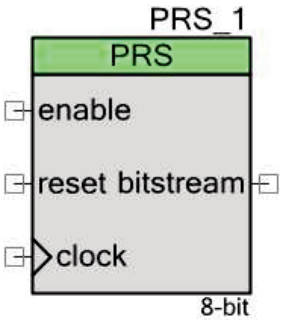
Pseudo Random Sequence (PRS)

2.20

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.

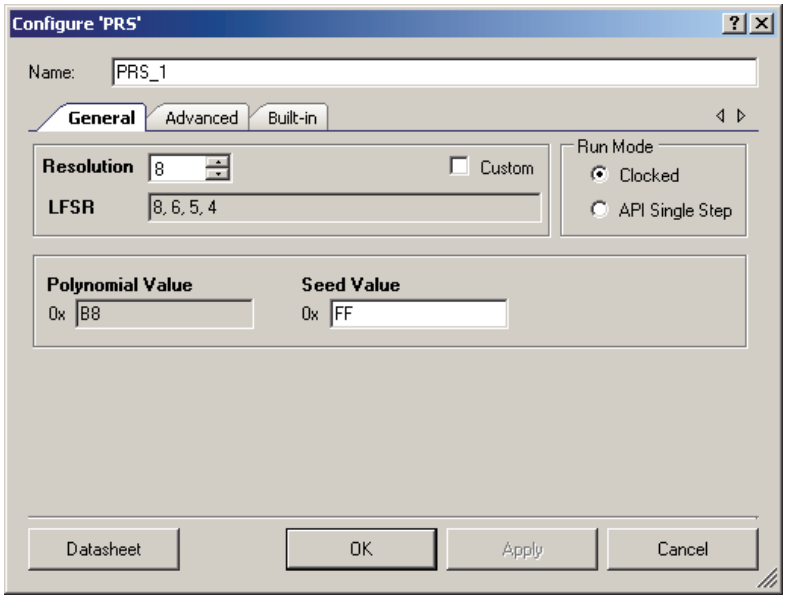
bitstream – Output

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.

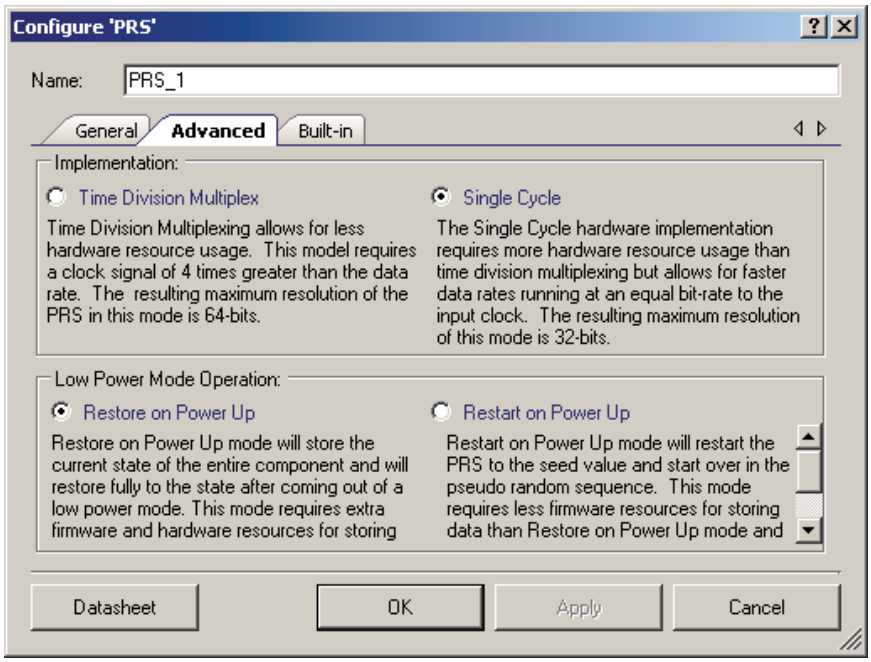
The Seed value, by default, is set to the maximum possible value ($2^{Resolution} - 1$). Its value can be changed to any other except 0. The Seed value is shown in hexadecimal form.

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.

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.

Function	Description
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.
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

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The PRS component has not been verified for MISRA-C:2004 coding guidelines compliance.

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

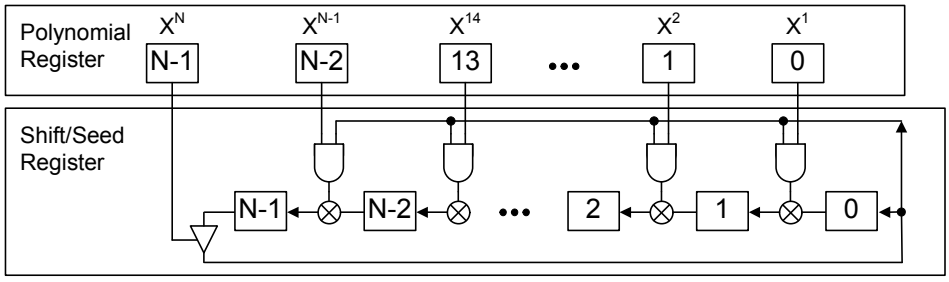
In this mode, the PRS component runs continuously after it starts and as long as the Enable input is held high.

PRS Run Mode: API Single Step

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

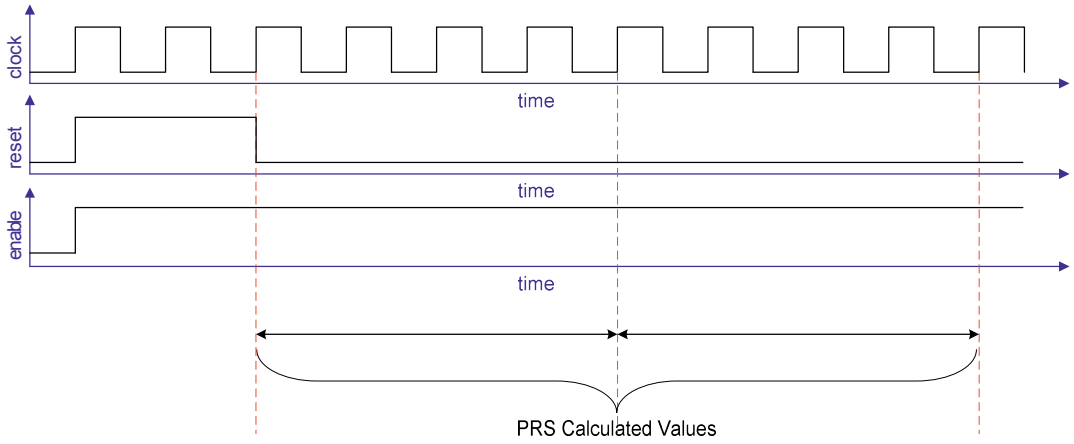
Block Diagram and Configuration

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

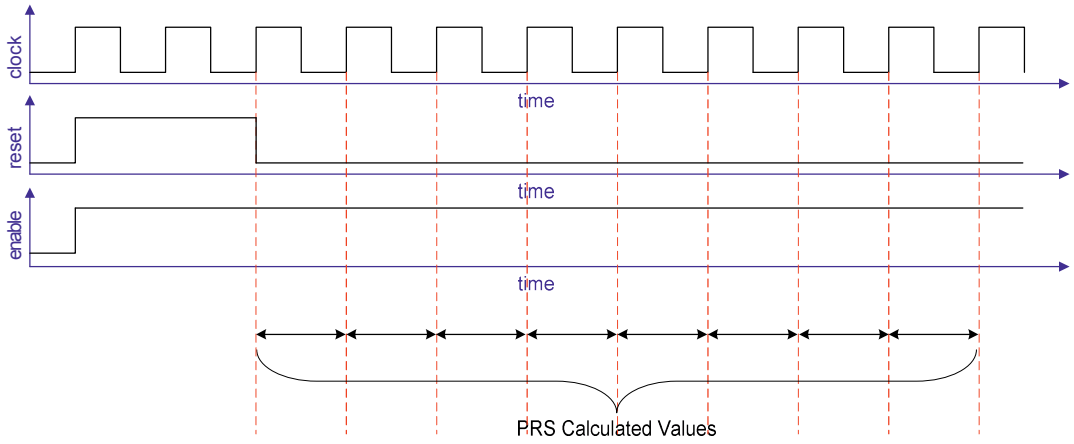


Timing Diagrams

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().

Resources

The PRS component is placed throughout the UDB array. The component utilizes the following resources.

Configuration	Resource Type					
	Datapath Cells	Macrocells[1]	Status Cells	Control Cells	DMA Channels	Interrupts
8-Bits Single Cycle	1	1	–	1	–	–
16-Bits Single Cycle	2	1	–	1	–	–
24-Bits Single Cycle	3	1	–	1	–	–
32-Bits Single Cycle	4	1	–	1	–	–
16-Bits Time Division	1	9	1	1	–	–
24-Bits Time Division	2	10	1	1	–	–
32-Bits Time Division	2	9	1	1	–	–
40-Bits Time Division	3	10	1	1	–	–
48-Bits Time Division	3	9	1	1	–	–
56-Bits Time Division	4	10	1	1	–	–
64-Bits Time Division	4	9	1	1	–	–

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

1. Additional macrocell is used for Single Cycle Implementation with API Single Step Run Mode.

Configuration	PSoC 3 (Keil_PK51)		PSoC 5 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
8-Bits Single Cycle	170	3	300	5	284	5
16-Bits Single Cycle	232	4	318	9	306	5
24-Bits Single Cycle	304	6	350	13	338	9
32-Bits Single Cycle	302	6	338	13	326	9
16-Bits Time Division	306	6	466	9	434	9
24-Bits Time Division	595	8	558	17	510	13
32-Bits Time Division	671	8	622	17	546	13
40-Bits Time Division	842	12	794	25	730	17
48-Bits Time Division	977	12	848	25	804	17
56-Bits Time Division	1083	12	904	25	880	17
64-Bits Time Division	1177	12	928	25	864	17

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Characteristics

Parameter	Description	Min	Typ[2]	Max	Units
I_{DD}	Component current consumption				
	8-Bits Single Cycle	–	11	–	µA/MHz
	16-Bits Single Cycle	–	17	–	µA/MHz
	24-Bits Single Cycle	–	23	–	µA/MHz
	32-Bits Single Cycle	–	31	–	µA/MHz
	16-Bits Time Division	–	22	–	µA/MHz
	24-Bits Time Division	–	30	–	µA/MHz
	32-Bits Time Division	–	31	–	µA/MHz
	40-Bits Time Division	–	40	–	µA/MHz

2. Device IO and clock distribution current not included. The values are at 25 °C.

Parameter	Description	Min	Typ[2]	Max	Units
	48-Bits Time Division	–	41	–	µA/MHz
	56-Bits Time Division	–	51	–	µA/MHz
	64-Bits Time Division	–	47	–	µA/MHz

AC Characteristics

Parameter	Description	Min	Typ	Max[3]	Units
f_{CLOCK}	Component clock frequency				
	8-Bits Single Cycle	–	–	39	MHz
	16-Bits Single Cycle	–	–	33	MHz
	24-Bits Single Cycle	–	–	30	MHz
	32-Bits Single Cycle	–	–	29	MHz
	16-Bits Time Division	–	–	29	MHz
	24-Bits Time Division	–	–	22	MHz
	32-Bits Time Division	–	–	28	MHz
	40-Bits Time Division	–	–	24	MHz
	48-Bits Time Division	–	–	28	MHz
	56-Bits Time Division	–	–	24	MHz
	64-Bits Time Division	–	–	26	MHz

3. The values provide a maximum safe operating frequency of the component. The component may run at higher clock frequencies, at which point you will need to validate the timing requirements with 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
2.20	Added MISRA Compliance section	The component was not verified for MISRA compliance
2.10	Added PSoC 5LP support	
	Added all 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.
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.
	Updated functions PRS_WriteSeed() and PRS_WriteSeedUpper().	The mask parameter was used to cut the seed value to define resolution while writing.

Version	Description of Changes	Reason for Changes / Impact
	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, 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.