Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

32-Bit Pseudo Random Sequence Generator Datasheet PRS32 V 3.4

Copyright © 2000-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA140, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8C28x43, CY8C28x52						
32-bit	4	0	0	87	0	1

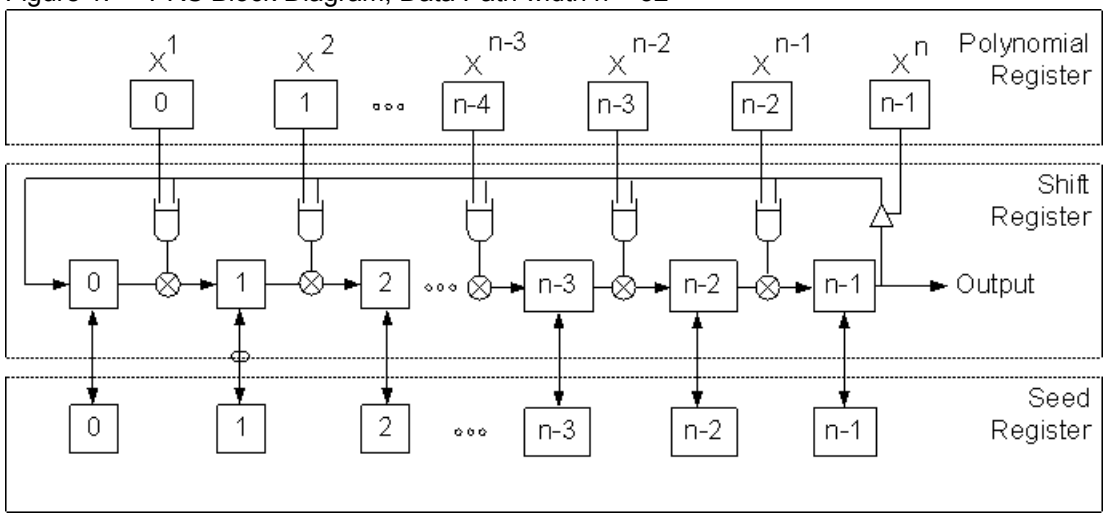
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 2- to 32-bit general purpose pseudo-random number generator uses two PSoC blocks
- Data input clocking up to 48 MHz
- Programmable polynomial and seed values
- Serial output bit stream
- Synchronization pulse output on selected value
- Computed pseudo-random number can be read directly from the linear feedback shift register (LFSR)

The PRS User Module is a modular LFSR that generates a pseudo random bit stream. The polynomial and starting seed values can be specified to define its output number sequence.

Figure 1. PRS Block Diagram, Data Path width n = 32



Functional Description

The PRS32 User Module employs two digital PSoC blocks. It implements a modular 2- to 24-bit LFSR that generates a pseudo-random bit stream. The modular form LFSR has an XOR between the output of each bit and the input of the following bit. The polynomial value gates the shift register output to the XOR at the following bit.

Polynomial, Shift, Seed, and Control registers refer to the combined registers of the two blocks, PRS32_MSB for most significant byte, PRS32_ISB2 and PRS32_ISB1 for the intermediate bytes, and PRS32_LSB for the least significant byte. Operation of the PSR32 is controlled by four registers per block.

The Polynomial register holds the polynomial that defines the length of the LFSR output bit sequence. The Shift register computes the LFSR function. The Seed register sets the sequence starting point and provides a compare value for synchronization. The Control register contains the start bit.

The Seed and Polynomial registers must be initialized before setting the start bit in the PRS's Control register. Writing the seed value into the Seed register while the PRS start bit is not set causes the seed value to be latched into the Shift register, initializing the starting data. Writing the seed value after the PRS has been started does not change the sequence, but it changes the synchronization value.

The following table lists the sequence length, taps, and polynomial value for each LFSR length from 2 to 32 bits. The maximum length bit sequence is not unique. There may be more than one polynomial that achieves the maximum length sequence. These polynomials have been verified to produce maximum length sequences.

Table 1. 32-bit Modular LFSR Polynomials

Bits	Sequence Length	Feedback Taps	16-bit Polynomial
2	3	2,1	0x0003
3	7	3,2	0x0006
4	15	4,3	0x000C
5	31	5,4,3,2	0x001E
6	63	6,5,3,2	0x0036
7	127	7,6,5,4	0x0078
8	255	8,6,5,4	0x00B8
9	511	9,8,6,5	0x01B0
10	1,023	10,9,7,6	0x0360
11	2,047	11,10,9,7	0x0740
12	4,095	12,11,8,6	0x0CA0
13	8,191	13,12,10,9	0x1B00
14	16,383	14,13,11,9	0x3500
15	32,767	15,14,13,11	0x7400
16	65,535	16,14,13,11	0xB400
17	131,071	17,16,15,14	0x1E000
18	262,143	18,17,16,14	0x3A000

Bits	Sequence Length	Feedback Taps	16-bit Polynomial
19	524,287	19,18,17,14	0x72000
20	1,048,575	20,19,16,14	0xCA000
21	2,097,151	21,20,19,16	0x1C8000
22	4,194,303	22,19,18,17	0x270000
23	8,388,607	23,22,20,18	0x6A0000
24	16,777,215	24,23,21,20	0xD80000
25	31,457,280	25,24,23,22	0x1E00000
26	59,244,544	26,25,24,20	0x3880000
27	119,537,664	27,26,25,22	0x7200000
28	211,812,352	28,27,24,22	0xCA00000
29	486,539,264	29,28,27,25	0x1D000000
30	847,249,408	30,39,26,24	0x32800000
31	2,013,265,920	31,30,29,28	0x78000000
32	2,734,686,208	32,20,26,25	0xA3000000

The maximum sequence code length for an N-bit LFSR is 2^{n-1}. Zero is the missing value, as this results in a terminal condition.

When the seed value and polynomial are initialized, the PRS32 User Module is started and a rising edge of the input clock generates the next state in the specified pseudo-random sequence. Tap bit N is output to the specified output as a bit stream synchronous with the clock.

The starting seed value must be set to a value between 1 and 2^{n-1}. If the seed value is set larger than 2^{n-1} (for example, 0x2fff for a 12-bit LFSR), the PRS32 will start, but will not generate a synchronization signal on the Compare output.

The PRS32 may be read in order to generate a random number to be used as part of a system process. Reading the computed pseudo-random number is a multi-step process.

1. Stop the PRS32 User Module before reading the pseudo-random value. This guarantees that the LFSR is not inadvertently clocked while reading the data.
2. Read the Shift register. This causes the Shift register to be latched into the Seed register.
3. Read the random number result is read from the Seed register. Note that this changes the value of the Seed register. If a specific starting value is required for synchronization, the Seed register should be rewritten before restarting.
4. Start the PRS32 User Module.

DC and AC Electrical Characteristics

Table 2. PRS AC Electrical Characteristics for the CY8C29/27/24/22/21xxx Device Family

Parameter	Typical	Limit	Units	Conditions and Notes
Maximum input frequency	--	48[1]	MHz	Vdd = 5.0V[2]
Maximum output frequency	--	24[1]	MHz	Vdd = 5.0V and 48 MHz input clock
	--	12[3]	MHz	Vdd = 3.3V and 24 MHz input clock

Electrical Characteristics Notes

- If the output or clock input is routed through the global buses, then the frequency is limited to a maximum of 12 MHz.
- Provided enable signal is always high; otherwise, the limit is 24 MHz.
- Fastest clock available to PSoC blocks is 24 MHz at 3.3-V operation.

Placement

The PRS32 uses four digital PSoC blocks. They are placed consecutively by the Device Editor in order of increasing block number from least-significant byte (LSB) to most significant (MSB). Each block is given a symbolic name displayed by the device editor during and after placement. The API qualifies all register names with user assigned instance name and block name to provide direct access to the PRS registers through the API include files. The block names used by the various widths are given in the following table.

PSoC Blocks	32-Bit PRS
1	PRS32_LSB
2	PRS32_ISB1
3	PRS32_ISB2
4	PRS32_MSB

Parameters and Resources

Clock

The PRS User Module is clocked by one of 16 possible sources. The 48-MHz clock, the CPU_32-kHz clock, one of the divided clocks (24V1 or 24V2), or another PSoC block output can be specified as the clock input. The Global I/O buses may be used to connect the clock input to an external pin or a clock function generated by a different PSoC block. When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy and sleep operation.

OutputBitStream

The output may be disabled or routed through row connections to one of 16 Global Output buses.

CompareType

Each cycle of the PRS compares the value of the Shift register to the value of the Seed register. This parameter sets the type of compare function to be performed.

Parameter	Description
Equal	Output goes high when Shift register equals seed value.
Less Than or Equal	Output goes high when Shift register is less than or equal to seed value.
Less Than	Output goes high when Shift register is less than seed value.

The normal usage is to set CompareType to Equal. This yields a single synchronization pulse at the Compare Output. Other settings will yield multiple output trigger values synced at different points in the sequence.

CompareOut

The result of the compare operation (see the CompareType parameter, above) produces and active-high The DAC Output may be routed to one of four Global Output buses or disabled.

ClockSync

In the PSoC devices, digital blocks may provide clock sources in addition to the system clocks. Digital clock sources may even be chained in ripple fashion. This introduces skew with respect to the system clocks. These skews are more critical in the CY8C29/27/24/22/21xxx PSoC device families because of various data-path optimizations, particularly those applied to the system busses. This parameter may be used to control clock skew and ensure proper operation when reading and writing PSoC block register values. Appropriate values for this parameter must be determined from the following table.

ClockSync Value	Use
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32KHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources must also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based clock unless the resulting frequency is 48 MHz (in other words, when the product of all divisors is 1).
Use SysClk Direct	Use when a 24 MHz (SysClk/1) clock is desired. This does not actually perform synchronization but provides low-skew access to the system clock itself. If selected, this option overrides the setting of the Clock parameter, above. It must always be used instead of VC1, VC2, VC3 or Digital Blocks where the net result of all dividers in combination produces a 24 Mhz output.
Unsynchronized	Use when the 48 MHz (SysClk*2) input is selected. Use when unsynchronized inputs are desired. In general this use is advisable only when interrupt generation is the sole application of the Counter.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include” files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

PRS32_Start

Description:

Enables the PRS32 User Module for operation. Before the module is started, the polynomial and seed values must be initialized.

C Prototype:

```
void  PRS32_Start(void)
```

Assembler:

```
lcall  PRS32_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

PRS32_Stop

Description:

Disables the PRS32 User Module.

C Prototype:

```
void  PRS32_Stop(void)
```

Assembler:

```
lcall  PRS32_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

Writing the seed value into the Seed register latches the seed value into the Shift register. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

PRS32_WriteSeed

Description:

Stops the user module by writing to the PRS32_Control register. Loads the PRS32 Seed register with an initial seed value. Upon completion of write operation, restores the PRS32 User Module to the previous state. While the PRS32 is running, a seed value written to the Seed register is not latched into the Shift register.

C Prototype:

```
void  PRS32_WriteSeed(DWORD dwSeed)
```

Assembler:

```
mov   X, dwSeed
lcall  PRS32_WriteSeed
```

Parameters:

dwSeed: 32-bit seed value. The X register points to dwSeed. Load address of the dwSeed into the X register.

Return Value:

None

Side Effects:

While the PRS32 is running, a seed value written to the Seed register is not latched into the Shift register. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

PRS32_WritePolynomial

Description:

Stops the user module by writing to the PRS32_Control register. Loads the Polynomial register with the LFSR function polynomial. Upon completion of write operation, restores the PRS32 User Module to the previous state.

C Prototype:

```
void  PRS32_WritePolynomial(DWORD dwPolynomial)
```

Assembler:

```
mov   X, dwPolynomial
lcall  PRS32_WritePolynomial
```

Parameters:

dwPolynomial: 32-bit polynomial value. See the PRS User Module description section for a discussion on how to set the polynomial value. The X register points to dwPolynomial. Load address of the dwSeed into the X register.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

PRS32_wReadPRS

Description:

Stops the user module by writing to the PRS32_Control register. The computed LFSR value in the Shift register is written to the Seed register. The output is then read from the Seed register. This overwrites the existing seed value. If a specific seed value is required for waveform synchronization, the desired seed value must be re-written using PRS32_WriteSeed. The user module must be restarted using the PRS32_Start API.

C Prototype:

```
void  PRS32_ReadPRS(DWORD * pdwPRSValue)
```

Assembler:

```
mov A, >pdwPRSValue
mov X, <pdwPRSValue
lcall  PRS32_ReadPRS
```

Parameters:

pdwPRSValue: Pointer to the buffer to hold the PRS generated value. The X register is loaded with the address of the return buffer.

Return Value:

PRS generated value is returned in the pdwPRSValue buffer pointed to by the X register.

Side Effects:

Overwrites the existing seed value. If a specific seed value is required for waveform synchronization, the desired seed value must be re-written using PRS32_WriteSeed. The user module must be restarted using the PRS32_Start API. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Sample Firmware Source Code

In the following examples, the correspondence between the C and assembly code is simple and direct. The values shown for period and compare value are each “off-by-1” from the cardinal values because the registers are zero-based; that is, zero is the terminal count in their down-count cycle. Passing a simple one byte parameter in the A register rather than on the stack is a performance optimization used by both the assembler and C compiler for user module APIs. The C compiler employs this mechanism for “INT”

types instead of pushing the argument on the stack when it sees the #pragma fastcall declarations in the PRS32.h file.

The following is assembly language source that illustrates the use of the APIs.

```
;*****************************************************************************
;
;  Setup the PRS32 to generate a 32-bit maximal sequence.
;
;*****************************************************************************
include  "PRS32.inc"
export   SetupPRS32Bit

dwPOLY:  equ    E0000200h        ; Modular Polynomial = [32,31,30,10]
dwSEED:  equ    FFFFFFFFh        ; Seed value - all bits set

SetupPRS32Bit:
    ; load the PRS polynomial
    mov   X, SP
    add   SP, 4
    mov   [X],   (dwPOLY >> 24) & ffh       // MSB
    mov   [X+1], (dwPOLY >> 16) & ffh       // ISB2
    mov   [X+2], (dwPOLY >> 8 ) & ffh       // ISB1
    mov   [X+3], (dwPOLY & ffh)             // LSB
    call  PRS32_WritePolynomial
    add   SP, -4

    ; load the PRS seed
    mov   X, SP
    add   SP, 4
    mov   [X],   (dwSEED >> 24) & ffh
    mov   [X+1], (dwSEED >> 16) & ffh
    mov   [X+2], (dwSEED >> 8 ) & ffh
    mov   [X+3], (dwSEED & ffh)
    call  PRS32_WriteSeed
    add   SP, -4

    ;start the PRS32
    call  PRS32_Start

    ret
```

The same code in C is as follows.

```
#include "PRS32.h"

#define  dwPOLY    0xE0000200    // Modular Polynomial = [32,31,30,10]
#define  dwSEED    0xFFFFFFFF    // Seed value

void SetupPRS32Bit(void)
{
    // load the PRS polynomial
    PRS32_WritePolynomial(dwPOLY);

    // load the PRS seed
```

```
    PRS32_WriteSeed(dwSEED);

// start the PRS32
    PRS32_Start();
}
```

Configuration Registers

Except where noted, the register specifications given in this section apply to all PSoC device families.

The 32-bit PRS uses four digital PSoC blocks. In placement order from left to right they are named PRS32_LSB, PRS32_ISB1, PRS32_ISB2 and PRS32_MSB. Each block is personalized and parameterized through 7 registers. The following tables give the “personality” values as constants and the parameters as named bit-fields with brief descriptions. Symbolic names for these registers are defined in the user module instance’s C and assembly language interface files (the “.h” and “.inc” files).

Table 3. Function Register, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	0	0	0	1	0
ISB2	0	0	0	0	0	0	1	0
ISB1	0	0	0	0	0	0	1	0
LSB	0	0	0	0	0	0	1	0

Table 4. Input Register, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	Clock			
ISB2	0	0	0	0	Clock			
ISB1	0	0	0	0	Clock			
LSB	0	0	0	0	Clock			

Clock selects the input clock from one of 16 sources. This parameter is set in the Device Editor.

Table 5. Output Register, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	OutputBitStream		
ISB2	0	0	0	0	0	0	0	0
ISB1	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	0

OutputBitStream selects output from one of four global busses. This parameter is set in the Device Editor.

Table 6. Shift Register (DR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Shift Register(MSB)							
ISB2	Shift Register(ISB2)							
ISB1	Shift Register(ISB2)							
LSB	Shift Register(LSB)							

Shift Register is the PRS32 Shift register MSB, ISB2, ISB1 and LSB. It is read and configured using the PRS32 API.

Table 7. Polynomial Register (DR1), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Polynomial Register(MSB)							
ISB2	Polynomial Register(ISB2)							
ISB1	Polynomial Register(ISB1)							
LSB	Polynomial Register(LSB)							

Polynomial Register is the PRS32 Polynomial register MSB, ISB2, ISB1 and LSB. It is modified using the PRS32 API.

Table 8. Seed Register (DR2), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Seed Register(MSB)							
ISB2	Seed Register(ISB2)							
ISB1	Seed Register(ISB1)							
LSB	Seed Register(LSB)							

Seed Register is the PRS32 Seed register MSB, ISB2, ISB1 and LSB. It is modified using the PRS32 API.

Table 9. Control Register (CR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
ISB2	0	0	0	0	0	0	0	0
ISB1	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the PRS32 is enabled when set. It is modified using the PRS32 API.

Version History

Version	Originator	Description
3.4	DHA	Added Version History

Note PSoC Designer 5.1 introduces a Version History in all user Module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2000-2015 Cypress Semiconductor Corporation. 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 Designer™ and Programmable System-on-Chip™ 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.