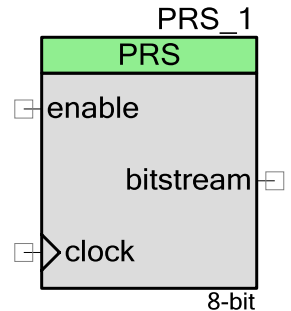


# Pseudo Random Sequence (PRS)

1.30

## Features

- 2 to 64-bit PRS sequence length
- 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 a LFSR to generate a pseudo random sequence, which outputs a pseudo random bitstream. The LFSR is of the Galois form (sometimes known as the modular form) and utilizes the provided maximal code length, or period. The PRS component runs continuously after started as long as Enable Input is held high. The PRS number generator may be started with any valid seed value excluding 0.

## When to use a PRS

LFSRs can be implemented in hardware, and this makes them useful in applications that require very fast generation of a pseudo-random sequence, such as 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.

## Uses as counters

The repeating sequence of states of an LFSR allows it to be used as a divider, or as a counter when a non-binary sequence is acceptable. LFSR counters have simpler feedback logic than natural binary counters or Gray code counters, and therefore can operate at higher clock rates. However it is necessary to ensure that the LFSR never enters an all-zeros state, for example by presetting it at start-up to any other state in the sequence.

**PRELIMINARY**

## Input/Output Connections

This section describes the various input and output connections for the PRS Component. An asterisk (\*) in the list of I/O's 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 PRS. This input is not available when the single step run mode is chosen.

### enable – Input

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

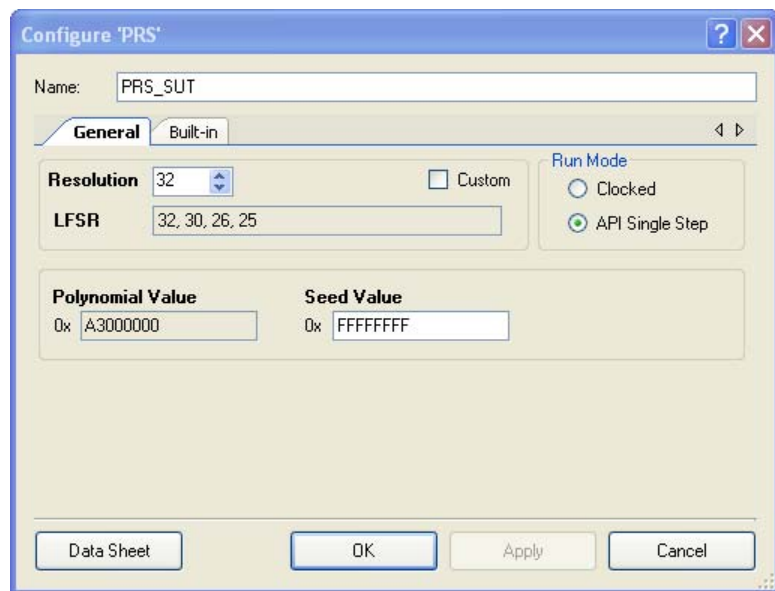
### bitstream – Output

Output of the LFSR.

## Parameters and Setup

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.

**Figure 1 Configure PRS Dialog**



The PRS dialog contains the following settings:

**PRELIMINARY**



## 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 ( $2^{\text{Resolution}} - 1$ )	Resolution	LFSR	Period ( $2^{\text{Resolution}} - 1$ )
2	2, 1	3	34	34, 31, 30, 26	17179869184
3	3, 2	7	35	35, 34, 28, 27	34359738368
4	4, 3	15	36	36, 35, 29, 28	68719476736
5	5, 4, 3, 2	31	37	37, 36, 33, 31	137438953472
6	6, 5, 3, 2	63	38	38, 37, 33, 32	274877906944
7	7, 6, 5, 4	127	39	39, 38, 35, 32	549755813888
8	8, 6, 5, 4	255	40	40, 37, 36, 35	1099511627776
9	9, 8, 6, 5	511	41	41, 40, 39, 38	2199023255552
10	10, 9, 7, 6	1023	42	42, 40, 37, 35	4398046511104
11	11, 10, 9, 7	2047	43	43, 42, 38, 37	8796093022208
12	12, 11, 8, 6	4095	44	44, 42, 39, 38	17592186044416
13	13, 12, 10, 9	8191	45	45, 44, 42, 41	35184372088832
14	14, 13, 11, 9	16383	46	46, 40, 39, 38	70368744177664
15	15, 14, 13, 11	32767	47	47, 46, 43, 42	140737488355328
16	16, 14, 13, 11	65535	48	48, 44, 41, 39	281474976710656
17	17, 16, 15, 14	131071	49	49, 45, 44, 43	562949953421312
18	18, 17, 16, 13	262143	50	50, 48, 47, 46	1125899906842624
19	19, 18, 17, 14	524187	51	51, 50, 48, 45	2251799813685248
20	20, 19, 16, 14	1048575	52	52, 51, 49, 46	4503599627370496
21	21, 20, 19, 16	2097151	53	53, 52, 51, 47	9007199254740992
22	22, 19, 18, 17	4194303	54	54, 51, 48, 46	18014398509481984
23	23, 22, 20, 18	8388607	55	55, 54, 53, 49	36028797018963968
24	24, 23, 21, 20	16777215	56	56, 54, 52, 49	72057594037927936
25	25, 24, 23, 22	33554431	57	57, 55, 54, 52	144115188075855872
26	26, 25, 24, 20	67108863	58	58, 57, 53, 52	288230376151711744
27	27, 26, 25, 22	134217727	59	59, 57, 55, 52	576460752303423488
28	28, 27, 24, 22	268435455	60	60, 58, 56, 55	1152921504606846976
29	29, 28, 27, 25	536870911	61	61, 60, 59, 56	2305843009213693952
30	30, 29, 26, 24	1073741823	62	62, 59, 57, 56	4611686018427387904
31	31, 30, 29, 28	2147483647	63	63, 62, 59, 58	9223372036854775808
32	32, 30, 26, 25	4294967295	64	64, 63, 61, 60	18446744073709551616
33	33, 32, 29, 27	8589934592			



PRELIMINARY

**To set LFSR coefficients manually:**

1. Define **Resolution**.
2. Check the **Custom** check box.
3. 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.

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

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

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

**Run Mode**

This parameter defines continuous or single step run modes. This parameter defines the component operation mode. Possible values include: "Clocked" (default) and "APISingleStep".

**Local Parameters (For API usage)**

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. 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. The default is 0x00h because the default Resolution is 8.
- **SeedValueLower(uint32)** – Contains the lower half of the seed value in hexadecimal. The default is 0xFFh because the default Resolution is 8.
- **SeedValueUpper(uint32)** – Contains the upper half of the seed value in hexadecimal. The default is 0 because the default Resolution is 8.

**Clock Selection**

There is no internal clock in this component. You must attach a clock source.

**Note** Generation of proper PRS sequence for Resolution, which is greater 8, requires two clock transitions.

**Placement**

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

**PRELIMINARY**



## Resources

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
8-Bits	1	?	0	1	0	?	?	?
16-Bits	1	?	0	1	0	?	?	?
24-Bits	2	?	0	1	0	?	?	?
32-Bits	2	?	0	1	0	?	?	?
40-Bits	3	?	0	1	0	?	?	?
48-Bits	3	?	0	1	0	?	?	?
56-Bits	4	?	0	1	0	?	?	?
64-Bits	4	?	0	1	0	?	?	?

## 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
void PRS_Start(void)	Initializes seed and polynomial registers. Computation of PRS starts on rising edge of input clock.
void PRS_Stop(void)	Stops PRS computation.
void PRS_Step(void)	Increments the PRS by one when API single step mode is used.
void PRS_WriteSeed(uint8/16/32 seed)	Writes seed value.
void PRS_WriteSeedUpper(uint32 seed)	Writes upper half of seed value. Only generated for 33-64-bit PRS.
void PRS_WriteSeedLower(uint32 seed)	Writes lower half of seed value. Only generated for 33-64-bit PRS.
uint8/16/32 PRS_Read(void)	Reads PRS value.



**PRELIMINARY**

Function	Description
uint32 PRS_ReadUpper(void)	Reads upper half of PRS value. Only generated for 33-64-bit PRS.
uint32 PRS_ReadLower(void)	Reads lower half of PRS value. Only generated for 33-64-bit PRS.
void PRS_WritePolynomial(uint8/16/32 polynomial)	Writes PRS polynomial value.
void PRS_WritePolynomialUpper(uint32 polynomial)	Writes upper half of PRS polynomial value. Only generated for 33-64-bit PRS.
void PRS_WritePolynomialLower(uint32 polynomial)	Writes lower half of PRS polynomial value. Only generated for 33-64-bit PRS.
uint8/16/32 PRS_ReadPolynomial(void)	Reads PRS polynomial value.
uint32 PRS_ReadPolynomialUpper(void)	Reads upper half of PRS polynomial value. Only generated for 33-64-bit PRS.
uint32 PRS_ReadPolynomialLower(void)	Reads lower half of PRS polynomial value. Only generated for 33-64-bit PRS.

### void PRS\_Start(void)

<b>Description:</b>	Initializes seed and polynomial registers. Computation of PRS starts on rising edge of input clock.
<b>Parameters:</b>	None
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

### void PRS\_Stop(void)

<b>Description:</b>	Stops PRS computation.
<b>Parameters:</b>	None
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

**PRELIMINARY**



**void PRS\_Step(void)**

<b>Description:</b>	Increments the PRS by one when API single step mode is used.
<b>Parameters:</b>	None
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

**void PRS\_WriteSeed(uint8/16/32 seed)**

<b>Description:</b>	Writes seed value.
<b>Parameters:</b>	(uint8/16/32) seed: Seed value.
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

**void PRS\_WriteSeedUpper(uint32 seed)**

<b>Description:</b>	Writes upper half of seed value. Only generated for 33-64-bit PRS.
<b>Parameters:</b>	(uint32) seed: Upper half of seed value.
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

**void PRS\_WriteSeedLower(uint32 seed)**

<b>Description:</b>	Writes lower half of seed value. Only generated for 33-64-bit PRS.
<b>Parameters:</b>	(uint32) seed: Lower half of seed value.
<b>Return Value:</b>	None
<b>Side Effects:</b>	None

**uint8/16/32 PRS\_Read(void)**

<b>Description:</b>	Reads PRS value
<b>Parameters:</b>	None
<b>Return Value:</b>	(uint8/16/32) Returns PRS value.
<b>Side Effects:</b>	None

**PRELIMINARY**

**uint32 PRS\_ReadUpper(void)**

**Description:** Reads upper half of PRS value. Only generated for 33-64-bit PRS.  
**Parameters:** None  
**Return Value:** (uint32) Returns upper half of PRS value.  
**Side Effects:** None

**uint32 PRS\_ReadLower(void)**

**Description:** Reads lower half of PRS value. Only generated for 33-64-bit PRS  
**Parameters:** None  
**Return Value:** (uint32) Returns lower half of PRS value.  
**Side Effects:** None

**void PRS\_WritePolynomial(uint8/16/32 polynomial)**

**Description:** Writes PRS polynomial value.  
**Parameters:** (uint8/16/32) polynomial: PRS polynomial.  
**Return Value:** None  
**Side Effects:** None

**void PRS\_WritePolynomialUpper(uint32 polynomial)**

**Description:** Writes upper half of PRS polynomial value. Only generated for 33-64-bit PRS.  
**Parameters:** (uint32) polynomial: Upper half PRS polynomial value.  
**Return Value:** None  
**Side Effects:** None

**void PRS\_WritePolynomialLower(uint32 polynomial)**

**Description:** Writes lower half of PRS polynomial value. Only generated for 33-64-bit PRS.  
**Parameters:** (uint32) polynomial: Lower half of PRS polynomial value.  
**Return Value:** None  
**Side Effects:** None

**PRELIMINARY**



**uint8/16/32 PRS\_ReadPolynomial(void)**

<b>Description:</b>	Reads PRS polynomial value.
<b>Parameters:</b>	None
<b>Return Value:</b>	(uint8/16/32) Returns PRS polynomial value.
<b>Side Effects:</b>	None

**uint32 PRS\_ReadPolynomialUpper(void)**

<b>Description:</b>	Reads upper half of PRS polynomial value. Only generated for 33-64-bit PRS.
<b>Parameters:</b>	None
<b>Return Value:</b>	(uint32) Returns upper half of PRS polynomial value.
<b>Side Effects:</b>	None

**uint32 PRS\_ReadPolynomialLower(void)**

<b>Description:</b>	Reads lower half of PRS polynomial value. Only generated for 33-64-bit PRS.
<b>Parameters:</b>	None
<b>Return Value:</b>	(uint32) Returns lower half of PRS polynomial value.
<b>Side Effects:</b>	None

## Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the PRS component. This example assumes the component has been placed in a design with the default name PRS\_1."

**Note** If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>

main()
{
    PRS_1_Start();
}
```



**PRELIMINARY**

## Functional Description

### PRS Run Mode: Clocked

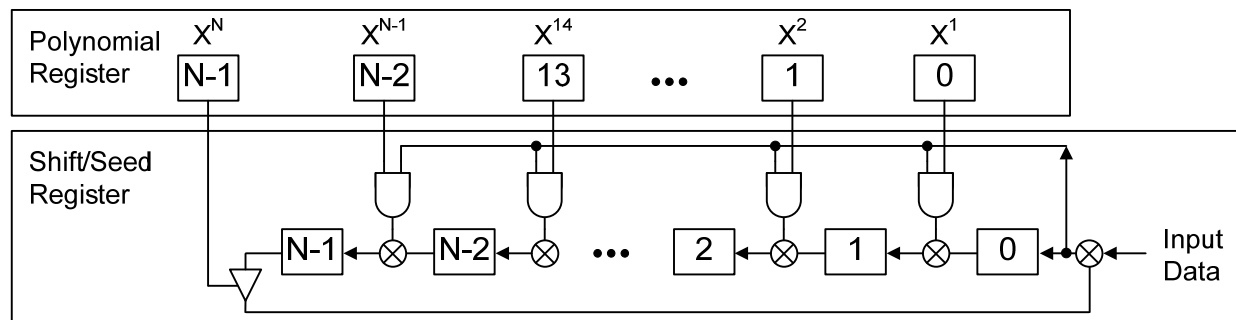
The PRS component runs continuously after it is started and as long as the Enable Input is held high.

### 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.



## Registers

### Polynomial register (from 2 to 64-bit based on Resolution)

The Polynomial register contains the polynomial value. It may be changed by the user with the `PRS_WritePolynomial()`, `PRS_WritePolynomialUpper()` or `PRS_WritePolynomialLower()` functions call. Also user may read current polynomial value using `PRS_ReadPolynomial()`, `PRS_ReadPolynomialUpper()` or `PRS_ReadPolynomialLower()`.

### Shift/Seed register (from 2 to 64-bit based on Resolution)

The Shift/Seed register contains the seed value. It may be changed by the user with the `PRS_WriteSeed()`, `PRS_WriteSeedUpper()` or `PRS_WriteSeedLower()` functions call. Also user may read current seed value using `PRS_ReadSeed()`, `PRS_ReadSeedUpper()` or `PRS_ReadSeedLower()`.

PRELIMINARY



## References

Not applicable

## DC and AC Electrical Characteristics

### 5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	$\Omega$	
Maximum Clock Rate	---		100	MHz	

## Component Changes

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

Version	Description of Changes
1.30	API customizer was removed. Refactored Verilog code. Updated schematic.

© Cypress Semiconductor Corporation, 2009. 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® Creator™, Programmable System-on-Chip™, and PSoC Express™ are trademarks and PSoC® is a registered trademark 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.

**PRELIMINARY**

