8-Bit Pseudo Random Sequence Generator Datasheet PRS8 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, CY8C21x45, CY8C22x45, CY8CTMA140, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8C28x43, CY8C28x52						
8-bit	1	0	0	41	0	1

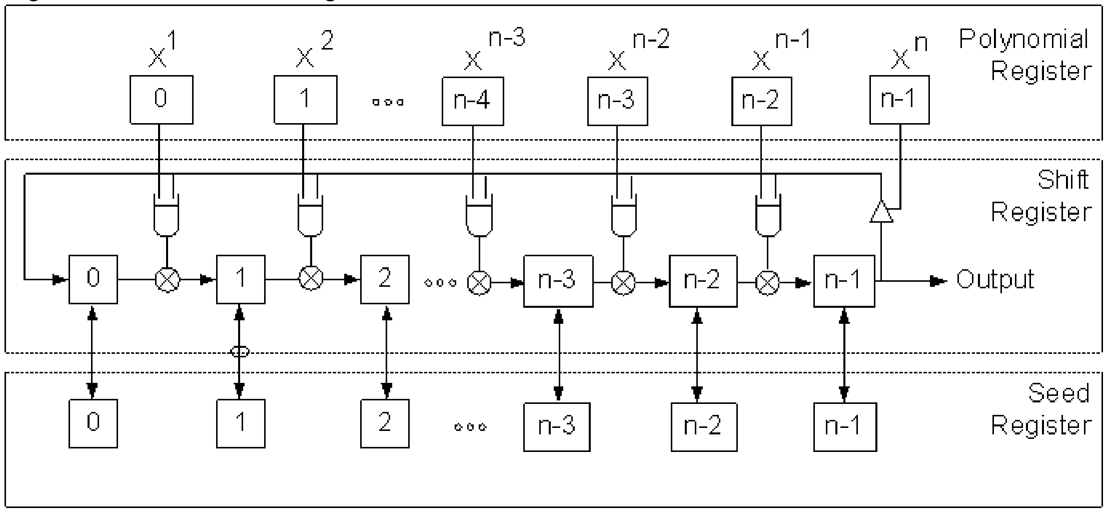
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 2 to 8-bit general purpose pseudo-random number generator uses one PSoC block
- Data input clocking up to 48 MHz
- Programmable polynomial and seed values
- Serial output bit stream
- 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 = 8



Functional Description

The PRS8 User Module employs one digital PSoC block. It implements a modular 2 to 8-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, PRS8_MSB for most significant byte and PRS8_LSB for the least significant byte. Operation of the PSR8 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 8 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. 8-bit Modular LFSR Polynomials

Bits	Sequence Length	Feedback Taps	8-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

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 PRS8 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, the PRS8 will start, but will not generate a synchronization signal on the Compare output.

The PRS8 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 PRS8 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. 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 PRS8 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 input or output 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 3.3V operation.

Placement

The PRS consumes one digital PSoC block. The block names used by the various widths are given in the following table.

PSoC Blocks	8-Bit PRS
1	PRS8

Parameters and Resources

Clock

The PRS8 User Module is clocked by one of 8 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 sixteen Global Output buses.

CompareType

During 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 and active-high The digital compare 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.

PRS8_Start

Description:

Enables the PRS8 User Module for operation. Before the module is started, the polynomial and seed values must be initialized.

C Prototype:

```
void  PRS8_Start(void)
```

Assembler:

```
lcall  PRS8_Start
```

Parameters:

None

Return Value:

None

Side Effects:

While the PRS8 is started, 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.

PRS8_Stop

Description:

Disables the PRS8 User Module.

C Prototype:

```
void  PRS8_Stop(void)
```

Assembler:

```
lcall  PRS8_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.

PRS8_WriteSeed

Description:

Initializes the PRS8 Shift register with an initial seed value. The user module is stopped while updating the Shift register with the new seed value. Upon exit, the start state is restored.

C Prototype:

```
void  PRS8_WriteSeed(BYTE bSeed)
```

Assembler:

```
mov   A, [bSeed]
lcall  PRS8_WriteSeed
```

Parameters:

bSeed: 8-bit seed value and is passed in the A 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.

PRS8_WritePolynomial

Description:

Loads the Polynomial register with the LFSR function polynomial. The PRS8 User Module is stopped while the Polynomial register is updated. Upon exit, the start state is restored.

C Prototype:

```
void  PRS8_WritePolynomial(BYTE bPolynomial)
```

Assembler:

```
mov   A, [bPolynomial]
lcall  PRS8_WritePolynomial
```

Parameters:

bPolynomial: 8-bit polynomial value. See the PRS User Module description section for a discussion on how to set the polynomial value. It is passed in the A 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.

PRS8_bReadPRS

Description:

Reads the computed LFSR resultant data. Calling this function while the PRS8 User Module is clocked gives inaccurate results.

C Prototype:

```
BYTE  PRS8_bReadPRS(void)
```

Assembler:

```
lcall  PRS8_bReadPRS
mov   [bPRSValue], A
```

Parameters:

None

Return Value:

Value read from the Shift register. It is returned in Accumulator.

Side Effects:

The Seed register is overwritten with the computed LFSR value. 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 PRS8.h file.

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

```
;**********************************************************************
;  Setup the PRS8 to generate an 8-bit maximal sequence.
;
;**********************************************************************
include "PRS8.inc"
export  SetupPRS255

bPOLY_255:  equ    %10111000 ; Modular Polynomial = [8,6,5,4]
bSEED_255:  equ    %11111111 ; Seed value - all bits set

SetupPRS255:
    ; load the PRS polynomial
    mov   A, bPOLY_255
    call  PRS8_WritePolynomial

    ; load the PRS seed
    mov   A, bSEED_255
    call  PRS8_WriteSeed

    ;start the PRS8
    call  PRS8_Start

    ret
```

The same code in C is as follows.

```
#include "PRS8.h"

#define bPOLY 0xB8
#define bSEED 0xFF
void SetupPRS8Bit(void)
{
// load the PRS polynomial
PRS8_WritePolynomial(bPOLY);
// load the PRS seed
PRS8_WriteSeed(bSEED);
// start the PRS8
PRS8_Start();
}
```

Configuration Registers

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

The 8-bit PRS uses a single digital PSoC block named PRS8. 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
Value	0	0	1	0	1	0	1	0

Table 4. Block PRS8 Register: Input

Bit	7	6	5	4	3	2	1	0
Value	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. Block PRS8 Register: Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	OutputBitStream		

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

Table 6. Block PRS8 Shift Register: DR0

Bit	7	6	5	4	3	2	1	0
Value	Shift Register							

Shift Register is the PRS8 Shift register. It is read and configured using the PRS8 API.

Table 7. Block PRS8 Polynomial Register: DR1

Bit	7	6	5	4	3	2	1	0
Value	Polynomial Register							

Polynomial Register is the PRS8 Polynomial register. It is modified using the PRS8 API.

Table 8. Block PRS8 Seed Register: DR2

Bit	7	6	5	4	3	2	1	0
Value	Seed Register							

Seed Register is the PRS8 Seed register. It is modified using the PRS8 API.

Table 9. Block PRS8 Control Register: CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the PRS8 is enabled when set. It is modified using the PRS8 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.