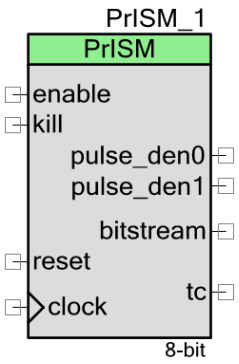
Precision Illumination Signal Modulation (PrISM)

2.10



Features

- Programmable flicker-free dimming resolution from 2 to 32 bit
- Two pulse density outputs
- Programmable output signal density
- Serial output bit stream
- Continuous run mode
- User-configurable sequence start value
- Standard or custom polynomials provided for all sequence lengths
- Kill input disables pulse density outputs and forces them low
- Enable input provides synchronized operation with other components
- Reset input allows restart at sequence start value for synchronization with other components
- Terminal Count Output for 8-, 16-, 24-, and 32-bit sequence lengths.

General Description

The Precision Illumination Signal Modulation (PrISM) component uses a linear feedback shift register (LFSR) to generate a pseudo random sequence. The sequence outputs a pseudo random bit stream, as well as up to two user-adjustable pseudo random pulse densities. The pulse densities may range from 0 to 100 percent.

The LFSR is of the Galois form (sometimes known as the modular form) and uses the provided maximal length codes. The PrISM component runs continuously after it starts and as long as the enable input is held high. The PrISM pseudo random number generator can be started with any valid seed value, excluding 0.

When to Use a PrISM

The PrISM component provides modulation technology that significantly reduces low-frequency flicker and radiated electromagnetic interference (EMI), which are common problems with high-

brightness LED designs. The PrISM is also useful in other applications that need this benefit, such as motor controls and power supplies.

Input/Output Connections

This section describes the various input and output connections for PrISM. An asterisk (*) in the list of I/Os indicates 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 the pseudo random sequence.

reset – Input

The reset input resets the pseudo random number to the start value at high state. This input is valid for the started component only and provides synchronized operation with other components.

kill – Input

The active-high kill input disables the PrISM pulse density outputs and sets them to 0 until kill is released low.

enable – Input

The PrISM component runs after it starts and as long as the enable input is held high and reset input is low. This input provides synchronized operation with other components.

pulse_den0/pulse_den1 – Outputs

Two pulse density outputs are available; both are derived from the same pseudo random sequence. Each output is generated by comparing the desired pulse density value with the current pseudo random number. If the pulse density type is configured as **Less Than or Equal**, then the output is high while the pseudo random number is less than or equal to the pulse density value. The second option is to set the pulse density type to **Greater Than or Equal** so the output is high while the pseudo random number is greater than or equal to the pulse density value.

bitstream – Output

The bitstream output continuously outputs the LSb of the LFSR.

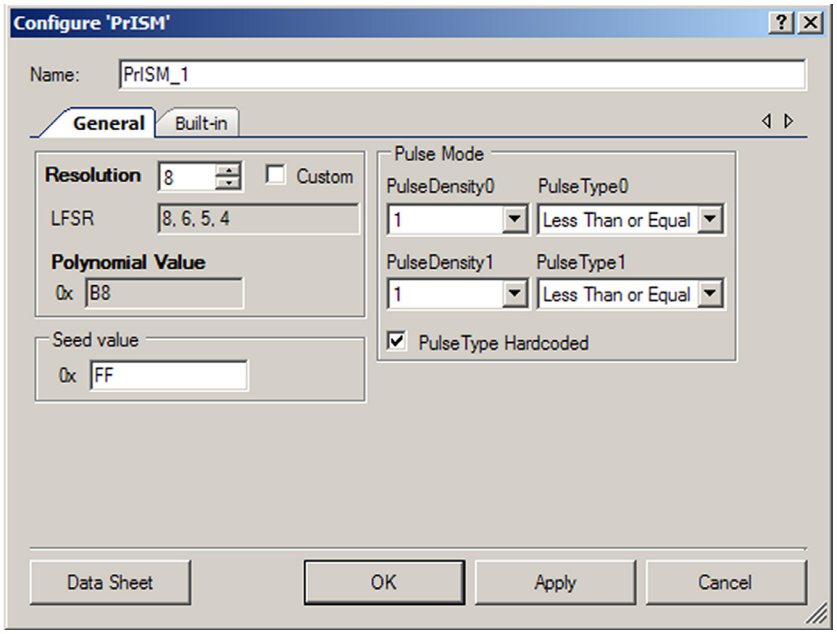
tc – Output *

The terminal count output is available for 8-, 16-, 24-, and 32-bit length PrISM components. The terminal count output goes high for one clock period each time the pseudo random number equals 0xFF (8-bit), 0xFFFF (16-bit), 0xFFFFFF (24-bit), or 0xFFFFFFFF (32-bit), which occurs once during each cycle of the pseudo random number generator.

Component Parameters

Drag a PrISM component onto your design and double-click it to open the **Configure** dialog box.

Figure 1. Configure Dialog Box



The PrISM component contains the following parameters:

Resolution

This parameter defines the PrISM maximal code length (period). The maximal code length is ($2^{Resolution}$ – 1). Possible values include 2 to 32 bits. The maximal length code sets the length of the pseudo random number generator and, therefore, the length of the sequence to be generated. Longer sequences increase the pulse density resolution and lower the radiated EMI. The maximal length codes listed in the following table are provided in the Galois form and require no conversion before you use them in the PSoC 3 UDB ALU.

Table 1. Maximal Code Lengths

Resolution	LFSR
2	2, 1
3	3, 2
4	4, 3
5	5, 4, 3, 2
6	6, 5, 3, 2
7	7, 6, 5, 4
8	8, 6, 5, 4
9	9, 8, 6, 5
10	10, 9, 7, 6
11	11, 10, 9, 7
12	12, 11, 8, 6

Resolution	LFSR
13	13, 12, 10, 9
14	14, 13, 11, 9
15	15, 14, 13, 11
16	16, 14, 13, 11
17	17, 16, 15, 14
18	18, 17, 16, 13
19	19, 18, 17, 14
20	20, 19, 16, 14
21	21, 20, 19, 16
22	22, 19, 18, 17
23	23, 22, 20, 18

Resolution	LFSR
24	24, 23, 21, 20
25	25, 24, 23, 22
26	26, 25, 24, 20
27	27, 26, 25, 22
28	28, 27, 24, 22
29	29, 28, 27, 25
30	30, 29, 26, 24
31	31, 30, 29, 28
32	32, 30, 26, 25

To Set LFSR Coefficients Manually:

Define **Resolution**.

Select the **Custom** check box.

Enter coefficients separated by comma in the LFSR text box and press [**Enter**]. The Polynomial Value is recalculated automatically.

The **Polynomial Value** is represented in hexadecimal format.

Note LFSR coefficient value cannot be greater than the **Resolution** value.

Polynomial Value

This parameter is represented in hexadecimal format. The correct polynomial is chosen based on the **Resolution** selected. A custom polynomial can be specified.

Seed Value

This parameter, by default, is set to the maximum possible value ($2^{Resolution} – 1$). This value can be changed to any value except 0. The **Seed value** is represented in the hexadecimal form.

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

Pulse Mode

These parameter values are chosen from combo boxes. Available values are from 1 to $2^{Resolution} - 1$ with a step $2^{Resolution}$. Pulse compare type can be set to **Less Than or Equal** or **Greater Than or Equal**.

PulseType Hardcoded

The **PulseType Hardcoded** parameter saves recourses (control register) when enabled, but makes it impossible to change the Pulse Type using the PrISM_SetPulse0Mode() or PrISM_SetPulse1Mode() APIs.

The PrISM_Stop() function is also not available if this parameter is enabled. To stop the PrISM in this case, use the “enable” input.

Local Parameters (For API usage)

These parameters are used in the API and not shown in the **Configure** dialog.

- **PolyValue(uint32) –** Contains the polynomial value in hexadecimal format. The default is 0xB8h (LFSR= [8,6,5,4]).
- **Density0(uint32) –** Contains density0 value in hexadecimal format.
- **Density1(uint32) –** Contains density1 value in hexadecimal format.
- **CompareType0(CompareType) –** Contains **Pulse Type** for Density0, which may be **Less Than or Equal** or **Greater Than or Equal**.
- **CompareType1(CompareType) –** Contains **Pulse Type** for Density1, which may be **Less Than or Equal** or **Greater Than or Equal**.

Clock Selection

There is no internal clock in this component. You must attach a clock source. This component operates from a single clock connected to the component.

Application Programming Interface

Application Programming Interface (API) routines allow configuration of 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 “PrISM_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 “PrISM.”

Table2. Function Interfaces

Function	Description
PrISM_Start()	The start function sets polynomial, seed, and pulse density registers provided by the customizer.
PrISM_Stop()	Stops PrISM computation.
PrISM_SetPulse0Mode()	Sets the pulse density type for Density0.
PrISM_SetPulse1Mode()	Sets the pulse density type for Density1.
PrISM_ReadSeed()	Reads the PrISM Seed register.
PrISM_WriteSeed()	Writes the PrISM Seed register with the start value.
PrISM_ReadPolynomial()	Reads the PrISM Polynomial register.
PrISM_WritePolynomial()	Writes the PrISM Polynomial register with the start value.
PrISM_ReadPulse0()	Reads the PrISM Pulse Density0 value register.
PrISM_WritePulse0()	Writes the PrISM Pulse Density0 value register with the new Pulse Density value.
PrISM_ReadPulse1()	Reads the PrISM Pulse Density1 value register.
PrISM_WritePulse1()	Writes the PrISM Pulse Density1 value register with the new Pulse Density value.
PrISM_Sleep()	Stops and saves the user configuration.
PrISM_Wakeup()	Restores and enables the user configuration
PrISM_Init()	Initializes the default configuration provided with the customizer.
PrISM_Enable()	Enables the PrISM block operation.
PrISM_SaveConfig()	Saves the current user configuration.
PrISM_RestoreConfig()	Restores the current user configuration.

Table3. Global Variables

Variable	Description
PrISM_initVar	Indicates whether the PrISM has been initialized. The variable is initialized to 0 and set to 1 the first time PrISM_Start() is called. This allows the component to restart without reinitialization after the first call to the PrISM_Start() routine. If reinitialization of the component is required, then the PrISM_Init() function can be called before the PrISM_Start() or PrISM_Enable() functions.

void PrISM_Start(void)

Description: This is the preferred method to begin component operation. PrISM_Start() sets the initVar variable, calls the PrISM_Init() function, and then calls the PrISM_Enable() function. The start function sets polynomial, seed, and pulse density registers provided by the customizer. PrISM computation starts on the rising edge of the input clock.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_Stop(void)

Description: Stops PrISM computation. Outputs remain constant.

Parameters: None

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

void PrISM_SetPulse0Mode(uint8 pulse0Type)

Description: Sets the pulse density type for Density0. Less Than or Equal(<=) or Greater Than or Equal(>=).

Parameters: uint8 pulse0Type: Selected pulse density type

Parameters Value	Description
PrISM_LESSTHAN_OR_EQUAL	The pulse_den0 output is high when the pseudo random number is less than or equal to the PulseDensity0 register value
PrISM_GREATERTHAN_OR_EQUAL	The pulse_den0 output is high when the pseudo random number is greater than or equal to the PulseDensity0 register value

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

void PrISM_SetPulse1Mode(uint8 pulse1Type)

Description: Sets the pulse density type for Density1. Less Than or Equal(<=) or Greater Than or Equal(>=).

Parameters: uint8 pulse1Type: Selected pulse density type

Parameters Value	Description
PrISM_LESSTHAN_OR_EQUAL	The pulse_den1 output is high when the pseudo random number is less than or equal to the PulseDensity1 register value
PrISM_GREATERTHAN_OR_EQUAL	The pulse_den1 output is high when the pseudo random number is greater than or equal to the PulseDensity1 register value

Return Value: None

Side Effects: Valid only if the **PulseType Hardcoded** parameter is disabled.

uint8/16/32 PrISM_ReadSeed(void)

Description: Reads the PrISM seed register.

Parameters: None

Return Value: uint8/16/32: Seed register value

Side Effects: None

void PrISM_WriteSeed(uint8/16/32 seed)

Description: Writes the PrISM seed register with the start value.

Parameters: uint8/16/32) seed: Seed register value

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPolynomial(void)

Description: Reads the PrISM polynomial.

Parameters: None

Return Value: uint8/16/32: Value of the polynomial

Side Effects: None

void PrISM_WritePolynomial(uint8/16/32 polynomial)

Description: Writes the PrISM polynomial.

Parameters: uint8/16/32 polynomial: Polynomial register value

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPulse0(void)

Description: Reads the PrISM PulseDensity0 value register.

Parameters: None

Return Value: uint8/16/32: PulseDensity0 register value

Side Effects: None

void PrISM_WritePulse0(uint8/16/32 pulseDensity0)

Description: Writes the PrISM Pulse Density0 value register with the new Pulse Density value.

Parameters: (unit8/16/32) pulseDensity0: Pulse Density value.

Return Value: None

Side Effects: None

uint8/16/32 PrISM_ReadPulse1(void)

Description: Reads the PrISM Pulse Density1 value register.

Parameters: None

Return Value: uint8/16/32: Pulse Density1 register value

Side Effects: None

void PrISM_WritePulse1(uint8/16/32 pulseDensity1)

Description: Writes the PrISM Pulse Density1 value register with the new Pulse Density value.

Parameters: uint8/16/32 pulseDensity1: Pulse Density value

Return Value: None

Side Effects: None

void PrISM_Sleep(void)

Description: This is the preferred API to prepare the component for sleep. The PrISM_Sleep() API saves the current component state. Then it calls the PrISM_Stop() function and calls PrISM_SaveConfig() to save the hardware configuration.

Call the PrISM_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power-management functions.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_Wakeup(void)

Description: This is the preferred API to restore the component to the state when PrISM_Sleep() was called. The PrISM_Wakeup() function calls the PrISM_RestoreConfig() function to restore the configuration. If the component was enabled before the PrISM_Sleep() function was called, the PrISM_Wakeup() function also re-enables the component.

Parameters: None

Return Value: None

Side Effects: Calling the PrISM_Wakeup() function without first calling the PrISM_Sleep() or PrISM_SaveConfig() function may produce unexpected behavior.

void PrISM_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call PrISM_Init() because the PrISM_Start() API calls this function and is the preferred method to begin the component operation.

Parameters: None

Return Value: None

Side Effects: All registers are set to values according to the customizer Configure dialog.

void PrISM_Enable(void)

Description: Activates the hardware and begins component operation. It is not necessary to call PrISM_Enable() because the PrISM_Start() API calls this function, which is the preferred method to begin the component operation.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_SaveConfig(void)

Description: This function saves the component configuration and nonretention registers. It also saves the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the PrISM_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void PrISM_RestoreConfig(void)

Description: This function restores the component configuration and nonretention registers. It also restores the component parameter values to what they were before calling the PrISM_Sleep() function.

Parameters: None

Return Value: None

Side Effects: Calling this function without first calling the PrISM_Sleep() or PrISM_SaveConfig() function may produce unexpected behavior.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

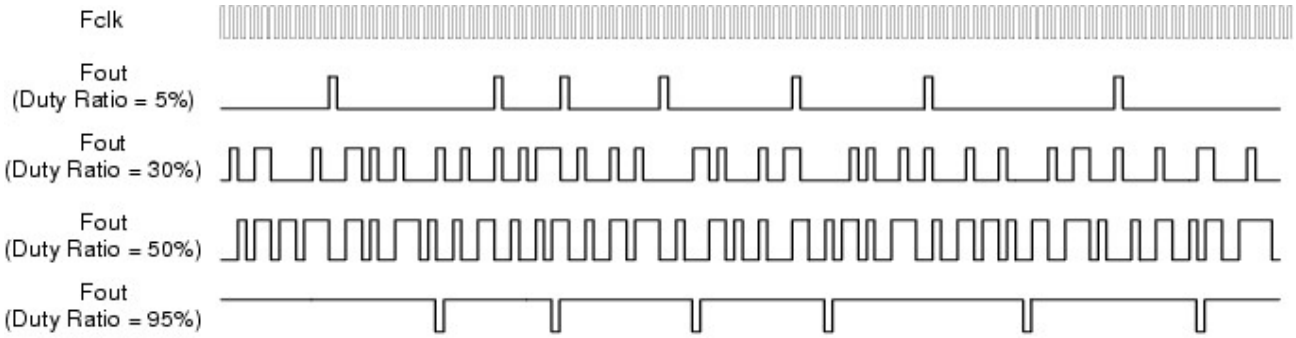
Functional Description

The PrISM component runs continuously after starting and as long as the “enable” input is kept high. The PrISM pseudo random number generator may be started with any valid value excluding 0. This allows multiple PrISM components to run out of phase of each other to further reduce EMI. The “reset” input resets the pseudo random number to the start value. The active-high “kill” input disables the PrISM pulse density outputs and sets them to 0 until kill is released low. The “bitstream” output continuously outputs the LSb of the LFSR.

Two Pulse Density outputs are available; both are derived from the same pseudo random sequence. Each output is generated by comparing the desired pulse density value with the current pseudo random number.

The following timing diagram shows the PrISM output based on several pulse density ratios.

Figure 2. Timing of PrISM Output

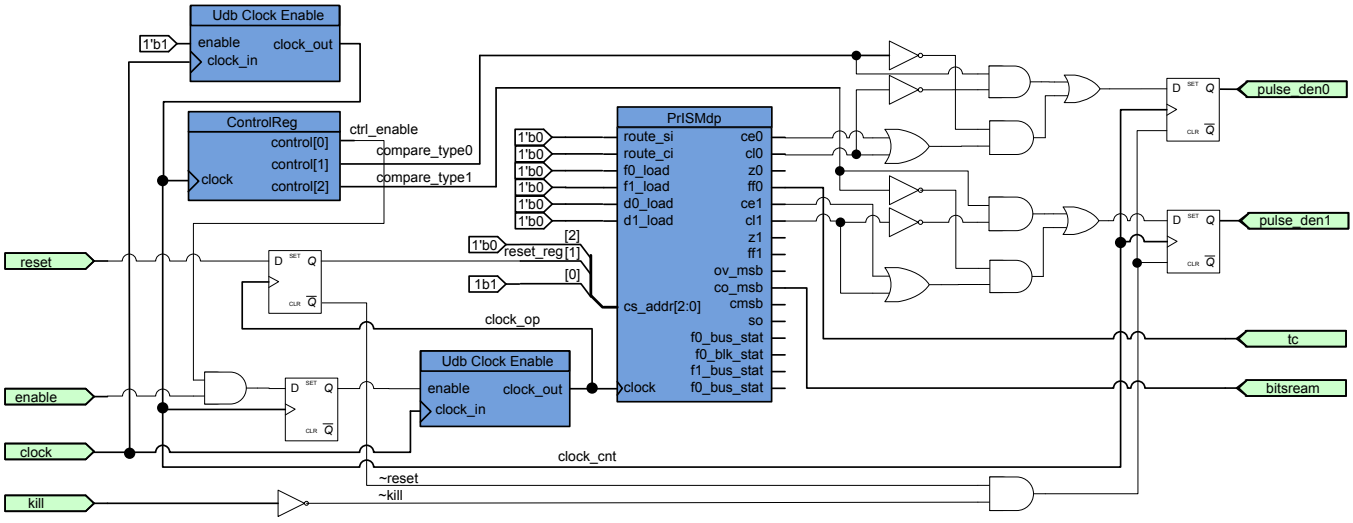


Block Diagram and Configuration

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

The implementation is described in the following block diagram.

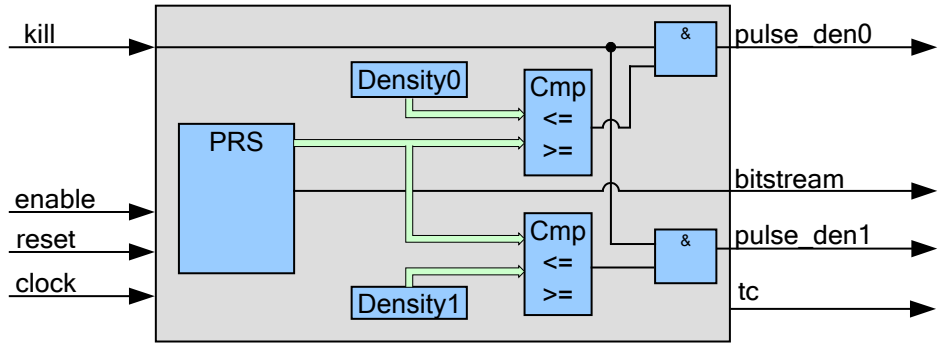
Figure 3. PrISM Implementation



Top-Level Architecture

The 2- to 32-bit hardware PrISM component compares the output of a pseudo random counter with a signal density value. The comparator output asserts when the count value is less than (or greater than) or equal to the value in the Density value register.

Figure 4. PrISM Top-Level Architecture



Registers

PrISM_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	reserved					compare type1	compare type0	ctrl enable

- ctrl enable: This bit enables generation of all internal signals described in the previous sections. The value can be changed by the PrISM_Start() and PrISM_Stop() functions.
- compare type0: This bit performs compare type for the pulse_den0 output. The value of this bit is determined by the choice made for the pulse compare type parameter in the component Configure dialog. Also, the value can be changed by the PrISM_SetPulse0Mode() function.
- compare type1: This bit performs compare type for pulse_den1 output. The value of this bit is determined by the choice made for the pulse compare type parameter in the component Configure dialog. Also, the value can be changed by the PrISM_SetPulse1Mode() function.

The control register is not used if the **PulseType Hardcoded** option is selected.

PrISM_SEED

Bits	7	6	5	4	3	2	1	0
Value	Seed							

- Seed: Contains the initial Seed value and PRS residual value at the end of the computation. The value of this register is determined by the **Seed value** parameter in the component Configure dialog. Also, the value can be changed by the PrISM_WriteSeed() function and can be read by PrISM_ReadSeed().

PrISM_SEED_COPY

Bits	7	6	5	4	3	2	1	0
Value	Seed_Copy							

- Seed_Copy: Contains the start Seed value for automatically loading PrISM_SEED register when the “reset” input is active. The value of this register is determined by the **Seed value** parameter in the component Configure dialog and automatically updates if the PrISM_WriteSeed() function is called.

PrISM_POLYNOM

Bits	7	6	5	4	3	2	1	0
Value	Polynomial							

- Polynomial: The correct polynomial chosen based on the resolution selected. The value can be changed by the PrISM_WritePolynomial() function and can be read by the PrISM_ReadPolynomial() function.

PrISM_DENSITY0

Bits	7	6	5	4	3	2	1	0
Value	Pulse density0							

- Pulse density0 determines the value for the PrISM pulse_den0 output. The value of this register is determined by the **PulseDensity0** parameter in the Configure dialog. This value can be changed by the PrISM_WritePulse0() function.

PrISM_DENSITY1

Bits	7	6	5	4	3	2	1	0
Value	Pulse density1							

- Pulse density1 determines the value for the PrISM pulse_den1 output. The value of this register is determined by the **PulseDensity1** parameter in the Configure dialog. This value can be changed by the PrISM_WritePulse1() function.

References

Refer also to the PRS component datasheet.

Resources

The PrISM component is placed throughout the UDB array. The component utilizes the following resources.

Table 4. Resource Types

Configuration	Resource Type					
	Datapath Cells	Macrocells	Status Cells	Control Cells[1]	DMA Channels	Interrupts
8-bit	1	4	–	1	–	–
16-bit	2	4	–	1	–	–
24-bit	3	4	–	1	–	–
32-bit	4	4	–	1	–	–

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Table 5. API Memory Resource Usage

Configuration	PSoC 3 (Keil_PK51)		PSoC 5 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
8-bit	284	6	480	9	428	9
16-bit	431	9	496	13	440	9
24-bit	444	15	624	25	532	17
32-bit	446	15	520	25	452	17

1. Control cells are not used if the **PulseType Hardcoded** parameter is checked.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

Table 6. DC Characteristics

Parameter	Description	Min	Typ[2]	Max	Units
I_{DD}	Component current consumption				
	8-bit	–	15	–	µA/MHz
	16-bit	–	22	–	µA/MHz
	24-bit	–	28	–	µA/MHz
	32-bit	–	35	–	µA/MHz

Table 7. AC Characteristics

Parameter	Description	Min	Typ	Max[3]	Units
f_{CLOCK}	Component clock frequency				
	8-bit			66	MHz
	16-bit			55	MHz
	24-bit			48	MHz
	32-bit			40	MHz

2. Device IO and clock distribution current not included. The values are at 25 °C.

3. The values provide a maximum safe operating frequency of the component. The component may run at higher clock frequencies, at which point validation of the timing requirements with STA results is necessary.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.10	Added all APIs with the CYREENTRANT keyword when they are included in the .cyre file.	Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates. This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections.
	Added support PSoC 5LP silicon.	
2.0.a	Minor datasheet edits and updates	
2.0	Pulse Density outputs registered for removing possible glitching.	Any combinatorial output can glitch, depending on placement and delay between signals. To remove glitching, the outputs should be registered.
	Enable and reset inputs registered to improve maximum speed operation.	These inputs had combinatorial usage, therefore were not automatically registered by Creator and had violations. Registering improves maximum speed and protects from possible glitching.
	Added characterization data to datasheet	
	Minor datasheet edits and updates	

© Cypress Semiconductor Corporation, 2012. 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® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks 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.