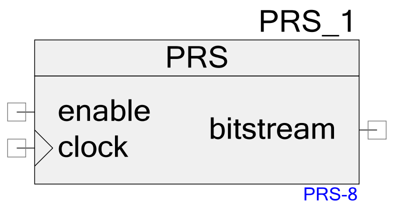
Pseudo Random Sequence (PRS)

0.5

Features

- 2 to 32-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 LFSR



General Description

The PRS component uses a Linear Feedback Shift Register (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 length codes. The PRS component runs continuously after started and as long as the Enable input is held high. The PRS pseudo random 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.

The Global Positioning System uses an LFSR to rapidly transmit a sequence that indicates high-precision relative time offsets. The Nintendo Entertainment System video game console also has an LFSR as part of its 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.

Uses in cryptography

LFSRs have long been used as pseudo-random number generators for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs. However, an LFSR is a linear system, leading to fairly easy cryptanalysis. For example, given a stretch of known plaintext and corresponding ciphertext, a stretch of LFSR output used in the system described above can be recovered, and from the output sequence one can construct an LFSR of minimal size by using the Berlekamp-Massey algorithm, which with the known output can be used to simulate the intended receiver to recover the remaining plaintext.

Three general methods are employed to reduce this problem in LFSR-based stream ciphers:

- Non-linear combination of several bits from the LFSR state;
- Non-linear combination of the outputs of two or more LFSRs; or
- Irregular clocking of the LFSR, as in the alternating step generator.

Important LFSR-based stream ciphers include A5/1 and A5/2, used in GSM cell phones, E0, used in Bluetooth, and the shrinking generator. The A5/2 cipher has been broken and both A5/1 and E0 have serious weaknesses.

Uses in digital broadcasting and communications

To prevent short repeating sequences (e.g., runs of 0's or 1's) from forming spectral lines that may complicate symbol tracking at the receiver or interfere with other transmissions, linear feedback registers are often used to "randomize" the transmitted bitstream. This randomization is removed at the receiver after demodulation. When the LFSR runs at the same rate as the transmitted symbol stream, this technique is referred to as scrambling. When the LFSR runs considerably faster than the symbol stream, expanding the bandwidth of the transmitted signal, this is direct-sequence spread spectrum.

Neither scheme should be confused with encryption or encipherment; scrambling and spreading with LFSRs do not protect the information from eavesdropping.

Digital broadcasting systems that use linear feedback registers:

- ATSC Standards (HDTV transmission system – North America)
- DAB (Digital audio broadcasting system -- for radio)
- DVB-T (HDTV transmission system – Europe, Australasia)
- NICAM (digital audio system for television)

Other digital communications systems using LFSRs:

- IBS (INTELSAT business service)
- IDR (Intermediate Data Rate service)
- SDI (Serial Digital Interface transmission)

- Data transfer over PSTN (according to the ITU-T V-series recommendations)
- CDMA (Code Division Multiple Access) cellular telephony
- 100BASE-T2 "fast" Ethernet scrambles bits using a LFSR
- 1000BASE-T Ethernet, the most common form of Gigabit Ethernet, scrambles bits using a LFSR

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.

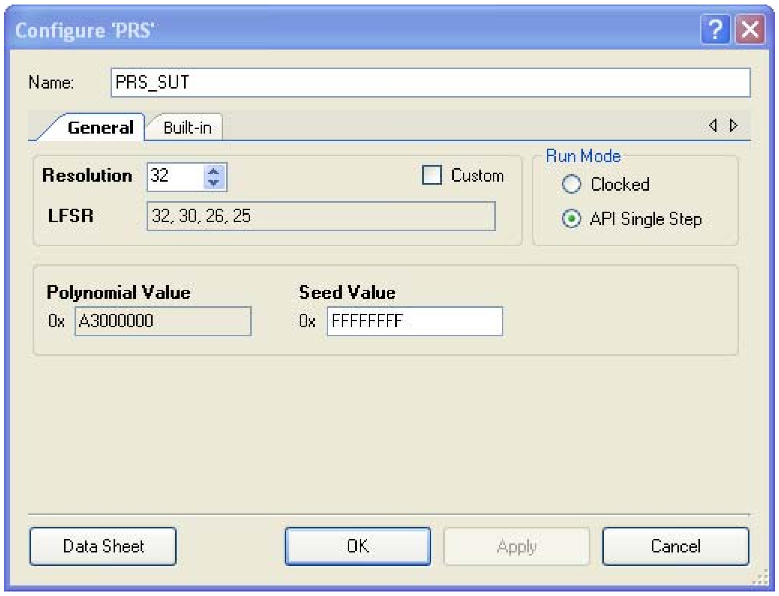
bitstream – Output

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



Configure 'PRS'

Name: PRS_SUT

Resolution: 32
[ ] Custom
LFSR: 32, 30, 26, 25

Run Mode
○ Clocked
[✓ API Single Step]

Polynomial Value: 0x A3000000
Seed Value: 0x FFFFFFFF

The PRS dialog contains the following settings:

Resolution

This defines the PRS sequence length. This value can be set from 2 to 32. The default is 8.

By default, Resolution defines LFSR coefficients and Polynomial Value. Coefficients are taken from the following table.

This parameters defines the maximal code length (period). The maximal code length is ($2^{Resolution}$ – 1). Possible value is 2 - 32 bits.

Resolution	LFSR	Period ($2^{Resolution}$-1)
Custom	User defined	$2^{Resolution}$-1
2	2, 1	3
3	3, 2	7
4	4, 3	15
5	5, 4, 3, 2	31
6	6, 5, 3, 2	63
7	7, 6, 5, 4	127
8	8, 6, 5, 4	255

PRELIMINARY

Resolution	LFSR	Period ($2^{Resolution}$-1)
9	9, 8, 6, 5	511
10	10, 9, 7, 6	1023
11	11, 10, 9, 7	2047
12	12, 11, 8, 6	4095
13	13, 12, 10, 9	8191
14	14, 13, 11, 9	16383
15	15, 14, 13, 11	32767
16	16, 14, 13, 11	65535
17	17, 16, 15, 14	131071
18	18, 17, 16, 13	262143
19	19, 18, 17, 14	524187
20	20, 19, 16, 14	1048575
21	21, 20, 19, 16	2097151
22	22, 19, 18, 17	4194303
23	23, 22, 20, 18	8388607
24	24, 23, 21, 20	16777215
25	25, 24, 23, 22	33554431
26	26, 25, 24, 20	67108863
27	27, 26, 25, 22	134217727
28	28, 27, 24, 22	268435455
29	29, 28, 27, 25	536870911
30	30, 29, 26, 24	1073741823
31	31, 30, 29, 28	2147483647
32	32, 30, 26, 25	4294967295

To set LFSR coefficients manually:

1. Define Resolution.
2. Check the Custom checkbox.
3. Enter coefficients separated by comma in the LFSR textbox and press Enter.
   The Polynomial value will be recalculated automatically.

Polynomial value is represented in the hexadecimal form.

Note Coefficients values could not be greater than Resolution.

Seed Value by default is set to the maximum possible value (2N-1, where N is Resolution). Its value can be changed to any other except 0. Seed value is represented in the hexadecimal form.

Note Change of Resolution causes the Seed value set to the default value.

Run Mode

This parameter defines continuous or single step run modes. This parameters defines the component operation mode. Possible value is “Clocked” (default) and “APISingleStep”.

Local Parameters (For API usage)

These parameters are used in the API and not exposed in the GUI; however, these are used in the APIs.

PolyValueLower (uint32)

Contains lower half of polynomial value in the hexadecimal form. The default is 0xB8h (LFSR= [8,6,5,4]).

PolyValueUpper (uint32)

Contain upper half of polynomial value in the hexadecimal form. The default is 0x00h.

SeedValueLower(uint32)

Contain lower half of seed value in the hexadecimal form. The default is 0xFFh.

SeedValueUpper(uint32)

Contain upper half of seed value in the hexadecimal form. The default is 0xFFh.

Clock Selection

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

Placement

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

Resources

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

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
64-Bits	8	?	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 riseing edge of input clock.
void PRS_Stop(void)	Stops PRS computation, PRS store in PRS register.
void PRS_ Step(void)	Increments the PRS by one when in API single step mode.
void PRS_WriteSeed(uint8/16/32 seed)	Writes the PRS Seed register with the start value.
void PRS_WriteSeedUpper(uint32 seed)	Writes the upper half of Seed register with the start value. Only generated for 33-64-bit PRS.
void PRS_WriteSeedLower(uint32 seed)	Writes the lower half of Seed register with the start value. Only generated for 33-64-bit PRS.
uint8/16/32 PRS_Read(void)	Reads the current PRS value.
uint32 PRS_ReadUpper(void)	Reads the current upper half of PRS value. Only generated for 33-64-bit PRS.
uint32 PRS_ReadLower(void)	Reads the current lower half of PRS value. Only generated for 33-64-bit PRS.
void PRS_ WritePolynomial(uint8/16/32 polynomial)	Writes the PRS polynomial.
void PRS_WritePolynomialUpper(uint32 polynomial)	Writes the upper half of PRS polynomial. Only generated for 33-64-bit PRS.
void PRS_WritePolynomialLower(uint32	Writes the lower half of PRS polynomial. Only generated for

PRELIMINARY

Function	Description
polynomial)	33-64-bit PRS.
uint8/16/32 PRS_ReadPolynomial(void)	Reads the PRS polynomial.
uint32 PRS_ReadPolynomialUpper(void)	Reads the upper half of PRS polynomial. Only generated for 33-64-bit PRS.
uint32 PRS_ReadPolynomialLower(void)	Reads the lower half of PRS polynomia. Only generated for 33-64-bit PRS.

void PRS_Start(void)

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

void PRS_Stop(void)

Description:	Stops PRS computation, PRS store in PRS register.
Parameters:	None
Return Value:	None
Side Effects:	None

void PRS_Step(void)

Description:	Increments the PRS by one when in API single step mode.
Parameters:	None
Return Value:	None
Side Effects:	None

void PRS_WriteSeed(uint8/16/32 seed)

Description:	Writes the PRS Seed register with the start value.
Parameters:	(uint8/16/32) seed: Seed register start value.
Return Value:	None
Side Effects:	None

void PRS_WriteSeedUpper(uint32 seed)

Description:	Writes the upper half of Seed register with the start value. Only generated for 33-64-bit PRS.
Parameters:	(uint32) seed: Upper half of Seed register start value.
Return Value:	None
Side Effects:	None

void PRS_WriteSeedLower(uint32 seed)

Description:	Writes the lower half of Seed register with the start value. Only generated for 33-64-bit PRS.
Parameters:	(uint32) seed: Lower half of Seed LSB register start value.
Return Value:	None
Side Effects:	None

uint8/16/32 PRS_Read(void)

Description:	Reads the current PRS value.
Parameters:	None
Return Value:	(uint8/16/32) Current PRS value.
Side Effects:	None

uint32 PRS_ReadUpper(void)

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

uint32 PRS_ReadLower(void)

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

void PRS_ WritePolynomial(uint8/16/32 polynomial)

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

void PRS_ WritePolynomialUpper(uint32 polynomial)

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

void PRS_ WritePolynomialLower(uint32 polynomial)

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

uint8/16/32 PRS_ReadPolynomial(void)

Description:	Reads the PRS polynomial.
Parameters:	None
Return Value:	(uint8/16/32) PRS polynomial.
Side Effects:	None

uint32 PRS_ReadPolynomialUpper(void)

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

uint32 PRS_ReadPolynomialLower(void)

Description:	Reads the lower half of PRS polynomial. Only generated for 33-64-bit PRS.
Parameters:	None
Return Value:	(uint32) Lower half of PRS polynomial.
Side Effects:	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>
#include "PRS_1.h"

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

Functional Description

PRS Run Mode: Clocked

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

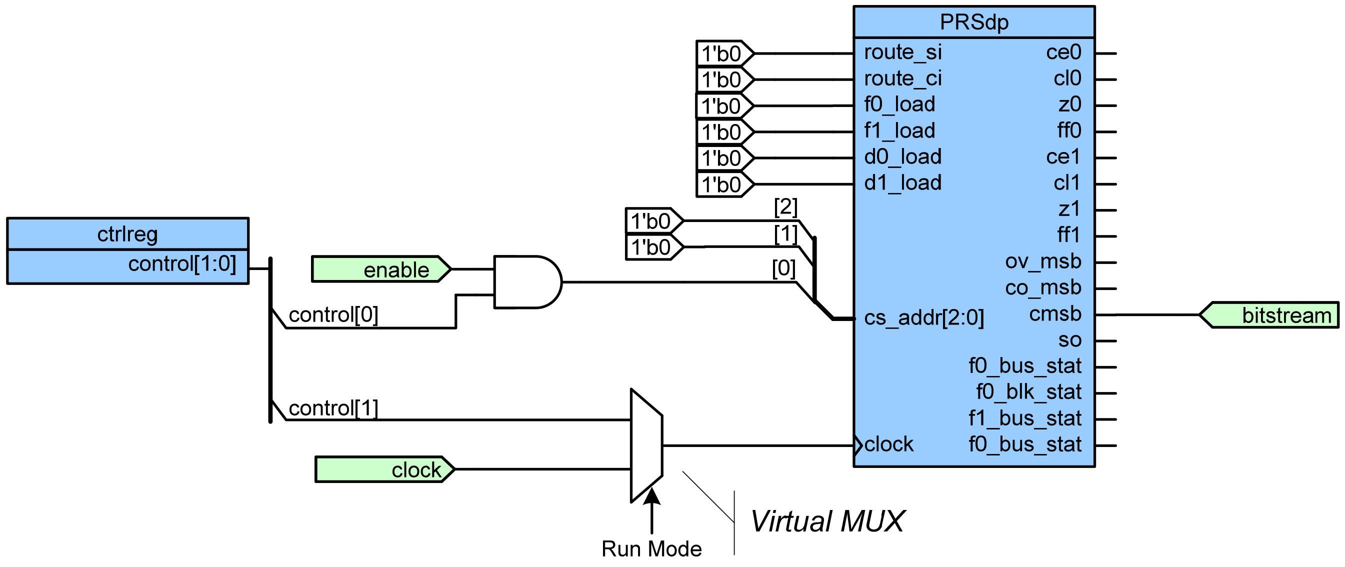
PRS Run Mode: API Single Step

In this mode PRS is incremented by API means.

Block Diagram and Configuration

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

The implementation is described in the following block diagram.



Registers

Not applicable

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	---		---	Ω	

PRELIMINARY

Parameter	Typical	Min	Max	Units	Conditions and Notes
Maximum Clock Rate	---		100	MHz	