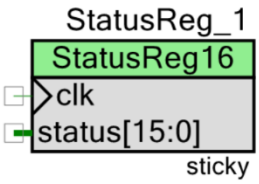
StatusReg16: 16-bit status register
0.0

Features



- Up to 16-bit Status Register
- 16-bit DMA access

General description

The StatusReg16(*) component implements 16-bit status register, which allows firmware to read digital signals. Functionality of the component is similar to the standard 8-bit status register, which has been expanded to 16-bit by combining two status registers in the adjacent UDB cells [1-3]. Such placement allows direct reading the register using atomic 16-bit access, providing better performance than when using two standard registers with 8-bit access. The component is not a substitute for the standard status register, which is preferred way for 8-bit applications.

When to use StatusRegister component

Component was developed for testing custom UDB components with wide digital output bus. It can be used whenever 16-bit data must be read by firmware, such as array of digital input pins or digital ports. Component is compatible exclusively with PSoC5 and was tested using CY8CKIT-059 PSoC5LP Prototyping Kit. Demo projects are provided.

* Hereafter referred to as “StatusRegister”

Input-output connections

clk – clock input

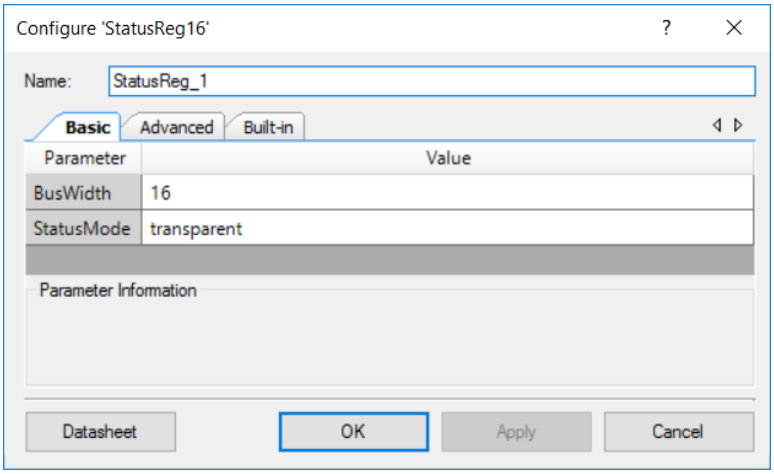
This pin is present when the **StatusMode** parameter is set to **sticky** mode. It is not available in **transparent** mode. When visible, the pin must be connected to valid digital source. The bus is sampled on the rising edge of this input.

status[N:0] – input

The StatusRegister contains up to 16 digital inputs, which are displayed as bus. The firmware queries the input signals by reading the status register. The number of input bits is set by the **BusWidth** 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.

Parameters and Settings

Basic dialog provides following parameters[*]:



BusWidth (unt8)

Input bus width (number of digital inputs). Valid range: [1 to 16]. Default bus width is 16. Bit 0 corresponds to the LSB.

StatusMode [transparent / sticky]

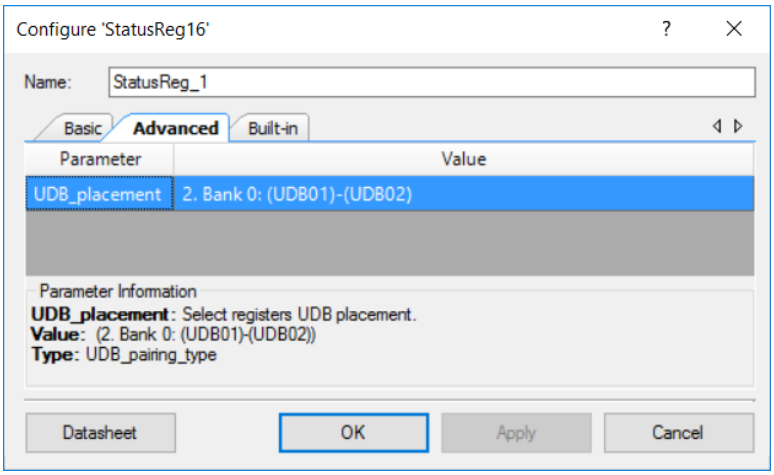
Sets all bits of the register to the **transparent** or **sticky** mode. Default setting is **transparent**.

- **transparent** – firmware reads bus state asynchronously to the block clock
- **sticky** – the bus state is sampled on the control clock. The signal captured in the status bit remains in that state, until cleared by reading the status register.

The selected status mode is applied to all bits simultaneously. Component does not support per-bit control mode settings.

[*] Component was intentionally compiled using Creator 4.0 for compatibility with older versions.

Advanced dialog provides following parameters:



UDB_placement

Shows the list of supported UDB placements for two contiguous 8-bit registers combined into a 16-bit register. Default value is Bank 0, (UDB00)-(UDB01). Selection of the UDB placement is done manually, component does not provide automatic UDB cell placement. The placement can't be changed during the run-time. Once UDB placement is selected, it can't be used by another StatusRegister. The user must select non-intersecting locations for each StatusRegister if multiple instances of the component are present in the design. See **Implementation** section for details.

Application Programming Interface

Function	Description
`StatusReg_Read()`	Reads value assigned to the control register

uint16 ControlReg_Read(void)

Description: Reads the value of the status register

Parameters: none

Return Value: Returns the current value of the status register

Implementation

The StatusRegister is implemented using a pair of 8-bit status registers placed into adjacent (contiguous) UDB cells[*], Figure 1.

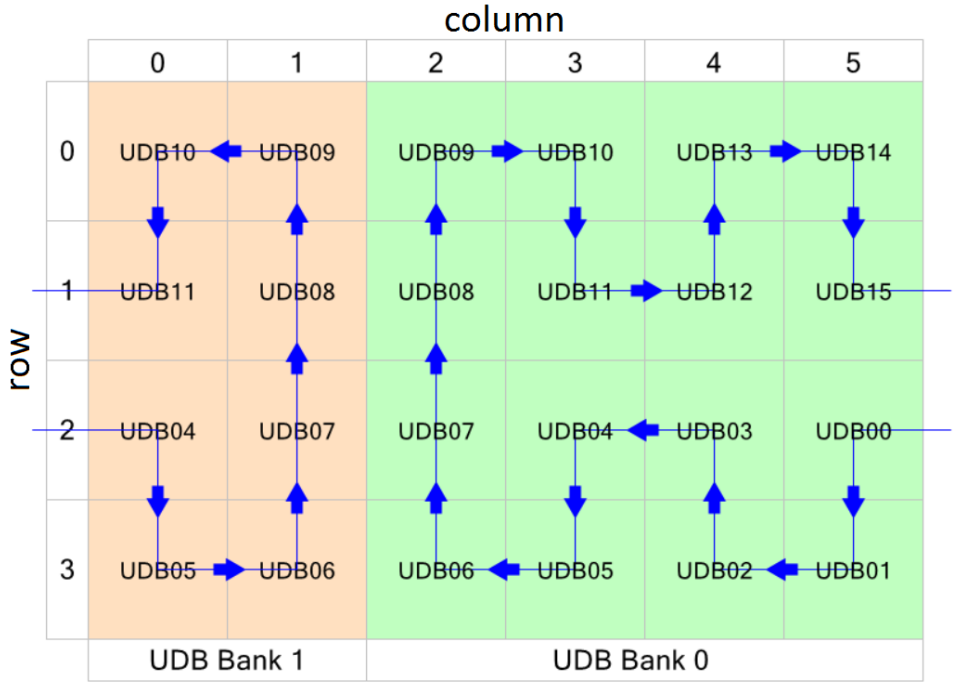


Figure 1. PSoC5 UDB map showing UDB cells arranged in rows and columns. Each UDB cell is defined by (row, column). For example, cell UDB00 has placement (2, 5). Arrows connect contiguous UDB cells, which can be paired to form a single 16-bit StatusRegister (for example UDB00 and UDB01). Note, that UDB cells from a different UDB Banks can't be paired.

Contiguous placement allows for atomic reading of the StatusRegister, as well as 16-bit DMA access. By default, the fitter mechanism of the Creator does not guarantee automatic contiguous placement of the registers, so their mapping into UDB space should be done by hard placement, using the control file directives:

```
csattribute placement_force of \ControlRegister_1:sStatus:byte0:Status\ : label is "U(2,5)";
csattribute placement_force of \ControlRegister_1:sStatus:byte1:Status\ : label is "U(3,5)";
```

In this example, the byte0 of the StatusRegister is placed into cell UDB00, located at row/column coordinates U(2, 5); and byte1 is placed into next consecutive cell UDB01, located at coordinates U(3, 5). Such placement has to be manually performed by the user at design time, and won't be affected by the fitter optimization process.

[*] The concept can be expanded up to 32-bit register, see [1-3].

To simplify the assignment, the Advanced dialog page offers **UDB_placement** selector, which automatically generates control file for the component (Figure 2). It lists available placement options, which can potentially be used by the StatusRegister. Note, that once UDB placement is selected, it can't be used by another StatusRegister. To avoid build errors, user must select non-intersecting locations for each of the StatusRegister in the project if multiple instances of the component are present in the design.

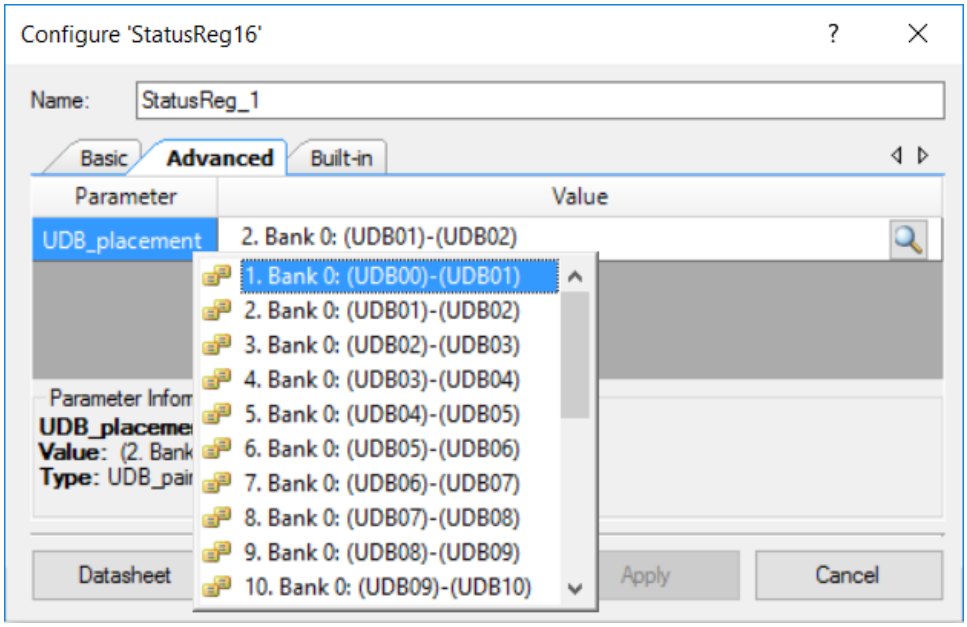


Figure 2. UDB manual placement options, showing a Bank and a pair of contiguous UDB cells. If multiple instances of the StatusRegister component are used in the project, each other must be assigned a different UDB placement.

The StatusRegister supports many, but not all of the features available to the stock control register. For 8-bit applications it is recommended to use standard 8-bit control register.

Features not implemented

- Per-bit status mode (transparent, sticky)
- Interrupts

Interrupts are not implemented in current version of the component. When enabled in a standard 8-bit status register, interrupt consumes one bit, creating a "bit hole". That renders it a cumbersome task to map two 7-bit status registers into a wide StatusRegister. Reading such StatusRegister by firmware would require additional steps of bits shifting and masking, which yields no advantages compared to using two standard status registers.

Resources

In 9 to 16-bit mode component uses two status cells, in 1 to 8-bit mode – one status cell.

DMA

DMA can be used to read data directly from the StatusRegister using destination address StatusRegister_Status_PTR. 16-bit DMA can't cross Bank boundary [3].

Low Power mode

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

Sample Firmware Source Code

Basic example of the StatusRegister reading the state of the digital input ports by firmware is shown on Figure 3. For advanced DMA project see **Appendix 1**.

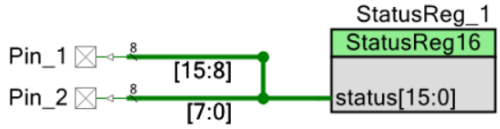


Figure 3. Sampling two 8-bit digital input ports using StatusRegister

Component Changes

Version	Description of changes	Reason for changes/impact
0.0	Version 0.0 is the first beta release of the component	

References

1. 32-bit parallel non-contiguous GPIO write technique, https://community.cypress.com/message/177785
2. Control register be expanded to 16bit or 32bit?, https://community.cypress.com/message/167599
3. PSoC 5LP UDB Placement Cheatsheet, https://imgur.com/a/IYeba

Appendix 1

DMA transfer from StatusRegister to RAM

PSoC5 example of DMA transfer from StatusRegister to RAM is shown on Figure 4[(*)].

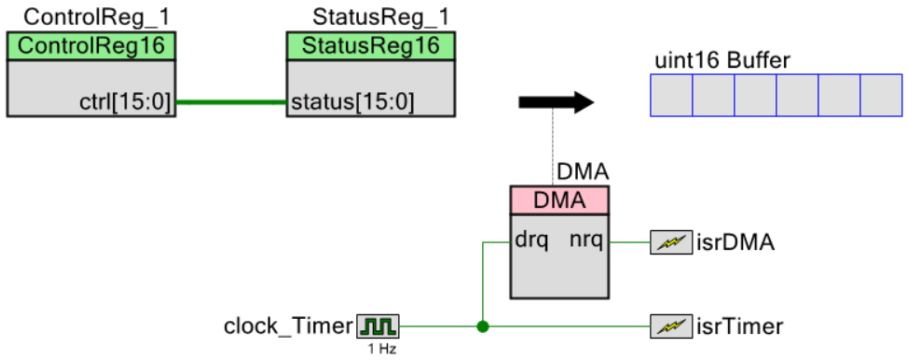


Figure 4. Project schematic. DMA transfers data from StatusReg_1 to RAM buffer on each Timer clock. When DMA transfer is completed, the program terminates.

DMA transfers data from 16-bit StatuslReg_1 to RAM Buffer on each clock_Timer. DMA configuration is listed below. DMA is configured for 2-byte transfer on single burst. It increments destination address on each data request, and produces **nrq** pulse when completed. The ControlReg_1 serves as source of digital data for the StatusReg_1.

```
uint8_t DMA_Chan;                          // variable declarations for DMA
uint8_t DMA_TD[1];                         //

#define DMA_BYTES_PER_BURST    (2u)        // transfer 2 bytes
#define DMA_REQUEST_PER_BURST  (1u)        // per each clock
#define DMA_SRC_BASE  (CYDEV_PERIPH_BASE)  // from hardware
#define DMA_DST_BASE  (CYDEV_SRAM_BASE)    // to RAM

DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                                           HI16(DMA_SRC_BASE),HI16(DMA_DST_BASE));
DMA_TD[0] = CyDmaTdAllocate();

/* transfer data to RAM Buffer, produce TERMOUT pulse and stop DMA */
CyDmaTdSetConfiguration(DMA_TD[0], NO_SAMPLES*2, DMA_DISABLE_TD, DMA__TD_TERMOUT_EN
                                                                 | TD_INC_DST_ADR);

CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)StatusReg_1_Status_PTR),
                             LO16((uint32)Buffer));

CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);
CyDmaChEnable(DMA_Chan, 1);
```

[*] Associated demo project: SReg16-DMA-RAM_01b.cyproj

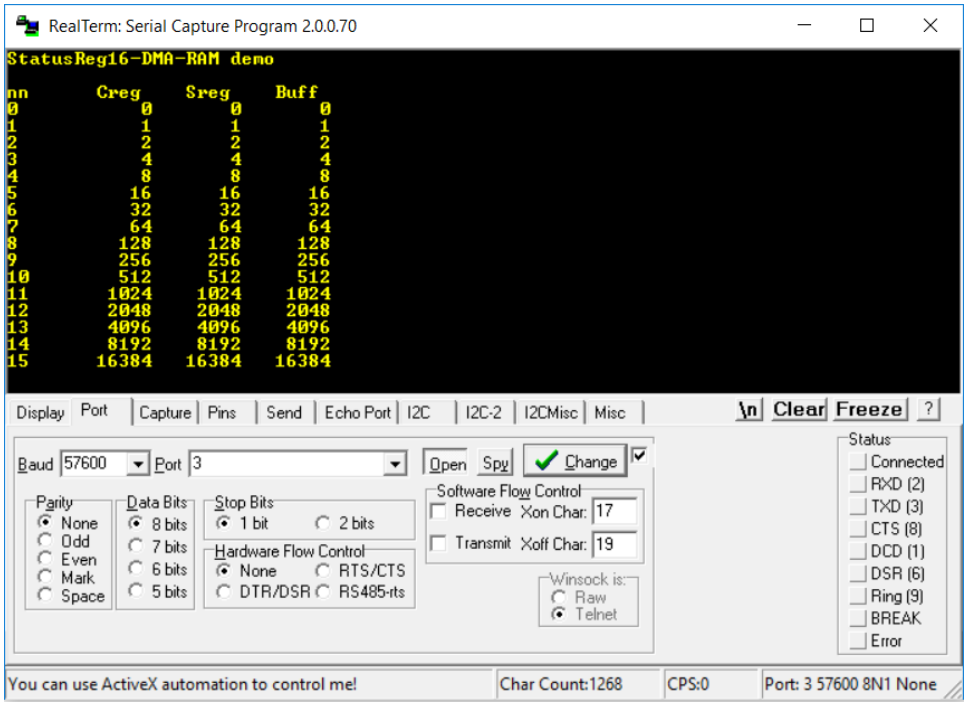


Figure 5. UART Terminal output showing transfer counter, content of the ControlReg_1 and StatusReg_1 and RAM Buffer.

Data is sent to the external Terminal program on each Timer interrupt using UART (Figure 5).