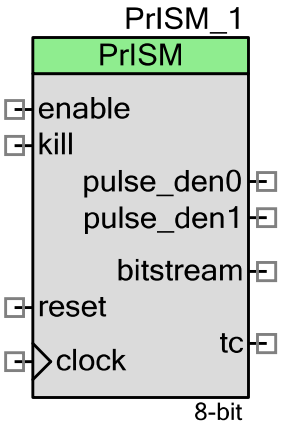
Precision Illumination Signal Modulation (PrISM)

2.0

Features



- Programmable flicker-free dimming resolution from 2 to 32 bit
- Two pulse density outputs
- Programmable output signal density
- Serial output bit stream
- Continuous run mode
- User-configurable sequence start value
- Standard or custom polynomials provided for all sequence lengths
- Kill input disables pulse density outputs and forces them low
- Enable input provides synchronized operation with other components
- Reset input allows restart at sequence start value for synchronization with other components
- Terminal Count Output for 8-, 16-, 24-, and 32-bit sequence lengths.

General Description

The Precision Illumination Signal Modulation (PrISM) component uses a linear feedback shift register (LFSR) to generate a pseudo random sequence. The sequence outputs a pseudo random bit stream, as well as up to two user-adjustable pseudo random pulse densities. The pulse densities may range from 0 to 100 percent.

The LFSR is of the Galois form (sometimes known as the modular form) and uses the provided maximal length codes. The PrISM component runs continuously after it starts and as long as the enable input is held high. The PrISM pseudo random number generator can be started with any valid seed value, excluding 0.

When to Use a PrISM

The PrISM component provides modulation technology that significantly reduces low-frequency flicker and radiated electromagnetic interference (EMI), which are common problems with high-

brightness LED designs. The PrISM is also useful in other applications that need this benefit, such as motor controls and power supplies.

Input/Output Connections

This section describes the various input and output connections for PrISM. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input

The clock input defines the signal to compute the pseudo random sequence.

reset – Input

The reset input resets the pseudo random number to the start value at high state. This input valid for started component only and provides synchronized operation with other components.

kill – Input

The active-high kill input disables the PrISM pulse density outputs and sets them to 0 untill kill is released low.

enable – Input

The PrISM component runs after it starts and as long as the enable input is held high and reset input is low. This input provides synchronized operation with other components.

pulse_den0/pulse_den1 – Outputs

Two pulse density outputs are available; both are derived from the same pseudo random sequence. Each output is generated by comparing the desired pulse density value with the current pseudo random number. If the pulse density type is configured as **Less Than or Equal**, the output is high while the pseudo random number is less than or equal to the pulse density value. The second option is to set the pulse density type of to **Greater Than or Equal** so the output is high while the pseudo random number is greater than or equal to the pulse density value.

bitstream – Output

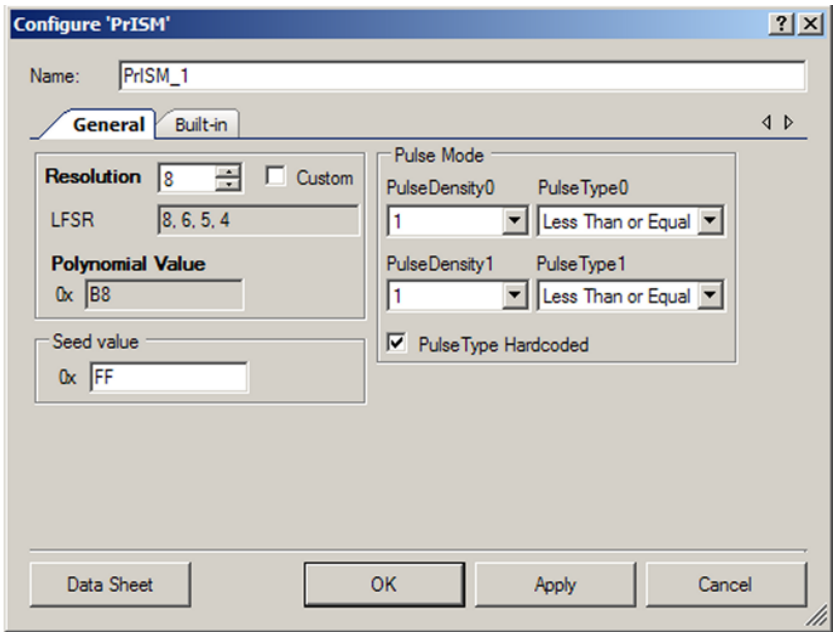
The bitstream output continuously outputs the LSb of the LFSR.

tc – Output *

The terminal count output is available for 8-, 16-, 24-, and 32-bit length PrISM components. The terminal count output goes high for one clock period each time the pseudo random number equals 0xFF (8-bit), 0xFFFF (16-bit), 0xFFFFFF (24-bit), or 0xFFFFFFFF (32-bit), which occurs once each cycle of the pseudo random number generator.

Component Parameters

Drag a PrISM component onto your design and double-click it to open the **Configure** dialog.



The PrISM component contains the following parameters:

Resolution

This parameter defines the PrISM maximal code length (period). The maximal code length is ($2^{Resolution}$ – 1). Possible values include 2 to 32 bits. The maximal length code sets the length of the pseudo random number generator and therefore the length of the sequence to be generated. Longer sequences increase the pulse density resolution and lower the radiated EMI. The maximal length codes listed in the following table are provided in the Galois form and require no conversion before you use them in the PSoC 3 UDB ALU.

Resolution	LFSR		Resolution	LFSR		Resolution	LFSR
2	2, 1		13	13, 12, 10, 9		24	24, 23, 21, 20
3	3, 2		14	14, 13, 11, 9		25	25, 24, 23, 22
4	4, 3		15	15, 14, 13, 11		26	26, 25, 24, 20
5	5, 4, 3, 2		16	16, 14, 13, 11		27	27, 26, 25, 22

Resolution	LFSR		Resolution	LFSR		Resolution	LFSR
6	6, 5, 3, 2		17	17, 16, 15, 14		28	28, 27, 24, 22
7	7, 6, 5, 4		18	18, 17, 16, 13		29	29, 28, 27, 25
8	8, 6, 5, 4		19	19, 18, 17, 14		30	30, 29, 26, 24
9	9, 8, 6, 5		20	20, 19, 16, 14		31	31, 30, 29, 28
10	10, 9, 7, 6		21	21, 20, 19, 16		32	32, 30, 26, 25
11	11, 10, 9, 7		22	22, 19, 18, 17			
12	12, 11, 8, 6		23	23, 22, 20, 18			

To set LFSR coefficients manually:

Define **Resolution**.

Select the **Custom** check box.

Enter coefficients separated by comma in the LFSR text box and press [**Enter**]. The **Polynomial Value** will be recalculated automatically.

The **Polynomial Value** is represented in hexadecimal format.

Note LFSR coefficient value cannot be greater than the **Resolution** value.

Polynomial Value

This parameter is represented in hexadecimal format. The correct polynomial is chosen based on the **Resolution** selected. You can optionally specify a custom polynomial.

Seed value

This parameter by default is set to the maximum possible value ($2^{Resolution} – 1$). This value can be changed to any value except 0. The **Seed value** is represented in the hexadecimal form.

Note Changing the **Resolution** sets the **Seed value** to the default value.

Pulse Mode

These parameter values are chosen from combo boxes. Available values are from 1 to $2^{Resolution} – 1$ with a step $2^{Resolution}$. Pulse compare type can be set to **Less Than or Equal** or **Greater Than or Equal**.

PulseType Hardcoded

The **PulseType Hardcoded** parameter saves recourses (control register) when enabled, but makes it impossible to change the Pulse Type using the PrISM_SetPulse0Mode() or PrISM_SetPulse1Mode() APIs.

The PrISM_Stop() function is also not available if this parameter is enabled. To stop the PrISM in this case, use the “enable” input.

Local Parameters (For API usage)

These parameters are used in the API and not shown in the **Configure** dialog.

- **PolyValue(uint32) –** Contains the polynomial value in hexadecimal format. The default is 0xB8h (LFSR= [8,6,5,4]).
- **Density0(uint32) –** Contains density0 value in hexadecimal format.
- **Density1(uint32) –** Contains density1 value in hexadecimal format.
- **CompareType0(CompareType) –** Contains **Pulse Type** for Density0 which can be **Less Than or Equal** or **Greater Than or Equal**.
- **CompareType1 (CompareType) –** Contains **Pulse Type** for Density1 which may be **Less Than or Equal** or **Greater Than or Equal**.

Clock Selection

There is no internal clock in this component. You must attach a clock source. The maximum frequency input is 67 MHz.

Placement

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

Resources

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Flash	RAM	
8 Bits	1	3	0	1	423	6	8
8 Bits *	1	3	0	0	423	6	8
16 Bits	2	3	0	1	543	13	8
24 Bits	3	3	0	1	569	23	8
32 Bits	4	3	0	1	569	23	8

* Parameter **PulseType Hardcoded** enabled.

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 cover each function in more detail.

By default, PSoC Creator assigns the instance name “PrISM_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 “PrISM.”

Function	Description
PrISM_Start()	The start function sets polynomial, seed, and pulse density registers provided by the customizer.
PrISM_Stop()	Stops PrISM computation.
PrISM_SetPulse0Mode()	Sets the pulse density type for Density0.
PrISM_SetPulse1Mode()	Sets the pulse density type for Density1.
PrISM_ReadSeed()	Reads the PrISM Seed register.
PrISM_WriteSeed()	Writes the PrISM Seed register with the start value.
PrISM_ReadPolynomial()	Reads the PrISM Polynomial register.
PrISM_WritePolynomial()	Writes the PrISM Polynomial register with the start value.
PrISM_ReadPulse0()	Reads the PrISM Pulse Density0 value register.
PrISM_WritePulse0()	Writes the PrISM Pulse Density0 value register with the new Pulse Density value.
PrISM_ReadPulse1()	Reads the PrISM Pulse Density1 value register.
PrISM_WritePulse1()	Writes the PrISM Pulse Density1 value register with the new Pulse Density value.
PrISM_Sleep()	Stops and saves the user configuration.
PrISM_Wakeup()	Restores and enables the user configuration
PrISM_Init()	Initializes the default configuration provided with the customizer.
PrISM_Enable()	Enables the PrISM block operation.
PrISM_SaveConfig()	Saves the current user configuration.
PrISM_RestoreConfig()	Restores the current user configuration.

Global Variables

Variable	Description
PrISM_initVar	Indicates whether the PrISM has been initialized. The variable is initialized to 0 and set to 1 the first time PrISM_Start() is called. This allows the component to restart without reinitialization after the first call to the PrISM_Start() routine. If reinitialization of the component is required, then the PrISM_Init() function can be called before the PrISM_Start() or PrISM_Enable() functions.

void PrISM_Start(void)

Description: This is the preferred method to begin component operation. PrISM_Start() sets the initVar variable, calls the PrISM_Init() function, and then calls the PrISM_Enable() function. The start function sets polynomial, seed, and pulse density registers provided by the customizer. PrISM computation starts on the rising edge of the input clock.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_Stop(void)

Description: Stops PrISM computation. Outputs remain constant.

Parameters: None

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

void PrISM_SetPulse0Mode(uint8 pulse0Type)

Description: Sets the pulse density type for Density0. Less Than or Equal(<=) or Greater Than or Equal(>=).

Parameters: uint8 pulse0Type: Selected pulse density type

Parameters Value	Description
PrISM_LESSTHAN_OR_EQUAL	The pulse_den0 output is high when the pseudo random number is less than or equal to the PulseDensity0 register value
PrISM_GREATERTHAN_OR_EQUAL	The pulse_den0 output is high when the pseudo random number is greater than or equal to the PulseDensity0 register value

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

void PrISM_SetPulse1Mode(uint8 pulse1Type)

Description: Sets the pulse density type for Density1. Less Than or Equal(<=) or Greater Than or Equal(>=).

Parameters: uint8 pulse1Type: Selected pulse density type

Parameters Value	Description
PrISM_LESSTHAN_OR_EQUAL	The pulse_den1 output is high when the pseudo random number is less than or equal to the PulseDensity1 register value
PrISM_GREATERTHAN_OR_EQUAL	The pulse_den1 output is high when the pseudo random number is greater than or equal to the PulseDensity1 register value

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

uint8/16/32 PrISM_ReadSeed(void)

Description: Reads the PrISM seed register.

Parameters: None

Return Value: uint8/16/32: Seed register value

Side Effects: None

void PrISM_WriteSeed(uint8/16/32 seed)

Description: Writes the PrISM seed register with the start value.

Parameters: uint8/16/32) seed: Seed register value

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPolynomial(void)

Description: Reads the PrISM polynomial.

Parameters: None

Return Value: uint8/16/32: Value of the polynomial

Side Effects: None

void PrISM_WritePolynomial(uint8/16/32 polynomial)

Description: Writes the PrISM polynomial.

Parameters: uint8/16/32 polynomial: Polynomial register value

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPulse0(void)

Description: Reads the PrISM PulseDensity0 value register.

Parameters: None

Return Value: uint8/16/32: PulseDensity0 register value

Side Effects: None

void PrISM_WritePulse0(uint8/16/32 pulseDensity0)

Description: Writes the PrISM Pulse Density0 value register with the new Pulse Density value.

Parameters: (unit8/16/32) pulseDensity0: Pulse Density value.

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPulse1(void)

Description: Reads the PrISM Pulse Density1 value register.

Parameters: None

Return Value: uint8/16/32: Pulse Density1 register value

Side Effects: None

void PrISM_WritePulse1(uint8/16/32 pulseDensity1)

Description: Writes the PrISM Pulse Density1 value register with the new Pulse Density value.

Parameters: uint8/16/32 pulseDensity1: Pulse Density value

Return Value: None

Side Effects: None

void PrISM_Sleep(void)

Description: This is the preferred API to prepare the component for sleep. The PrISM_Sleep() API saves the current component state. Then it calls the PrISM_Stop() function and calls PrISM_SaveConfig() to save the hardware configuration.

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

Parameters: None

Return Value: None

Side Effects: None

void PrISM_Wakeup(void)

Description: This is the preferred API to restore the component to the state when PrISM_Sleep() was called. The PrISM_Wakeup() function calls the PrISM_RestoreConfig() function to restore the configuration. If the component was enabled before the PrISM_Sleep() function was called, the PrISM_Wakeup() function will also re-enable the component.

Parameters: None

Return Value: None

Side Effects: Calling the PrISM_Wakeup() function without first calling the PrISM_Sleep() or PrISM_SaveConfig() function may produce unexpected behavior.

void PrISM_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call PrISM_Init() because the PrISM_Start() API calls this function and is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: All registers will be set to values according to the customizer Configure dialog.

void PrISM_Enable(void)

Description: Activates the hardware and begins component operation. It is not necessary to call PrISM_Enable() because the PrISM_Start() API calls this function, which is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_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 PrISM_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_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 PrISM_Sleep() function.

Parameters: None

Return Value: None

Side Effects: Calling this function without first calling the PrISM_Sleep() or PrISM_SaveConfig() function may produce unexpected behavior.

Sample Firmware Source Code

PSoC Creator provides numerous 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.

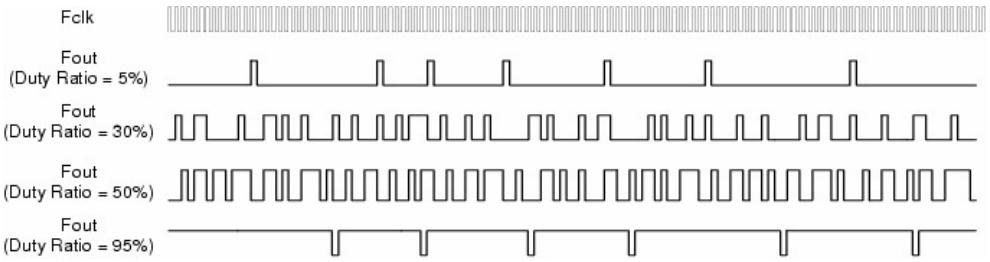
Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

Functional Description

The PrISM component runs continuously after starting and as long as the “enable” input is held high. The PrISM pseudo random number generator may be started with any valid value excluding 0. This allows multiple PrISM components to run out of phase of each other to further reduce EMI. The “reset” input resets the pseudo random number to the start value. The active-high “kill” input disables the PrISM pulse density outputs and sets them to 0 until kill is released low. The “bitstream” output continuously outputs the LSb of the LFSR.

Two Pulse Density outputs are available; both are derived from the same pseudo random sequence. Each output is generated by comparing the desired pulse density value with the current pseudo random number.

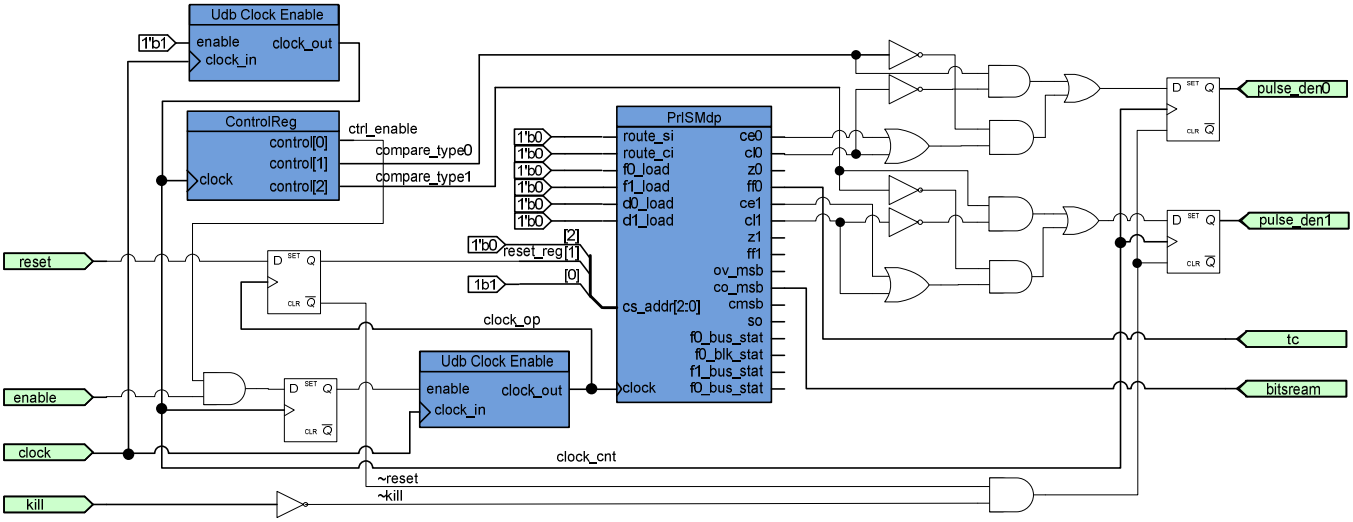
The following timing diagram shows the PrISM output based on several pulse density ratios.



Block Diagram and Configuration

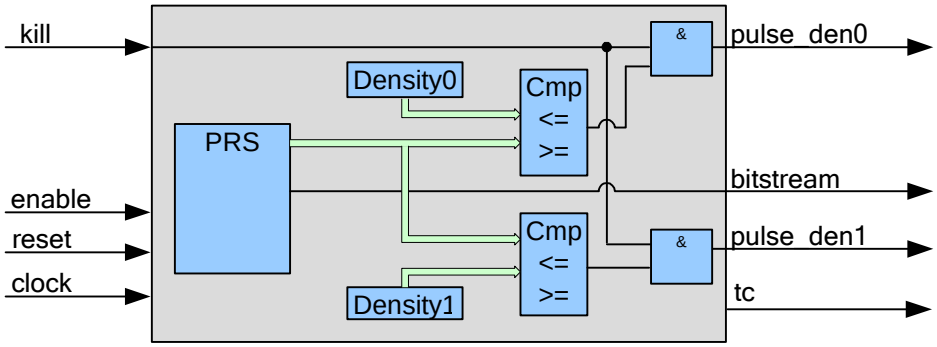
The PrISM is only available as a UDB configuration. The API is described above and the registers are described here to define the overall implementation of the PrISM.

The implementation is described in the following block diagram.



To-Level Architecture

The 2- to 32-bit hardware PrISM component compares the output of a pseudo random counter with a signal density value. The comparator output asserts when the count value is less than (or greater than) or equal to the value in the Density value register.



Registers

PrISM_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	reserved					compare type1	compare type0	ctrl enable

- ctrl enable: This bit enables generation of all internal signals described in the previous sections. The value can be changed by the PrISM_Start() and PrISM_Stop() functions.
- compare type0: This bit performs compare type for the pulse_den0 output. The value of this bit is determined by the choice made for the pulse compare type parameter in the component Configure dialog. Also, the value can be changed by the PrISM_SetPulse0Mode() function.
- compare type1: This bit performs compare type for pulse_den1 output. The value of this bit is determined by the choice made for the pulse compare type parameter in the component Configure dialog. Also, the value can be changed by the PrISM_SetPulse1Mode() function.

The control register is not used if the **PulseType Hardcoded** option is selected.

PrISM_SEED

Bits	7	6	5	4	3	2	1	0
Value	Seed							

- Seed: Contains the initial Seed value and PRS residual value at the end of the computation. The value of this register is determined by the **Seed value** parameter in the component Configure dialog. Also, the value can be changed by the PrISM_WriteSeed() function and can be read by PrISM_ReadSeed().

PrISM_SEED_COPY

Bits	7	6	5	4	3	2	1	0
Value	Seed_Copy							

- Seed_Copy: Contains the start Seed value for automatically loading PrISM_SEED register when the “reset” input is active. The value of this register is determined by the **Seed value** parameter in the component Configure dialog and automatically updates if the PrISM_WriteSeed() function is called.

PrISM_POLYNOM

Bits	7	6	5	4	3	2	1	0
Value	Polynomial							

- Polynomial: The correct polynomial chosen based on the resolution selected. The value can be changed by the PrISM_WritePolynomial() function and can be read by the PrISM_ReadPolynomial() function.

PrISM_DENSITY0

Bits	7	6	5	4	3	2	1	0
Value	Pulse density0							

- Pulse density0 determines the value for the PrISM pulse_den0 output. The value of this register is determined by the **PulseDensity0** parameter in the Configure dialog. This value can be changed by the PrISM_WritePulse0() function.

PrISM_DENSITY1

Bits	7	6	5	4	3	2	1	0
Value	Pulse density1							

- Pulse density1 determines the value for the PrISM pulse_den1 output. The value of this register is determined by the **PulseDensity1** parameter in the Configure dialog. This value can be changed by the PrISM_WritePulse1() function.

References

Refer also to the PRS component datasheet.

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	Config.	Min	Typ	Max	Units
f_{CLOCK}	Component clock frequency	8-bit			66	MHz
		16-bit			46	MHz
		24-bit			42	MHz
		32-bit			38	MHz
t_{clockH}	Input clock high time [1]	N/A		0.5		$1/f_{CLOCK}$
t_{clockL}	Input clock low time [1]	N/A		0.5		$1/f_{CLOCK}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [2]	1			STA [3]	ns
t_{PD_ps}	Input path delay, pin to sync [4]	2			8.5	ns
t_{PD_si}	Sync output to input path delay (route)	1,2,3,4			STA [3]	ns
t_{I_clk}	Alignment of clockX and clock	1,2,3,4	0		1	t_{CY_clock}
t_{PD_IE}	Input path delay to component clock (edge-sensitive input)	1,2	$t_{PD_ps} + t_{SYNC} + t_{PD_si}$		$t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	Input path delay to component clock (edge-sensitive input)	3,4	$t_{SYNC} + t_{PD_si}$		$t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	Input high Time	1,2,3,4	t_{CY_clock}			ns
t_{IL}	Input low time	1,2,3,4	t_{CY_clock}			ns

[1] $t_{CY_clock} = 1/f_{CLOCK}$. This is the cycle time of one clock period

[2] t_{PD_ps} is found in the Static Timing Results as described later. The number listed here is a nominal value based on STA analysis on many inputs.

[3] t_{PD_ps} and t_{PD_si} are route path delays. Because routing is dynamic, these values can change and directly affect the maximum component clock and sync clock frequencies. The values must be found in the Static Timing Analysis results.

[4] t_{PD_ps} in configuration 2 is a fixed value defined per pin of the device. The number listed here is a nominal value of all of the pins available on the device.

Timing Characteristics “Maximum with All Routing”

Parameter	Description	Config.	Min	Typ	Max[1]	Units
f_{CLOCK}	Component clock frequency	8-bit			40	MHz
		16-bit			23	MHz
		24-bit			21	MHz
		32-bit			19	MHz
t_{clockH}	Input clock high time [2]	N/A		0.5		$1/f_{CLOCK}$
t_{clockL}	Input clock low time [2]	N/A		0.5		$1/f_{CLOCK}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [3]	1			STA [4]	ns
t_{PD_ps}	Input path delay, pin to sync [5]	2			8.5	ns
t_{PD_si}	Sync output to input path delay (route)	1,2,3,4			STA [4]	ns
t_{I_clk}	Alignment of clockX and clock	1,2,3,4	0		1	t_{CY_clock}
t_{PD_IE}	Input path delay to component clock (edge-sensitive input)	1,2	$t_{PD_ps} + t_{SYNC} + t_{PD_si}$		$t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	Input path delay to component clock (edge-sensitive input)	3,4	$t_{SYNC} + t_{PD_si}$		$t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	Input high Time	1,2,3,4	t_{CY_clock}			ns
t_{IL}	Input low time	1,2,3,4	t_{CY_clock}			ns

[1] Maximum for “All Routing” is calculated by <nominal>/2 rounded to the nearest integer. This value makes it unnecessary for you to worry about meeting timing if the component is running at or below this frequency.

[2] $t_{CY_clock} = 1/f_{CLOCK}$. This is the cycle time of one clock period

[3] t_{PD_ps} is found in the Static Timing Results as described later. The number listed here is a nominal value based on STA analysis on many inputs.

[4] t_{PD_ps} and t_{PD_si} are route path delays. Because routing is dynamic, these values can change and directly affect the maximum component clock and sync clock frequencies. The values must be found in the Static Timing Analysis results.

[5] t_{PD_ps} in configuration 2 is a fixed value defined per pin of the device. The number listed here is a nominal value of all of the pins available on the device.

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 the Timing results in the clock summary as the named external clock. The graphic below shows an example of the clock limitations from the *_timing.html*:

-Clock Summary

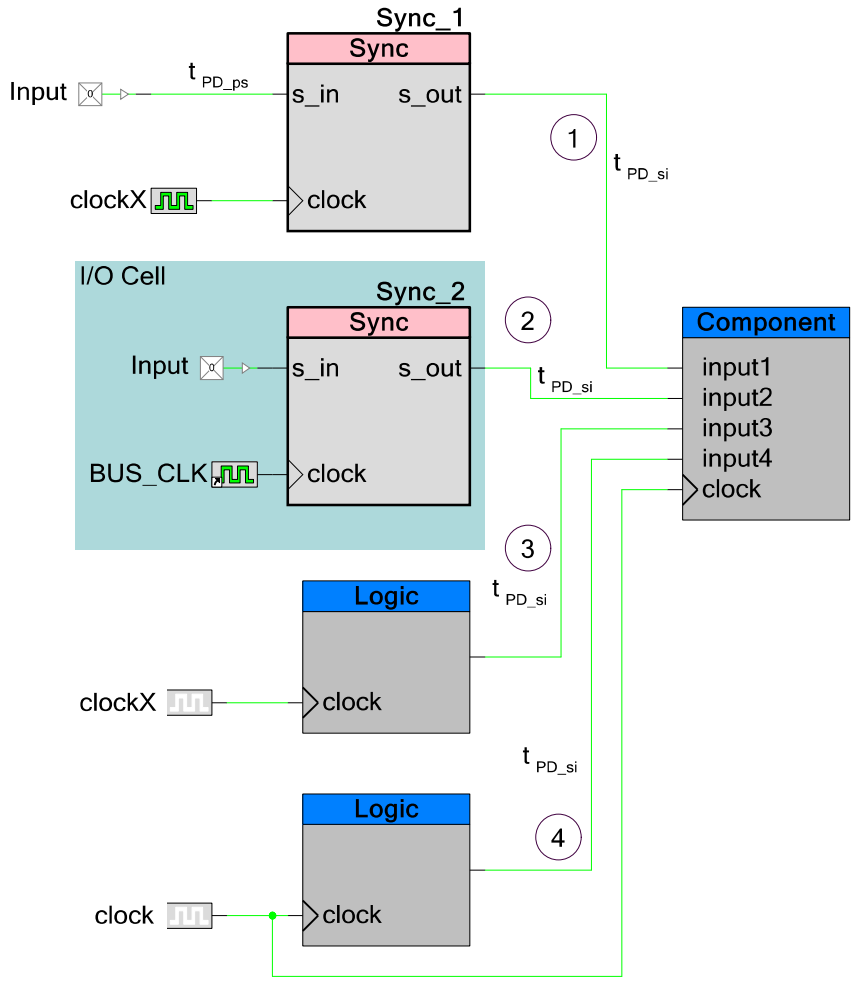
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

Input Path Delay and Pulse Width

When characterizing the functionality of inputs, all inputs, no matter how you have configured them, look like one of four possible configurations, as shown in Figure 1.

All inputs must be synchronized. The synchronization mechanism depends on the source of the input to the component. To fully interpret how your system will work, you must understand which input configuration you have set up for each input and the clock configuration of your system. This section describes how to use the Static Timing Analysis (STA) results to determine the characteristics of your system.

Figure 1. Input Configurations for Component Timing Specifications



Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
1	master_clock	master_clock	Figure 6
1	clock	master_clock	Figure 4
1	clock	clockX = clock [1]	Figure 2
1	clock	clockX > clock	Figure 3
1	clock	clockX < clock	Figure 5
2	master_clock	master_clock	Figure 6
2	clock	master_clock	Figure 4
3	master_clock	master_clock	Figure 11

[1] Clock frequencies are equal but alignment of rising edges is not guaranteed.

Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
3	clock	master_clock	Figure 9
3	clock	clockX = clock [1]	Figure 7
3	clock	clockX > clock	Figure 8
3	clock	clockX < clock	Figure 10
4	master_clock	master_clock	Figure 11
4	clock	clock	Figure 7

1. The input is driven by a device pin and synchronized internally with a “sync” component. This component is clocked using a different internal clock than the clock the component uses (all internal clocks are derived from master_clock).

   When characterizing inputs configured in this way, clockX may be faster than, equal to, or slower than the component clock. It may also be equal to master_clock. This produces the characterization parameters shown in Figure 2, Figure 3, Figure 5, and Figure 6.

2. The input is driven by a device pin and synchronized at the pin using master_clock.

   When characterizing inputs configured in this way, master_clock is faster than or equal to the component clock (it is never slower than). This produces the characterization parameters shown in Figure 3 and Figure 6.

Figure 2. Input Configuration 1 and 2; Sync Clock Frequency = Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)

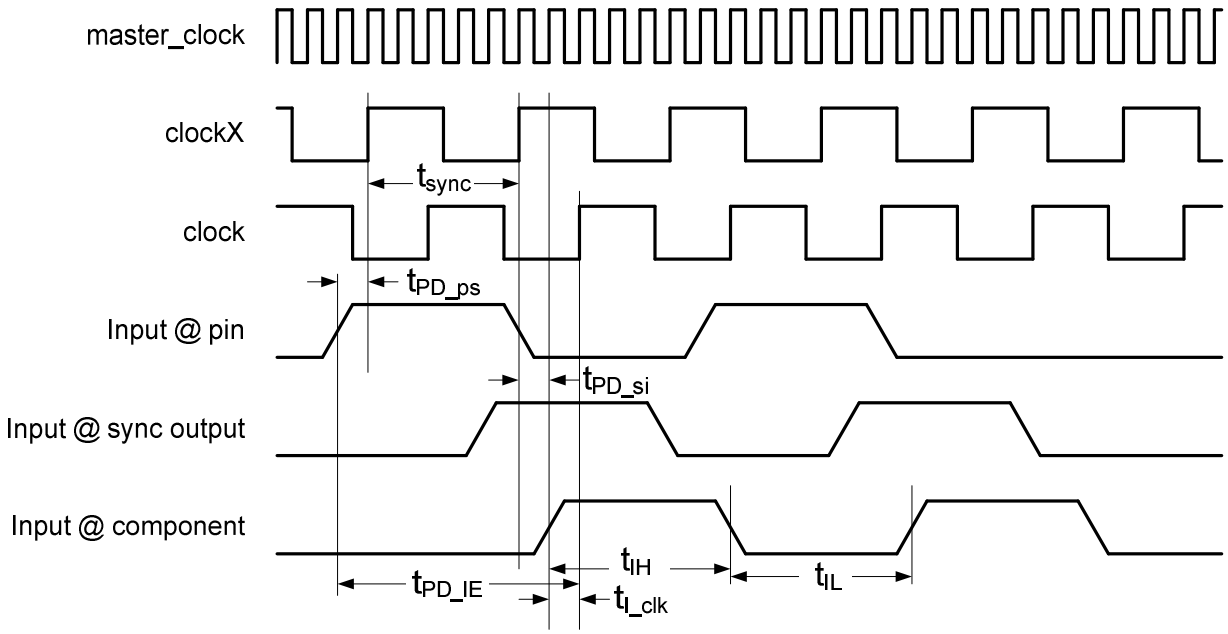


Figure 3. Input Configuration 1 and 2; Sync. Clock Frequency > Component Clock Frequency

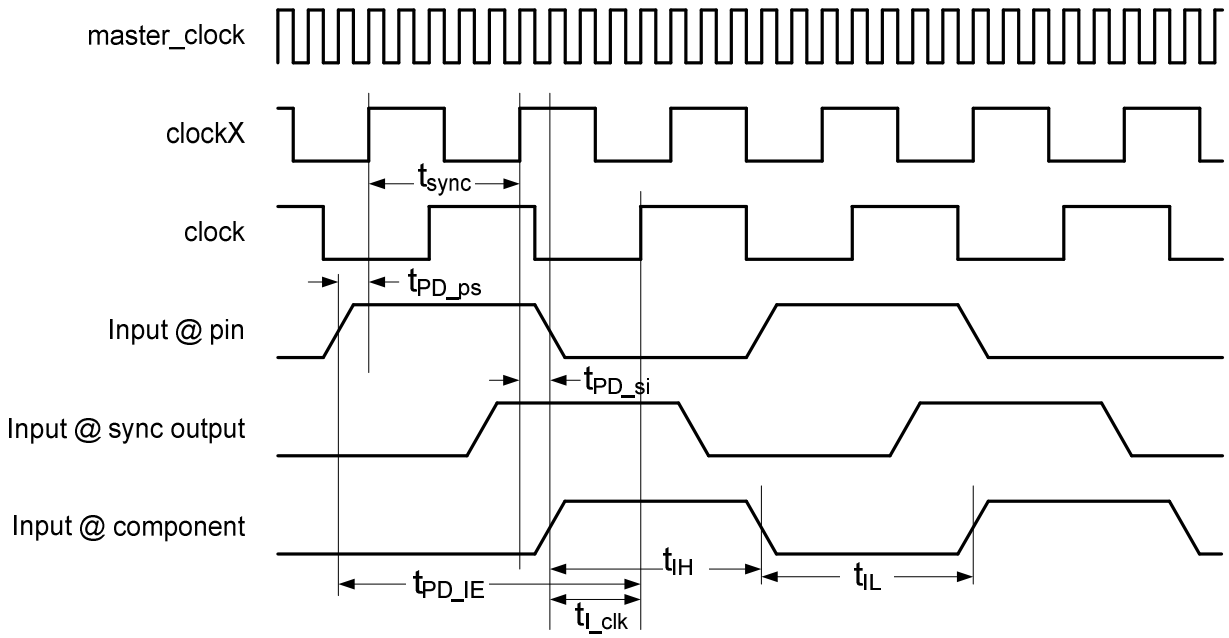


Figure 4. Input Configuration 1 and 2; [Sync. Clock Frequency == master_clock] > Component Clock Frequency

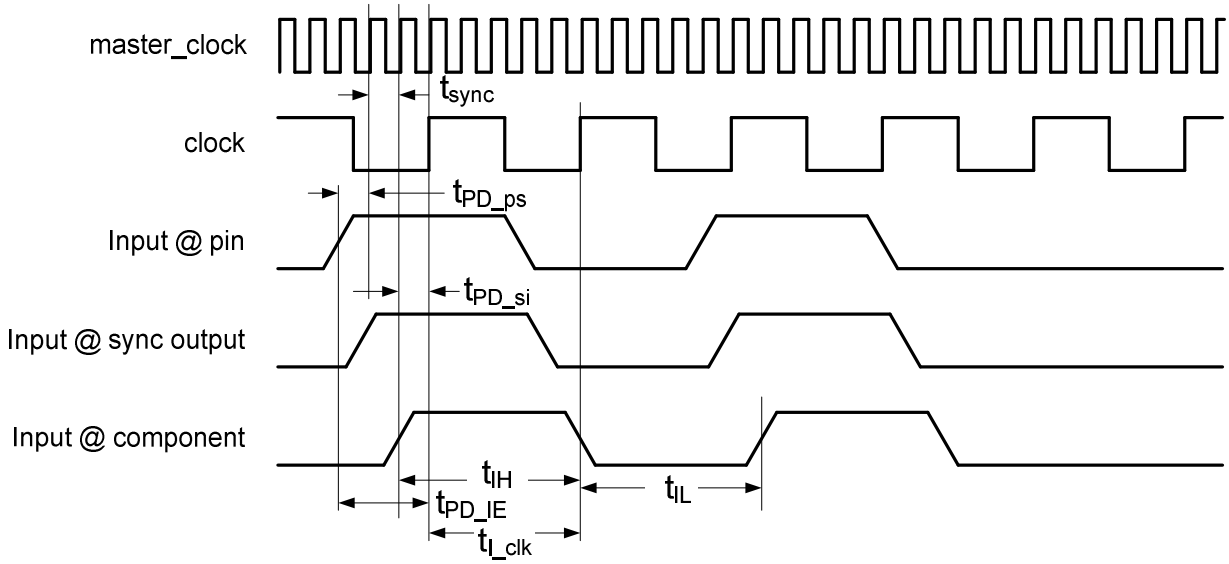


Figure 5. Input Configuration 1; Sync. Clock Frequency < Component Clock Frequency

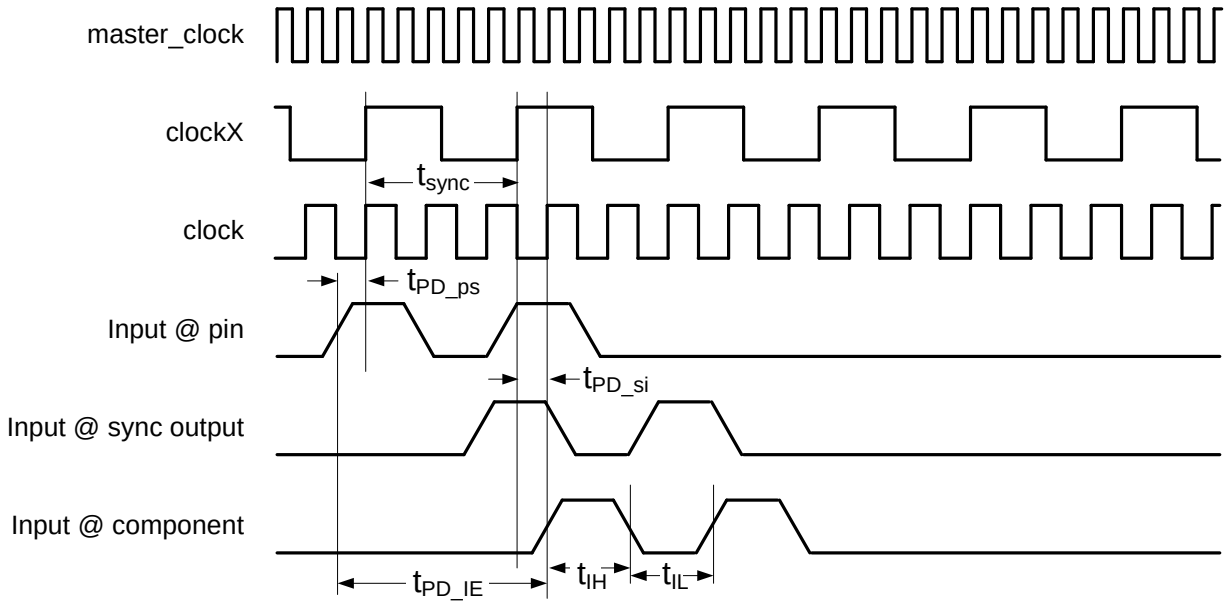
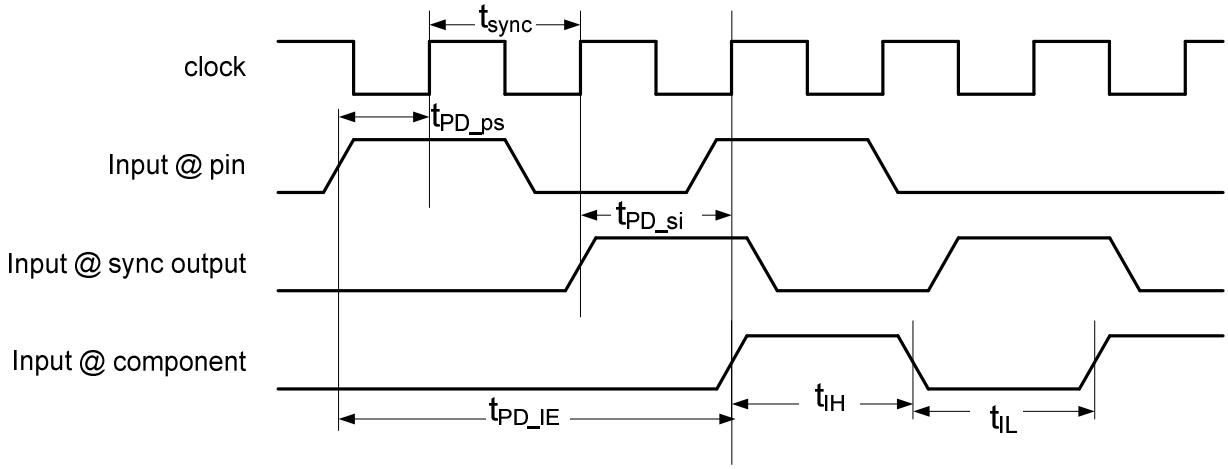


Figure 6. Input Configuration 1 and 2; Sync. Clock = Component Clock = master_clock



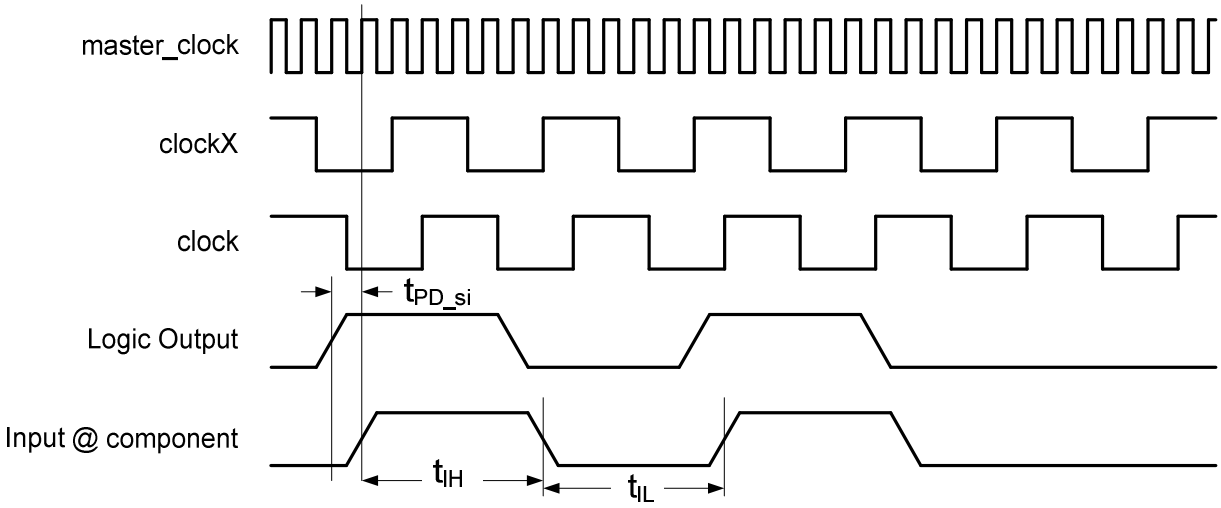
3. The input is driven by logic internal to the PSoC, which is synchronous based on a clock other than the clock the component uses (all internal clocks are derived from master_clock).

   When characterizing inputs configured in this way, the synchronizer clock is faster than, slower than, or equal to the component clock. This produces the characterization parameters shown in Figure 7, Figure 8, and Figure 10.

4. The input is driven by logic internal to the PSoC, which is synchronous based on the same clock the component uses.

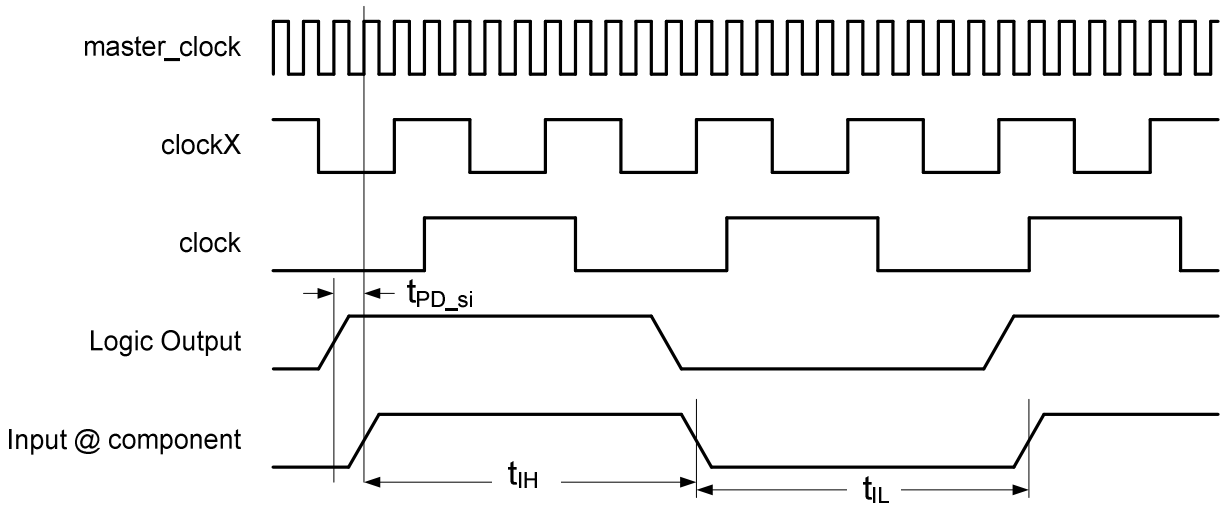
When characterizing inputs configured in this way, the synchronizer clock is equal to the component clock. This produces the characterization parameters shown in Figure 11.

Figure 7. Input Configuration 3 only; Sync. Clock Frequency = Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)



This figure represents the information that Static Timing Analysis has about the clocks. All clocks in the digital clock domain are synchronous to master_clock. However, it is possible for two clocks with the same frequency to not be rising-edge-aligned. Therefore, the static timing analysis tool does not know which edge the clocks are synchronous to and must assume the minimum of one master_clock cycle. This means that t_{PD_si} now has a limiting effect on master_clock of the system. master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run master_clock at a slower frequency.

Figure 8. Input Configuration 3; Sync. Clock Frequency > Component Clock Frequency



In much the same way as shown in Figure 7, all clocks are derived from master_clock. STA indicates the t_{PD_si} limitations on master_clock for one master_clock cycle in this configuration. master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run the master_clock at a slower frequency.

Figure 9. Input Configuration 3; Synchronizer Clock Frequency = master_clock > Component Clock Frequency

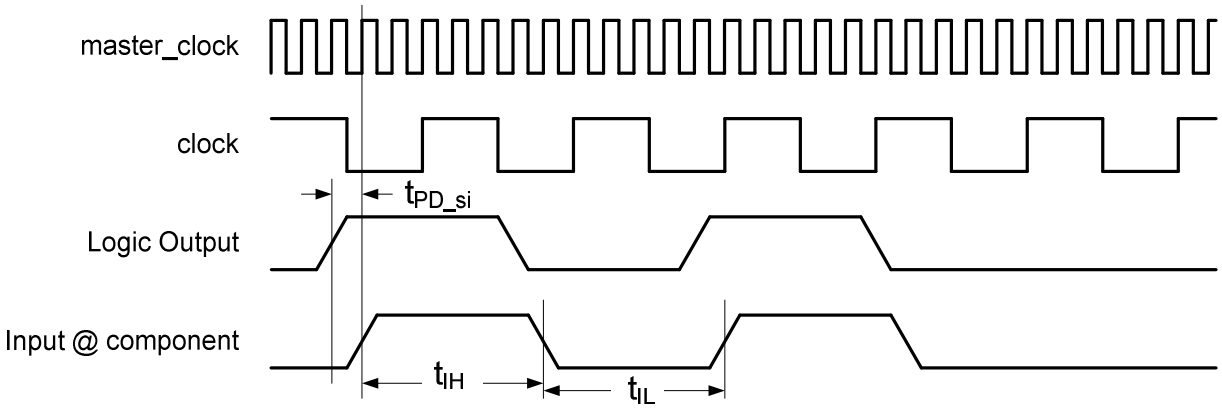
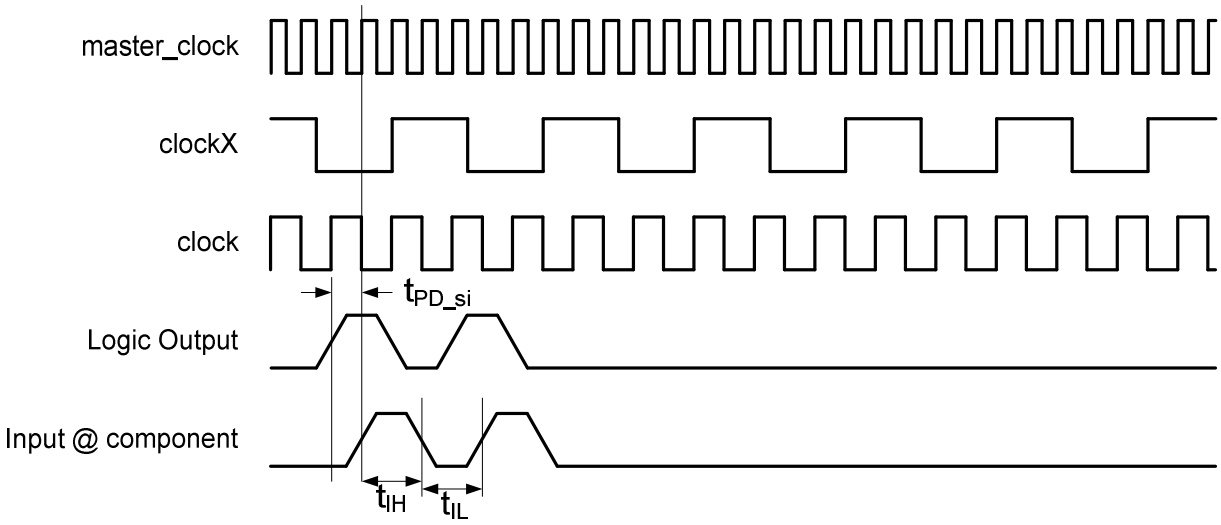
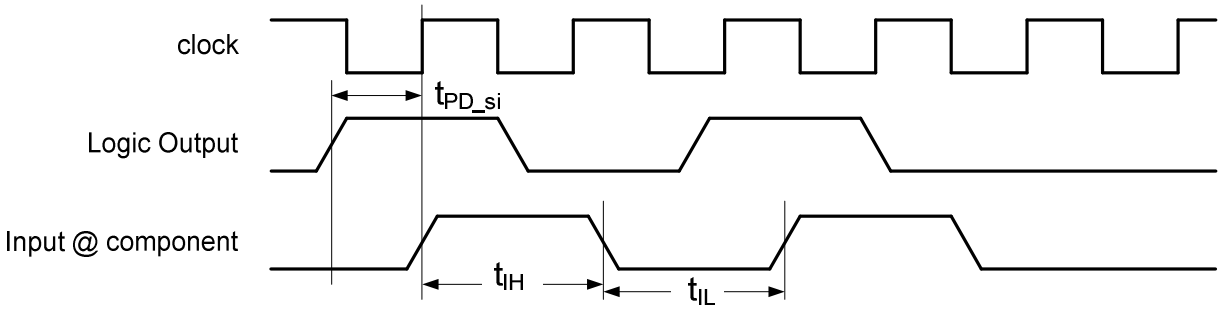


Figure 10. Input Configuration 3; Synchronizer Clock Frequency < Component Clock Frequency



In much the same way as shown in Figure 7, all clocks are derived from master_clock. STA indicates the t_{PD_si} limitations on master_clock for one master_clock cycle in this configuration. master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run master_clock at a slower frequency.

Figure 11. Input Configuration 4 only; Synchronizer Clock = Component Clock



In all previous figures in this section, the most critical parameters to use when understanding your implementation are f_{CLOCK} and t_{PD_IE}. t_{PD_IE} is defined by t_{PD_ps} and t_{SYNC} (for configurations 1 and 2 only), t_{PD_si}, and t_{I_Clk}. It is critical to note that t_{PD_si} defines the maximum component clock frequency. t_{I_Clk} does not come from the STA results but is used to represent when t_{PD_IE} is registered. This is the margin left over after the route between the synchronizer and the component clock.

t_{PD_ps} and t_{PD_si} are included in the STA results.

To find t_{PD_ps}, look at the input setup times defined in the *_timing.html* file. The fanout of this input may be more than 1 so you will need to evaluate the maximum of these paths.

-Setup times

-Setup times to clock BUS_CLK

Start	Register	Clock	Delay (ns)
input1(0):iocell.pad_in	input1(0):iocell.ind	BUS_CLK	16.500

t_{PD_si} is defined in the Register-to-register times. You will need to know the name of the net to use the *_timing.html* file. The fanout of this path may be more than 1 so you will need to evaluate the maximum of these paths.

-Register-to-register times

-Destination clock clock

Destination clock clock (Actual freq: 24.000 MHz)

+Source clock clock

-Source clock clock_1

Source clock clock_1 (Actual freq: 24.000 MHz)
Affected clock: BUS_CLK (Actual freq: 24.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
\Sync_1:genblk1[0]:INST\:synccell.syncq	\PWM_1:PWMUDB:runmode_enable\:macrocell.mc_d	7.843	127.508 MHz	24.000 MHz	

Output Path Delays

When characterizing the path delays of outputs, you must consider where the output is going in order to know where you can find the data in the STA results. For this component, all outputs are synchronized to the component clock. Outputs fall into one of two categories. The output goes either to another component inside the device, or to a pin to the outside of the device. In the first case, you must look at the Register-to-register times shown for the Logic-to-input descriptions above (the source clock is the component clock). For the second case, you can look at the Clock-to-Output times in the *_timing.html* STA results.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.0.a	Minor datasheet edits and updates	
2.0	Pulse Density outputs registered for removing possible glitching.	Any combinatorial output can glitch, depending on placement and delay between signals. To remove glitching, the outputs should be registered.

Version	Description of Changes	Reason for Changes / Impact
	Enable and reset inputs registered to improve maximum speed operation.	These inputs had combinatorial usage, therefore were not automatically registered by Creator and had violations. Registering improves maximum speed and protects from possible glitching.
	Added characterization data to datasheet	
	Minor datasheet edits and updates	

© Cypress Semiconductor Corporation, 2011. 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.