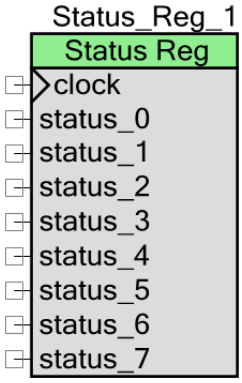
Status Register

1.90



Features

- Up to 8-bit Status Register
- Interrupt support

General Description

The Status Register allows the firmware to read digital signals.

When to Use a Status Register

Use the Status Register when the firmware needs to query the state of internal digital signals.

Input/Output Connections

This section describes the input connections for the status register. 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

Status register clock. The clock input signal is ignored for bits configured as Transparent.

status_0 - status_7 – Input *

Status register input. The firmware queries the input signals by reading the status register. The number of inputs depends on the **Inputs** parameter. These inputs may be left floating with no external connection. If nothing is connected to these lines, the component will assign a constant logic 0.

status[N:0] – Input *

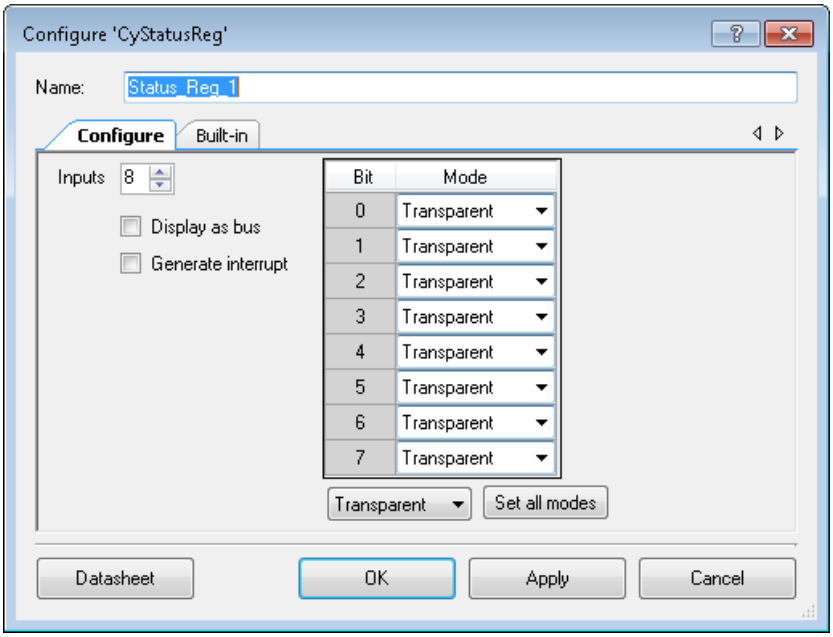
This optional input sweeps the individual input terminals into a single bus terminal. This pin is visible when the **Display as bus** parameter is enabled. N is the number of inputs - 1. This input may be left floating with no external connection. If nothing is connected to this line, the component will assign a constant logic 0.

intr – Output *

This optional pin is shown on the symbol when the **Generate interrupt** parameter is enabled. This option is only valid if less than 8 inputs are selected.

Component Parameters

Drag a Status Register onto your design and double-click it to open the **Configure** dialog.



Inputs

Number of input terminals (1 to 8). The default value is **8**.

Display as bus

This parameter displays the input as a bus instead of individual terminals. This option is unchecked by default.

Generate interrupt

This parameter displays the interrupt input on the symbol. It is not selected by default. An Interrupt is valid only if the number of inputs is less than 8. In addition to checking the **Generate Interrupt** box, the Interrupt mask must be set and the StatusReg_InterruptEnable()API must be called in order for the component to generate an interrupt. The Interrupt signal generation depends on the **Mode** parameter:

- **Transparent –** Goes high on the rising edge of the status input signal and goes low on the falling edge of the status input signal.

- **Sticky –** Goes high on the rising edge of the Clock (if the status input signal is high). It goes low after the Status register is read.

The interrupt generation is tied to the setting of status bits. This feature is built into the status register logic as the masking (Mask register) and OR reduction of status. Only the lower 7 bits of the status input can be used with the built-in interrupt generation circuitry.

Set all modes

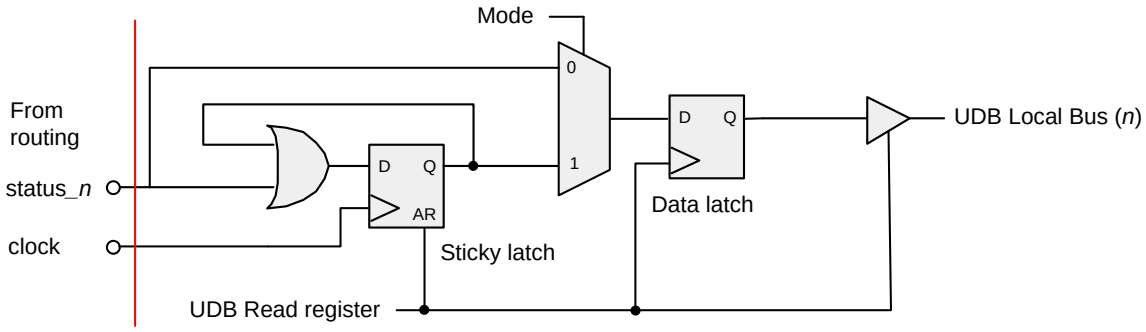
This button sets all bits to either **Transparent** or **Sticky** mode, depending on the mode selected from the combo box in the left hand side of this button.

Mode

These parameters are used to set specific bits of the Status Register to be held high after being registered, until a read is executed. That read clears all registered values. The settings are:

- **Transparent** – By default, a CPU read of this register transparently reads the state of the associated routing net and is asynchronous to the block clock. This mode can be used for transient state that is computed and registered internally in the UDB.
- **Sticky (Clear on Read)** – In this mode, the associated routing net is sampled on each cycle of the status and control clock. If the signal is high in a given sample, it is captured in the status bit and remains high, regardless of the subsequent state of the associated route. When CPU firmware reads the status register, the bit is cleared. The status register clearing is independent of mode and will occur even if the block clock is disabled; it is based on the bus clock and occurs as part of the read operation.

Figure 1. Behavior of Transparent versus Sticky Modes



Interrupt mask

This option is shown in the Configure dialog only when the **Generate interrupt** is checked. These parameters allow you to set the interrupt mask value for each bit in the Status Register. By default, the interrupt mask value is 0 (disabled).

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software.

By default, PSoC Creator assigns the instance name “Status_Reg_1” to the first instance of a status register in any given design. You can rename the component 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 function is “StatusReg.”

Functions

Function	Description
StatusReg_Read()	Reads the current value of the status register
StatusReg_InterruptEnable()	Enables the status register interrupt
StatusReg_InterruptDisable()	Disables the status register interrupt
StatusReg_WriteMask()	Writes the value assigned to the mask register
StatusReg_ReadMask()	Returns the current interrupt mask value from the mask register

uint8 StatusReg_Read (void)

Description: Reads the value of a status register.

Return Value: Returns the current value of a status register.

void StatusReg_InterruptEnable (void)

Description: Enables the status register interrupt. The default behavior is disabled. This is only valid if the status register generates an interrupt.

void StatusReg_InterruptDisable (void)

Description: Disables the status register interrupt. This is only valid if the status register generates an interrupt.

void StatusReg_WriteMask (uint8 mask)

Description: Writes the current mask value assigned to the status register. This is only valid if the status register generates an interrupt.

Parameters: mask: The bitwise value to write into the mask register. A 1 enables the corresponding bit in the status register to generate an interrupt. A 0 disables the corresponding bit in the status register from generating an interrupt.

uint8 StatusReg_ReadMask (void)

Description: Reads the current interrupt mask value assigned for the status register. This is only valid if the status register generates an interrupt.

Return Value: Returns the current bitwise value of the interrupt mask. A 1 in the corresponding bit of the status register enables generating an interrupt. A 0 in the corresponding bit of the status register disables generating an interrupt.

Sample Firmware Source Code

PSoC Creator provides numerous code examples that include schematics and example code in the Find Code Example 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 Code Example” topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Non PSoC 6 project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment. For PSoC 6, refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The Status Register component does not have any specific deviations.

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.

Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)		PSoC 6 (GCC)	
	Flash (Bytes)	RAM (Bytes)	Flash (Bytes)	RAM (Bytes)	Flash (Bytes)	RAM (Bytes)	Flash (Bytes)	RAM (Bytes)
Default	46	0	92	0	92	0	92	0

Functional Description

Low Power Mode Behavior

None of the Status Register content is retained during low power modes (sleep, deep sleep, and hibernate). Each bit of the Status Register component is initialized with a ‘0’ value when the device wakes up from low power mode.

DMA

The DMA component can be used to read data directly from Status Register. The DMA Wizard can be used to configure DMA operations as follows:

Name of DMA Source/Destination in DMA Wizard	Direction	DMA Req Signal	DMA Req Type	Description
StatusReg_Status_PTR	Source	N/A	N/A	Stores Status Register value.

Resources

The Status Register component uses one status cell in the UDB array.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.90.a	Updated MISRA section. Updated API Memory section. Added support for PSoC 6.	
1.90	Fixed MISRA violation	MISRA violation occurres in the StatusReg_WriteMask API if the number of inputs is less than 8.
1.80.c	Datasheet updates.	Clarified the API section (StatusReg_WriteMask, StatusReg_ReadMask APIs). Clarified the Component Parameters section (Generate interrupt, Interrupt mask parameters).
1.80.b	Datasheet updates.	Updated Figure 1 to better illustrate Transparent Mode versus Sticky Mode. Added low power mode behavior section.
	Updated the table appearance in the Configure dialog.	Corrected a display problem at 120 dpi resolution.
1.80.a	Updated datasheet with memory usage for PSoC 4.	
1.80	Added MISRA Compliance section.	The component does not have any specific deviations.
1.70	Added PSoC 5LP support.	
	Implemented StatusReg_InterruptEnable(), StatusReg_InterruptDisable(), StatusReg_WriteMask() and StatusReg_ReadMask() APIs.	To support interrupt functionality.
	Updated the Configure dialog.	Added Display as bus, Generate interrupt, Set all modes, Interrupt mask parameters and minor design changes.
	Added interrupt pin and display as bus option for input terminals. Also implemented DMA capabilities and Debug window support.	To have input terminals as bus and to support interrupt generation.
1.60	Updated the Configure dialog	Changed the Bit display and addressed minor Configure dialog issues
1.50.b	Datasheet edits	
1.50.a	Datasheet edits	
1.50	Updated the Configure dialog.	Created a customized interface. Added "Set All" buttons and changed Number of Inputs field to allow keyboard entry. Updated the dialog to comply with corporate standards.

© Cypress Semiconductor Corporation, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.