Features and Overview

- Programmable flicker-free dimming resolution from 2 to 16 bits
- Dedicated PrISM User Module enables you to use PSoC core digital blocks for other needs
- Programmable output signal density
- Selectable software output frequency compensation
- Reduced EMI

The 16-Bit Hardware Precise Intensity Signal Modulation (PrISM™) User Module 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 less than or equal to) the value in the Signal Density register.

Figure 1. PrISM16HW Block Diagram

The application note AN47372 - PowerPSoC - PrISM(TM) Technology for LED Dimming addresses the challenges faced in implementing high resolution PrISM and recommends solutions to address these issues.
Functional Description

The PrISM16HW User Module produces a stochastic signal density modulation. It uses a pseudo-random counter that is implemented as a linear feedback shift register (LFSR). The counter operates in the range of \([1..2^{\text{DimmingResolution}}-1]\). Zero is the missing value. The pseudorandom count is compared with the Signal Density value. If the random value is less than (or less than or equal to) the signal density value, the comparator outputs a high level. This principle of operation is very similar to a standard Pulse Width Modulator (PWM). In both cases the counter generates a similar set of binary numbers from Period to zero (PWM) or from Period to one (PrISM). However, a PWM produces a linearly ordered series of numbers where PrISM produces the numbers in random order. That is why the PrISM16HW output signal looks like series of random pulses.

The following time diagram shows PrISM16HW dimming frequency output depending on the duty ratio.

Figure 2. PRISM16HW Timing Diagram

![Timing Diagram](image)

The following table lists modular irreducible polynomials for the pseudo-random counter. These polynomials define 2 to 16-bit pseudo-random number sequences (including 8, 16 bits). Note that the table is not complete. Only one of the many polynomials is specified. This table is for reference purposes. The polynomial is set automatically by the PrISM16HW User Module.

Table 1. Modular Irreducible Polynomials (Continued)

<table>
<thead>
<tr>
<th>Number of Bits</th>
<th>Code Length</th>
<th>Modular Form Polynomial</th>
<th>Polynomial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>[2,1]</td>
<td>0x03</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>[3,2]</td>
<td>0x06</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>[4,3]</td>
<td>0x0C</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>[5,4,3,2]</td>
<td>0x1E</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>[6,5,4,1]</td>
<td>0x39</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>[7,6,5,2]</td>
<td>0x72</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
<td>[8,6,5,4]</td>
<td>0xB8</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
<td>[9,6,5,3]</td>
<td>0x134</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
<td>[10,8,7,2]</td>
<td>0x2C2</td>
</tr>
<tr>
<td>11</td>
<td>2047</td>
<td>[11,9,6,3]</td>
<td>0x524</td>
</tr>
<tr>
<td>12</td>
<td>4095</td>
<td>[12,11,8,6]</td>
<td>0xCA0</td>
</tr>
</tbody>
</table>
Signal Density is a 16-bit value that defines the average output duty ratio:

\[
\text{DutyRatio} = \frac{\text{SignalDensity} - 1}{\text{Period}}, \text{ for CompareType = Less Than}
\]

\[
\text{DutyRatio} = \frac{\text{SignalDensity}}{\text{Period}}, \text{ for CompareType = Less Than or Equal To}
\]

In this equation, Period depends on the Dimming Resolution (polynomial order):

\[
\text{Period} = 2^{\text{DimmingResolution}} - 1
\]

**Frequency Compensation**

Figure 2 shows that output the frequency depends on the Signal Density value (or to the duty ratio). If the duty ratio is near zero or one then the output frequency is the lowest. The middle of the duty ratio values range has the highest value.

In general, the output frequency can be calculated using the following formula:

\[
\text{Fout} = 0.5 \times \text{DutyRatio} \times \text{Fclk} \quad \text{for } 0 < \text{DutyRatio} \leq 0.5
\]

\[
\text{Fout} = 0.5 \times (1 - \text{DutyRatio}) \times \text{Fclk} \quad \text{for } 0.5 < \text{DutyRatio} < 1
\]

In Equation 3, Fclk is the input frequency of the pseudo-random counter.

<table>
<thead>
<tr>
<th>Number of Bits</th>
<th>Code Length</th>
<th>Modular Form Polynomial</th>
<th>Polynomial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>8191</td>
<td>[13,12,10,9]</td>
<td>0x1B00</td>
</tr>
<tr>
<td>14</td>
<td>16383</td>
<td>[14,13,12,2]</td>
<td>0x3802</td>
</tr>
<tr>
<td>15</td>
<td>32767</td>
<td>[15,13,10,8]</td>
<td>0x5280</td>
</tr>
<tr>
<td>16</td>
<td>65535</td>
<td>[16,15,13,4]</td>
<td>0xD008</td>
</tr>
</tbody>
</table>
The lowest frequency value is $F_{\text{clk}}/\text{Period}$ and the highest is $F_{\text{clk}}/4$:

Figure 3. Output Frequency

Taking into account that $F_{\text{clk}}$ depends on the block input frequency of the ModClock (24 or 48 MHz) and the ClockScaler value, the minimum value of $F_{\text{out}}$ is:

Equation 4

$$F_{\text{out}} = \frac{\text{ModClock}}{\text{ClockScaler} \cdot \text{Period}}$$

In an LED application, dimming frequencies that are too low cause visible flicker in the LEDs. 300Hz is the recommended minimum for $F_{\text{out}}$. Increasing the dimming frequency at middle of the Signal Density values may reduce efficiency due to the higher switching losses. The Software Frequency Compensation mechanism solves this problem by limiting the range of variability on the $F_{\text{out}}$ signal to a single frequency value (or close). This is done by an API that changes the ClockScaler value in such a way to keep $F_{\text{out}}$ at this desired single value.

The value of the ClockScaler that compensates $F_{\text{out}}$ changes can be calculated as:

Equation 5

$$\text{ClockScaler} = \frac{1}{2} \cdot \text{ClockScaler}_{\text{def}} \cdot \text{SignalDensity} \quad \text{for} \quad 0 < \text{SignalDensity} < \frac{\text{Period}}{2}$$

$$\text{ckScaler} = \frac{1}{2} \cdot \text{ClockScaler}_{\text{def}} \cdot (\text{Period} - \text{SignalDensity}) \quad \text{for} \quad \frac{\text{Period}}{2} < \text{SignalDensity} < \text{Pei}$$

The $\text{ClockScaler}_{\text{def}}$ is an initial user module parameter value.

As long as the ClockScaler is limited to a $[1:256]$ range, not all signal density values can be compensated. This is especially true at resolutions higher than 9. At the point where the calculated scaler exceeds the maximum as shown, the compensation is not complete.
DC and AC Electrical Characteristics

Table 2. AC Specifications

<table>
<thead>
<tr>
<th>AC Parameter</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Input frequency</td>
<td>Min 24, Max 48 MHz</td>
</tr>
<tr>
<td>ClockScaler</td>
<td>Input frequency scaler</td>
<td>Min 1, Max 256</td>
</tr>
</tbody>
</table>
| Fout         | Fout is MOD block output frequency for PrISM mode | \[
\frac{\text{Clock}_{\text{min}}}{\text{Period} \cdot \text{ClockScaler}_{\text{max}}} \leq 256
\]

Placement

The PrISM16HW User Module can be placed in one of the available modulator blocks; MOD0, MOD1, MOD2, or MOD3.
Parameters and Resources
The following parameters configure the PrISM16HW User Module.

ClockScaler
The ClockScaler parameter allows the input clock (48 MHz or 24 MHz) to be scaled down by a factor of ClockScaler. Allowed values are between 1 and 256.

DimmingResolution
The DimmingResolution parameter contains a value from 2-16. This parameter defines the pseudo-random counter period.

\[
\text{Period} = 2^{\text{DimmingResolution}} - 1
\]

Equation 6

SignalDensity
The SignalDensity parameter contains values from 0 to \(2^n-1\) where \(n\) is the value selected in the Dimming Resolution parameter. The output average Duty Ratio is the SignalDensity divided by Period. For a CompareType of less than or equal to, use the following equation:

\[
\text{Duty Ratio} = \frac{\text{SignalDensity}}{\text{Period}}
\]

Equation 7

For a CompareType of Less Than, as the following equation:

\[
\text{Duty Ratio} = \frac{\text{SignalDensity} - 1}{\text{Period}}
\]

Equation 8

FrequencyCompensation
Enables or disables software frequency compensation.

CompareType
This parameter sets the compare function type "LessThan" or "LessThanEqual". This affects output signal duty ratio. A compare type must be chosen for the user module to function properly.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LessThan</td>
<td>PrISM16HW output goes high when Counter value is less than Signal Density value.</td>
</tr>
<tr>
<td>LessThanEqual</td>
<td>PrISM16HW output goes high when Counter value is less than or equal to Signal Density value.</td>
</tr>
</tbody>
</table>
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.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the PrISM16HW_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions, the instance name has been shortened to PrISM16HW for simplicity.

Note

** In this, as in all user module APIs, you can alter the values of the A and X register 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 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 functions may leave A and X unchanged, there is no guarantee they may 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.

PRISM16HW_Start

Description:

Starts the PrISM16HW operation. Until started output of PrISM16HW is asserted high.

C Prototype:

void PRISM16HW_Start(void);

Assembler:

lcall PRISM16HW_Start

Parameters:

None

Return Value:

None

Side Effects:

After this function is called ClockScaler is returned to value defined in the User Module ClockScaler parameter. See Note ** at the beginning of the API section.

PRISM16HW_Stop

Description:

Stops the PrISM16HW operation. The output of the PrISM16HW is high when the user module is stopped.
C Prototype:
void PRISM16HW_Stop(void);

Assembler:
lcall PRISM16HW_Stop

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

PRISM16HW_SetClockScaler

Description:
Writes the Clock Scaler with a value from 1 to 256. The input clock (48 MHz or 24 MHz) is scaled
don down by a factor of ClockScaler.

C Prototype:
void PRISM16HW_SetClockScaler(WORD wClockScaler);

Assembler:
mov X, [wClockScaler] ; MSB
mov A, [wClockScaler+1] ; LSB
lcall PRISM16HW_SetClockScaler

Parameters:

wClockScaler is a value from 1 to 256.

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section

PRISM16HW_SetSignalDensity

Description:
Sets the PrISM16HW Signal Density of the output of the modulator block. If the Frequency Compensa-
tion mode is enabled, this API also adjusts the Frequency Scaler block to keep a stable output
frequency.

C Prototype:
void PRISM16HW_SetSignalDensity(WORD wSignalDensity);

Assembler:
mov X, [wSignalDensity]
mov A, [wSignalDensity+1]
lcall PRISM16HW_SetSignalDensity
Parameters:
  wSignalDensity - 16-bit signal density value. MSB is passed in the X register and LSB is passed in the Accumulator.

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section

PRISM16HW_SetDimmingResolution

Description:
  Initializes the pseudorandom counter Polynomial Register with a polynomial that matches the desired resolution.

C Prototype:
  void PRISM16HW_SetDimmingResolution(BYTE bResolution);

Assembler:
  mov A, bResolution
  lcall PRISM16HW_SetDimmingResolution

Parameters:
  bResolution - the resolution parameter contains a value from 2 to 16 (which be presented by 1 byte).
  This parameter defines the pseudo-random counter period as $2^n-1$ where n is resolution.

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section

PRISM16HW_EnableFrequencyComp

Description:
  Enables the software frequency compensation.

C Prototype:
  void PRISM16HW_EnableFrequencyComp(void);

Assembler:
  lcall PRISM16HW_EnableFrequencyComp

Parameters:
  None.

Return Value:
  None.

Side Effects:
  See Note ** at the beginning of the API section
**PRISM16HW_DisableFrequencyComp**

**Description:**
Disables the software frequency compensation.

**C Prototype:**
```c
void PRISM16HW_DisableFrequencyComp(void);
```

**Assembler:**
```assembly
lcall PRISM16HW_DisableFrequencyComp
```

**Parameters:**
None.

**Return Value:**
None.

**Side Effects:**
See Note ** at the beginning of the API section

**Sample Firmware Source Code**
The C code illustrated here shows you how to use the PrISM16HW User Module.

```c
PRISM16HW_SetDimmingResolution(10);
PRISM16HW_SetSignalDensity(256);
PRISM16HW_Start();
```

The same code in assembly is as follows.
```assembly
mov   A, 0Ah
call  PRISM16HW_SetDimmingResolution
mov   X, 01h
mov   A, 00h
call  PRISM16HW_SetSignalDensity
call  PRISM16HW_Start
```

**Configuration Registers**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Programmable Clock Frequency Scaler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programmable Clock Frequency Scaler determines the clock scaler for the PrISM block. The value of this register is determined by the choice made for the ClockScaler parameter in the user module parameters of the Device Editor. Also value can be changed by PRISM16HW_SetClockScaler() API.
Table 4. PRISM16HW_PDH_REG (MSB), PRISM16HW_PDL_REG (LSB)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>Polynomial Register (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>Polynomial Register (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Polynomial determines the period value for the PrISM16HW block. The value of this register is determined by the choice made for the DimmingResolution parameter in the user module parameters of the Device Editor. Also value can be changed by PRISM16HW_SetDimmingResolution() API.

Table 5. PRISM16HW_PWH_REG (MSB), PRISM16HW_PWL_REG (LSB)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>Signal Density Register (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>Signal Density Register (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signal Density determines the value for the PrISM16HW block output. The value of this register is determined by the choice made for the SignalDensity parameter in the user module parameters of the Device Editor. The value can be changed by PRISM16HW_SetSignalDensity() API.

Table 6. PRISM16HW_PCFG_REG

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CompType</td>
</tr>
</tbody>
</table>

CompType determines the PrISM16HW compare type. The value of this bit is determined by the choice made for the CompareType parameter in the user module parameter of the Device Editor.

Table 7. PRISM16HW_GCFG_REG

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Enable</td>
</tr>
</tbody>
</table>

Enable turns on the PrISM16HW block. The value of this bit can be changed by PRISM16HW_Start() and PRISM16HW_Stop() API.
## Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>DHA</td>
<td>Initial version</td>
</tr>
</tbody>
</table>

**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.