | ((color->G&0xe0)>>5);
    b2 = ((color->G&0x1c)<<3) | ((color->B&0xf8)>>3);

    3 LCD_24S_Write_Command(0x002c);
    LCD_24S_Write_Data(b1);
    LCD_24S_Write_Data(b2);
}

```

# Commands

## 9.1.22 RAMWR (2Ch): Memory Write

2CH	RAMWR (Memory Write)												
Inst / Para	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
RAMWR	0	↑	1	-	0	0	1	0	1	1	0	0	(2Ch)
1 <sup>st</sup> parameter	1	↑	1	D1[17]-1[8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]	
...	1	↑	1	Dx[17]-x[8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]	
N parameter	1	↑	1	Dn[17]-n[8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]	
Description	<p>-This command is used to transfer data from MCU to frame memory.</p> <p>-When this command is accepted, the column register and the page register are reset to the start column/start page positions.</p> <p>-The start column/start page positions are different in accordance with MADCTL setting.</p> <p>-Sending any other command can stop frame write.</p>												

# Defining Rectangle Start and End Addresses

## 9.1.20 CASET (2Ah): Column Address Set

2AH	CASET (Column Address Set)												
Inst / Para	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
CASET	0	↑	1	-	0	0	1	0	1	0	1	0	(2Ah)
1 <sup>st</sup> parameter	1	↑	1	-	XS15	XS14	XS13	XS12	XS11	XS10	XS9	XS8	
2 <sup>nd</sup> parameter	1	↑	1	-	XS7	XS6	XS5	XS4	XS3	XS2	XS1	XS0	
3 <sup>rd</sup> parameter	1	↑	1	-	XE15	XE14	XE13	XE12	XE11	XE10	XE9	XE8	
4 <sup>th</sup> parameter	1	↑	1	-	XE7	XE6	XE5	XE4	XE3	XE2	XE1	XE0	
-The value of XS [7:0] and XE [7:0] are referred when RAMWR command comes. -Each value represents one column line in the Frame Memory.													

## 9.1.21 RASET (2Bh): Row Address Set

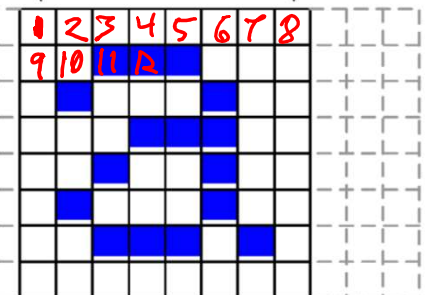
2BH	RASET (Row Address Set)												
Inst / Para	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
RASET	0	↑	1	-	0	0	1	0	1	0	1	1	(2Bh)
1 <sup>st</sup> parameter	1	↑	1	-	YS15	YS14	YS13	YS12	YS11	YS10	YS9	YS8	
2 <sup>nd</sup> parameter	1	↑	1	-	YS7	YS6	YS5	YS4	YS3	YS2	YS1	YS0	
3 <sup>rd</sup> parameter	1	↑	1	-	YE15	YE14	YE13	YE12	YE11	YE10	YE9	YE8	
4 <sup>th</sup> parameter	1	↑	1	-	YE7	YE6	YE5	YE4	YE3	YE2	YE1	YE0	
-This command is used to defined area of frame memory where MCU can access. -The value of YS [15:0] and YE [15:0] are referred when RAMWR command comes. -Each value represents one page line in the Frame Memory.													

YS[15:0]

XS[7:0]

XE[7:0]

YE[15:0]





## Drawing Rectangles

- Controller can accept multiple data values
  - Will store data in consecutive locations (increasing addresses)
  - Will wrap address based on XS and XE, YS and YE

### 9.1.22 RAMWR (2Ch): Memory Write

2CH	RAMWR (Memory Write)												
Inst / Para	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
RAMWR	0	↑	1	-	0	0	1	0	1	1	0	0	(2Ch)
1 <sup>st</sup> parameter	1	↑	1	D1[17]-1[8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]	
...	1	↑	1	Dx[17]-x[8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]	
N parameter	1	↑	1	Dn[17]-n[8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]	
Description	<p>-This command is used to transfer data from MCU to frame memory.</p> <p>-When this command is accepted, the column register and the page register are reset to the start column/start page positions.</p> <p>-The start column/start page positions are different in accordance with MADCTL setting.</p> <p>-Sending any other command can stop frame write.</p>												

## Drawing Rectangles

### Set-up operations

- Set column address: 5 writes
- Set page (row) address: 5 writes
- Start the 0x2C write command, but don't send data yet: 1 write

### Data write operation

- MSB, LSB: 2 writes

### Total LCD Operations: 11 + 2/pixel

- Compare with 13 LCD operations/pixel in LCD\_Plot\_Pixel

```
uint32_t LCD_Start_Rectangle(PT_T * p1, PT_T * p2) {
    uint32_t n;
    uint16_t c_min, c_max, r_min, r_max;

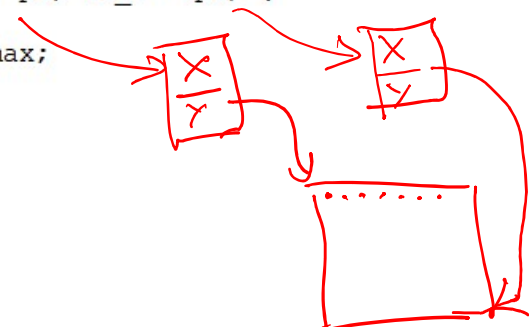
    // Find bounds of rectangle
    c_min = MIN(p1->X, p2->X);
    c_max = MAX(p1->X, p2->X);

    r_min = MIN(p1->Y, p2->Y);
    r_max = MAX(p1->Y, p2->Y);

    // Clip to display size
    c_max = MIN(c_max, LCD_WIDTH-1);
    r_max = MIN(r_max, LCD_HEIGHT-1);

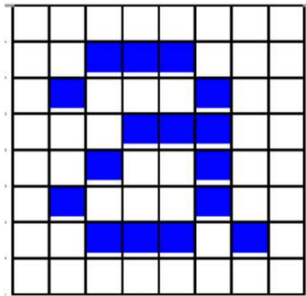
    n = (c_max - c_min + 1) * (r_max - r_min + 1);
    if (n > 0) {
        // Enable access to full screen, reset write pointer to origin
        LCD_24S_Write_Command(0x002A); //column address set
        LCD_24S_Write_Data(c_min >> 8);
        LCD_24S_Write_Data(c_min & 0xff); //start
        LCD_24S_Write_Data(c_max >> 8);
        LCD_24S_Write_Data(c_max & 0xff); //end
        LCD_24S_Write_Command(0x002B); //page address set
        LCD_24S_Write_Data(r_min >> 8);
        LCD_24S_Write_Data(r_min & 0xff); //start
        LCD_24S_Write_Data(r_max >> 8);
        LCD_24S_Write_Data(r_max & 0xff); //end

        // Memory Write 0x2c
        LCD_24S_Write_Command(0x002c);
    }
    return n;
}
```

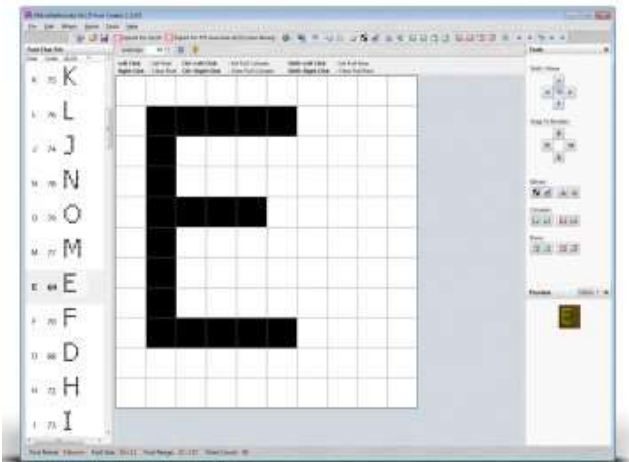


# LCD BITMAPPED TEXT

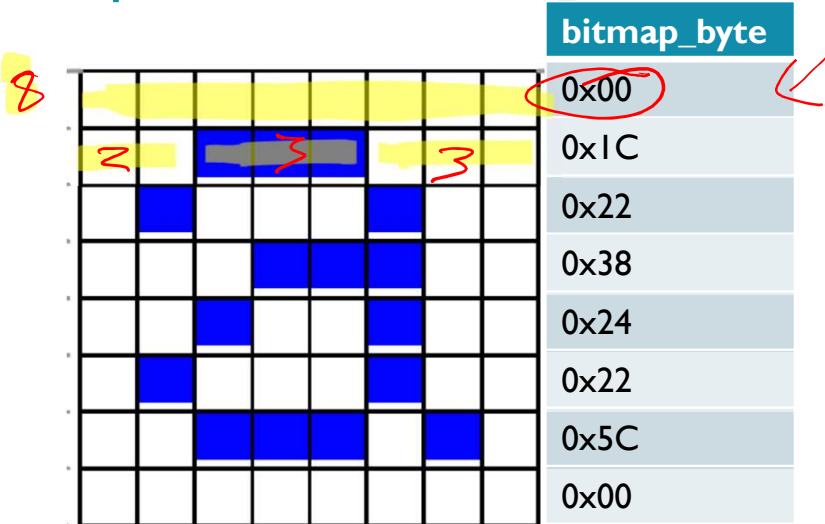
## Providing Text for LCD



- LCD controller does not have built-in character bitmaps
- Instead, user code has to render (draw) text
- Two options
  - Bitmap: Set pixels based on bitmap – fast (but ugly if scaled much)
  - Vector: Draw a series of lines – slow (but beautiful with scaling)
- Will use bitmap (for speed)
  - Use free tool (GLCD Font Creator) to generate bitmaps from Windows fonts: <https://www.mikroe.com/glcd-font-creator>
  - Bitmap uses 1 for character foreground, 0 for background
  - Code needs to read bitmap and output foreground or background color for each pixel in bitmap



## Simple Text Rendering Code



- Loop iterates over each column in bitmap
- Calls LCD\_Plot\_Pixel per pixel
- Slow because of set-up overhead before each pixel of data

```
pixel_pos.Y = pos->Y;
```

```
for (row = 0; row < CHAR_HEIGHT; row++) {
    pixel_pos.X = pos->X;
    x_bm = 0;
    do {
        bitmap_byte = *glyph_data;
        for (col = 0; col < 8; col++) {
            if (bitmap_byte & 0x01) // if pixel is to be set
                pixel_color = fg_color;
            else
                pixel_color = bg_color;
            LCD_Plot_Pixel(&pixel_pos, pixel_color);
            bitmap_byte >>= 1;
            pixel_pos.X++;
            x_bm++;
        }
        glyph_data++;
    } while (x_bm < width);
    pixel_pos.Y++;
}
```



## LCD\_PrintChar Optimized for Pixel Runs



- Draws a rectangle for each run of pixels
- Identifies runs of pixels by examining existing bitmap data
  - If 0000 0000, then draw run of 8 background pixels
  - If 1111 1111, then draw run of 8 foreground pixels
  - If x000 0000, then draw run of 7 background pixels
  - Et cetera
  - Draw any remaining pixels in byte individually
- Could improve performance further by changing bitmap data to encode run information

```
// Special cases with run starting at LSB
// Up to 8 bit run
if (bitmap_byte == 0x00) {
    num_pixels = MIN(8, glyph_width - x_bm);
    LCD_Write_Rectangle_Pixel(&bg, num_pixels);
    x_bm += num_pixels;
} else if (bitmap_byte == 0xff) {
    num_pixels = MIN(8, glyph_width - x_bm);
    LCD_Write_Rectangle_Pixel(&fg, num_pixels);
    x_bm += num_pixels;
} else {
    col = 0;
    num_pixels = 0;
    if ((bitmap_byte & 0x7f) == 0) { // Up to 7 bit run ✓
        num_pixels = MIN(7, glyph_width - x_bm);
        LCD_Write_Rectangle_Pixel(&bg, num_pixels);
    } else if ((bitmap_byte & 0x7f) == 0x7f) {
        num_pixels = MIN(7, glyph_width - x_bm);
        LCD_Write_Rectangle_Pixel(&fg, num_pixels);
    } else if ((bitmap_byte & 0x3f) == 0) { // Up to 6 bit run
```

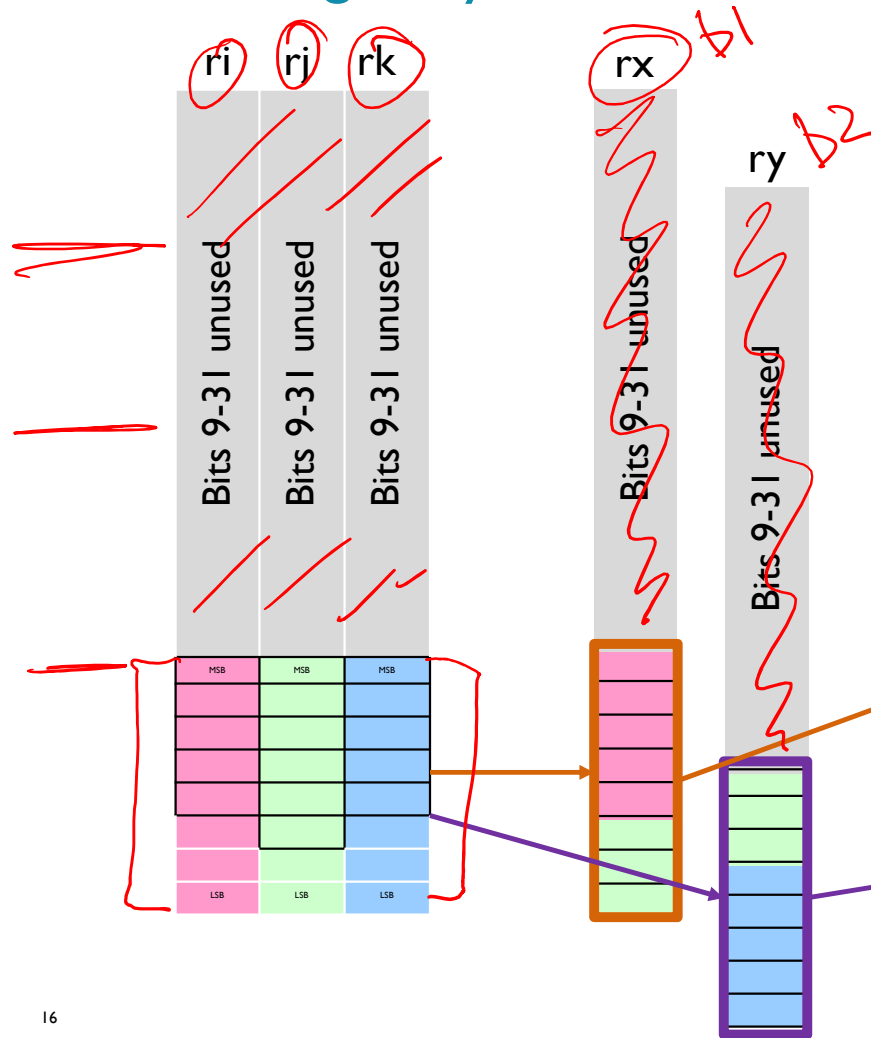
*Background color*

*foreground color*

# USING SIMD FOR PIXEL DATA REFORMATTING

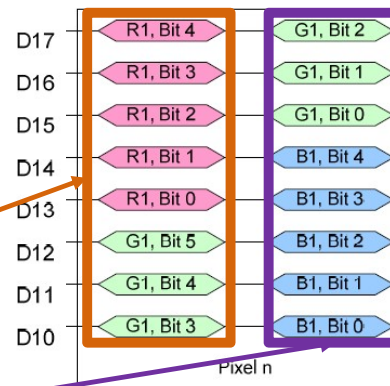


## We're Using Only Part of 32-bit Data Path in CPU



```
// Memory Write 0x2c
// 16 bpp, 5-6-5. Assume color channel data is left-aligned
b1 = (color->R&0xf8) | ((color->G&0xe0)>>5);
b2 = ((color->G&0x1c)<<3) | ((color->B&0xf8)>>3);

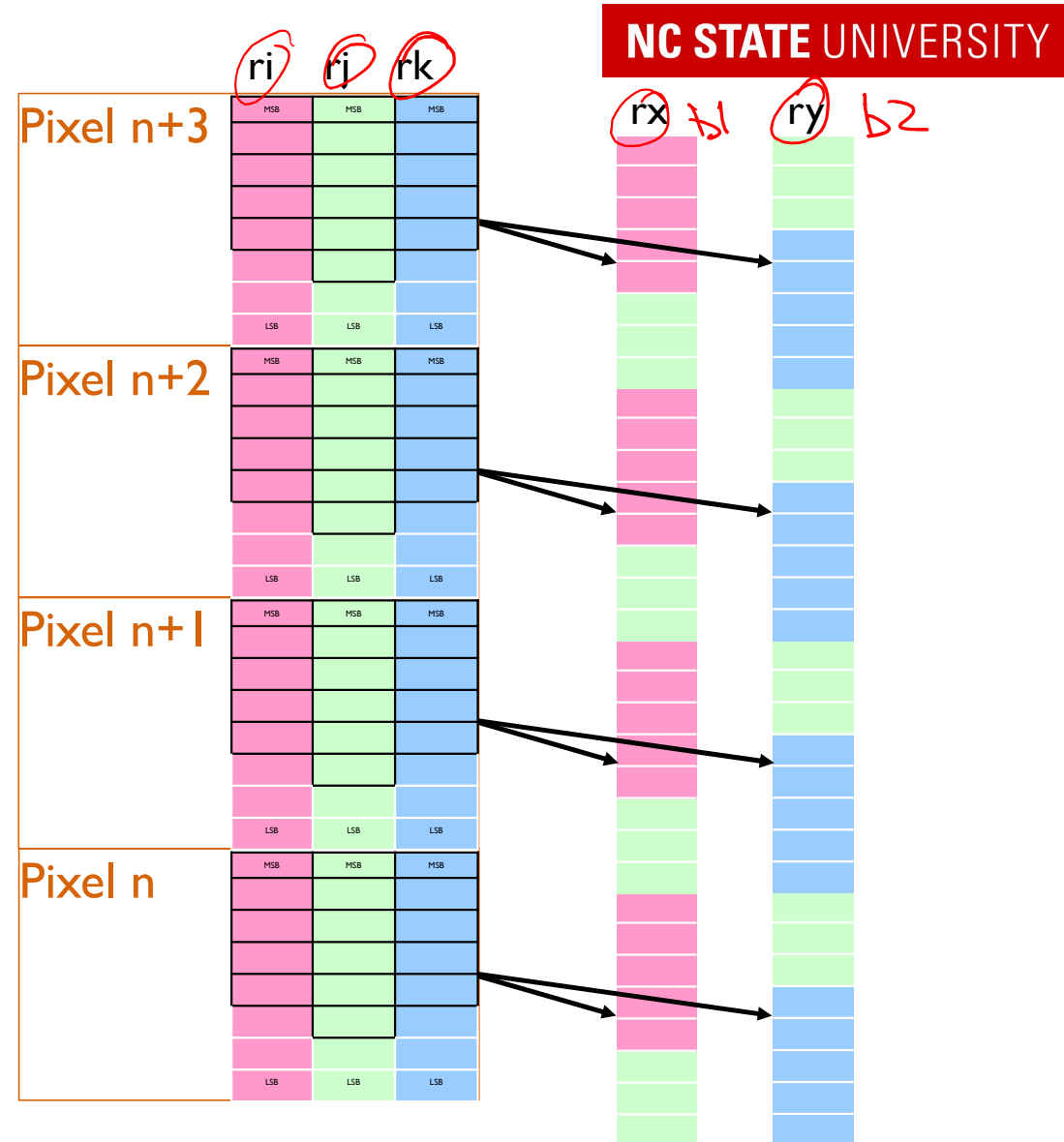
LCD_24S_Write_Command(0x002c);
LCD_24S_Write_Data(b1);
LCD_24S_Write_Data(b2);
```

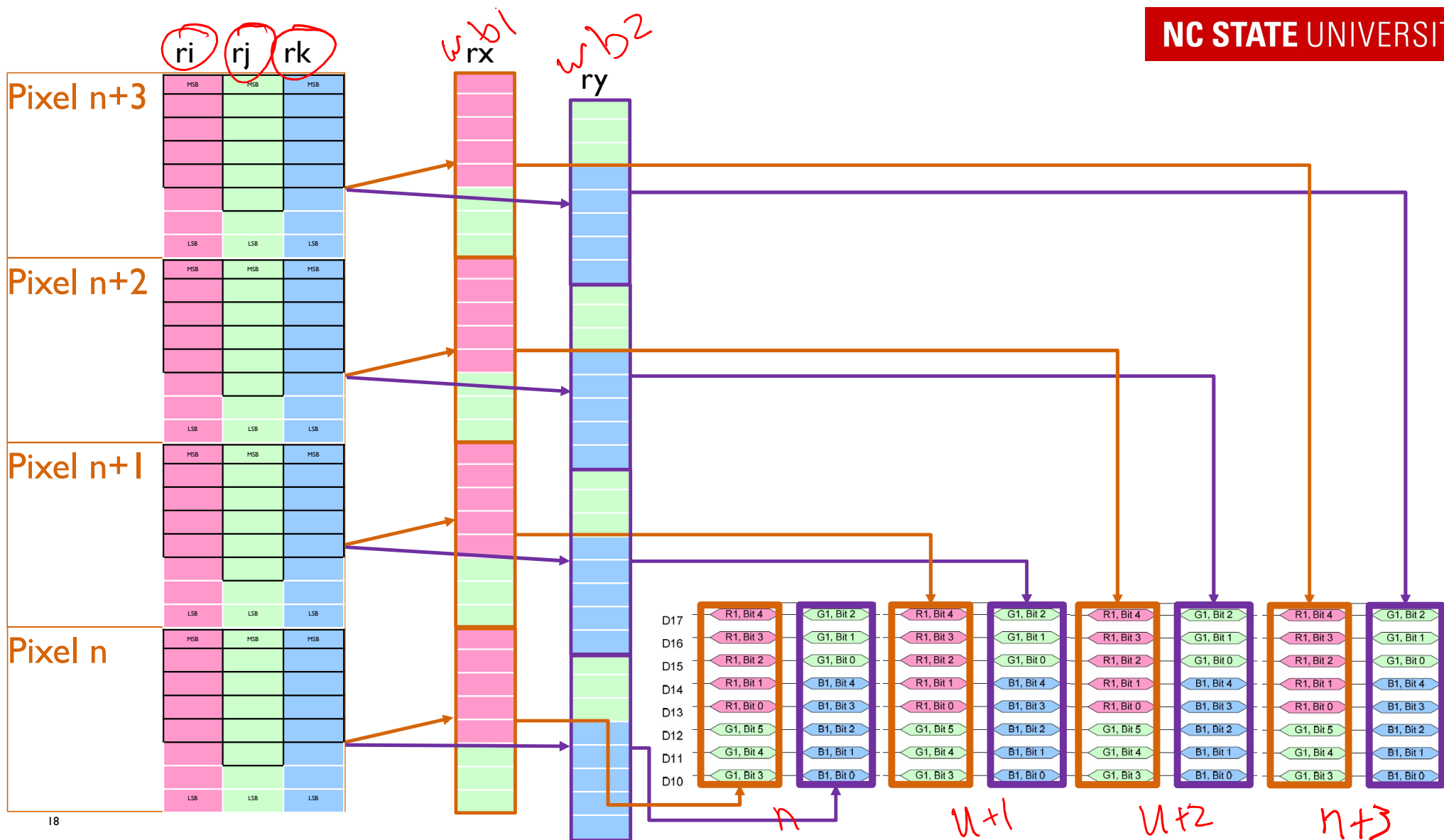




## Four-Wide SIMD?

- SIMD: Single Instruction, Multiple Data
  - Have each register hold multiple data elements (mini-vector)
  - Now one instruction can process multiple data elements
- Want to process four bytes in parallel in a 32-bit register





## Must Reorganize Data and Interface

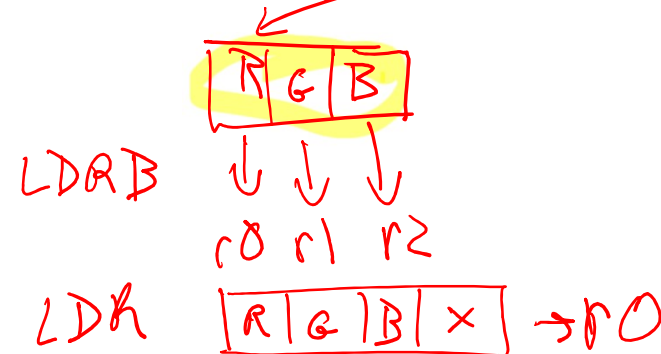
### Original code

- Pixel color is in a structure with color components
  - struct { uint8\_t R, G, B; } COLOR\_T;
- Image is an array of structures
  - COLOR\_T image[W][H]
- One component of one pixel is loaded into each register
- Loading register with full word from memory? Will just get R, G, B components of single pixel: not useful

### Changes needed

- Pass at least four pixels of data to function at a time
- Reorganize data in memory so loading a register with a word will get a component (e.g. R) from four adjacent pixels

```
void LCD_Plot_Pixel(Pixel_T * pos, COLOR_T * color) {
    uint8_t b1, b2;
    b1 = (color->R&0xf8) | ((color->G&0xe0)>>5);
    b2 = ((color->G&0x1c)<<3) | ((color->B&0xf8)>>3);
}
```



## New Data Organization and Interface

- Pass at least four pixels of data to function at a time
- Reorganize data in memory
  - Loading a register (LDR) should get a component (e.g. R) from four adjacent pixels
  - Reorganize data into structure of arrays
    - struct {
 uint8\_t R[W\*H],
 G[W\*H],
 B[W\*H];
 }

```
void LCD_Write_Rectangle_N_Quad_Pixel_Components(
    uint32_t * aR, uint32_t * aG,
    uint32_t * aB, int32_t n){
```



## Four-Wide SIMD

- Load three color components into three registers
  - R: Four reds, G: four greens, B: four blues
- Mask off and shift color component bits of interest
  - Four reds, four high greens, four low greens, four blues
- Merge to create W1 (set of four first bytes)
- Merge to create W2 (set of four second bytes)
- Send out four pairs of bytes sequentially
  - Extract b1 and b2 from W1 and W2
  - Write the data
  - Shift W1 and W2 to prep for next pair of bytes

```
void LCD_Write_Rectangle_N_Quad_Pixel_Components(
    uint32_t * aR, uint32_t * aG,
    uint32_t * aB, int32_t n){
```

```
    uint8_t b1, b2; uint8_t i;
    uint32_t R, G, B, GH, GL, W1, W2;
    do {
```

```
        R = *aR++;
        G = *aG++;
        B = *aB++;
        R &= 0xf8f8f8f8;
        GH = (G & 0xe0e0e0e0) >> 5;
        GL = (G & 0x1c1c1c1c) >> 2;
        B = (B & 0xf8f8f8f8) >> 3;
        W1 = R | GH;
        W2 = GL | B;
        for (i=0; i<4; i++) {
            b1 = W1 & 0x000000ff;
            b2 = W2 & 0x000000ff;
            LCD_24S_Write_Data(b1);
            LCD_24S_Write_Data(b2);
            W1 >>= 8;
            W2 >>= 8;
```

```
        }
```

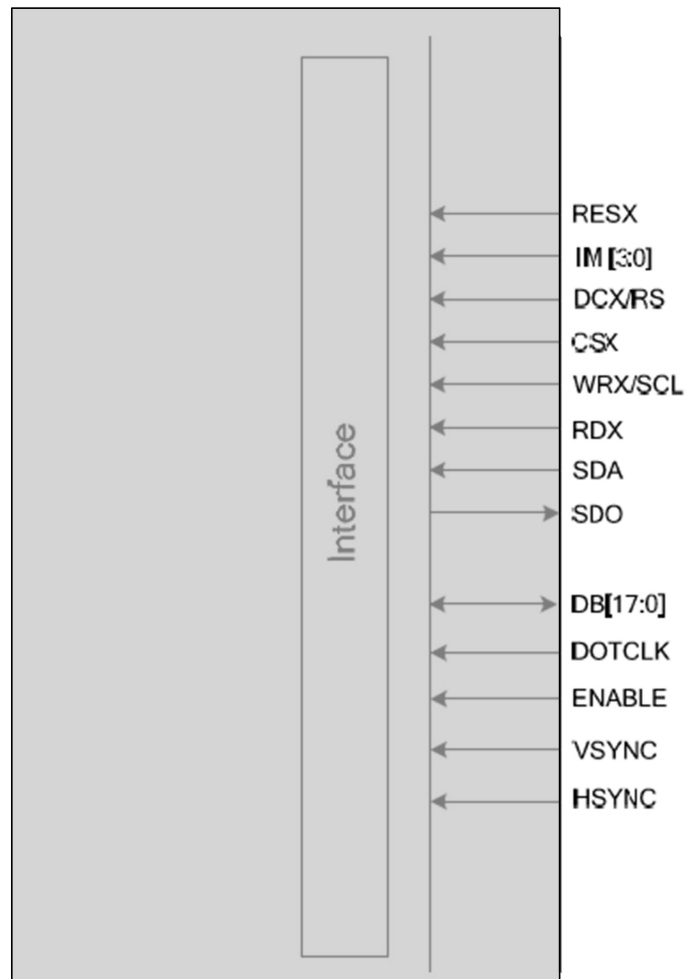
```
    }
```

# APPENDIX

## Commands

Software Reset	Vertical Scrolling Definition	Read ID3	Register Value Selection 2
Read Display ID	Tearing Effect Line Off/On	RAM Control	Power Control 1
Read Display Status	Memory Data Access Control	RGB Interface Control	Enable VAP/VAN signal output
Read Display Power Mode	Vertical Scroll Start Address of RAM	Porch Setting	Positive/Negative Voltage
Read Display MADCTL	Idle Mode Off/On	Frame Rate Control 1 (In partial mode/ idle colors)	Gamma Control
Read Display Pixel Format	Interface Pixel Format	Gate Control	Digital Gamma Look-up Table for Red/Blue
Read Display Image Mode	Read/Write Memory Continue	Digital Gamma Enable	Gate Control
Read Display Signal Mode	Get/Set Tear Scanline	VCOM Setting	SPI2 Enable
Read Display Self-Diagnostic Result	Read/Write Display Brightness	LCM Control	Power Control 2
Sleep In/Out	Read/Write CTRL Display	ID Code Setting	Equalize time control
Partial Display Mode On	Read/Write Content Adaptive	VDV and VRH Command Enable	Program Mode Control
Normal Display Mode On	Brightness Control and Color Enhancement	VRH Set	Program Mode Enable
Display Inversion Off/On	Read/Write CABC Minimum Brightness	VDV Set	NVM Setting
Gamma Set	Read ID1	VCOM Offset Set	Program action
Display Off/On	Read ID2	Frame Rate Control in Normal Mode	
Column Address Set		CABC Control	
Row Address Set		Register Value Selection 1	
Memory Read/Write			
Partial Area			

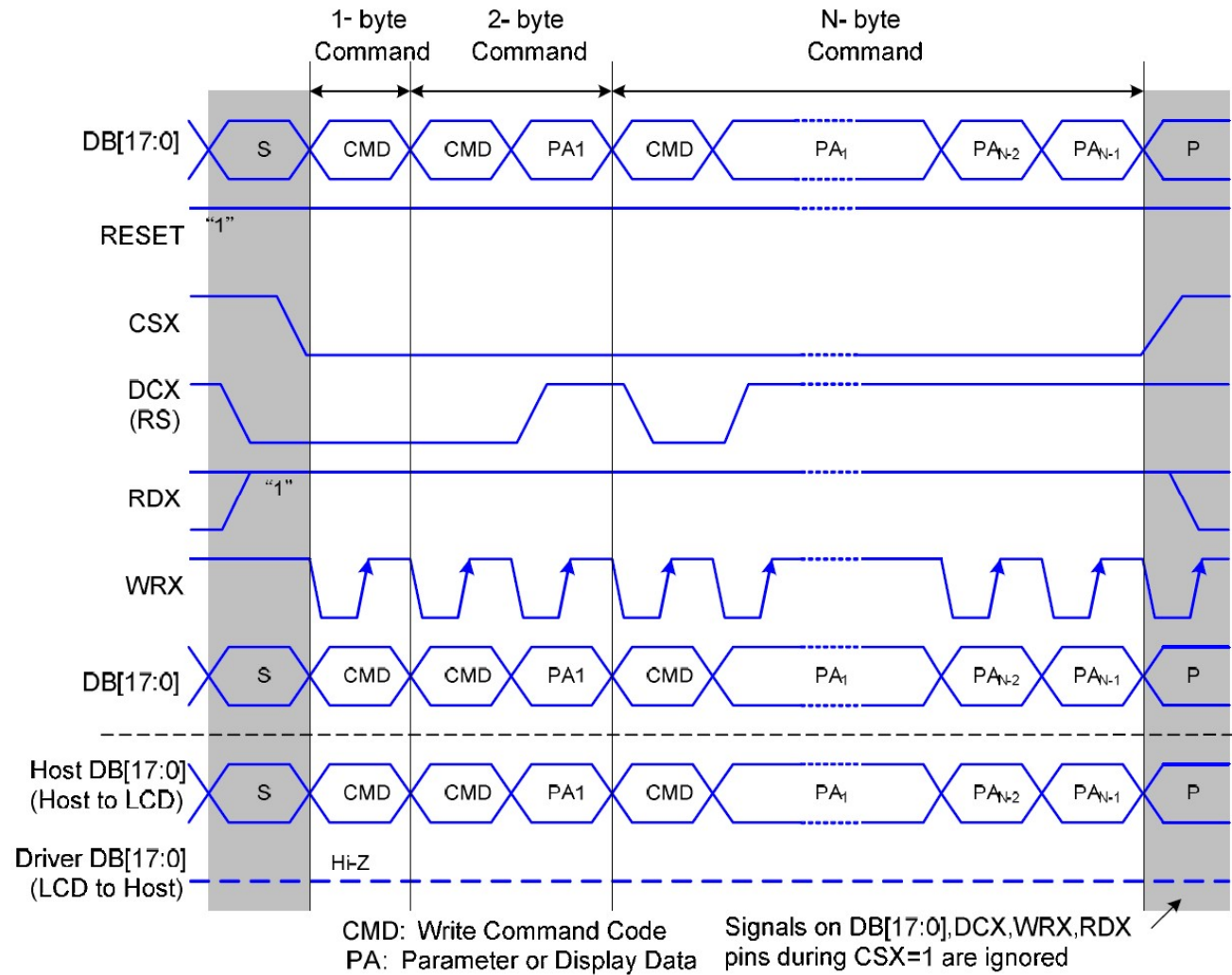
## Interface Signals



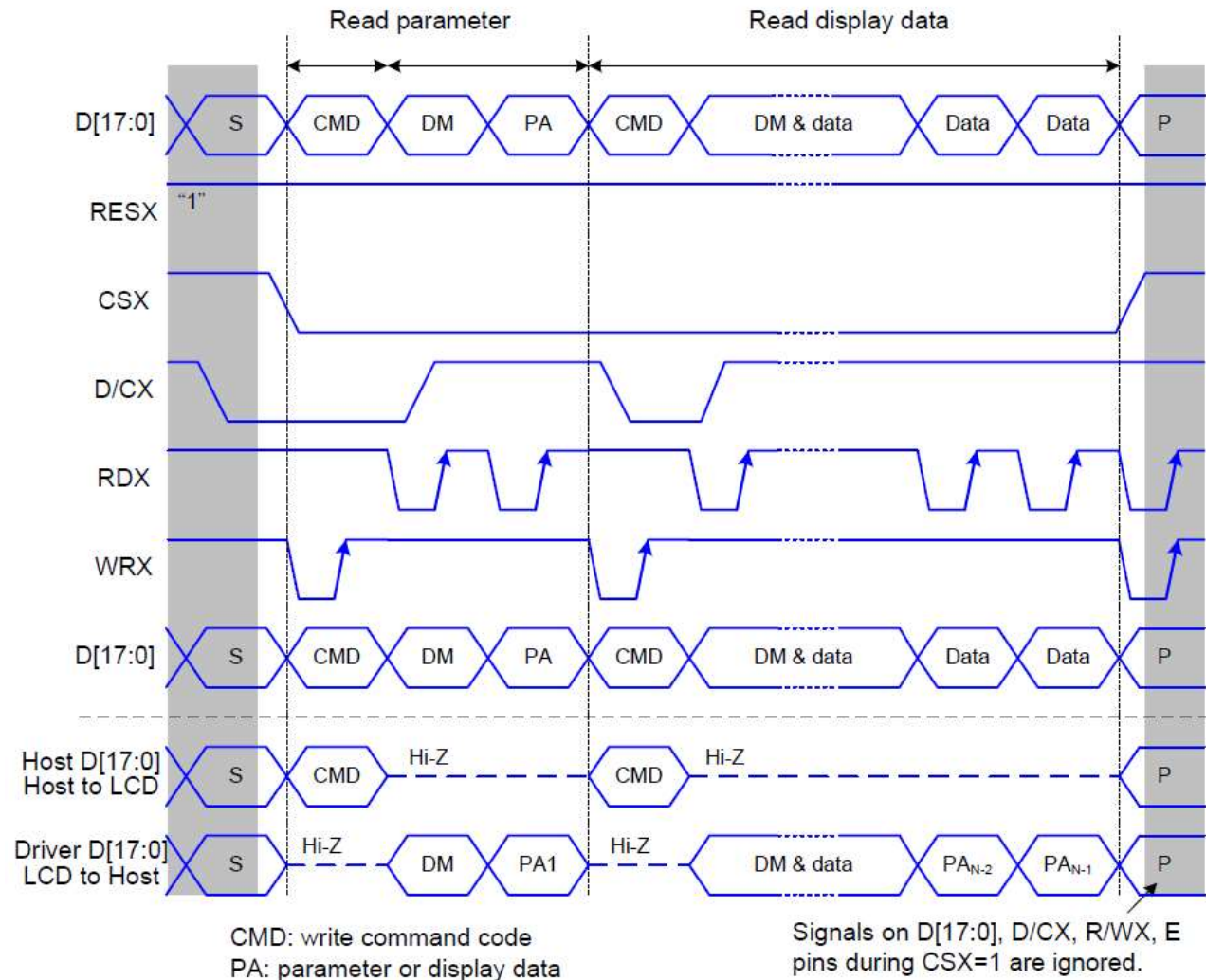
- Reset (X = Active Low)
- Interface Mode
- Data (1) / Command (0)
- Chip Select X
- Write X
- Read X
- Data Bus (18 bits)



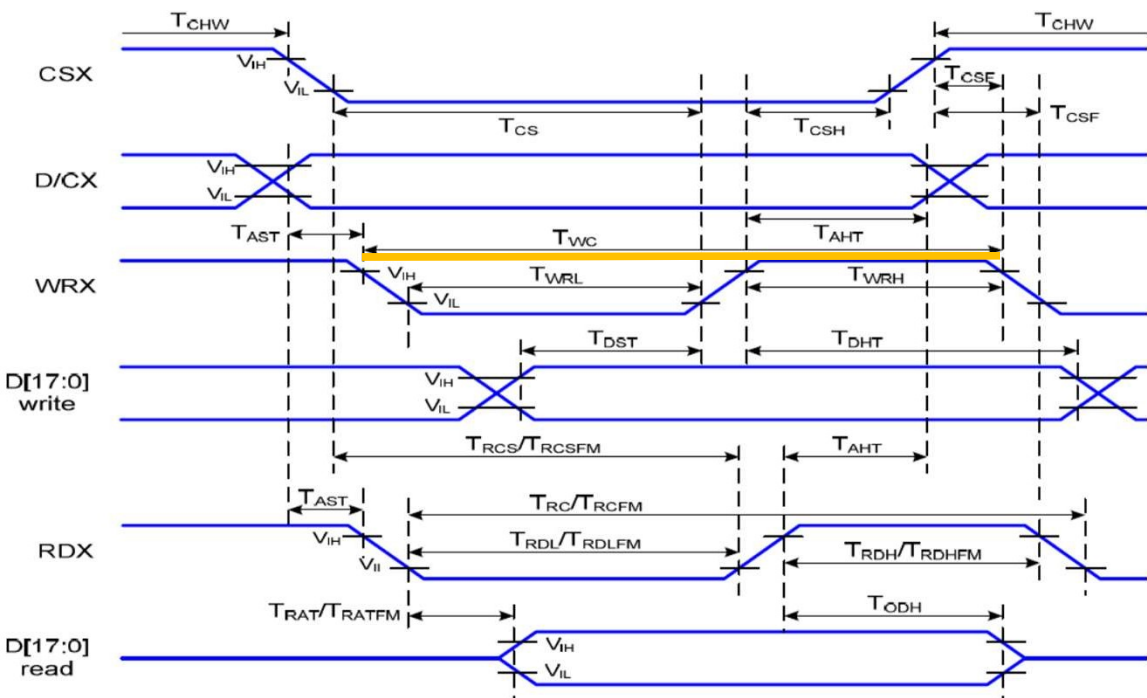
# Write Command or Data



## Read Parameter or Display Data



Parallel Interface Timing



- $66\text{ ns} * 48\text{ MHz} = 3.168$  instruction cycles
- Software-implemented bus **probably** won't exceed this minimum timing requirement
- DMA would be able to (if configured correctly)

VDDI=1.65 to 3.3V, VDD=2.4 to 3.3V, AGND=DGND=0V, Ta= -30 to 70 °C

Signal	Symbol	Parameter	Min	Max	Unit	Description
D/CX	$T_{AST}$	Address setup time	0		ns	-
	$T_{AHT}$	Address hold time (Write/Read)	10		ns	
CSX	$T_{CHW}$	Chip select "H" pulse width	0		ns	-
	$T_{CS}$	Chip select setup time (Write)	15		ns	
	$T_{RCS}$	Chip select setup time (Read ID)	45		ns	
	$T_{RCSFM}$	Chip select setup time (Read FM)	355		ns	
	$T_{CSF}$	Chip select wait time (Write/Read)	10		ns	
	$T_{CSH}$	Chip select hold time	10		ns	
WRX	$T_{WC}$	Write cycle	66		ns	
	$T_{WRH}$	Control pulse "H" duration	15		ns	
	$T_{WRL}$	Control pulse "L" duration	15		ns	
RDX (ID)	$T_{RC}$	Read cycle (ID)	160		ns	When read ID data
	$T_{RDH}$	Control pulse "H" duration (ID)	90		ns	
	$T_{RDL}$	Control pulse "L" duration (ID)	45		ns	
RDX (FM)	$T_{RCFM}$	Read cycle (FM)	450		ns	When read from frame memory
	$T_{RDHFM}$	Control pulse "H" duration (FM)	90		ns	
	$T_{RDLFM}$	Control pulse "L" duration (FM)	355		ns	
D[17:0]	$T_{DST}$	Data setup time	10		ns	For CL=30pF

# Font Data Structures

