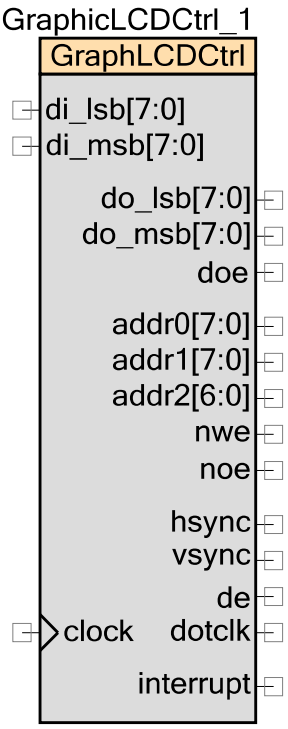
Graphic LCD Controller (GraphicLCDCtrl)

1.61

Features



- Fully programmable screen size support up to HVGA resolution including:
  - QVGA (320x240) @ 60 Hz 16 bpp
  - WQVGA (480x272) @ 60 Hz 16 bpp
  - HVGA (480x320) @ 60 Hz 16 bpp
- Supports virtual screen operation
- Interfaces with SEGGER emWin graphics library
- Performs read and write transactions during the blanking intervals
- Generates continuous timing signals to the panel without CPU intervention
- Supports up to a 23-bit address and a 16-bit data async SRAM device used as externally provided frame buffer
- Generates a selectable interrupt pulse at the entry and exit of the horizontal and vertical blanking intervals

General Description

The Graphic LCD Controller (GraphicLCDCtrl) component provides the interface to an LCD panel that has an LCD driver, but not an LCD controller. This type of panel does not include a frame buffer. The frame buffer must be provided externally.

This component also interfaces to an externally provided frame buffer implemented using a 16-bit-wide async SRAM device.

This component is designed to work with the SEGGER emWin graphics library. This graphics library is provided by Cypress to use with Cypress devices and is available on the Cypress website at www.cypress.com/go/comp_emWin. This graphics library provides a full-featured set of graphics functions for drawing and rendering text and images.

When to Use a GraphicLCDCtrl

The GraphicLCDCtrl component supports many LCD panels. It directly drives the control signals and manages the frame buffer in an external SRAM. The component generates the dotclk, hsync, vsync, and de outputs to access data from the SRAM and display it on the LCD using the.

The frame buffer SRAM can only be accessed for reads and writes when it is not refreshing the LCD panel. If a read or write is requested during the refresh period, the API functions provided will wait until the refresh gets to a blanking period. During the blanking period, the read or write is completed.

An interrupt can be used to indicate the entry and exit from blanking periods. This is particularly useful when coupled with an RTOS, which can swap in or swap out tasks that require access to the frame buffer when a blanking period is entered and exited.

Input/Output Connections

This section describes the input and output connections for the GraphicLCDCtrl component. Some I/Os may be hidden on the symbol under the conditions listed in the description of that I/O.

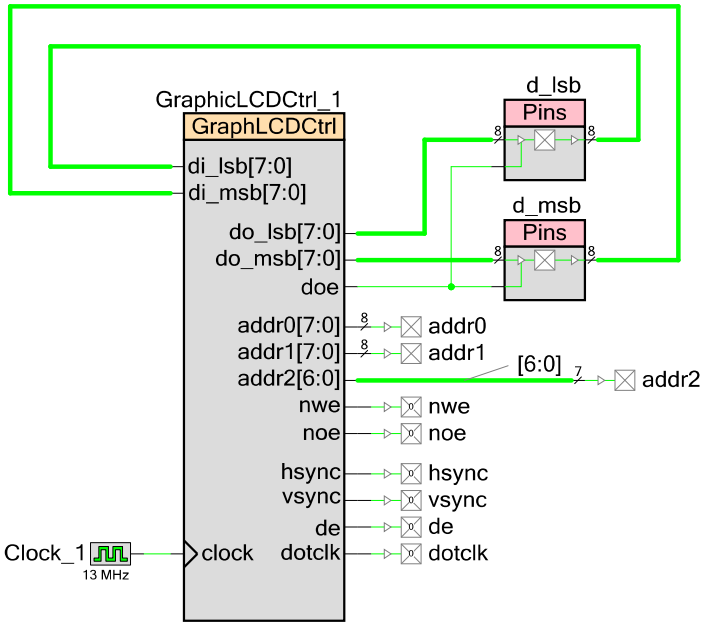
Input	May Be Hidden	Description
di_lsb[7:0]	N	The lower eight bits of the input data bus. They are used for data during a read transaction. Connect these signals to an input pin on the device and disable the “Input Synchronized” selection for these pins. The signals themselves are synchronized already because they are driven based on synchronous output signals.
di_msb[7:0]	N	The upper eight bits of the input data bus. They are used for data during a read transaction. Connect these signals to an input pin on the device and disable the “Input Synchronized” selection for these pins. The signals themselves are synchronized already because they are driven based on synchronous output signals.
clock	N	The clock that operates this component. It is twice the frequency of the dotclk.

Output	May Be Hidden	Description
do_lsb[7:0]	N	The lower eight bits of the output data bus. They are used for data during a write transaction.
do_msb[7:0]	N	The upper eight bits of the output data bus. They are used for data during a write transaction.
doe	N	The output enable for the data bus component within PSoC. It is normally connected to the output enable of the Input/Output pin component for the data buses.
addr0[7:0]	N	The lowest eight bits of the address bus connected to the frame buffer.
addr1[7:0]	N	The middle eight bits of the address bus connected to the frame buffer.
addr2[6:0]	N	The upper seven bits of the address bus connected to the frame buffer. The number of data bits needed by the frame buffer depends on the SRAM device you use.

Output	May Be Hidden	Description
nwe	N	Active-low write enable for the frame buffer SRAM.
noe	N	Active-low output enable for the frame buffer.
de	N	Data enable for the panel.
hsync	N	Horizontal sync timing signal for the panel.
vsync	N	Vertical sync timing signal for the panel.
dotclk	N	Clock driven to the panel. This clock is one-half the rate of the incoming clock.
interrupt	Y	Edge-triggered interrupt signal. This output is hidden if no interrupt generation is selected.

Schematic Macro Information

The macro is configured with the default settings to interface with the Optrex T-55343GD035JU-LW-AEN panel. The clock included in the macro is set to 13 MHz, which results in a 6.5-MHz dotclk provided to the Optrex QVGA panel.



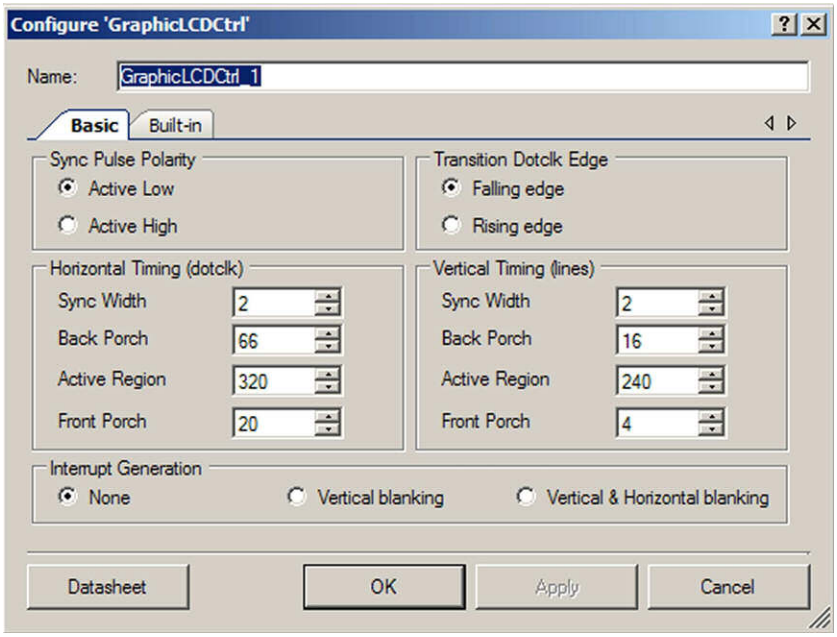
Typically, only some of the bits of the upper address bits (addr2) are used to connect to the frame buffer. How many depends on the size of the SRAM you use. Based on the number of address bits you need, you can use the following steps to adjust the size of that bus.

- Configure the addr2 output pin component and set the “Number of Pins.”
- Right-click on the signal driving the output pin and select “Edit Name And Width.” Then adjust the “Left Index” to reflect the width of the output pin.

The "Input Synchronized" option is unchecked on all of the data pins and the generation of API functions for all of the pins is turned off.

Component Parameters

Drag a GraphicLCDCtrl component onto your design and double-click it to open the **Configure** dialog. The default GraphicLCDCtrl settings are configured for operation with the Optrex panel.



The **Configure GraphicLCDCtrl** dialog contains the following parameters. All of these settings are compile-time selections; you do not need to change these settings at run time. They are all characteristics of the panel and frame buffer SRAM being used.

Sync Pulse Polarity

Based on this setting, the hsync and vsync signals are either **Active High** (pulse generated is a high pulse) or **Active Low**. This selection controls both hsync and vsync polarity. The default setting is **Active Low**.

Transition Dotclk Edge

Dotclk is the clock that is sent to the panel, off of which the panel operates. When the transition is set to **Rising edge**, all of the associated signals such as hsync, vsync, and de, transition on the rising edge of dotclk. When set to **Falling edge**, they transition on the falling edge of dotclk.

Typically, if the panel specifies a setup and hold time to one edge of dotclk, you should configure this setting to the other edge of the clock. This provides approximately one-half clock cycle of both setup and hold. The default setting is **Falling edge**.

Horizontal Timing (dotclk)

- **Sync Width** – Defines the horizontal sync width in dotclks. This value can be set between 1 and 256 clock cycles. The default setting is **2**.
- **Back Porch** – Defines the horizontal back porch width in dotclks. This value can be set between 6 and 256 clock cycles. A minimum of 6 gives an early enough indication to the state machine to prevent a read or write access that could not complete before the active screen area begins. Some panel specifications measure the back porch as the region between the end of the sync signal and the start of the active region. Other panel specifications consider the back porch to be the region from the start of the sync pulse to the start of the active region. This component measures the back porch as the period from the end of the sync pulse to the start of the active region. The default setting is **66**.
- **Active Region** – Defines the horizontal active region width in dotclks. The active region is implemented using a setting that is a multiple of 4. This allows for regions as large as 1024 x 1024 while using only 8-bit counters. All popular screen sizes are multiples of 4 in both directions. This value can be set between 4 and 1024 (must be a multiple of 4) clock cycles. The default setting is **320**.
- **Front Porch** – Defines the horizontal front porch width in dotclks. This value can be set between 1 and 256 clock cycles. The default setting is **20**.

Vertical Timing (lines)

- **Sync Width** – Defines the vertical sync width in lines. This value can be set between 1 and 256 clock cycles. The default setting is **2**.
- **Back Porch** – Defines the vertical back porch width in lines. This value can be set between 1 and 256 lines. Some panel specifications measure the back porch as the region between the end of the sync signal and the start of the active region. Other panel specifications consider the back porch to be the region from the start of the sync pulse to the start of the active region. This component measures the back porch as the period from the end of the sync pulse to the start of the active region. The default setting is **16**.
- **Active Region** – Defines the vertical active region width in lines. The active region is implemented using a setting that is a multiple of 4. This allows for regions as large as 1024 x 1024 while using only 8-bit counters. All popular screen sizes are multiples of 4 in both directions. This value can be set between 4 and 1024 (must be a multiple of 4) lines. The default setting is **240**.
- **Front Porch** – Defines the vertical front porch width in lines. This value can be set between 1 and 256 lines. The default setting is **4**.

Interrupt Generation

Defines the settings for interrupt generation. The default setting is **None**.

- If set to **Vertical blanking**, an interrupt pulse is generated at the start and end of the vertical blanking interval.
- If set to **Vertical & Horizontal blanking**, an interrupt pulse is generated at the start and end of every active region.

  During the active vertical region, this is an interrupt at the start and end of the active region for each line. For the vertical blanking region, this is a single interrupt at the end of the last active line and another interrupt at the start of the first active line.

Clock Selection

There is no internal clock in this component. You must attach a clock source. The clock rate provided must be two times the desired clock rate for the output dotclk clock to the panel.

Placement

The GraphicLCDCtrl is placed throughout the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Resources

Resource Type				API Memory (Bytes)		
Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Flash	RAM	Pins (per External I/O)
7	11	1	1	421	6	45

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections discuss each function in more detail.

By default, PSoC Creator assigns the instance name “GraphicLCDCtrl_1” to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “GraphicLCDCtrl.”

Function	Description
GraphicLCDCtrl_Start()	Starts the GraphicLCDCtrl interface.
GraphicLCDCtrl_Stop()	Disables the GraphicLCDCtrl interface.
GraphicLCDCtrl_Write()	Initiates a write transaction to the frame buffer.
GraphicLCDCtrl_Read()	Initiates a read transaction from the frame buffer.
GraphicLCDCtrl_WriteFrameAddr()	Sets the starting frame buffer address used when refreshing the screen.
GraphicLCDCtrl_ReadFrameAddr()	Reads the starting frame buffer address used when refreshing the screen.
GraphicLCDCtrl_WriteLineIncr()	Sets the address spacing between adjacent lines.
GraphicLCDCtrl_ReadLineIncr()	Reads the address increment between lines.
GraphicLCDCtrl_Sleep()	Saves the configuration and disables the GraphicLCDCtrl.
GraphicLCDCtrl_Wakeup()	Restores the configuration and enables the GraphicLCDCtrl.
GraphicLCDCtrl_Init()	Initializes or restores the component parameters to the settings provided with the component customizer.
GraphicLCDCtrl_Enable()	Enables the GraphicLCDCtrl.
GraphicLCDCtrl_SaveConfig()	Saves the configuration of the GraphicLCDCtrl.
GraphicLCDCtrl_RestoreConfig()	Restores the configuration of the GraphicLCDCtrl.

Global Variables

Variable	Description
GraphicLCDCtrl_initVar	GraphicLCDCtrl_initVar indicates whether the Graphic LCD Controller has been initialized. The variable is initialized to 0 and set to 1 the first time GraphicLCDCtrl_Start() is called. This allows the component to restart without reinitialization after the first call to the GraphicLCDCtrl_Start() routine. If reinitialization of the component is required, then the GraphicLCDCtrl_Init() function can be called before the GraphicLCDCtrl_Start() or GraphicLCDCtrl_Enable() function.

void GraphicLCDCtrl_Start(void)

Description: This function enables Active mode power template bits or clock gating as appropriate. This is the preferred method to begin component operation. GraphicLCDCtrl_Start() sets the initVar variable, calls the GraphicLCDCtrl_Init() function, and then calls the GraphicLCDCtrl_Enable() function.

Parameters: None

Return Value: None

Side Effects: If the initVar variable is already set, this function only calls the GraphicLCDCtrl_Enable() function.

void GraphicLCDCtrl_Stop(void)

Description: This function disables Active mode power template bits or gates clocks as appropriate.

Parameters: None

Return Value: None

Side Effects: None

void GraphicLCDCtrl_Write(uint32 addr, uint16 data)

Description: This function initiates a write transaction to the frame buffer using the address and data provided. The write is a posted write, so this function returns before the write has actually completed on the interface. If the command queue is full, this function does not return until space is available to queue this write request

Parameters: addr: Address to be sent on the address lines of the component (addr2[6:0], addr1[7:0], addr0[7:0]).

data: Data sent on the do_msb[7:0] (most significant byte) and do_lsb[7:0] (least significant byte) pins

Return Value: None

Side Effects: None

uint16 GraphicLCDCtrl_Read(uint32 addr)

Description: This function initiates a read transaction from the frame buffer. The read executes after all currently posted writes have completed. The function waits until the read completes and then returns the read value.

Parameters: addr: Address to be sent on the address lines of the component (addr2[6:0], addr1[7:0], addr0[7:0])

Return Value: Read value from the di_msb[7:0] (most significant byte) and di_lsb[7:0] (least significant byte) pins

Side Effects: None

void GraphicLCDCtrl_WriteFrameAddr(uint32 addr)

Description: This function sets the starting frame buffer address used when refreshing the screen. This register is read during each vertical blanking interval. To implement an atomic update of this register it should be written during the active refresh region.

Parameters: addr: Address of the start of the frame buffer

Return Value: None

Side Effects: None

uint32 GraphicLCDCtrl_ReadFrameAddr(void)

Description: This function reads the starting frame buffer address used when refreshing the screen.

Parameters: None

Return Value: Address of the start of the frame buffer

Side Effects: None

void GraphicLCDCtrl_WriteLineIncr(uint32 incr)

Description: This function sets the address spacing between adjacent lines. By default, this is the display size of a line. This setting can be used to align lines to a different word boundary or to implement a virtual line length that is larger than the display region.

Parameters: incr: Address increment between lines. Must be at least the display size of a line.

Return Value: None

Side Effects: None

uint32 GraphicLCDCtrl_ReadLineIncr(void)

Description: This function reads the address increment between lines.

Parameters: None

Return Value: Address increment between lines

Side Effects: None

void GraphicLCDCtrl_Sleep(void)

Description: This is the preferred routine to prepare the component for sleep. The GraphicLCDCtrl_Sleep() routine saves the current component state. Then it calls the GraphicLCDCtrl_Stop() function and calls GraphicLCDCtrl_SaveConfig() to save the hardware configuration.

Call the GraphicLCDCtrl_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. See the PSoC Creator *System Reference Guide* for more information about power-management functions.

Parameters: None

Return Value: None

Side Effects: None

void GraphicLCDCtrl_Wakeup(void)

Description: This is the preferred routine to restore the component to the state when GraphicLCDCtrl_Sleep() was called. The GraphicLCDCtrl_Wakeup() function calls the GraphicLCDCtrl_RestoreConfig() function to restore the configuration. If the component was enabled before the GraphicLCDCtrl_Sleep() function was called, the GraphicLCDCtrl_Wakeup() function also re-enables the component.

Parameters: None

Return Value: None

Side Effects: Calling the GraphicLCDCtrl_Wakeup() function without first calling the GraphicLCDCtrl_Sleep() or GraphicLCDCtrl_SaveConfig() function can produce unexpected behavior.

void GraphicLCDCtrl_Init(void)

Description: This function initializes or restores the component parameters to the settings provided with the component customizer. The compile time configuration that defines timing generation is restored to the settings provided with the customizer. The run-time configuration for the frame buffer address is set to 0; for the line increment it is set to the display line size.

Parameters: None

Return Value: None

Side Effects: The component must be disabled by GraphicLCDCtrl_Stop() before this function call, otherwise the component's behavior can be unexpected. This reinitializes the component with the following exceptions: it does not clear data from the FIFOs and does not reset component hardware state machines.

void GraphicLCDCtrl_Enable(void)

Description: This function activates the hardware and begins component operation. It is not necessary to call GraphicLCDCtrl_Enable() because the GraphicLCDCtrl_Start() routine calls this function, which is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: None

void GraphicLCDCtrl_SaveConfig(void)

Description: This function saves the component configuration and nonretention registers. It also saves the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the GraphicLCDCtrl_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void GraphicLCDCtrl_RestoreConfig(void)

Description: This function restores the component configuration and nonretention registers. It also restores the component parameter values to what they were before calling the GraphicLCDCtrl_Sleep() function.

Parameters: None

Return Value: None

Side Effects: If this API is called before GraphicLCDCtrl_SaveConfig(), the component configurations will be restored to their default settings. The run-time configuration for the frame buffer address is set to 0; for the line increment it is set to the display line size.

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

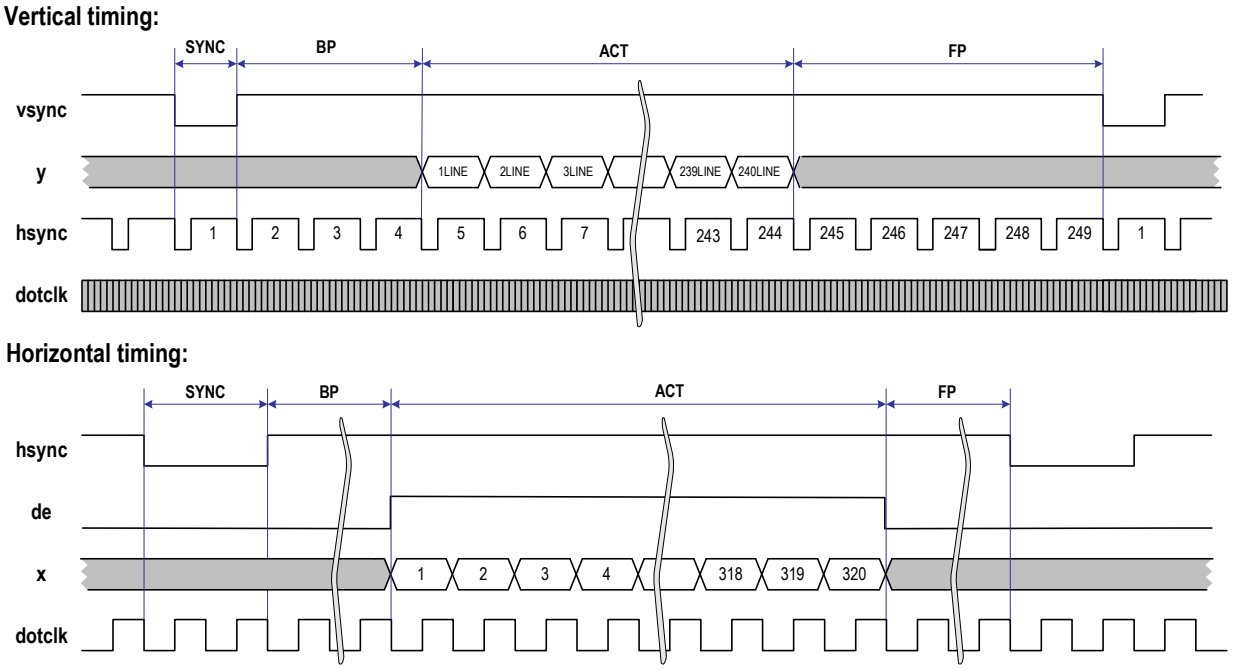
Functional Description

This component generates continuous timing signals to the panel without CPU intervention. During the refresh period, the component also generates read requests to the frame buffer, scanning through a frame of 16-bit pixel data. During the blanking intervals (horizontal and vertical) the component can generate read or write transactions on the frame buffer interface.

Screen Refresh and Timing

Throughout a frame time, the component generates the configured vertical timing pattern on vsync, and throughout each line of the frame the component generates the configured horizontal pattern on hsync. In addition to hsync and vsync, some panels require a de (data enable) signal that is active high during the active portion of the screen refresh. All panels operate in the same way, although the timing of each of the segments of the refresh period differs. Figure 1 shows the timing diagram for a typical panel.

Figure 1. Typical Panel Timing



The sequence for each frame for the vertical signal and the sequence for each line for the horizontal signal follow this pattern:

- Sync pulse: Period where the sync pulse is active
- Back Porch: Period from the end of the sync pulse until the active display area
- Active: Display area on the screen
- Front Porch: Period from the end of the active display until the sync pulse starts

Address Generation

As the screen is refreshed, the component must scan through the frame buffer generating the addresses for the pixels on the screen. Each pixel requires one 16-bit read from the frame buffer. For the beginning of each frame, the index into the frame buffer is reset to the designated starting point for the frame buffer. The value is set to 0 initially and can then be changed using API functions. The frame buffer address does not affect the read and write API functions, it only affects the refresh operation.

Frame Buffer Transactions

The controller component can perform either read or write transactions. These transactions have the following parameters:

- Read or write
- Address: Up to a 23-bit address
- Data: 16-bit value. Sent on “do” (data out) for writes and read on “di” (data in) for reads.

The implementation used for this component combines the 23 bits of address with a 1-bit read/write indicator. This allows the address and transaction type to be transferred to the component in three bytes. It also allows the transaction type and address to stay together in the datapath FIFO.

Read and write transactions are performed during the horizontal and vertical blanking intervals.

Idle Condition

When neither a read nor a write is occurring on the frame buffer interface, the interface is in the idle state. The idle state control signals are the same as the values for reading. The values for the output pins in the idle condition are as follows:

- do: Don’t care (may be left at its last state)
- doe: 0
- addr: Don’t care (may be left at its last state)
- nwe: 1
- noe: 0

Any signal not listed in the description of the read and write transactions is in the idle state.

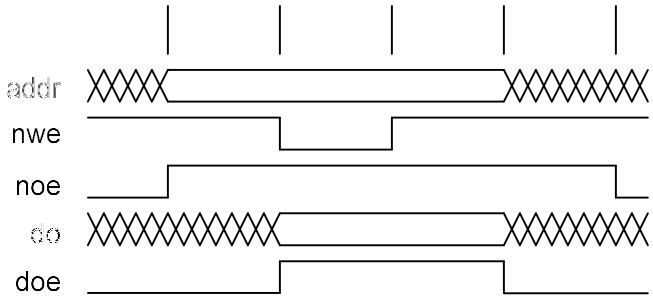
Write Transaction

The component implements the timing diagram shown in Figure 2 for a write transaction. This diagram shows that the write transaction requires four dotclk cycles (all diagrams are in dotclk cycles). This transaction can be immediately preceded or followed by another read or write transaction or may be in the idle state before or after a write transaction.

The interface to the CPU allows the CPU to make posted write requests (request a write providing the address and data and then proceed before the transaction is actually completed to the frame buffer). The implementation allows the CPU to have four write requests outstanding without stalling.

Note the pattern of the noe and doe signals, which prevents the data bus from being driven by both the component and the frame buffer regardless of the skew of the signals.

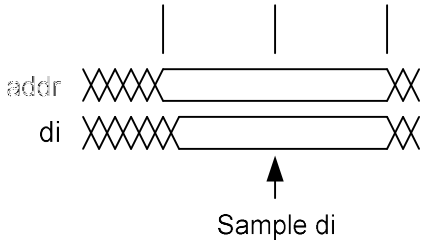
Figure 2. Write Transaction Timing Diagram



Read Transaction

This component implements the timing diagram shown in Figure 3 for a read transaction. This transaction can be immediately preceded or followed by another read or write transaction or may be in the idle state before or after a write transaction.

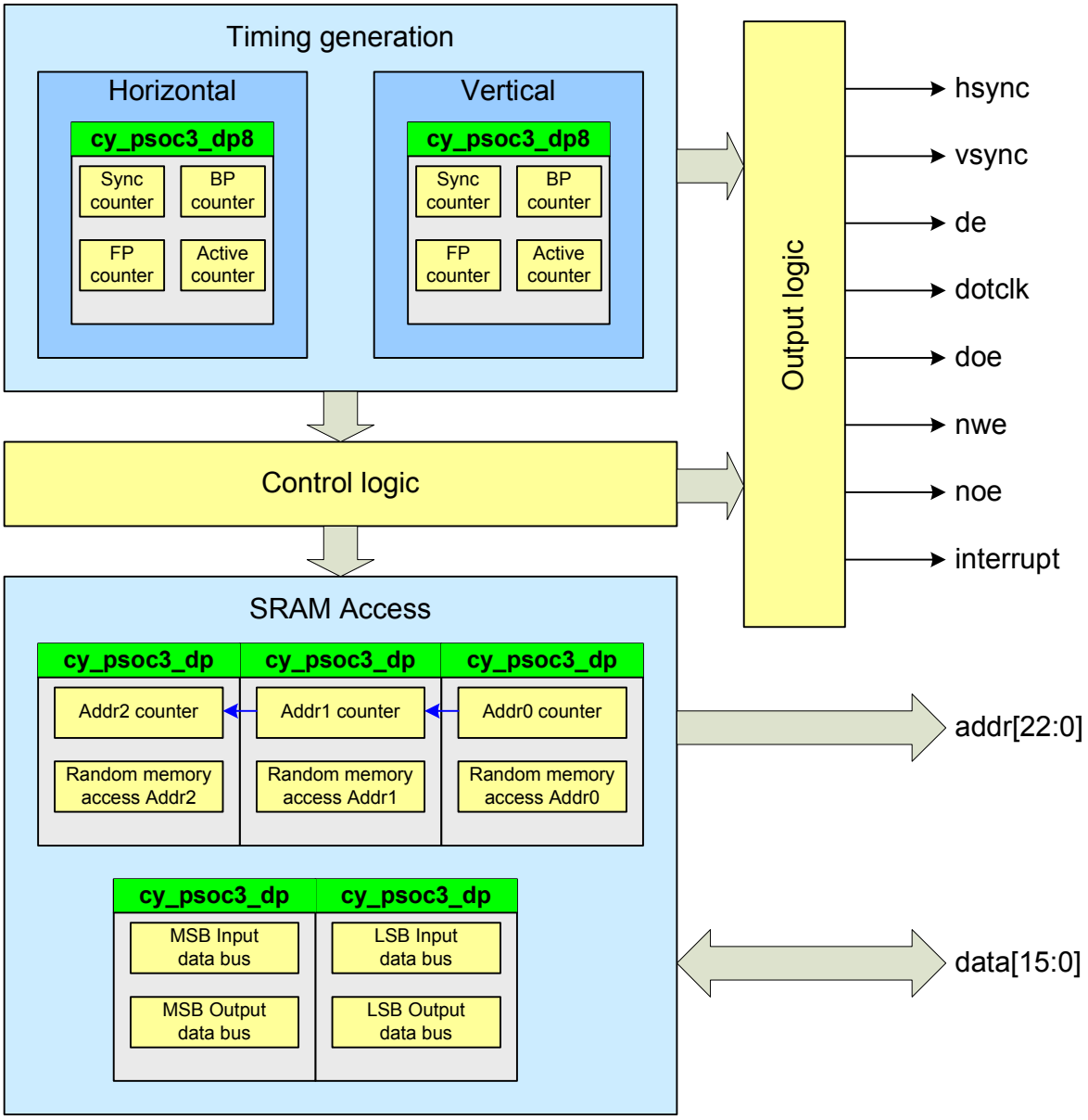
Figure 3. Read Transaction Timing Diagram



Block Diagram and Configuration

The GraphicLCDCtrl component is implemented as a set of configured UDBs. Figure 4 shows this implementation.

Figure 4. Block Diagram



Registers

GraphicLCDCtrl_STATUS_REG

Bits	7	6	5	4	3	2	1	0
Value	reserved				v_blanking	h_blanking	avail	full

- full: Set if the command/data FIFO is full
- avail: Set if the read data is valid for the CPU
- h_blanking: Set during the horizontal blanking interval
- v_blanking: Set during the vertical blanking interval

DC and AC Electrical Characteristics

The following values indicate expected performance and are based on initial characterization data.

Timing Characteristics “Maximum with Nominal Routing”

Parameter	Description	Min	Typ	Max [1]	Unit
f_{DOTCLK}	Dotclk frequency	–	–	20	MHz
f_{CLOCK}	Component clock frequency	$2*f_{DOTCLK}$	–	–	MHz
t_{DOTCLK}	Dotclk period	$1/f_{DOTCLK}$	–	–	ns
t_{CKL}	Dotclk low time	–	0.5	–	$1/f_{DOTCLK}$
t_{CKH}	Dotclk high time	–	0.5	–	$1/f_{DOTCLK}$
Screen Refresh and Data Transaction Timing					
t_{HSYNC}	Horizontal sync pulse period	1	–	256	t_{DOTCLK}
t_{HBP}	Horizontal back porch period	6	–	256	t_{DOTCLK}
$t_{HACTIVE}$	Horizontal active region period	4	–	1024	t_{DOTCLK}
t_{HFP}	Horizontal front porch period	1	–	256	t_{DOTCLK}
t_{HBLANK}	Horizontal blanking period	–	$t_{HSYNC} + t_{HBP} + t_{HFP}$	–	t_{DOTCLK}

[1] These “Nominal” numbers provide a maximum safe operating frequency of the component under nominal routing conditions. You can run the component at higher clock frequencies, but you will need to validate the timing requirements with STA results.

Parameter	Description	Min	Typ	Max [1]	Unit
H_{CYCLE}	Horizontal cycle	–	t_{HBLANK} + $t_{HACTIVE}$	–	t_{DOTCLK}
t_{VSYNC}	Vertical sync pulse period	1	–	256	H_{CYCLE}
t_{VBP}	Vertical back porch period	1	–	256	H_{CYCLE}
$t_{VACTIVE}$	Vertical active region period	4	–	1024	H_{CYCLE}
t_{VFP}	Vertical front porch period	1	–	256	H_{CYCLE}
t_{VBLANK}	Horizontal cycle	–	t_{VSYNC} + t_{VBP} + t_{VFP}	–	H_{CYCLE}
V_{CYCLE}	Vertical cycle	–	t_{VBLANK} + $t_{VACTIVE}$	–	H_{CYCLE}
Pixel Timing					
t_{HV}	Phase difference of sync signal falling edge	–	t_{HFP}	–	t_{DOTCLK}
t_{VSYS}	Vertical sync setup time	–	0.5	–	t_{DOTCLK}
t_{VSYH}	Vertical sync hold time	–	0.5	–	t_{DOTCLK}
t_{HSYS}	Horizontal sync setup time	–	0.5	–	t_{DOTCLK}
t_{HSYH}	Horizontal sync hold time	–	0.5	–	t_{DOTCLK}
t_{DS}	Data setup time to LCD panel	–	0.5	–	t_{DOTCLK}
t_{DH}	Data hold time to LCD panel	–	0.5	–	t_{DOTCLK}
Frame Buffer Transaction Timing					
t_{AS}	Address setup time	1	–	–	t_{DOTCLK}
t_{AH}	Address hold time	–	2	–	t_{DOTCLK}
t_{PWE}	NWE pulse width	–	1	–	t_{DOTCLK}
t_{DSW}	Data setup time to frame buffer	–	1	–	t_{DOTCLK}
t_{DHW}	Data hold time to frame buffer	–	1	–	t_{DOTCLK}
t_{CYCLE}	Clock cycle time				
	Write cycle	4	–	–	t_{DOTCLK}
	Read cycle	2	–	–	t_{DOTCLK}
t_{ACC}	Data access time	–	1	–	t_{DOTCLK}
t_{OH}	Output hold time	–	0	–	t_{DOTCLK}

Timing Characteristics "Maximum with All Routing"

Parameter	Description	Min	Typ	Max [2]	Unit
f_{DOTCLK}	Dotclk frequency	–	–	10	MHz
f_{CLOCK}	Component clock frequency	$2*f_{DOTCLK}$	–	–	MHz
t_{DOTCLK}	Dotclk period	$1/f_{DOTCLK}$	–	–	ns
t_{CKL}	Dotclk low time	–	0.5	–	$1/f_{DOTCLK}$
t_{CKH}	Dotclk high time	–	0.5	–	$1/f_{DOTCLK}$
Screen Refresh and Data Transaction Timing					
t_{HSYNC}	Horizontal sync pulse period	1	–	256	t_{DOTCLK}
t_{HBP}	Horizontal back porch period	6	–	256	t_{DOTCLK}
$t_{HACTIVE}$	Horizontal active region period	4	–	1024	t_{DOTCLK}
t_{HFP}	Horizontal front porch period	1	–	256	t_{DOTCLK}
t_{HBLANK}	Horizontal blanking period	–	$t_{HSYNC} + t_{HBP} + t_{HFP}$	–	t_{DOTCLK}
H_{CYCLE}	Horizontal cycle	–	$t_{HBLANK} + t_{HACTIVE}$	–	t_{DOTCLK}
t_{VSYNC}	Vertical sync pulse period	1	–	256	H_{CYCLE}
t_{VBP}	Vertical back porch period	1	–	256	H_{CYCLE}
$t_{VACTIVE}$	Vertical active region period	4	–	1024	H_{CYCLE}
t_{VFP}	Vertical front porch period	1	–	256	H_{CYCLE}
t_{VBLANK}	Horizontal cycle	–	$t_{VSYNC} + t_{VBP} + t_{VFP}$	–	H_{CYCLE}
V_{CYCLE}	Vertical cycle	–	$t_{VBLANK} + t_{VACTIVE}$	–	H_{CYCLE}
Pixel Timing					
t_{HV}	Phase difference of sync signal falling edge	–	t_{HFP}	–	t_{DOTCLK}
t_{VSYS}	Vertical sync setup time	–	0.5	–	t_{DOTCLK}
t_{VSYH}	Vertical sync hold time	–	0.5	–	t_{DOTCLK}
t_{HSYS}	Horizontal sync setup time	–	0.5	–	t_{DOTCLK}
t_{HSYH}	Horizontal sync hold time	–	0.5	–	t_{DOTCLK}

[2] Maximum for "All Routing" is calculated by <nominal>/2 rounded to the nearest integer. If your component instance operates at or below these speeds, then meeting timing should not be a concern for this component.

Parameter	Description		Min	Typ	Max [2]	Unit
t_{DS}	Data setup time to LCD panel		–	0.5	–	t_{DOTCLK}
t_{DH}	Data hold time to LCD panel		–	0.5	–	t_{DOTCLK}
Frame Buffer Transaction Timing						
t_{AS}	Address setup time		1	–	–	t_{DOTCLK}
t_{AH}	Address hold time		–	2	–	t_{DOTCLK}
t_{PWE}	NWE pulse width		–	1	–	t_{DOTCLK}
t_{DSW}	Data setup time to frame buffer		–	1	–	t_{DOTCLK}
t_{DHW}	Data hold time to frame buffer		–	1	–	t_{DOTCLK}
t_{CYCLE}	Clock cycle time					
		Write cycle	4	–	–	t_{DOTCLK}
		Read cycle	2	–	–	t_{DOTCLK}
t_{ACC}	Data access time		–	1	–	t_{DOTCLK}
t_{OH}	Output hold time		–	0	–	t_{DOTCLK}

Figure 5. Screen Refresh and Data Transaction Timing Diagram

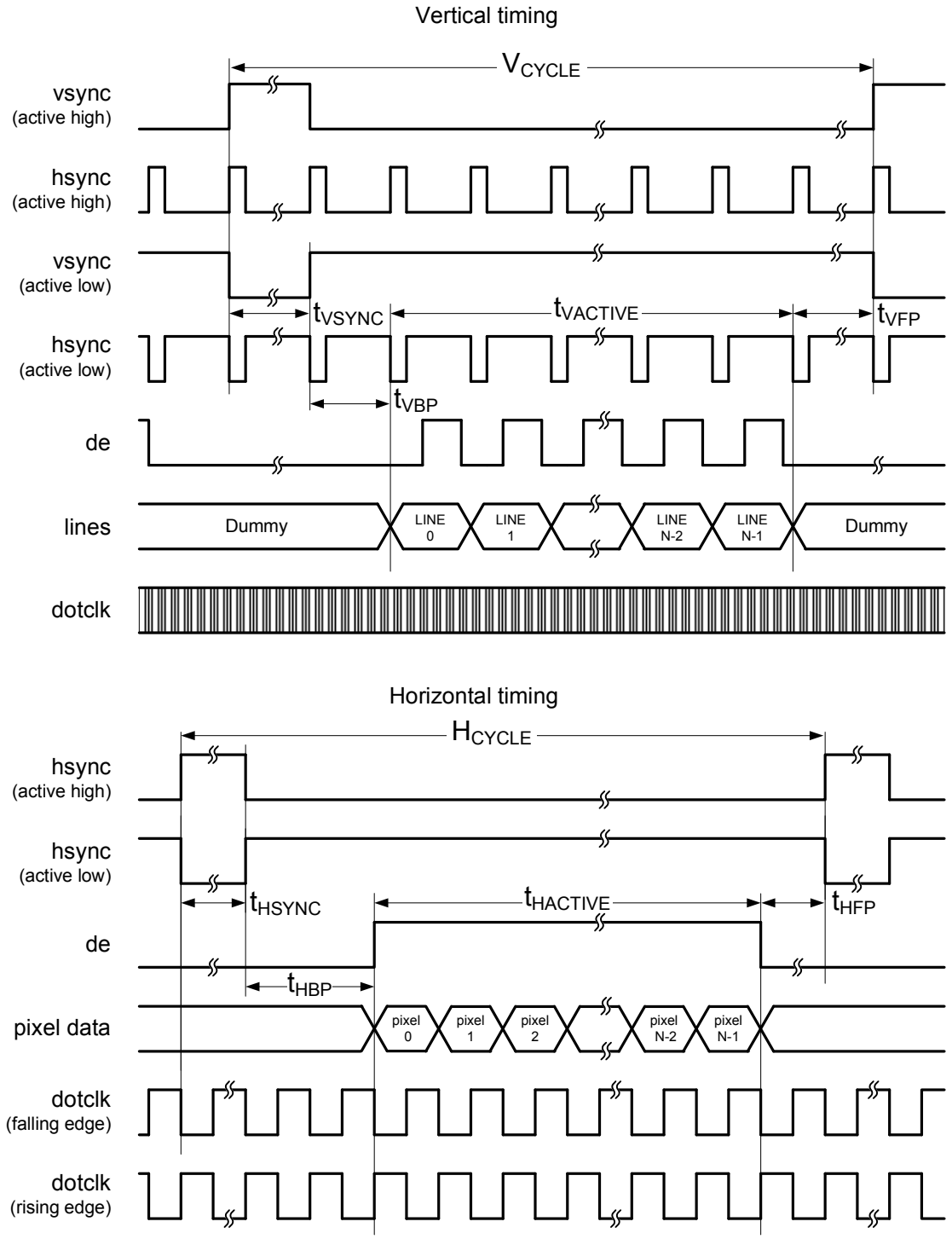


Figure 6. Pixel Timing Diagram

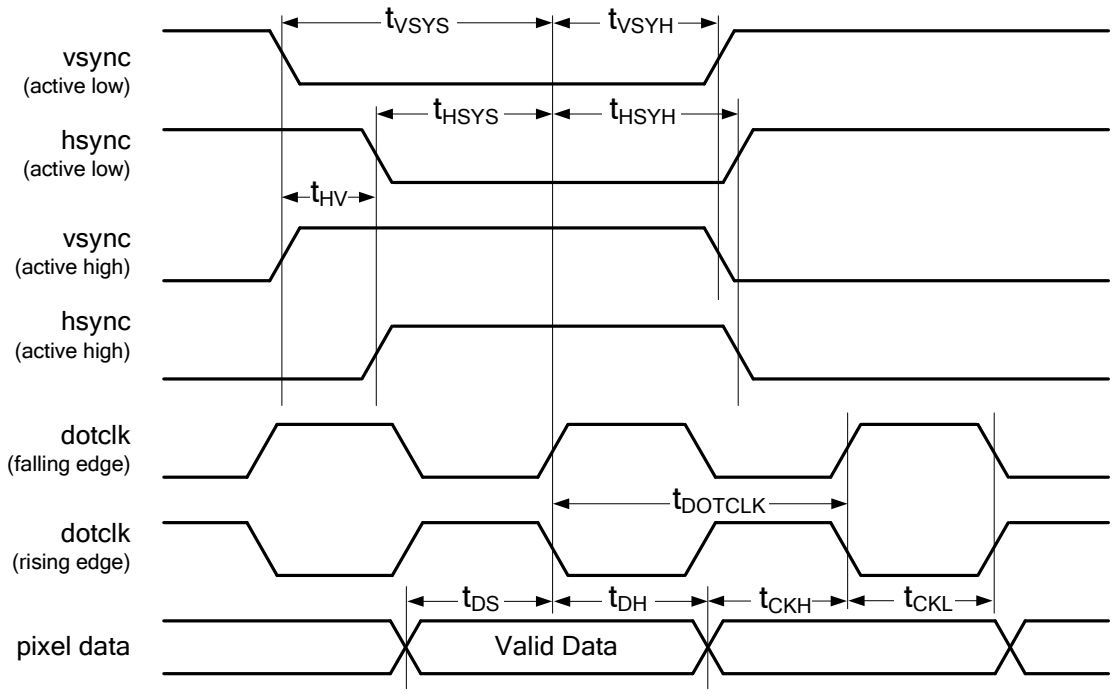
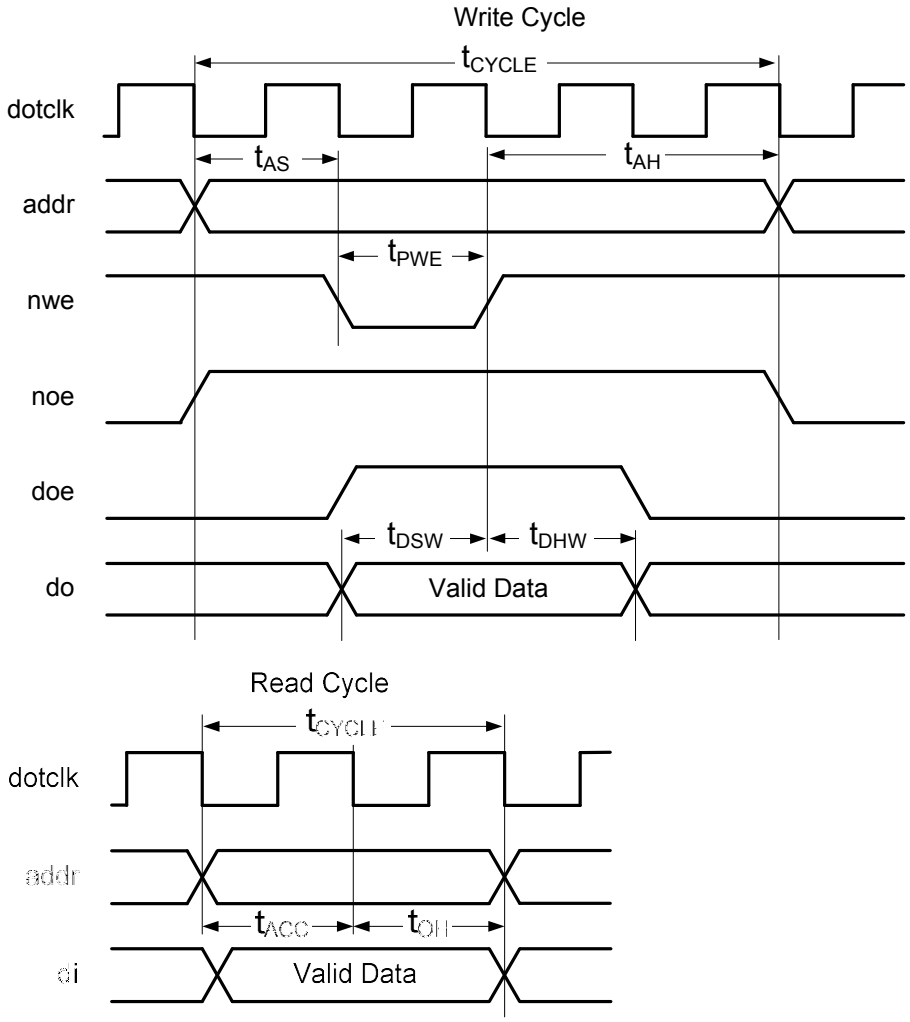


Figure 7. Frame Buffer Data Transaction Timing Diagram



How to Use STA Results for Characteristics Data

Nominal route maximums are gathered through multiple test passes with Static Timing Analysis (STA). You can calculate the maximums for your designs with the STA results using the following methods:

f_{clock} Maximum component clock frequency appears in Timing results in the clock summary as the named external clock. The following graphic shows an example of the clock limitations.

- Clock Summary Section

Clock	Type	Nominal Frequency	Required Frequency	Maximum Frequency	Violation
CLK	Sync	20.000 MHz	20.000 MHz	41.545 MHz	
ClockBlock/clk_bus	Async	60.000 MHz	60.000 MHz	N/A	
ClockBlock/dclk_0	Async	20.000 MHz	20.000 MHz	N/A	
CyBUS_CLK	Sync	60.000 MHz	60.000 MHz	N/A	
CyILO	Async	1.000 kHz	1.000 kHz	N/A	
CyIMO	Async	3.000 MHz	3.000 MHz	N/A	
CyMASTER_CLK	Sync	60.000 MHz	60.000 MHz	N/A	
CyPLL_OUT	Async	60.000 MHz	60.000 MHz	N/A	

The remaining parameters are implementation-specific and are measured in clock cycles. They can be divided into two categories.

- The parameters that are used to configure the component:

  Screen Refresh and Data Transaction Timing Parameters

  f_{DOTCLK} Clock driven to the panel. This clock is one-half the rate of the incoming clock. The component allows you to change the dotclk edge for the signal transition to the panel. The parameter can be set to ‘rising edge’ or ‘falling edge’. If you set it to the rising edge, all output signals change on the rising edge of dotclk. If you set it to the falling edge, the output transitions occur at the same time as the falling edge of dotclk. That allows those signals to then be sampled by the panel on the opposite edge of dotclk and satisfy setup and hold times.

  t_{HSYNC} The period that the horizontal hsync pulse is active in dotclks. The signal can either be active high (pulse generated is a high pulse) or active low (pulse generated is a low pulse). The polarity of the signal is set in the component customizer.

  t_{HBP} The period from the end of the hsync pulse to the start of the active region in dotclks.

  $t_{HACTIVE}$ The horizontal active region period (display area) in dotclks.

  t_{HFP} The period from the end of the active display until the hsync pulse starts in dotclks.

  t_{VSYNC} The period that the vertical sync pulse is active in H_{CYCLE}. The signal can either be active high (pulse generated is a high pulse) or active low (pulse generated is a low pulse). The polarity of the signal is set in the component customizer.

  t_{VBP} The period from the end of the vsync pulse to the start of the active region in H_{CYCLE}.

  $t_{VACTIVE}$ The vertical active region period (display area) in H_{CYCLE}.

  t_{VFP} The period from the end of the active display until the vsync pulse starts in H_{CYCLE}.

  V_{CYCLE} The period during which one whole frame is updated. This is defined as the sum of t_{VSYNC}, t_{VBP}, $t_{VACTIVE}$ and t_{VFP} periods.

  t_{VBLANK} The number of blanking lines for the frame period. During this period the frame buffer can be updated (the component initiates a write/read transaction to the frame buffer). There is no data flow to an LCD panel during the blanking period. The period is the sum of the t_{VSYNC}, t_{VBP}, and t_{VFP} intervals.

H_{CYCLE} The period during which one horizontal line is updated. Defined as the sum of the t_{HSYNC}, t_{HBP}, $t_{HACTIVE}$, and t_{HFP} periods.

t_{HBLANK} The number of blanking pixels for one horizontal line. During this period the frame buffer can be updated (the component initiates a write/read transaction to the frame buffer). There is no data flow to an LCD panel during the blanking period. The period is the sum of the t_{HSYNC}, t_{HBP}, and t_{HFP} intervals.

- The parameters that are fixed based on the component implementation:

  Pixel Timing Parameters

  t_{DOTCLK} Period of dotclk signal.

  t_{CKL} The component generates a 50-percent duty cycle dotclk.

  t_{CKH} The component generates a 50-percent duty cycle dotclk.

  t_{VSYS} The minimum amount of time the vsync signal is valid before the active edge of the dotclk signal.

  t_{VSYH} The minimum amount of time the vsync signal is valid after the active edge of the dotclk signal.

  t_{HSYS} The minimum amount of time the hsync signal is valid before the active edge of the dotclk signal.

  t_{HSYH} The minimum amount of time the hsync signal is valid after the active edge of the dotclk signal.

  Note that t_{VSYS}, t_{VSYH}, t_{HSYS}, t_{HSYH} parameters are defined by the relation between dotclk and vsync for vertical timing and dotclk and hsync for horizontal timing. You can change the dotclk edge for the signal transition to the panel. That allows those signals to then be sampled by the panel on the opposite edge of dotclk and satisfy setup and hold times. That allows you to have almost a full half dotclk cycle of setup and hold time that t_{VSYS}, t_{VSYH}, t_{HSYS}, t_{HSYH} signals.

  t_{HV} The phase difference of the sync signal active edge. In the component implementation, vertical counting is done at the first cycle of the horizontal front porch, so the phase difference of vsync before hsync is equal to the horizontal front porch period (t_{HFP}).

  t_{DS} The minimum amount of time the data is valid on the input to the panel before the active edge of the dotclk signal.

  t_{DH} The minimum amount of time the data is valid on the input to the panel after the active edge of the dotclk signal.

  To determine these parameters, the timing for the SRAM that is used as frame buffer must be considered together with GraphicLCDCtrl component implementation. As the screen refreshes, the component scans through the frame buffer, generating the addresses for the pixels on the screen. The addresses to the frame buffer change on active dotclk edge. The delay between dotclk and address signals is near zero,

because both of these signals are generated on the internal component clock and then propagated to the output pins. This allows almost a full half dotclk cycle of hold time. Setup time is calculated as a full half period of dotclk minus t_{AA} for the SRAM frame buffer. t_{AA} can be found in the respective SRAM datasheet.

Frame Buffer Data Transaction Parameters

t_{AS} The minimum amount of time the address signal is valid before the falling edge of the nwe signal.

t_{AH} The minimum amount of time the address signal is valid after the rising edge of the nwe signal.

t_{PWE} The minimum pulse width low time for the write signal.

t_{CYCLE} The period of time during which a single transaction (write/read) is performed on the interface to the frame buffer.

t_{DSW} The minimum amount of time the data is valid before the falling edge of the write signal.

t_{DHW} The minimum amount of time the data is valid after the rising edge of the write signal.

t_{ACC} The minimum amount of time the data is sampled after the address is valid for the read transaction.

t_{OH} The minimum amount of time the data should be valid after active edge of the dotclk signal the data is sampled.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.61	Added all component APIs with the CYREENTRANT keyword when they are included in the .cyre file.	Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates. This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections.
	Added timing constraints to mark false timing paths in the component.	Removes paths that are not used from timing analysis. This avoids false timing violation messages.
1.60.a	Removed references to associated kits from datasheet.	
1.60	Resampled FIFO block status signals to DP clock.	Allows the component to function with the same timing results for all PSoC 3 and PSoC 5 silicons.
	Minor datasheet edits and updates	

© Cypress Semiconductor Corporation, 2012. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.