Features and Overview

- PS/2 Device Interface
- Selectable command sets for mouse or keyboard applications
- Custom feature unlock mechanism
- Integrated with the USB SIE for combo USB-PS/2 applications

Figure 1. Block Diagrams

<table>
<thead>
<tr>
<th>Resources</th>
<th>API Memory (Bytes)</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY7C639/638/633/602/601xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mouse Support</td>
<td>935</td>
<td>17 (including 4 byte transfer buffer)</td>
</tr>
<tr>
<td>Keyboard Support</td>
<td>810</td>
<td>17 (including 4 byte transfer buffer)</td>
</tr>
</tbody>
</table>
Functional Description

The PS/2 Device User Module supports a standard PS/2 keyboard or a standard PS/2 mouse. A PS/2 connector is a 6-pin mini-DIN connector. It can be used for connecting either a keyboard or a mouse with a compatible computer. You must choose to support either a keyboard or a mouse with the PS/2 Device User Module. Choose to support either the keyboard command set or the mouse command set when you select the user module. If you wish to change from one to the other after the user module is selected, right click on the user module icon and select **User Module Selection Options**. By convention, connectors that support keyboards are color coded purple, and ports that support mice are color coded green. With the addition of the USB Device User Module, you can create a single device that supports a USB and PS/2 combination device.

Data Format

Host to Device Data Format (12 bits):
- Request to Send. (0)
- 8 data bits.
- Parity bit. (Odd Parity)
- Stop bit. (1)
- Acknowledge. (1)

Device to Host Data Format (11 bits):
- Start bit. (0)
- 8 data bits.
- Parity bit. (Odd Parity)
- Stop bit. (1)

DC and AC Electrical Characteristics

See the device datasheet for your enCoRe device for electrical characteristics.

Timing

The following timing diagrams describe the PS/2 timing requirements for proper PS/2 data transmission and reception. The PS/2 Device is responsible for generating all of the clock signals on SCLK for both Host to Device and Device to Host transfers.

Figure 2. PS/2 Device to Host Transfer Timing

```
<table>
<thead>
<tr>
<th>SCLK</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SDATA</td>
<td>S</td>
</tr>
</tbody>
</table>
```

S 0 1 2 3 4 5 6 7 P Y S
During Device to Host transfers, the Device controls both SCLK and SDATA. The Device maintains a consistent SCLK cycle between 10 kHz and 14.6 kHz. The Device updates SDATA on while SCLK is high, and the Host sample SDATA when SCLK is low.

Figure 3. PS/2 Host to Device Transfer Timing

Four distinct phases occur during a Host to Device transfer. First, the Host pulls SCLK low, indicating an Host Inhibit condition. During an Inhibit, the Device must stop any transfer in progress and release both SCLK and SDATA. Next, the Host pulls SDATA low, and releases SCLK, signaling a Request to Send to the Device. During the data transfer phase, the Device cycles SCLK. The Host updates SDATA when SCLK is low. The Device samples SDATA when SCLK is high. The falling edge of the parity bit starts the Acknowledgment Phase. The Host releases SDATA, forming the stop bit. After sampling SDATA, the Device acknowledges receipt of the data, by driving SDATA low. The Host samples SDATA on the low SCLK for the acknowledgment.

**Placement**

The PS2 Device User Module is software only and does not consume enCoRe blocks.

**Parameters and Resources**

**PS2D Port**

Choose the port for the PS2 data and clock pins. If you plan to create a dual PS2/USB device, choose the port that the USB transceiver is attached to.

**TxBufferSize**

This parameter determines how many RAM locations are reserved for the transmit buffer. The transit buffer must accommodate the largest data packet generated by the application.
Application Programming Interface

The Application Programming Interface routines in this section allow programmatic control of the PS/2 User Module. The following tables list the basic and device specific API functions.

**Note** When using this the PS2D User Module and GPIO pins from Port n (Pn.2-Pn.7), the application should use the Port_n_Data_SHADE shadow register to ensure consistent data handling. From assembly language, you can access the Port_n_Data_SHADE RAM location directly. From C language, you should include an extern reference:

```c
extern BYTE Port_n_Data_SHADE; // Where n is the port number
```

Table 1. Basic PS/2 Device API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void PS2D_Start(void)</td>
<td>Enable user module.</td>
</tr>
<tr>
<td>void PS2D_Stop(void)</td>
<td>Disable user module.</td>
</tr>
<tr>
<td>BYTE PS2D_DoCommand(void)</td>
<td>Receives and processes a PS/2 Host Command if the Host requests to send one. This function should be called at least once every 1-2 milliseconds.</td>
</tr>
<tr>
<td>void PS2D_SendNextByte(void)</td>
<td>Transmit the next byte in the transmit buffer.</td>
</tr>
<tr>
<td>BYTE PS2D_TransferInProgress(void)</td>
<td>Returns 1 if a transfer to the PS/2 Host is in progress, otherwise returns 0</td>
</tr>
<tr>
<td>void PS2D_SendResponseResend(void)</td>
<td>Resend the contents of the PS/2 transmit buffer.</td>
</tr>
<tr>
<td>void PS2D_AbortTransfer(void)</td>
<td>Abort the current transfer.</td>
</tr>
</tbody>
</table>

Table 2. Device Specific API (Mouse)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE PS2D_GetDeviceID(void);</td>
<td>Returns the Device ID. The PS2D supports two mouse Device IDs.</td>
</tr>
<tr>
<td></td>
<td>Device ID 0 is returned as the default Mouse Device Id. The host expects a three byte mouse data packet for Device ID 0.</td>
</tr>
<tr>
<td></td>
<td>Device ID 3 is return for scroll wheel mice. This feature is automatically activated by a series of Set Resolutions Commands from the host. (200, 100, 80). The host expects a four byte mouse data packet for Device ID 3.</td>
</tr>
<tr>
<td>BYTE PS2D_IsEnabled(void)</td>
<td>Returns 1 if the host has enabled Data Reporting with the &quot;Enable Data Reporting&quot;, otherwise returns 0.</td>
</tr>
<tr>
<td>BYTE PS2D_IsStreamMode(void);</td>
<td>Returns 1 if the interface is in streaming mode, otherwise 0.</td>
</tr>
<tr>
<td>BYTE PS2D_GetSampleInterval(void);</td>
<td>Returns the sample interval (milliseconds).</td>
</tr>
</tbody>
</table>
Table 3. Device Specific API (Keyboard)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD PS2D_GetDeviceID(void);</td>
<td>Returns the Device ID.</td>
</tr>
<tr>
<td>BYTE PS2D_IsEnabled(void)</td>
<td>Returns 1 if the host has enabled the keyboard, otherwise returns 0.</td>
</tr>
<tr>
<td>BYTE PS2D_GetScanCodeSet(void);</td>
<td>Returns the current Scan Code Set selected by the PS/2 Host</td>
</tr>
<tr>
<td>BYTE PS2D_GetRepeatRate(void);</td>
<td>Returns the current Typematic Repeat Rate selected by the PS/2 Host</td>
</tr>
<tr>
<td>BYTE PS2D_GetDelay(void);</td>
<td>Returns the current Typematic Delay selected by the PS/2 Host</td>
</tr>
<tr>
<td>BYTE PS2D_GetLED(void);</td>
<td>Returns the current LED settings selected by the PS/2 Host</td>
</tr>
</tbody>
</table>

**PS2D_Start**

**Description:**
Performs all required initialization for PS/2 Device User Module.

**C Prototype:**
```c
void PS2D_Start(void)
```

**Assembly:**
```asm
lcall PS2D_Start
```

**Parameters:**
None

**Return Value:**
None

**Side Effects:**
The A and X registers may be altered by this function.

**PS2D_Stop**

**Description:**
Performs all necessary shutdown task required for the PS/2 User Module. This function should be called in PS/2-USB combi applications when the actual interface is determined to be connected to a USB host.

**C Prototype:**
```c
void PS2D_Stop(void)
```

**Assembly:**
```asm
lcall PS2D_Stop
```

**Parameters:**
None
Return Value:
None

Side Effects:
The A and X registers may be altered by this function.

PS2D_DoCommand

Description:
Receive and process a command from the PS/2 Host

C Prototype:
BYTE PS2D_DoCommand(void)

Assembly:
lcall PS2D_DoCommand

Parameters:
None

Return Value:
Returns a non-zero value if a PS/2 command had been received and processed.

Side Effects:
The A and X registers may be altered by this function.

PS2D_SendNextByte

Description:
Sends the next byte in the transmit buffer to the PS/2 Host.

C Prototype:
void PS2D_SendNextByte(void)

Assembly:
lcall PS2D_SendNextByte

Parameters:
None

Return Value:
None

Side Effects:
The A and X registers may be altered by this function.
PS2D_TransferInProgress

Description:
Determines if a PS/2 transfer is in progress.

C Prototype:
BYTE PS2D_TransferInProgress(void)

Assembly:
lcall PS2D_TransferInProgress

Parameters:
None

Return Value:
Returns a non-zero value if a PS/2 transfer is in progress.

Side Effects:
The A and X registers may be altered by this function.

PS2D_SendResponseResend

Description:
Resend the contents of the PS/2 transmit buffer.

C Prototype:
void PS2D_SendResponseResend(void)

Assembly:
lcall PS2D_SendResponseResend

Parameters:
None

Return Value:
None

Side Effects:
The A and X registers may be altered by this function.

PS2D_AbortTransfer

Description:
Abort the current transfer.

C Prototype:
void PS2D_AbortTransfer(void)

Assembly:
lcall PS2D_AbortTransfer

Parameters:
None
Return Value:
  None

Side Effects:
  The A and X registers may be altered by this function.

Device Specific API (Mouse)

PS2D_GetDeviceID

Description:
  Returns the Device ID.

C Prototype:
  BYTE PS2D_GetDeviceID(void)

Assembly:
  lcall PS2D_GetDeviceID

Parameters:
  None

Return Value:
  Device ID

Side Effects:
  The A and X registers may be altered by this function.

PS2D_IsEnabled

Description:
  Returns 1 if the interface is enabled, otherwise returns 0.

C Prototype:
  BYTE PS2D_IsEnabled(void)

Assembly:
  lcall PS2D_IsEnabled

Parameters:
  None

Return Value:
  None

Side Effects:
  The A and X registers may be altered by this function.
**PS2D_IsStreamMode**

**Description:**
Returns 1 if the interface is in streaming mode, otherwise 0.

**C Prototype:**
```c
BYTE PS2D_IsStreamMode(void)
```

**Assembly:**
```assembly
lcall PS2D_IsStreamMode
```

**Parameters:**
None

**Return Value:**
Stream Mode

**Side Effects:**
The A and X registers may be altered by this function.

**PS2D_GetSampleInterval**

**Description:**
Returns the sample interval (milliseconds) set by the PS/2 Host

**C Prototype:**
```c
BYTE PS2D_GetSampleInterval(void)
```

**Assembly:**
```assembly
lcall PS2D_GetSampleInterval
```

**Parameters:**
None

**Return Value:**
Sample Interval

**Side Effects:**
The A and X registers may be altered by this function.
Device Specific API (Keyboard)

**PS2D_GetDeviceID**

**Description:**
Returns the PS/2 ID for keyboards

**C Prototype:**
```c
WORD PS2D_GetDeviceID(void)
```

**Assembly:**
```assembly
lcall PS2D_GetDeviceID
```

**Parameters:**
None

**Return Value:**
PS/2 ID for keyboards (0xAB83)

**Side Effects:**
The A and X registers may be altered by this function.

**PS2D_IsEnabled**

**Description:**
Returns 1 if the interface is enabled, otherwise returns 0.

**C Prototype:**
```c
BYTE PS2D_IsEnabled(void)
```

**Assembly:**
```assembly
lcall PS2D_IsEnabled
```

**Parameters:**
None

**Return Value:**
None

**Side Effects:**
The A and X registers may be altered by this function.

**PS2D_GetScanCodeSet**

**Description:**
Returns the Scan Code Set selected by the PS/2 Host

**C Prototype:**
```c
BYTE PS2D_GetScanCodeSet(void)
```

**Assembly:**
```assembly
lcall PS2D_GetScanCodeSet
```
Parameters:
None

Return Value:
Scan Code Set

Side Effects:
The A and X registers may be altered by this function.

**PS2D_GetRepeatRate**

Description:
Returns the Typematic Rate selected by the PS/2 Host

**C Prototype:**
BYTE PS2D_GetRepeatRate(void)

**Assembly:**
lcall PS2D_GetRepeatRate

Parameters:
None

Return Value:
Repeat Rate

Side Effects:
The A and X registers may be altered by this function.

**PS2D_GetDelay**

Description:
Returns the Typematic Delay selected by the PS/2 Host

**C Prototype:**
BYTE PS2D_GetDelay(void)

**Assembly:**
lcall PS2D_GetDelay

Parameters:
None

Return Value:
Typematic Delay

Side Effects:
The A and X registers may be altered by this function.
**PS2D_GetLED**

**Description:**

Returns the current LED setting set by the PS/2 Host

**C Prototype:**

```c
BYTE PS2D_GetLED(void)
```

**Assembly:**

```assembly
lcall PS2D_GetLED
```

**Parameters:**

None

**Return Value:**

LED Setting Bitmap.
- Bit 0 Scroll Lock (0=Off, 1=On)
- Bit 1 Num Lock (0=Off, 1=On)
- Bit 2 Caps Lock (0=Off, 1=On)

**Side Effects:**

The A and X registers may be altered by this function.

**Application Data Transfer API**

For efficiency considerations, the PS2D User Module does not provide a function interface for loading the transfer buffer. Instead, the application loads the transfer buffer directly. The application is responsible for making sure no transfer is underway before loading the transfer buffer by calling `PS2D_TransferInProgress`. The application must guard against a buffer overrun using the `PS2D_TX_BUFFER_SIZE`. After the buffer is loaded, a transfer is started by invoking the `PS2D_StartTransfer` macro, specifying the transfer size as the only parameter.

**PS/2 Low Level Driver**

The PS2D User Module low level driver handles all data transfer between the application and the PS/2 Host. The driver is invoked by periodic calls to `PS2D_DoCommand` and `PS2D_SendNextByte`. `PS2D_DoCommand` must be called at least every 5 milliseconds. When `PS2D_TransferInProgress` indicates that a transfer is active, `PS2D_SendNextByte` should be called at intervals of not greater than 2 ms.

In order to guarantee proper responses to host commands, the application must not start a data transfer unless the preceding call to `PS2D_DoCommand` indicates that no command was not processed. Certain commands require additional data exchange for specific command parameters. Of course, the application must not start a new data transfer while an existing transfer is in progress.
**Mouse Application Support**

The following table displays a list of the commands that may be sent to the mouse. Before sending any other commands, check the status of the mouse - if the mouse is in stream mode, the host should disable the reporting of data. Command 0xF5 may be used to disable data reporting.

Table 4. Mouse Command Set

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>S=Supported</th>
<th>U=Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF (Reset)</td>
<td>The mouse responds to this command with &quot;acknowledge&quot; (0xFA) then enters Reset Mode.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xFE (Resend)</td>
<td>The host sends this command whenever it receives invalid data from the mouse. The mouse responds by resending the last packet it sent to the host. If the mouse responds to the &quot;Resend&quot; command with another invalid packet, the host may either issue another &quot;Resend&quot; command, issue an &quot;Error&quot; command, cycle the mouse's power supply to reset the mouse, or it may inhibit communication (by bringing the Clock line low). The action taken depends on the host.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF6 (Set Defaults)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then loads the following values: Sampling rate = 100, Resolution = 4 counts/mm, Scaling = 1:1, Disable Data Reporting. The mouse then resets its movement counters and enters stream mode.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF5 (Disable Data Reporting)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then disables data reporting and resets its movement counters. This only effects data reporting in Stream mode and does not disable sampling. Disabled stream mode functions the same as remote mode.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF4 (Enable Data Reporting)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then enables data reporting and resets its movement counters. This command may be issued while the mouse is in Remote Mode (or Stream mode), but it only effects data reporting in Stream mode.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF3 (Set Sample Rate)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then reads one more byte from the host. The mouse saves this byte as the new sample rate. After receiving the sample rate, the mouse again responds with &quot;acknowledge&quot; (0xFA) and resets its movement counters. Valid sample rates are 10, 20, 40, 60, 80, 100, and 200 samples/sec.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF2 (Get Device ID)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) followed by its device ID (0x00 for the standard PS/2 mouse.) The mouse should also reset its movement counters.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xF0 (Set Remote Mode)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then resets its movement counters and enters remote mode.</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0xEE (Set Wrap Mode)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then resets its movement counters and enters wrap mode.</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>
The following table displays a list of all the commands the host may send to the keyboard:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>S=Supported U=Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xEC (Reset Wrap Mode)</td>
<td>The mouse responds with &quot;acknowledge&quot; (0xFA) then resets its movement counters and enters the mode it was in before wrap mode (Stream Mode or Remote Mode.)</td>
<td>S</td>
</tr>
<tr>
<td>0xEB (Read Data)</td>
<td>The mouse responds with acknowledge (0xFA) then sends a movement data packet. This is the only way to read data in Remote Mode. After the data packets has been successfully sent, it resets its movement counters.</td>
<td>S</td>
</tr>
<tr>
<td>0xEA (Set Stream Mode)</td>
<td>The mouse responds with &quot;acknowledge&quot; then resets its movement counters and enters stream mode.</td>
<td>S</td>
</tr>
<tr>
<td>0xE9 (Status Request)</td>
<td>The mouse responds with &quot;acknowledge&quot; then sends the following 3-byte status packet (then resets its movement counters):</td>
<td>S</td>
</tr>
<tr>
<td>0xE8 (Set Resolution)</td>
<td>The mouse responds with acknowledge (0xFA) then reads one byte from the host and again responds with acknowledge (0xFA) then resets its movement counters. The byte read from the host determines the resolution.</td>
<td>S</td>
</tr>
<tr>
<td>0xE7 (Set Scaling 2:1)</td>
<td>The mouse responds with acknowledge (0xFA) then enables 2:1 scaling.</td>
<td>S</td>
</tr>
<tr>
<td>0xE6 (Set Scaling 1:1)</td>
<td>The mouse responds with acknowledge (0xFA) then enables 1:1 scaling.</td>
<td>S</td>
</tr>
</tbody>
</table>

**Keyboard Application Support**

The following table displays a list of all the commands the host may send to the keyboard:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>S=Supported U=Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF (Reset)</td>
<td>Keyboard responds with &quot;ack&quot; (0xFA), then enters &quot;Reset&quot; mode.</td>
<td>S</td>
</tr>
<tr>
<td>0xFE (Resend)</td>
<td>Keyboard responds by resending the last-sent byte. The exception to this is if the last-sent byte was &quot;resend&quot; (0xFE). If this is the case, the keyboard resends the last non-0xFE byte. This command is used by the host to indicate an error in reception.</td>
<td>S</td>
</tr>
<tr>
<td>0xFD (Set Key Type Make)</td>
<td>Disable break codes and typematic repeat for specified keys. Keyboard responds with &quot;ack&quot; (0xFA), then disables scanning (if enabled) and reads a list of keys from the host. These keys are specified by their set 3 make codes. Keyboard responds to each make code with &quot;ack&quot;. Host terminates this list by sending an invalid set 3 make code (for example, a valid command.) The keyboard then re-enables scanning (if previously disabled).</td>
<td>U</td>
</tr>
<tr>
<td>0xFC (Set Key Type Make/Break)</td>
<td>Similar to the Set Key Type Make command, except this one only disables typematic repeat.</td>
<td>U</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>S=Supported</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0xFB</td>
<td>Similar to the Set Key Type Make command and the Set Key Type Make/Break command, except this command only disables break codes.</td>
<td>U</td>
</tr>
<tr>
<td>0xFA</td>
<td>Keyboard responds with &quot;ack&quot; (0xFA). Sets all keys to their normal setting (generate scan codes on make, break, and typematic repeat)</td>
<td>U</td>
</tr>
<tr>
<td>0xF9</td>
<td>Keyboard responds with &quot;ack&quot; (0xFA). Similar to 0xFD, except applies to all keys.</td>
<td>U</td>
</tr>
<tr>
<td>0xF8</td>
<td>Keyboard responds with &quot;ack&quot; (0xFA). Similar to 0xFC, except applies to all keys.</td>
<td>U</td>
</tr>
<tr>
<td>0xF7</td>
<td>Keyboard responds with &quot;ack&quot; (0xFA). Similar to 0xFB, except applies to all keys.</td>
<td>U</td>
</tr>
<tr>
<td>0xF6</td>
<td>Load default typematic rate/delay (10.9cps / 500ms), key types (all keys typematic/make/break), and scan code set.</td>
<td>S</td>
</tr>
<tr>
<td>0xF5</td>
<td>Keyboard stops scanning, loads default values, and waits further instructions.</td>
<td>S</td>
</tr>
<tr>
<td>0xF4</td>
<td>Re-enables keyboard after disabled using previous command.</td>
<td>S</td>
</tr>
<tr>
<td>0xF3</td>
<td>Host follows this command with a one-argument byte that defines the typematic rate and delay.</td>
<td>S</td>
</tr>
<tr>
<td>0xF2</td>
<td>The keyboard responds by sending a two-byte device ID of 0xAB, 0x83. (0xAB is sent first, followed by 0x83.)</td>
<td>S</td>
</tr>
<tr>
<td>0xF0</td>
<td>Keyboard responds with &quot;ack&quot;, then reads argument byte from the host. This argument byte may be 0x01, 0x02, or 0x03 to select scan code set 1, 2, or 3, respectively. The keyboard responds to this argument byte with &quot;ack&quot;. If the argument byte is 0x00, the keyboard responds with &quot;ack&quot; followed by the current scan code set.</td>
<td>S</td>
</tr>
<tr>
<td>0xEE</td>
<td>The keyboard responds with &quot;Echo&quot; (0xEE).</td>
<td>S</td>
</tr>
<tr>
<td>0xED</td>
<td>The host follows this command with a one-argument byte that specifies the state of the keyboard's Num Lock, Caps Lock, and Scroll Lock LEDs.</td>
<td>S</td>
</tr>
</tbody>
</table>
Sample Firmware Source Code

For examples of this user module usage, please refer to the CY4623 Kit projects available at www.cypress.com/go/cy4623.

Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>DHA</td>
<td>Added Version History.</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.