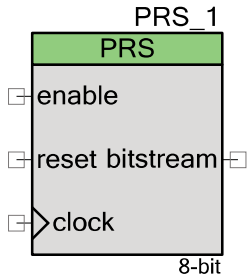
Pseudo Random Sequence (PRS)

2.0

Features



- 2 to 64 bits PRS sequence length
- Time Division Multiplexing mode
- Serial output bit stream
- Continuous or single-step run modes
- Standard or custom polynomial
- Standard or custom seed value
- Enable input provides synchronized operation with other components
- Computed pseudo random number can be read directly from the linear feedback shift register (LFSR)

General Description

The Pseudo Random Sequence (PRS) component uses an LFSR to generate a pseudo random sequence, which outputs a pseudo random bit stream. The LFSR is of the Galois form (sometimes known as the modular form) and uses the provided maximal code length, or period. The PRS component runs continuously after starting as long as the Enable Input is held high. The PRS number generator can be started with any valid seed value other than 0.

When to Use a PRS

LFSRs can be implemented in hardware. This makes them useful in applications that require very fast generation of a pseudo random sequence, such as a direct-sequence spread-spectrum radio.

Global positioning systems use an LFSR to rapidly transmit a sequence that indicates high-precision relative time offsets. Some video game consoles also use an LFSR as part of the sound system.

Used as a Counter

The repeating sequence of states of an LFSR allows it to be used as a divider, or as a counter when a nonbinary sequence is acceptable. LFSR counters have simpler feedback logic than

natural binary counters or Gray code counters, and can therefore operate at higher clock rates. However, you must make sure that the LFSR never enters an all-zeros state, for example by presetting it at startup to any other state in the sequence.

Input/Output Connections

This section describes the various input and output connections for the PRS Component. An asterisk (*) in the list of I/Os states 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 PRS. This input is not available when you choose the API Single Step Run Mode.

reset – Input *

The reset input defines the signal to synchronous reset the PRS. This input is available when you choose clocked mode. You can only reset the PRS if the Enable input is held high.

enable – Input

The PRS component runs after starting and as long as the Enable input is held high. This input provides synchronized operation with other components.

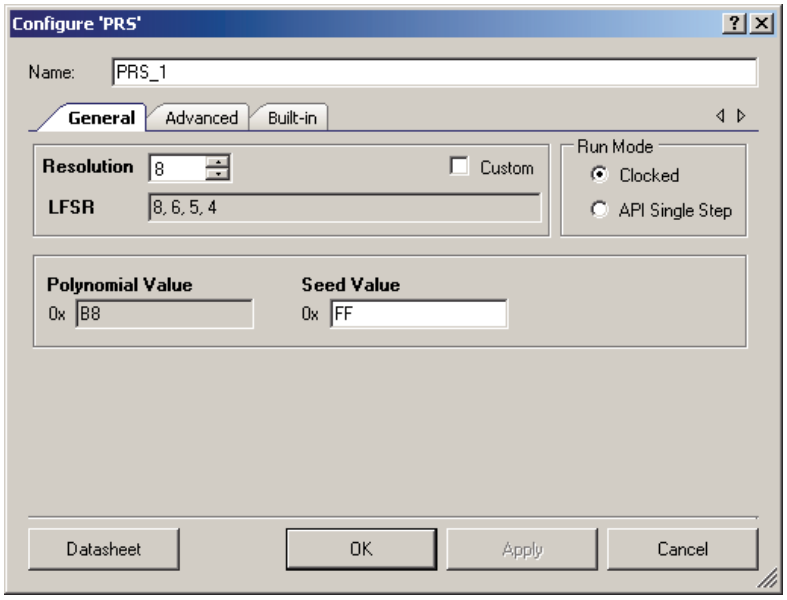
bitstream – Output

Output of the LFSR.

Component Parameters

Drag a PRS component onto your design and double-click it to open the **Configure** dialog. This dialog has several tabs to guide you through the process of setting up the PRS component.

General Tab



Resolution

This defines the PRS sequence length. This value can be set from 2 to 64. The default is **8**.

By default, **Resolution** defines **LFSR** coefficients and **Polynomial Value**. Coefficients are taken from the following table. This parameter also defines the maximal code length, or period, as shown in the following table.

Resolution	LFSR	Period ($2^{Resolution} - 1$)		Resolution	LFSR	Period ($2^{Resolution} - 1$)
2	2, 1	3		34	34, 31, 30, 26	17179869183
3	3, 2	7		35	35, 34, 28, 27	34359738367
4	4, 3	15		36	36, 35, 29, 28	68719476735
5	5, 4, 3, 2	31		37	37, 36, 33, 31	137438953471
6	6, 5, 3, 2	63		38	38, 37, 33, 32	274877906943
7	7, 6, 5, 4	127		39	39, 38, 35, 32	549755813887
8	8, 6, 5, 4	255		40	40, 37, 36, 35	1099511627775
9	9, 8, 6, 5	511		41	41, 40, 39, 38	2199023255551
10	10, 9, 7, 6	1023		42	42, 40, 37, 35	4398046511103

Resolution	LFSR	Period ($2^{Resolution} - 1$)		Resolution	LFSR	Period ($2^{Resolution} - 1$)
11	11, 10, 9, 7	2047		43	43, 42, 38, 37	8796093022207
12	12, 11, 8, 6	4095		44	44, 42, 39, 38	17592186044415
13	13, 12, 10, 9	8191		45	45, 44, 42, 41	35184372088831
14	14, 13, 11, 9	16383		46	46, 40, 39, 38	70368744177663
15	15, 14, 13, 11	32767		47	47, 46, 43, 42	140737488355327
16	16, 14, 13, 11	65535		48	48, 44, 41, 39	281474976710655
17	17, 16, 15, 14	131071		49	49, 45, 44, 43	562949953421311
18	18, 17, 16, 13	262143		50	50, 48, 47, 46	1125899906842623
19	19, 18, 17, 14	524187		51	51, 50, 48, 45	2251799813685247
20	20, 19, 16, 14	1048575		52	52, 51, 49, 46	4503599627370495
21	21, 20, 19, 16	2097151		53	53, 52, 51, 47	9007199254740991
22	22, 19, 18, 17	4194303		54	54, 51, 48, 46	18014398509481983
23	23, 22, 20, 18	8388607		55	55, 54, 53, 49	36028797018963967
24	24, 23, 21, 20	16777215		56	56, 54, 52, 49	72057594037927935
25	25, 24, 23, 22	33554431		57	57, 55, 54, 52	144115188075855871
26	26, 25, 24, 20	67108863		58	58, 57, 53, 52	288230376151711743
27	27, 26, 25, 22	134217727		59	59, 57, 55, 52	576460752303423487
28	28, 27, 24, 22	268435455		60	60, 58, 56, 55	1152921504606846975
29	29, 28, 27, 25	536870911		61	61, 60, 59, 56	2305843009213693951
30	30, 29, 26, 24	1073741823		62	62, 59, 57, 56	4611686018427387903
31	31, 30, 29, 28	2147483647		63	63, 62, 59, 58	9223372036854775807
32	32, 30, 26, 25	4294967295		64	64, 63, 61, 60	18446744073709551615
33	33, 32, 29, 27	8589934591				

To set LFSR coefficients manually:

Define **Resolution**.

Check the **Custom** check box.

Enter coefficients, separated by a comma, in the **LFSR** text box and press [**Enter**]. The Polynomial value is recalculated automatically.

The Polynomial value is shown in hexadecimal form.

Note No LFSR coefficient value can be greater than the **Resolution** value.

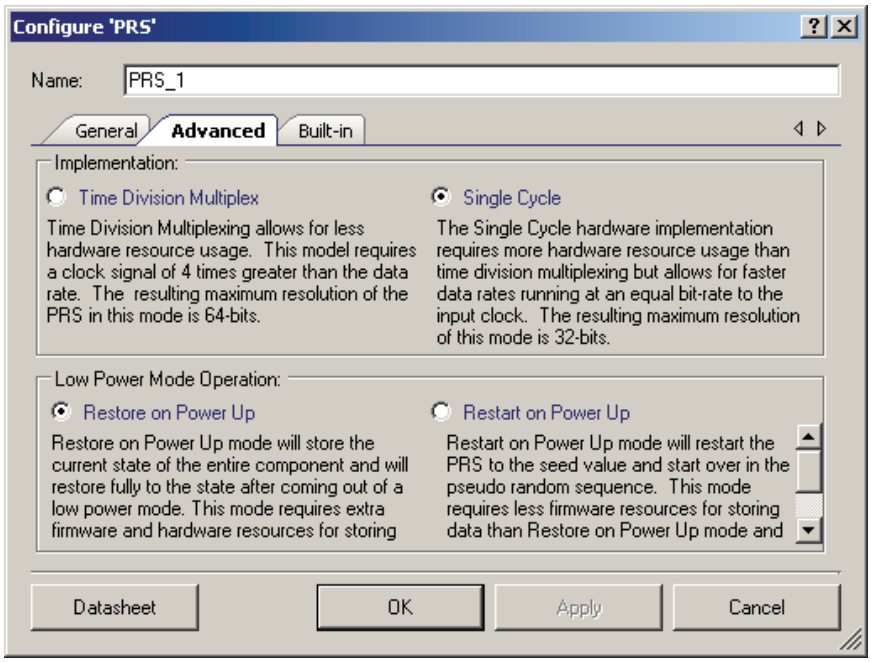
The Seed value, by default, is set to the maximum possible value ($2^{Resolution} - 1$). Its value can be changed to any other except 0. The Seed value is shown in hexadecimal form.

Note Changing the **Resolution** resets **Seed Value** to the default value.

Run Mode

This parameter defines the component operation mode as continuous or single-step run. You can choose **Clocked** (default) or **API Single Step**. If PRS values read continuously or you need one value read, you must stop the clock or set enable to low in Clocked mode.

Advanced Tab



The PRS **Advanced** tab contains the following settings:

Implementation

This defines implementation of PRS component: with time multiplexing or without it (**Single Cycle**). The default is **Single Cycle**.

Low Power Mode Operation

This defines PRS behavior after low-power mode. The default is **Restore on Power Up**.

Local Parameters (For API use)

These parameters are used in the API and are not exposed in the GUI:

- **PolyValueLower(uint32)** – Contains the lower half of the polynomial value in hexadecimal format. The default is 0xB8h (**LFSR=** [8,6,5,4]) because the default resolution is 8.
- **PolyValueUpper(uint32)** – Contains the upper half of the polynomial value in hexadecimal format. The default is 0x00h because the default resolution is 8.
- **SeedValueLower (uint32) –** Contains the lower half of the seed value in hexadecimal format. The default is 0xFFh because the default resolution is 8.
- **SeedValueUpper (uint32) –** Contains the upper half of the seed value in hexadecimal format. The default is 0 because the default resolution is 8.

Clock Selection

You must attach a clock source if you select the **Clocked** option for the **Run Mode** parameter.

Note Generation of the proper PRS sequence for a resolution of greater than 8 requires a clock signal four times greater than the data rate, if you select **Time Division Multiplex** for the **Implementation** parameter.

Placement

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

Resources

Single Cycle, API Single Step

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/Count7 Cells	Flash	RAM	
1..8-Bits Resolution	1	1	0	1	152	3	2
9..16-Bits Resolution	2	1	0	1	188	4	2
17..24-Bits Resolution	3	1	0	1	244	6	2
25..32-Bits Resolution	4	1	0	1	240	6	2

Time Division Multiplex, API Single Step

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/Count7 Cells	Flash	RAM	
9..16-Bits Resolution	1	3	1	1	302	6	2
17..24-Bits Resolution	2	4	1	1	548	8	2
25..32-Bits Resolution	2	3	1	1	624	8	2
33..40-Bits Resolution	3	4	1	1	753	12	2
41..48-Bits Resolution	3	3	1	1	870	12	2
49..56-Bits Resolution	4	4	1	1	956	12	2
57..64-Bits Resolution	4	3	1	1	1035	12	2

Single Cycle, Clocked

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/Count7 Cells	Flash	RAM	
1..8-Bits Resolution	1	1	0	1	180	3	4
9..16-Bits Resolution	2	1	0	1	244	4	4
17..24-Bits Resolution	3	1	0	1	319	6	4
25..32-Bits Resolution	4	1	0	1	302	6	4

Time Division Multiplex, Clocked

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/Count7 Cells	Flash	RAM	
9..16-Bits Resolution	1	3	1	1	318	6	4
17..24-Bits Resolution	2	4	1	1	512	8	4
25..32-Bits Resolution	2	3	1	1	686	8	4
33..40-Bits Resolution	3	4	1	1	855	12	4
41..48-Bits Resolution	3	3	1	1	990	12	4

Resources	Resource Type				API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/Count7 Cells	Flash	RAM	
49..56-Bits Resolution	**4**	4	1	1	1096	12	4
57..64-Bits Resolution	**4**	3	1	1	1175	12	4

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 “PRS_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 “PRS.”

Function	Description
PRS_Start()	Initializes seed and polynomial registers provided from customizer. PRS computation starts on rising edge of input clock.
PRS_Stop()	Stops PRS computation.
PRS_Sleep()	Stops PRS computation and saves PRS configuration.
PRS_Wakeup()	Restores PRS configuration and starts PRS computation on rising edge of input clock.
PRS_Init()	Initializes seed and polynomial registers with initial values.
PRS_Enable()	Starts PRS computation on rising edge of input clock.
PRS_SaveConfig()	Saves seed and polynomial registers.
PRS_RestoreConfig()	Restores seed and polynomial registers.
PRS_Step()	Increments the PRS by one when using API single-step mode.
PRS_WriteSeed()	Writes seed value.
PRS_WriteSeedUpper()	Writes upper half of seed value. Only generated for 33 to 64 bits PRS.
PRS_WriteSeedLower()	Writes lower half of seed value. Only generated for 33 to 64 bits PRS.
PRS_Read()	Reads PRS value.
PRS_ReadUpper()	Reads upper half of PRS value. Only generated for 33 to 64 bits PRS.
PRS_ReadLower()	Reads lower half of PRS value. Only generated for 33 to 64 bits PRS.

Function	Description
PRS_WritePolynomial()	Writes PRS polynomial value.
PRS_WritePolynomialUpper()	Writes upper half of PRS polynomial value. Only generated for 33 to 64 bits PRS.
PRS_WritePolynomialLower()	Writes lower half of PRS polynomial value. Only generated for 33 to 64 bits PRS.
PRS_ReadPolynomial()	Reads PRS polynomial value.
PRS_ReadPolynomialUpper()	Reads upper half of PRS polynomial value. Only generated for 33 to 64 bits PRS.
PRS_ReadPolynomialLower()	Reads lower half of PRS polynomial value. Only generated for 33 to 64 bits PRS.

Global Variables

Variable	Description
PRS_initVar	Indicates whether the PRS has been initialized. The variable is initialized to 0 and set to 1 the first time PRS_Start() is called. This allows the component to restart without reinitialization after the first call to the PRS_Start() routine. If reinitialization of the component is required, then the PRS_Init() function can be called before the PRS_Start() or PRS_Enable() function.

void PRS_Start(void)

Description: Initializes the seed and polynomial registers. PRS computation starts on the rising edge of the input clock.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Stop(void)

Description: Stops PRS computation.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Sleep(void)

Description: Stops PRS computation and saves the PRS configuration.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Wakeup(void)

Description: Restores the PRS configuration and starts PRS computation on the rising edge of the input clock.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Init(void)

Description: Initializes the seed and polynomial registers with initial values.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Enable(void)

Description: Starts PRS computation on the rising edge of the input clock.

Parameters: None

Return Value: None

Side Effects: None

void PRS_SaveConfig(void)

Description: Saves the seed and polynomial registers.

Parameters: None

Return Value: None

Side Effects: None

void PRS_RestoreConfig(void)

Description: Restores the seed and polynomial registers.

Parameters: None

Return Value: None

Side Effects: None

void PRS_Step(void)

Description: Increments the PRS by one when API single-step mode is used.

Parameters: None

Return Value: None

Side Effects: None

void PRS_WriteSeed(uint8/16/32 seed)

Description: Writes the seed value.

Parameters: uint8/16/32 seed: Seed value

Return Value: None

Side Effects: The seed value is cut according to mask = $2^{Resolution} - 1$.
For example, if PRS resolution is 14 bits, the mask value is: mask = $2^{14} - 1$ = 0x3FFFu.
The seed value = 0xFFFFu is cut: seed AND mask = 0xFFFFu AND 0x3FFFu = 0x3FFFu.

void PRS_WriteSeedUpper(uint32 seed)

Description: Writes the upper half of the seed value. Only generated for 33 to 64 bits PRS.

Parameters: uint32 seed: Upper half of the seed value

Return Value: None

Side Effects: The upper half of the seed value is cut according to mask = $2^{(Resolution - 32)} - 1$.
For example, if PRS Resolution is 35 bits the mask value is:
$2^{(35 - 32)} - 1 = 2^{\wedge 3} - 1$ = 0x0000 0007u.
The upper half of the seed value = 0x0000 00FFu is cut:
upper half of seed AND mask = 0x0000 00FFu AND 0x0000 0007u = 0x0000 0007u.

void PRS_WriteSeedLower(uint32 seed)

Description: Writes the lower half of the seed value. Only generated for 33 to 64 bits PRS.

Parameters: uint32 seed: Lower half of the seed value

Return Value: None

Side Effects: None

uint8/16/32 PRS_Read(void)

Description: Reads the PRS value.

Parameters: None

Return Value: uint8/16/32: Returns the PRS value.

Side Effects: None

uint32 PRS_ReadUpper(void)

Description: Reads the upper half of the PRS value. Only generated for 33 to 64 bits PRS.

Parameters: None

Return Value: uint32: Returns the upper half of the PRS value.

Side Effects: None

uint32 PRS_ReadLower(void)

Description: Reads the lower half of the PRS value. Only generated for 33 to 64 bits PRS

Parameters: None

Return Value: uint32: Returns the lower half of the PRS value.

Side Effects: None

void PRS_WritePolynomial(uint8/16/32 polynomial)

Description: Writes the PRS polynomial value.

Parameters: uint8/16/32 polynomial: PRS polynomial.

Return Value: None

Side Effects: The polynomial value is cut according to mask = $2^{Resolution} - 1$.
For example, if PRS Resolution is 14 bits the mask value is: mask = $2^{14} - 1$ = 0x3FFFu.
The polynomial value = 0xFFFFu is cut:
polynomial AND mask = 0xFFFFu AND 0x3FFFu = 0x3FFFu.

void PRS_WritePolynomialUpper(uint32 polynomial)

Description: Writes the upper half of the PRS polynomial value. Only generated for 33 to 64 bits PRS.

Parameters: uint32 polynomial: Upper half of the PRS polynomial value.

Return Value: None

Side Effects: The upper half or the polynomial value is cut according to mask = $2^{(Resolution - 32)} - 1$.
For example, if PRS Resolution is 35 bits the mask value is:
$2^{(35 - 32)} - 1 = 2^3 - 1$ = 0x0000 0007u.
The upper half of the polynomial value = 0x0000 00FFu is cut:
upper half of the polynomial AND mask = 0x0000 00FFu AND 0x0000 0007u = 0x0000 0007u.

void PRS_WritePolynomialLower(uint32 polynomial)

Description: Writes the lower half of the PRS polynomial value. Only generated for 33 to 64 bits PRS.

Parameters: uint32 polynomial: Lower half of the PRS polynomial value

Return Value: None

Side Effects: None

uint8/16/32 PRS_ReadPolynomial(void)

Description: Reads the PRS polynomial value.

Parameters: None

Return Value: uint8/16/32: Returns the PRS polynomial value.

Side Effects: None

uint32 PRS_ReadPolynomialUpper(void)

Description:	Reads the upper half of the PRS polynomial value. Only generated for 33 to 64 bits PRS.
Parameters:	None
Return Value:	uint32: Returns the upper half of the PRS polynomial value.
Side Effects:	None

uint32 PRS_ReadPolynomialLower(void)

Description:	Reads the lower half of the PRS polynomial value. Only generated for 33 to 64 bits PRS.
Parameters:	None
Return Value:	uint32: Returns the lower half of the PRS polynomial value.
Side Effects:	None

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

PRS Run Mode: Clocked

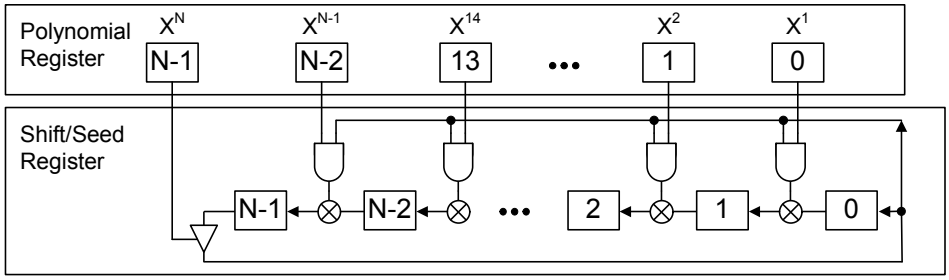
In this mode, the PRS component runs continuously after it starts and as long as the Enable input is held high.

PRS Run Mode: API Single Step

In this mode, the PRS is incremented by an API call.

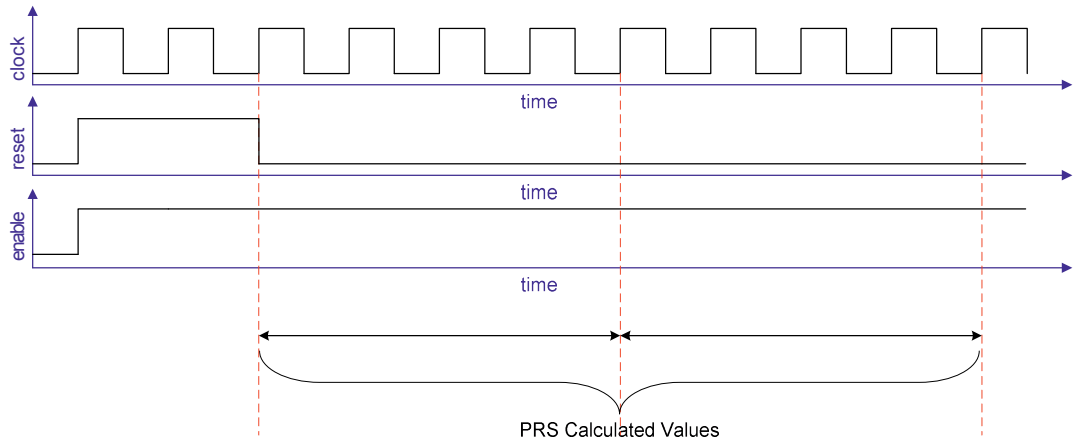
Block Diagram and Configuration

The PRS is implemented as a set of configured UDBs. The implementation is shown in the following block diagram.

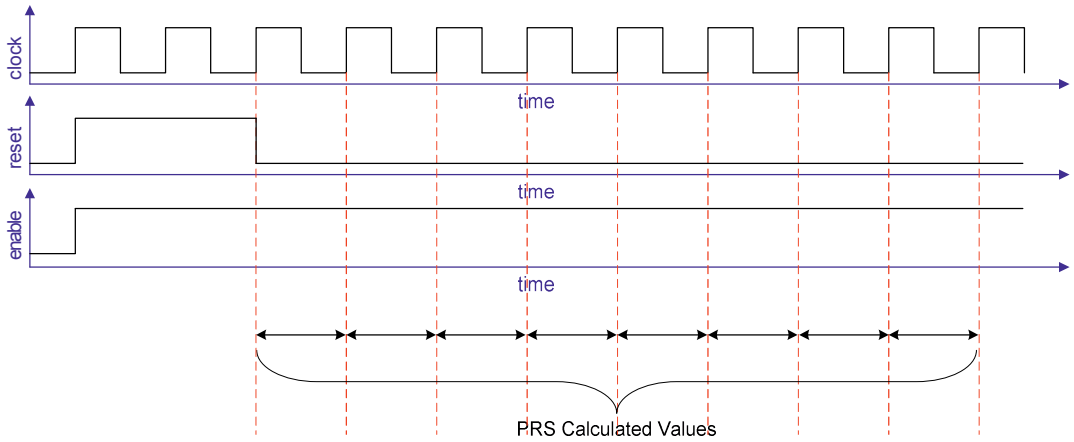


Timing Diagrams

Time Division Multiplex Implementation Mode



Single Cycle Implementation Mode



Registers

Polynomial Register (from 2 to 64 bits based on Resolution)

The Polynomial register contains the polynomial value. You can change it with the PRS_WritePolynomial(), PRS_WritePolynomialUpper(), or PRS_WritePolynomialLower() functions. You can also read the current polynomial value using PRS_ReadPolynomial(), PRS_ReadPolynomialUpper(), or PRS_ReadPolynomialLower().

Shift/Seed register (from 2 to 64 bits based on Resolution)

The Shift/Seed register contains the seed value. You can change it with the PRS_WriteSeed(), PRS_WriteSeedUpper(), or PRS_WriteSeedLower() functions. You can also read the current seed value using PRS_ReadSeed(), PRS_ReadSeedUpper(). or PRS_ReadSeedLower().

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. [1]	Min	Typ	Max	Units
f_{CLOCK}	Component clock frequency [2]	Config 1	–	–	57	MHz
		Config 2	–	–	57	MHz
		Config 3	–	–	35	MHz
		Config 4	–	–	30	MHz
		Config 5	–	–	43	MHz

[1] Configurations:

Config 1:
- Resolution: 8 bits
- Run Mode: API Single Step
- Low Power Mode Operation: Restore on Power Up
- Implementation: Single Cycle

Config 2:
- Resolution: 8 bits
- Run Mode: Clocked
- Low Power Mode Operation: Restore on Power Up
- Implementation: Single Cycle

Config 3:
- Resolution: 16 bits
- Run Mode: Clocked
- Low Power Mode Operation: Restore on Power Up
- Implementation: Time Division Multiplex

Config 4:
- Resolution: 16 bits
- Run Mode: API Single Step
- Low Power Mode Operation: Restore on Power Up
- Implementation: Time Division Multiplex

Config 5:
- Resolution: 32 bits
- Run Mode: API Single Step
- Low Power Mode Operation: Restore on Power Up
- Implementation: Single Cycle

Config 6:
- Resolution: 32 bits
- Run Mode: Clocked
- Low Power Mode Operation: Restore on Power Up
- Implementation: Single Cycle

Config 7:
- Resolution: 64 bits
- Run Mode: API Single Step
- Low Power Mode Operation: Restore on Power Up
- Implementation: Time Division Multiplex

Config 8:
- Resolution: 64 bits
- Run Mode: Clocked
- Low Power Mode Operation: Restore on Power Up
- Implementation: Time Division Multiplex

[2] If Time Division Multiplex Implementation is selected, then the component clock frequency must be four times greater than the data rate.

Parameter	Description	Config. [1]	Min	Typ	Max	Units
		Config 6	–	–	40	MHz
		Config 7	–	–	25	MHz
		Config 8	–	–	30	MHz
t_{clockH}	Input clock high time [3]	N/A	–	0.5	–	$1/f_{clock}$
t_{clockL}	Input clock low time [3]	N/A	–	0.5	–	$1/f_{clock}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [4]	1	–	–	STA [5]	ns
t_{PD_ps}	Input path delay, pin to sync [6]	2	–	–	8.5	ns
t_{PD_si}	Sync output to input path delay (route)	1,2,3,4	–	–	STA [5]	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} [7]	–	–	ns
t_{IL}	Input Low Time	1,2,3,4	t_{CY_clock} [7]	–	–	ns
Outputs						
	Output register-to-register time		–	–	–	
	Output to pin path delay		–	–	–	

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

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

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

[6] 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

[7] $t_{CY_clock} = 4\times [1/f_{CLOCK}]$ if Time Division Multiplex Implementation is selected.

Timing Characteristics "Maximum with All Routing"

Parameter	Description	Config.[1]	Min	Typ	Max[2]	Units
f_{CLOCK}	Component clock frequency [3]	Config 1	–	–	29	MHz
		Config 2	–	–	29	MHz
		Config 3	–	–	18	MHz

[1] Configurations:

Config 1:
Resolution: 8 bits
Run Mode: API Single Step
Low Power Mode Operation: Restore on Power Up
Implementation: Single Cycle

Config 2:
Resolution: 8 bits
Run Mode: Clocked
Low Power Mode Operation: Restore on Power Up
Implementation: Single Cycle

Config 3:
Resolution: 16 bits
Run Mode: Clocked
Low Power Mode Operation: Restore on Power Up
Implementation: Time Division Multiplex

Config 4:
Resolution: 16 bits
Run Mode: API Single Step
Low Power Mode Operation: Restore on Power Up
Implementation: Time Division Multiplex

Config 5:
Resolution: 32 bits
Run Mode: API Single Step
Low Power Mode Operation: Restore on Power Up
Implementation: Single Cycle

Config 6:
Resolution: 32 bits
Run Mode: Clocked
Low Power Mode Operation: Restore on Power Up
Implementation: Single Cycle

Config 7:
Resolution: 64 bits
Run Mode: API Single Step
Low Power Mode Operation: Restore on Power Up
Implementation: Time Division Multiplex

Config 8:
Resolution: 64 bits
Run Mode: Clocked
Low Power Mode Operation: Restore on Power Up
Implementation: Time Division Multiplex

[2] Maximum for "All Routing" is calculated by <nominal>/2 rounded to the nearest integer. This value provides a basis for you to not have to worry about meeting timing if the component is running at or below this frequency.

[3] If Time Division Multiplex Implementation is selected, then the component clock frequency must be four times greater than the data rate.

Parameter	Description	Config.[1]	Min	Typ	Max[2]	Units
		Config 4	–	–	15	MHz
		Config 5	–	–	22	MHz
		Config 6	–	–	20	MHz
		Config 7	–	–	13	MHz
		Config 8	–	–	15	MHz
t_{clockH}	Input clock high time [4]	N/A	–	0.5	–	$1/f_{clock}$
t_{clockL}	Input clock low time [4]	N/A	–	0.5	–	$1/f_{clock}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [5]	1	–	–	STA [6]	ns
t_{PD_ps}	Input path delay, pin to sync [7]	2	–	–	8.5	ns
t_{PD_si}	Sync output to input path delay (route)	1,2,3,4	–	–	STA [6]	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} [8]	–	–	ns
t_{IL}	Input low time	1,2,3,4	t_{CY_clock} [8]	–	–	ns
Outputs						
	Output register-to-register time		–	–	–	
	Output to pin path delay		–	–	–	

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

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

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

[7] 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

[8] $t_{CY_clock} = 4\times [1/f_{CLOCK}]$ if Time Division Multiplex Implementation is selected.

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

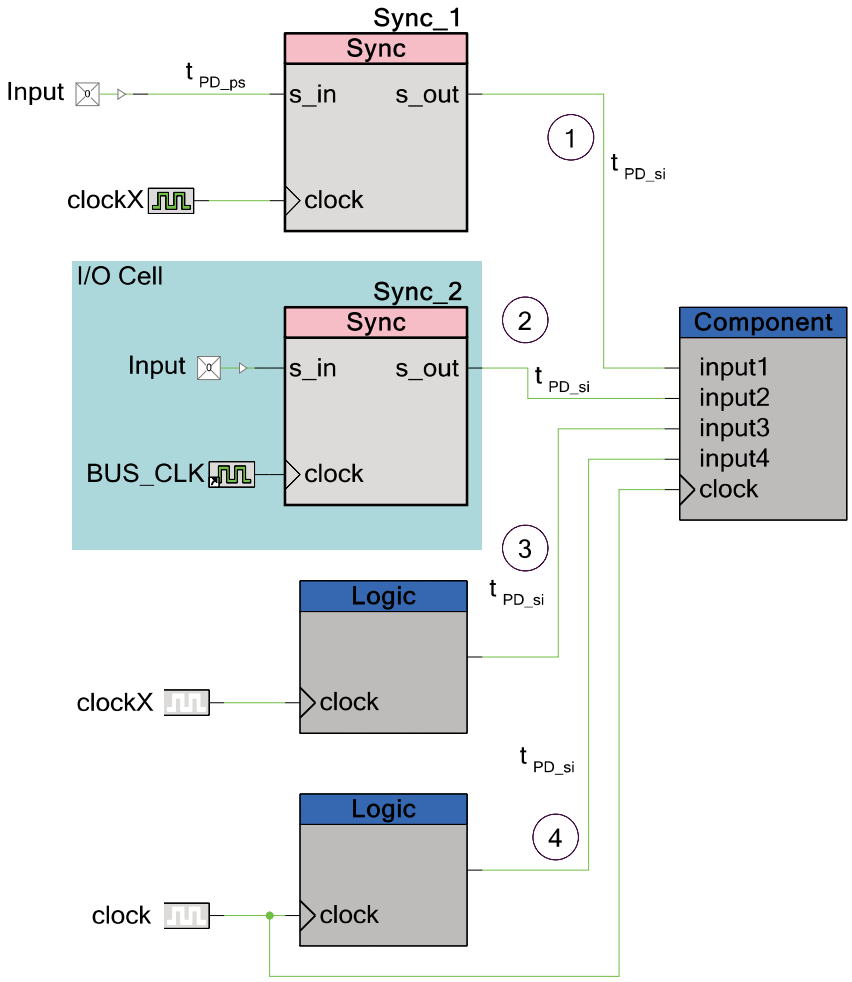
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	66.000 MHz	63.295 MHz	VIOLATION
CharComp_clock	33.000 MHz	57.594 MHz	

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 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)

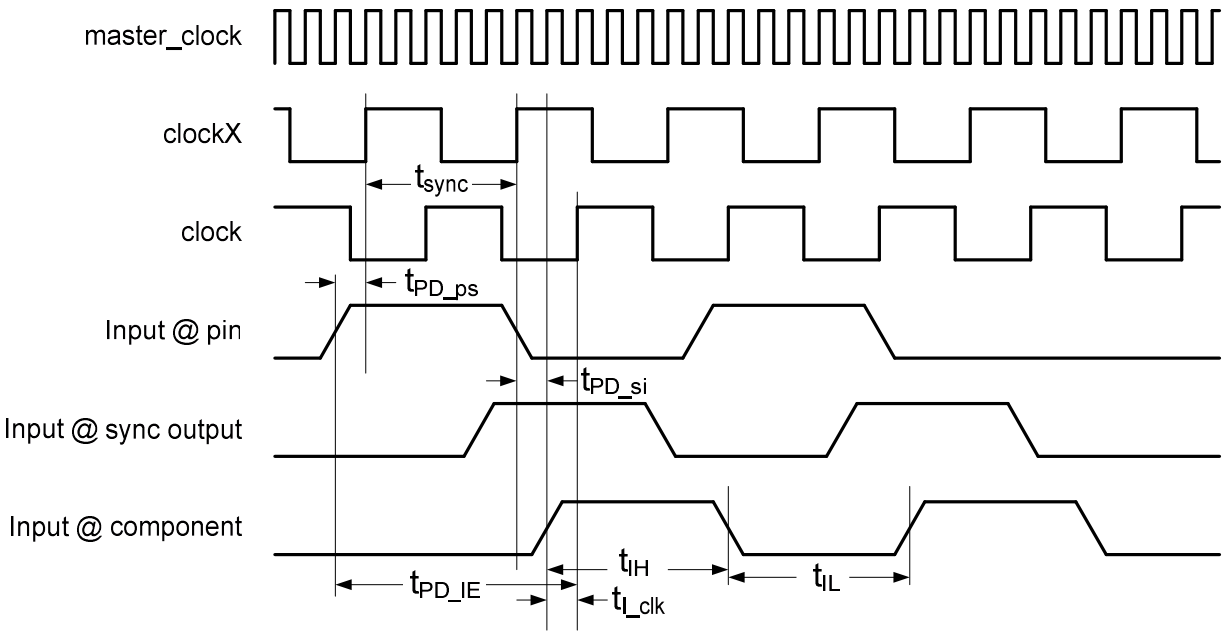


Figure 3. Input Configuration 1 and 2; Sync. Clock Frequency > Component Clock Frequency

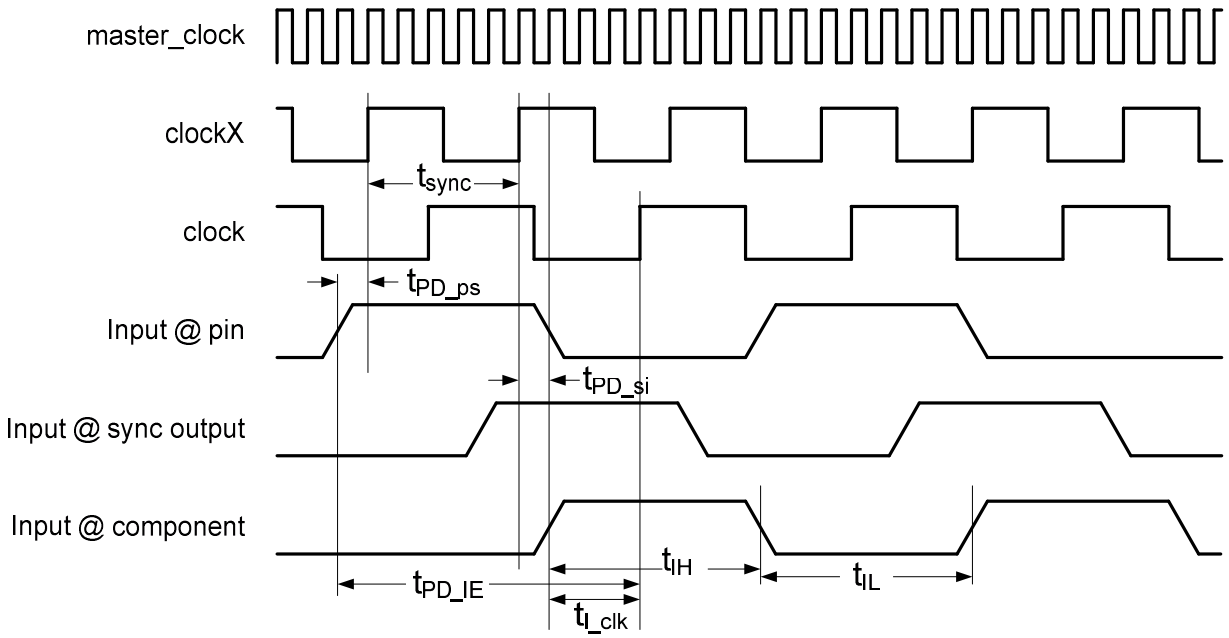


Figure 4. Input Configuration 1 and 2; [Sync. Clock Frequency == master_clock] > Component Clock Frequency

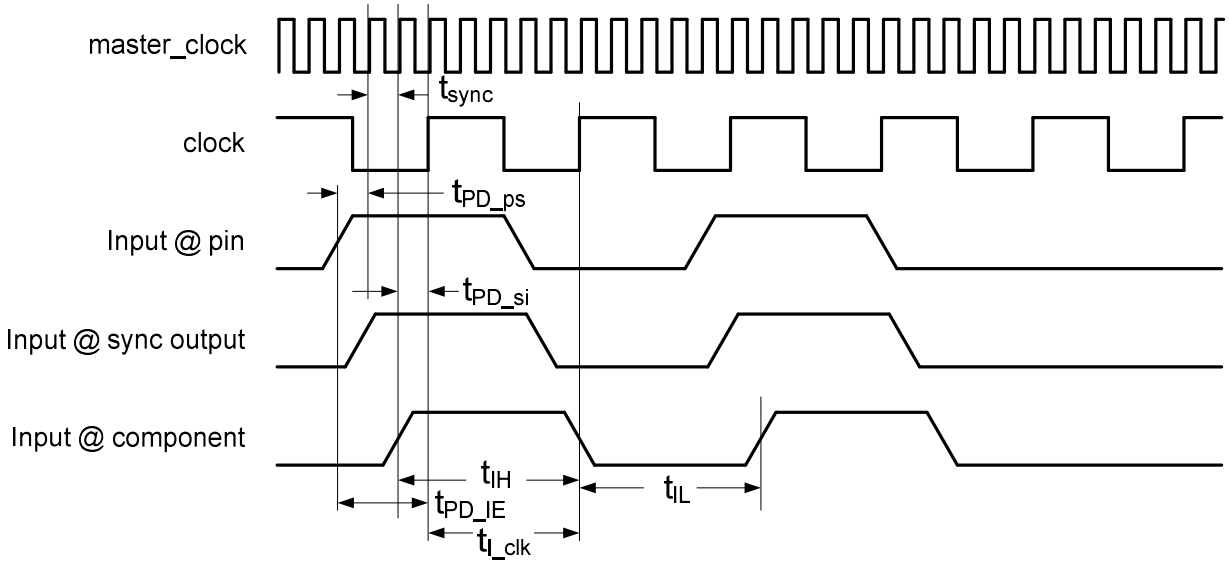


Figure 5. Input Configuration 1; Sync. Clock Frequency < Component Clock Frequency

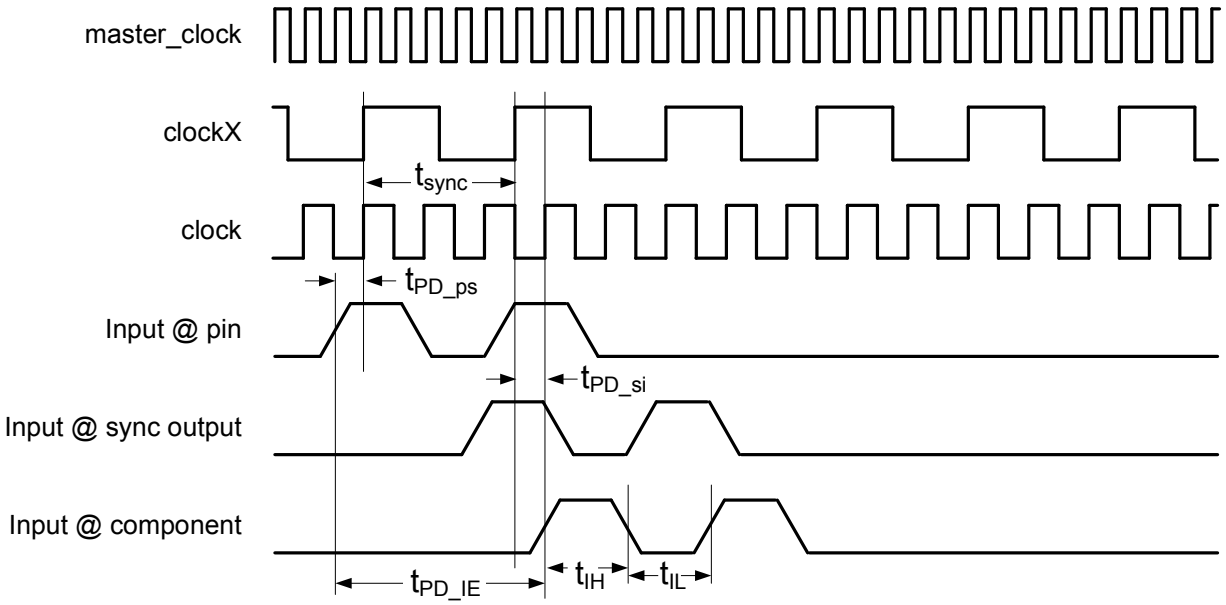
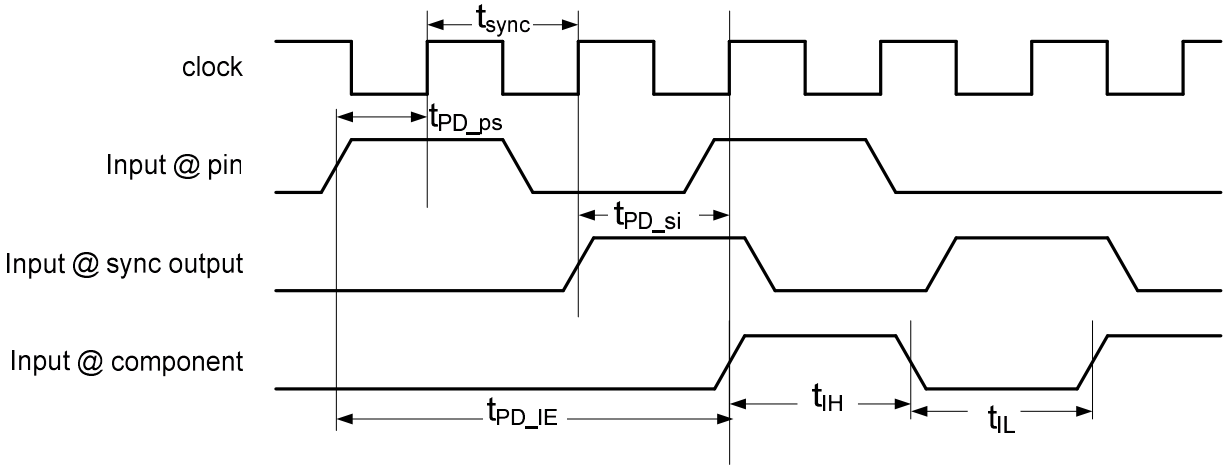


Figure 6. Input Configuration 1 and 2; Sync. Clock = Component Clock = master_clock



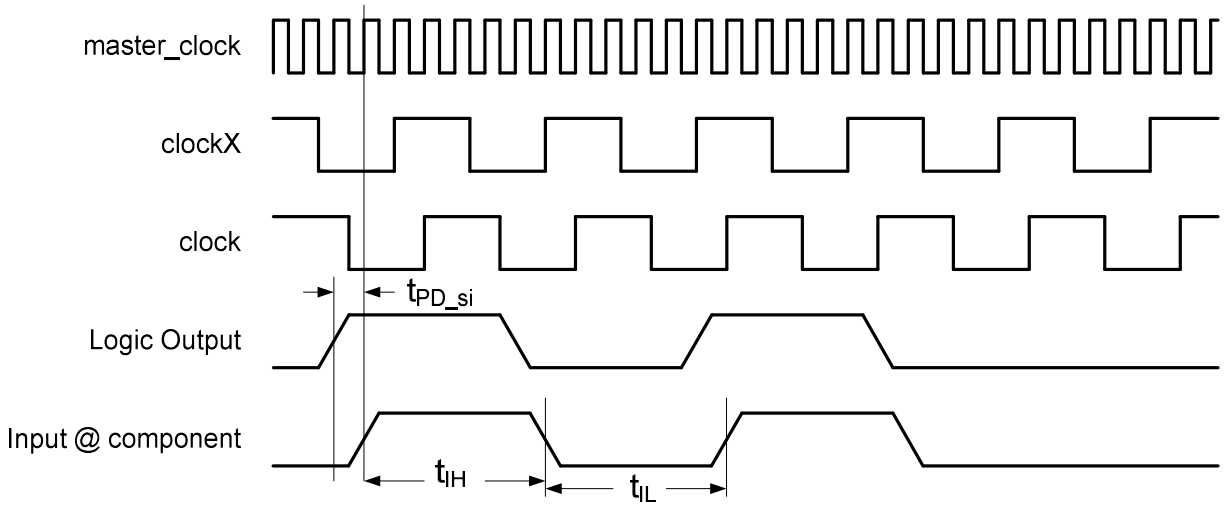
3. The input is driven by logic internal to the PSoC, which is synchronous based on a clock other than the clock the component uses (all internal clocks are derived from master_clock).

   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.

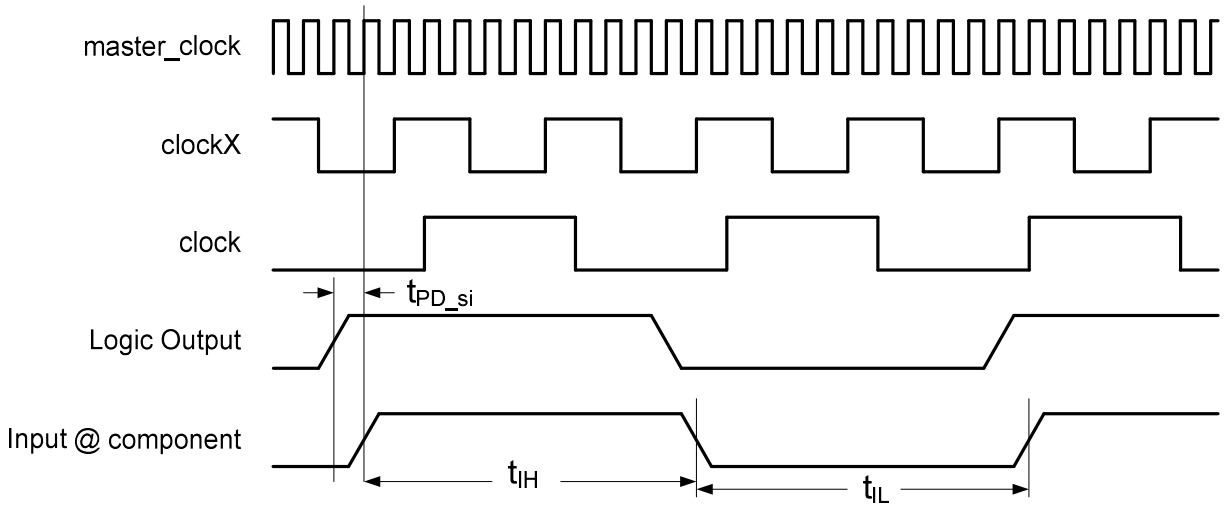
   When characterizing inputs configured in this way, the synchronizer clock is equal to the component clock. This produces the characterization parameters shown in Figure 11.

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, two clocks with the same frequency may 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 the system master_clock. 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

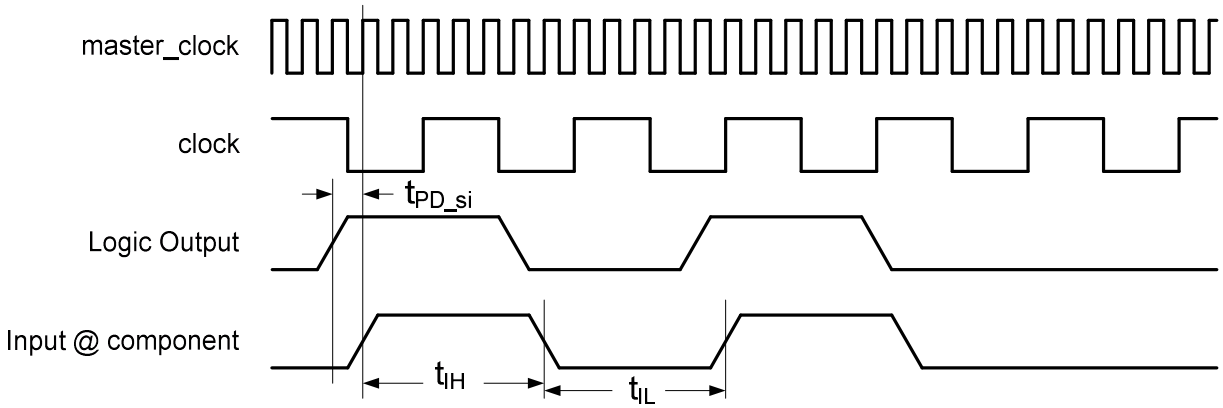
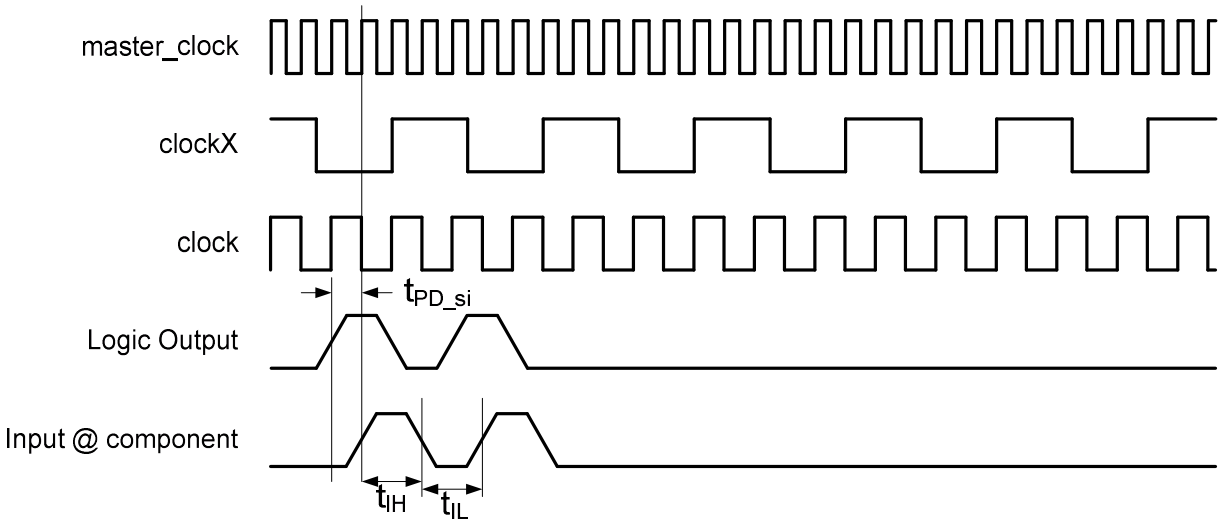
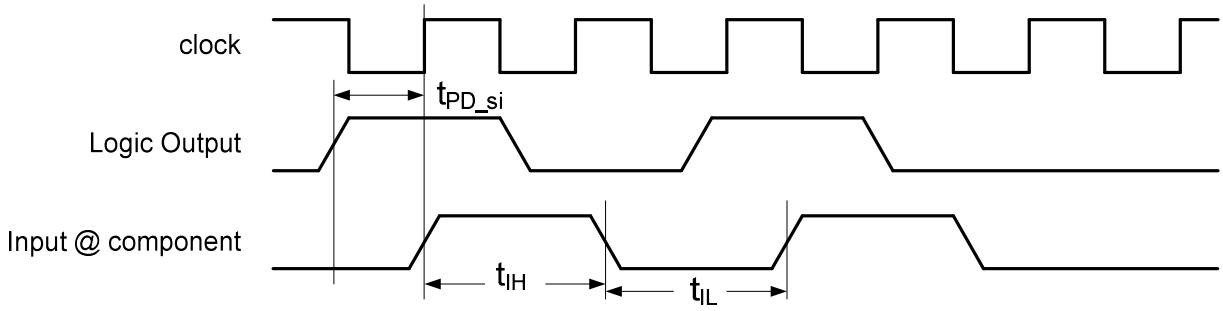


Figure 10. Input Configuration 3; Synchronizer Clock Frequency < Component Clock Frequency



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)_SYNC:synccell.syncd	BUS_CLK	13.246

t_{PD_si} is defined in the Register-to-register times. You 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 CharComp_clock

Destination clock CharComp_clock (Actual freq: 33.000 MHz)

+Source clock BUS_CLK

Source clock BUS_CLK (Actual freq: 66.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
input1(0)_SYNC:synccell.syncq	\CharComp:sC8:PRSdp:u0\:datapathcell.sync_ov_msb	15.799	63.295 MHz	66.000 MHz	SETUP

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 the device. In the first case, you must look at the Register-to-register times shown for the Logic-to-input descriptions (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.

+Clock to output times

+Clock to output times from clock CharComp_clock

Start	Register	End	Delay (ns)
CharComp_clock	\CharComp:sC8:PRSdp:u0\:datapathcell.regq a0	CharComp_bitstream(0):iocell.pad_out	27.253

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.d	Minor datasheet edit.	
2.0.c	Minor datasheet edit.	
2.0.b	Updated resource information in datasheet	
2.0.a	Added characterization data to datasheet	
	Minor datasheet edits and updates	
2.0	Added support for PSoC 3 Production silicon. Changes include: ▪ 4x clock for Time Division Multiplex Implementation added ▪ Single Cycle Implementation on 1x clock now available for 1 to 32 bits. ▪ Time Division Multiplex Implementation on 4x clock now available for 9 to 64 bits. ▪ Synchronous input signal Reset is added. ▪ Synchronous input signal Enable is added. ▪ Added new 'Advanced' page to the Configure dialog for the Implementation and Low Power Mode parameters.	New requirements to support the PSoC 3 Production device, thus a new 2.0 version of the PRS component was created.
	Added PRS_Sleep()/PRS_Wakeup() and PRS_Init()/PRS_Enable() APIs.	To support low-power modes, as well as to provide common interfaces to separate control of initialization and enabling of most components.

Version	Description of Changes	Reason for Changes / Impact
	Updated functions PRS_WriteSeed() and PRS_WriteSeedUpper().	The mask parameter was used to cut the seed value to define resolution while writing.
	Add reset DFF triggers to polynomial write functions: PRS_WritePolynomial(), PRS_WritePolynomialUpper() and PRS_WritePolynomialLower().	The DFF triggers must be set in proper state (most significant bit of polynomial, always 1) before starts calculation. To meet this condition any write to Seed or Polynomial register resets the DFF triggers.
	Updated Configure dialog to allow the Expression View for some parameters.	Expression View is used to directly access the symbol parameters. This view allows you to connect component parameters with external parameters, if desired.
	Updated Configure dialog to add error icons for various parameters.	If you enter an incorrect value in a text box, the error icon displays with a tool tip of the problem description. This provides easier use than a separate error message.

© Cypress Semiconductor Corporation, 2010-2013. 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.