16-Bit Pseudo Random Sequence Generator Datasheet PRS16 V 3.4

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, CY8CTMA140, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8C28x43, CY8C28x52						
16-bit	2	0	0	54	0	1

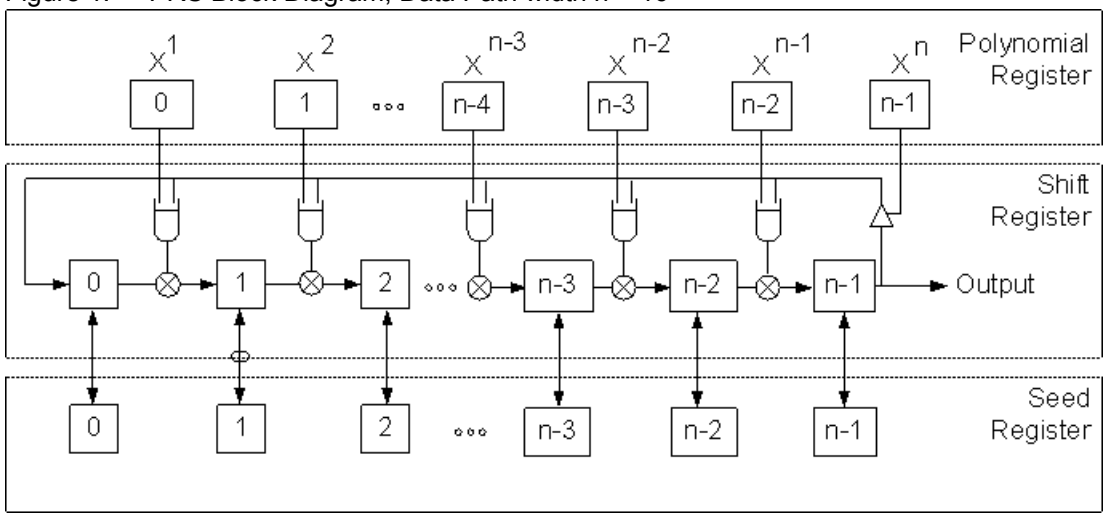
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 2- to 16-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 = 16



Functional Description

The PRS16 User Module employs two digital PSoC blocks. It implements a modular 2- to 16-bit linear feedback shift register (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, PRS16_MSB for most significant byte and PRS16_LSB for the least significant byte. Operation of the PSR16 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 user module'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 starts 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 16 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 tested and verified to produce maximum length sequences.

Table 1. 16-Bit Modular LFSR Polynomials

Bits	Sequence Length	Feedback Taps	16-bit Polynomial
2	3	2,1	0X003
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	8,4,5,4	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

The maximal sequence code length, for an N-bit LFSR pseudo random bit sequence generator, 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 PRS16 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, 0xff for a 12-bit LFSR), the PRS16 will start, but will not generate a synchronization signal on the Compare output.

The PRS16 may be read to generate a random number to be used as part of a system process. Follow these steps to read the computed pseudo-random number:

1. Stop the PRS16 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 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 PRS16 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 = 5.0 V[2]
Maximum output frequency	--	24[1]	MHz	VDD = 5.0 V and 48-MHz input clock
	--	12[3]	MHz	VDD = 3.3 V and 24-MHz input clock

Electrical Characteristics Notes

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

Placement

The PRS16 uses two 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 byte (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.

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 the 16 Global Output buses.

CompareType

Each cycle 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 an active-high. The Compare Output may be disabled or routed through the output row to one of the 16 Global Output buses.

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 a skew with respect to the system clocks. This parameter is 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), 32 kHz, 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 also ensure their code observes the policy. Though some user module API functions 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.

PRS16_Start

Description:

Enables the PRS16 User Module for operation. Before the module is started, the polynomial and seed values must be initialized.

C Prototype:

```
void  PRS16_Start(void)
```

Assembler:

```
lcall  PRS16_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.

PRS16_Stop

Description:

Disables the PRS16 User Module.

C Prototype:

```
void  PRS16_Stop(void)
```

Assembler:

```
lcall  PRS16_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.

PRS16_WriteSeed

Description:

Stops the user module by writing to the PRS16_Control register. Loads the PRS16 Seed register with an initial seed value. Upon completion of the write operation, restores the PRS16 User Module to the previous state. While the PRS16 is running, a seed value written to the Seed register is not latched into the Shift register.

C Prototype:

```
void  PRS16_WriteSeed(WORD wSeed)
```

Assembler:

```
mov   X, [wSeed]
mov   A, [wSeed+1]
lcall  PRS16_WriteSeed
```

Parameters:

wSeed: 16-bit seed value. MSB is passed in the X register and LSB is passed in the Accumulator.

Return Value:

None

Side Effects:

While the PRS16 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.

PRS16_WritePolynomial

Description:

Stops the user module by writing to the PRS16_Control register. Loads the Polynomial register with the LFSR function polynomial. Upon completion of the write operation, restores the PRS16 User Module to previous state.

C Prototype:

```
void   PRS16_WritePolynomial(WORD wPolynomial)
```

Assembler:

```
mov   X, [wPolynomial]
mov   A, [wPolynomial+1]
lcall  PRS16_WritePolynomial
```

Parameters:

wPolynomial: 16-bit polynomial value. See the PRS User Module description section for a discussion on how to set the polynomial value. MSB is passed in the X register and LSB is passed in the Accumulator.

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.

PRS16_wReadPRS

Description:

Stops the user module by writing to the PRS16_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 PRS16_WriteSeed. The user module must be restarted using the PRS16_Start API.

C Prototype:

```
WORD  PRS16_wReadPRS(void)
```

Assembler:

```
lcall  PRS16_wReadPRS
mov   [wPRSValue], X
mov   [wPRSValue+1], A
```

Parameters:

None

Return Value:

Value read from the Shift register. MSB is returned in the X registers and LSB is returned in the Accumulator.

Side Effects:

Overwrites the existing seed value. If a specific seed value is required for waveform synchronization, the desired seed value must be rewritten using PRS16_WriteSeed. The user module must be restarted using the PRS16_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 PRS16.h file.

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

```
;*****************************************************************************
;
;  Setup the PRS16 to generate a 12-bit maximal sequence.
;
;*****************************************************************************
include  "PRS16.inc"
export   SetupPRS12Bit

wPOLY:  equ    %110010100000      ; Modular Polynomial = [12,11,8,6]
wSEED:  equ    FFFFh              ; Seed value - all bits set

SetupPRS12Bit:
    mov   X, >wPOLY               ; load the PRS polynomial
    mov   A, <wPOLY
    call  PRS16_WritePolynomial

    mov   X, >wSEED               ; load the PRS seed
    mov   A, <wSEED
    call  PRS16_WriteSeed

    call PRS16_Start              ;start the PRS16
    ret
```

The same code in C is as follows.

```
#include "PRS16.h"
```

```
#define  wPOLY    0x0CA0        // Modular Polynomial = [12,11,8,6]
#define  wSEED    0xFFFF        // Seed value

void SetupPRS16Bit(void)
{
    // load the PRS polynomial
    PRS16_WritePolynomial(wPOLY);

    // load the PRS seed
    PRS16_WriteSeed(wSEED);

// start the PRS16
    PRS16_Start();
}
```

Configuration Registers

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

The 16-bit PRS uses two digital PSoC blocks. In placement order from left to right they are named PRS16_LSB and PRS16_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
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			
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		
LSB	0	0	0	0	0	0	0	0

OutputBitStream selects output to one of four row outputs. 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)							
LSB	Shift Register(LSB)							

Shift Register is the PRS16 Shift register MSB and LSB. It is read and configured using the PRS16 API.

Table 7. Polynomial Register (DR1), Bank 0

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

Polynomial Register is the PRS16 Polynomial register MSB and LSB. It is modified using the PRS16 API.

Table 8. Seed Register (DR2), Bank 0

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

Seed Register is the PRS16 Seed register MSB and LSB. It is modified using the PRS16 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
LSB	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the PRS16 is enabled when set. It is modified using the PRS16 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.