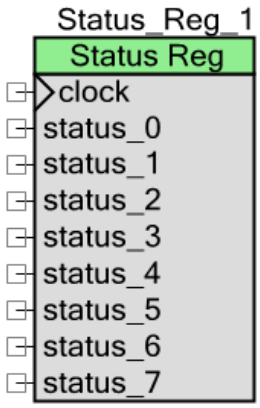
Status Register
1.70

Features

- Up to 8-bit Status Register



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 *

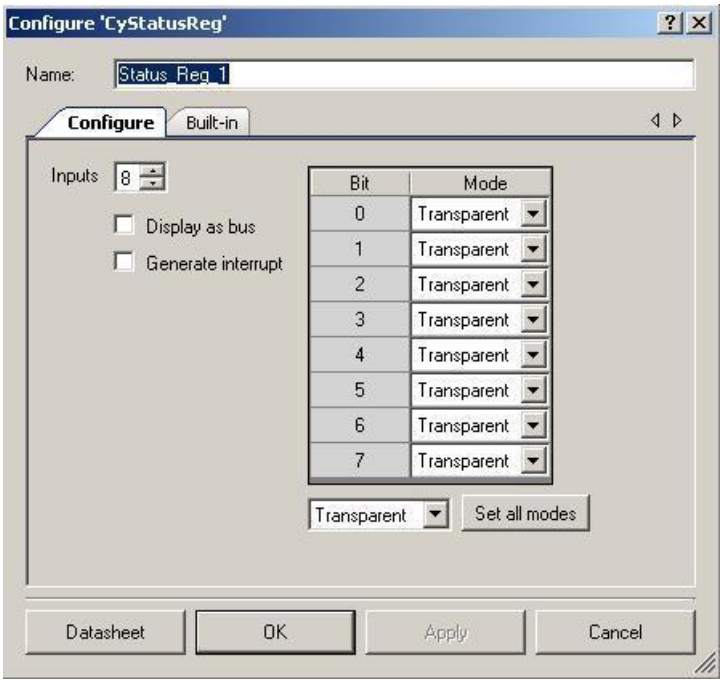
This optional input sweeps the individual input terminals into a single bus terminal. This pin is visible when the **Display as bus** parameter is enabled. N is the number of inputs - 1. This input may be left floating with no external connection. If nothing is connected to this line, the component will 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. This option is unchecked by default. Interrupt is valid only if the number of inputs is less than 8.

Set all modes

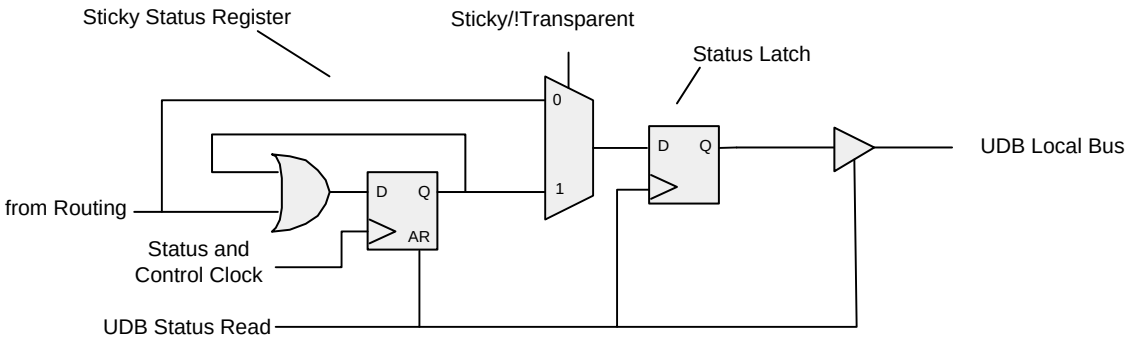
This button sets all bits to either **Transparent** or **Sticky** mode, depending on the mode selected in the combo box in the left hand side of this button.

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 **Generate interrupt** is checked. These parameters allow to set the interrupt mask value for each bit in the Status Register. By default, the interrupt mask value is 0.

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.”

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.

Parameters: None

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

Side Effects: None

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.

Parameters: None.

Return Value: None.

Side Effects: None.

void StatusReg_InterruptDisable (void)

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

Parameters: None.

Return Value: None.

Side Effects: None.

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 value to write into the mask register.

Return Value: None.

Side Effects: None.

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.

Parameters: None.

Return Value: Returns the current value of the interrupt mask.

Side Effects: None

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.

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.

Resources

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

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 5 (GCC)		PSoC 5LP (GCC)	
	Flash (Bytes)	RAM (Bytes)	Flash (Bytes)	RAM (Bytes)	Flash (Bytes)	RAM (Bytes)
Default	46	0	92	0	92	0

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.70.a	Minor datasheet edit.	
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, 2010-2015. 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.