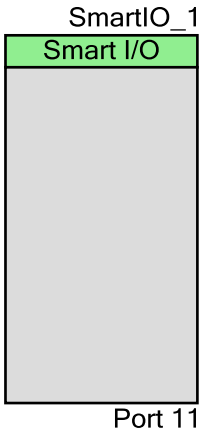
Smart I/O™ 1.10

Features



- Provides glue logic functionality at I/O ports
- Deep Sleep and Hibernate low-power mode operation
- Flexible Look-up Table array for user-defined logic
- 8-bit multi-function counter and shifter
- Combinatorial and clocked (registered) operation
- Low-latency deterministic delays
- Simple user interface for routing signals in the fabric

General Description

The Cypress Smart I/O Component provides a programmable logic fabric interposed between a General Purpose Input/Output (GPIO) port and the connections to it from various peripherals and UDB sources.

Inputs to the chip from the GPIO port can be logically operated upon before being routed to the peripheral blocks and connectivity of the chip. Likewise, outputs from the peripheral blocks and internal connectivity of the chip can be logically operated upon before being routed to the GPIO port.

The programmable logic fabric of the Smart I/O Component can be purely combinatorial or registered with a choice of clock selection. Its functionality is completely user-defined and each path can be selectively bypassed if certain routes are not required by the fabric.

Each Smart I/O Component is associated with a particular GPIO port and consumes the port entirely. If the Component is not used, then the Smart I/O functionality for that port is bypassed.

The Component can operate down to Deep Sleep (and Hibernate for select devices) low-power mode and can wake up the chip using the port interrupt, when required.

When to Use a Smart I/O Component

The Smart I/O Component should be used whenever simple logic operations and routing are required to be performed on signals going to and coming from IO pins. Typical applications include:

Clock divider (Deep Sleep capable)

The Smart I/O Component can be used to implement clock dividers in various power domains. For example, either the LUTs or the data unit can be used to generate very low frequency clock frequencies from LFCLK while operating in Deep-Sleep mode.

Change routing to/from GPIOs

Signals from fixed-function peripherals, such as a TCPWM, can be rerouted to non-dedicated pins on the same port by using the Smart I/O Component. For example, a TCPWM (line) signal can be routed to a non-dedicated pin on the port by passing through one of the LUTs in the Smart I/O Component.

Polarity inverter

Peripheral signals, such as the UART Tx (SCB) can be inverted before going out to a pin. The signal can go through a LUT configured as an inverter, which allows polarity inversion without consuming external hardware resources.

Signal/Clock buffer

When an input signal to a GPIO port is required to drive a heavier load than one pin can drive by itself, the Smart I/O Component can be used to drive the same signal through two GPIO buffers. The signal is duplicated through two LUTs and the LUT outputs are used to drive two GPIO buffers with the same signal. The two pins are then combined externally to drive a higher load.

Pattern detection

The LUTs may be configured to detect a particular pattern on the input signals (for example, logic to detect if four input signals are all low). The resulting LUT output is routed to a pin on the port, which is configured to generate a port interrupt. This is especially useful during Deep Sleep to generate a wake-up signal to the chip.

Logic elements

The Smart I/O LUTs can be used as general logic elements used to build custom functions. Examples include, four-to-two priority encoders, shift registers, glitch filters, and gates, etc.

Quick Start

The Smart I/O Component is a port-wide resource that has a close relationship with the port to which it is dedicated. Hence the connectivity of peripherals to the IOs of a Smart I/O Component will differ based on which device and which port is used. To use the Smart I/O Component, drag and drop it from the Component Catalog to the schematic.

1. Choose the port on which it is meant to be operated, and configure the source selections.
2. Connect the pins, peripheral terminals, and/or UDB signals to the Smart I/O Component.
3. Build and compile.

The Component will be configured as specified in the Configure dialog when you call the Component's Start() function. If you wish to change the behavior of the Smart I/O Component at run-time, you may dynamically reconfigure these using the provided API functions.

Note Run-time configuration must be performed with care as all the signals are set to bypass during reconfiguration.

Note The pins for the port where the Smart I/O Component is located may not be available in certain devices. You may still use the hardware for performing internal logic within the chip, but you cannot bring these out to the port's pins. Choose from the list of possible ports that the Smart I/O is located and avoid assigning pins to the Component.

Input/Output Connections

This section describes the various input and output connections for the Smart I/O Component. An asterisk (*) in the list of IOs indicates that the IO may be hidden on the symbol under the conditions listed in the description of that IO.



data[7…0] * - Input/Output

These terminals are located on the left side of the Component symbol, and allows connections to chip resources such as dedicated peripheral terminals or routed UDB signals. The terminals can be configured to be inputs or outputs, but not both. Bidirectional signals are not supported.

The direction of the terminals is defined using the **data 7…0 direction** parameter.

The legally allowed connections are defined on a port by port basis and are derived from the chosen port (refer to **Port** parameter) and the chosen terminal functionality (refer to **data 7…0 source** parameter).

A terminal is hidden if the channel is bypassed. If the **data 7…0 direction** parameter is set to **None**, the corresponding terminal will not be usable in your design.

gpio[7…0] * - Input/Output

These terminals are located on the right side of the Component symbol, and allows connections to the GPIO on the port. The terminals can be configured to be inputs or outputs, but not both. Bidirectional signals are not supported.

The direction of the terminals is defined using the **gpio 7…0 direction** parameter.

A terminal is hidden if the line is bypassed. If the **gpio 7…0 direction** parameter is set to **None**, the corresponding terminal will not be usable in your design.

clock * - Input

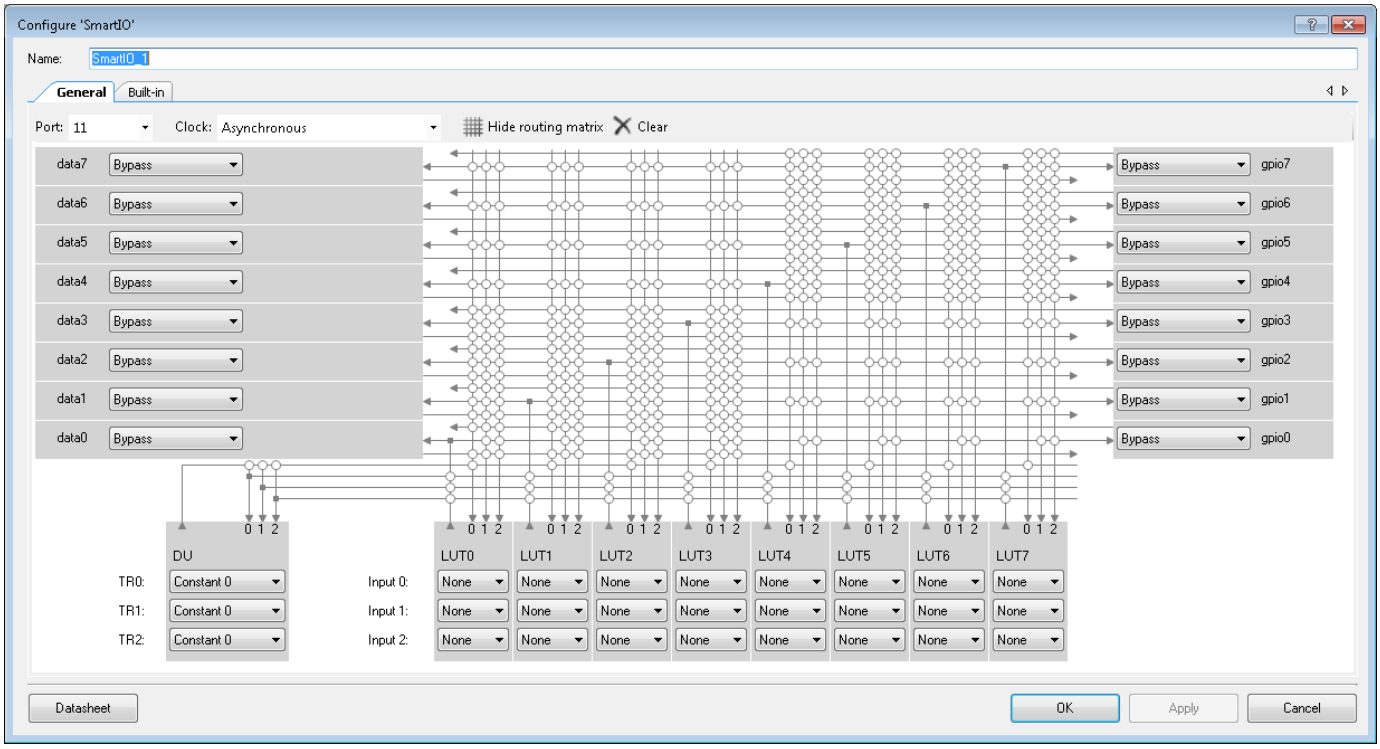
This terminal is available if the Component **Clock** parameter is set to **Divided Clock (Active), (Deep-Sleep), or (Hibernate)**. Only a clock Component may be connected to this terminal. All other **Clock** parameter selections automatically configure the source internally in the Component.

Component Parameters

Drag a Smart I/O Component onto the design schematic and double click it to open the **Configure** dialog.

General Tab

The **General** tab is used to define the general settings and the routing configuration of the Component.



This tab contains the following parameters:

Port

Valid peripheral connections of a Smart I/O Component is port and device specific. This parameter allows selecting a port that supports Smart I/O.

Clock

Selects the clock source used to drive all sequential logic in the block. This clock is global within the Smart I/O Component instance and have differing constraints suited to different applications. Refer to the Functional Description section for more information.

- **gpio 7…0** – Uses the selected gpio signal as the clock source. This signal may not be used in LUT inputs if used as the clock source. This clock can be used to drive the sequential elements during all power modes.

- **data 7…0** – Uses the selected data (peripheral or UDB) signal as the clock source. This signal may not be used in LUT inputs if used as the clock source. This clock can only be used during chip active and sleep modes.
- **Divided clock (Active)** – Divided clock from HFCLK. This clock is operational only in chip Active and Sleep modes. Sequential elements will be reset when entering Deep-Sleep or Hibernate mode and at POR.
- **Divided clock (Deep-Sleep)** – Divided clock from HFCLK. This clock is operational only in chip Active and Sleep modes. Sequential elements will be reset when entering Hibernate mode and at POR.
- **Divided clock (Hibernate)** – Divided clock from HFCLK. This clock is operational only in chip Active and Sleep modes. Sequential elements will be reset only at POR.
- **LFCLK** – Low Frequency clock (either from ILO or WCO). This clock operates during chip Active, Sleep and Deep-sleep and allows the Smart I/O sequential logic to be clocked during those power modes.
- **Asynchronous** – If the Smart I/O Component is used purely for combinatorial logic, this option allows the block to conserve power by not using a clock. There are no constraints to power modes with this selection.
- **Clock gated** – This configuration disables the clock connection and should only be used when turning off the block, or if making run-time reconfiguration for clock sensitive applications.

Hide routing matrix / Show routing matrix

This button hides or displays the Smart I/O routing matrix. If the routing matrix is shown, you may click on the switches in the fabric to make input connections to the LUTs.

Clear button

Click on this button to reset the routing matrix. All Data Unit (DU) and LUT inputs will be cleared and the data/gpio terminal directions will become bypassed.

Data configuration

data 7…0 source

When a **Port** is specified, these parameters allow each of the **data** terminals to choose a functionality allowed on that terminal. You may choose one of the supported options presented in the Component configuration dialog and then connect your peripheral or UDB signal to the exposed terminal.

For example, if you choose “SCB[0].uart_rx” as your data source in Smart I/O, you may then connect an SCB Component’s “uart_rx” terminal to it.

Note This parameter will not check for a valid connection. It is used for showing legal connections and to provide an easy way to label the functionality of the terminals. All options shown in this parameter are legal connections to that particular terminal.

data 7…0 direction

Defines the direction of the specified **data** terminal. Valid options are:

- **Input (Async/Sync)** – Changes the direction of the terminal to be input type and allows connecting the matching peripheral/UDB signal to it. Input signals can be used as inputs to the LUTs. These can be either synchronized to the Component clock or remain asynchronous.
- **Output** – Changes the direction of the terminal to be output type and allows connecting the matching peripheral/UDB signal to it. Output signals can only be driven by the corresponding LUT outputs.
- **Bypass** – The line is bypassed and do not go through the Smart I/O routing fabric. i.e. Connection between the chip resource and the pin are directly connected. **Bypass** option frees this line and allows PSoC Creator to use this pin location for placing resources not meant to be used in the Smart I/O fabric.
- **None** – The data terminal is consumed and cannot be used for connecting chip resources to it. This option is chosen automatically if the corresponding gpio terminal on the channel is specified as either **Input** or **Output**. You may use the terminal if the data direction is set to **Output** or **Input** (only allowed if corresponding gpio is **Output**).

GPIO Configuration

gpio 7…0 direction

Defines the direction of the specified **gpio** terminal. Valid options are,

- **Input (Async/Sync)** – Changes the direction of the terminal to be input type and allows connecting an input GPIO pin to it. Input signals can be used as inputs to the LUTs. These can be either synchronized to the Component clock (**Sync**) or remain asynchronous (**Async**).
- **Output** – Changes the direction of the terminal to be output type and allows connecting an output GPIO pin to it. Output signals can only be driven by the corresponding LUT outputs.
- **Bypass** – The line is bypassed and do not go through the Smart I/O fabric. i.e. Connection between the chip resource and the pin are directly connected. **Bypass** option frees this line and allows PSoC Creator to use this pin location for placing resources not meant to be used in the Smart I/O fabric.

- **None** – The gpio terminal is consumed and cannot be used for connecting a GPIO Component to it. This option is chosen automatically if the corresponding data terminal on the channel is specified as either **Input** or **Output**. You may use the terminal if the gpio direction is set to **Output** or **Input** (only allowed if corresponding data is **Output**).

DU Input Configuration

DU TR0, TR1, TR2

Defines the inputs for DU triggers TR0, TR1 and TR2. The purpose of the triggers are dependent on the chosen Opcode as specified in the **Data Unit** tab. The triggers are active high.

- **Constant 0** – Default input selection. This effectively makes the trigger input to not perform any operation on the DU.
- **Constant 1** – Ties the trigger high, which activates and maintains the chosen DU operation.
- **DU** – Feeds back the single 1-bit DU output.
- **LUT 0…7** – Accepts any of the LUT outputs as an input.

LUT Input Configuration

LUT 7…0 input 0

Defines the input 0 of the specified LUT. The actual valid selection depends on the enabled/used resources and terminals in your design. The LUT input 0 may accept any LUT outputs as an input **with the exception of LUT 0**. Instead, it may accept the DU output as an input.

- **DU** – The 1-bit output from the DU.
- **LUT 1…7** – Accepts the outputs of LUT1 to LUT 7.
- If using LUT 0…3, **gpio/data 0…3** signals are allowed as potential inputs.
- If using LUT 4…7, **gpio/data 4…7** signals are allowed as potential inputs.

LUT 7…0 input 1

Defines the input 1 of the specified LUT. The actual valid selection depends on the enabled/used resources and terminals in your design.

- **LUT 0…7** – Accepts the outputs of LUT0 to LUT 7.
- If using LUT 0…3, **gpio/data 0…3** signals are allowed as potential inputs.
- If using LUT 4…7, **gpio/data 4…7** signals are allowed as potential inputs.

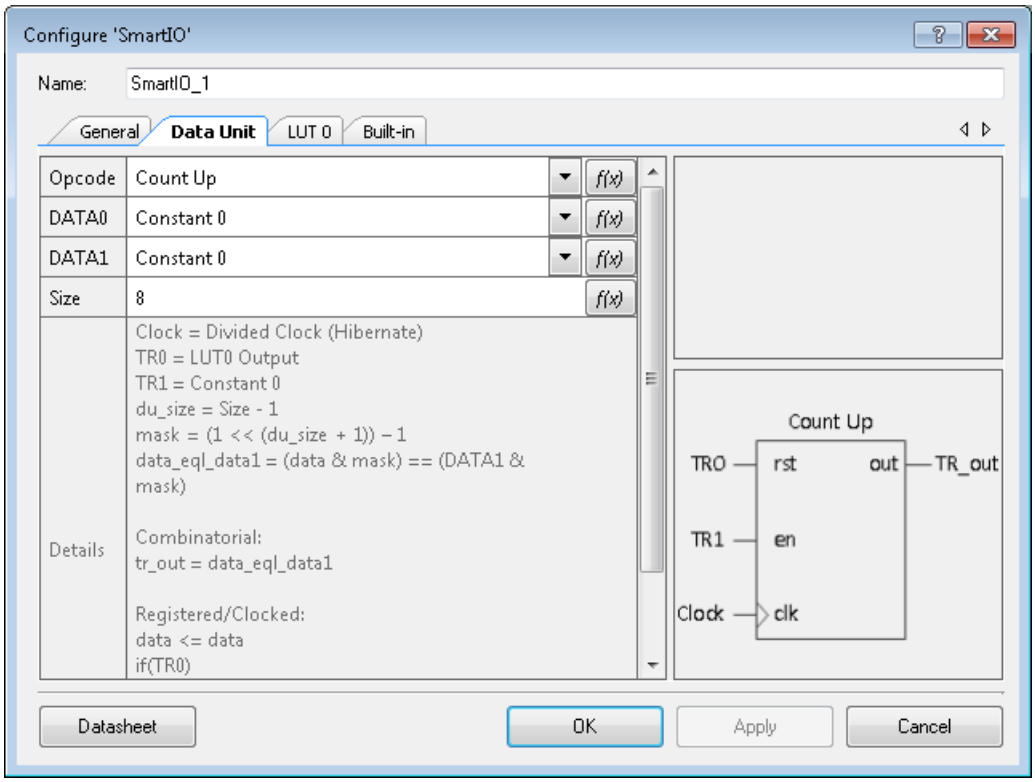
LUT 7…0 input 2

Defines the input 2 of the specified LUT. The actual valid selection depends on the enabled/used resources and terminals in your design.

- **LUT 0…7** – Accepts the outputs of LUT0 to LUT 7.
- If using LUT 0…3, **gpio/data 0…3** signals are allowed as potential inputs.
- If using LUT 4…7, **gpio/data 4…7** signals are allowed as potential inputs.

Data Unit Tab

When the Data Unit (DU) in the **General** tab is configured to accept an input other than a Constant 0, the corresponding **Data Unit** tab will appear.



This tab contains the following parameters:

Opcode

Defines the Data Unit operation. Each opcode performs a unique function that can be controlled using the DU trigger inputs TR0, TR1 and TR2.

Note Not all trigger inputs are required for a chosen Opcode. Refer to the pseudo Verilog code in the **Details** window and also to the Functional Description section for more information.

- **Count Up** – Implements an 8-bit One-shot Up Counter. The DU output goes high when the internal DU working data register is equal to DU DATA1 value.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the increment operation
- **Count Down** – Implements an 8-bit One-shot Down Counter. The DU output goes high when the internal DU working data register is equal to 0.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the decrement operation
- **Count Up Wrap** – Implements an 8-bit Up Counter that wraps when it reaches DU DATA1 value. The DU output is a single clock pulse when the internal DU working data register is equal to DU DATA1 value.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the increment operation
- **Count Down Wrap** – Implements an 8-bit Down Counter that wraps when it reaches 0. The DU output is a single clock pulse when the internal DU working data register is equal to 0.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the decrement operation
- **Count Up/Down** – Implements an 8-bit Up/Down Counter. The DU output goes high when the internal DU working data register is equal to either DU DATA1 register or if it is equal to 0.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the increment operation
  - TR2 = enable signal to start the decrement operation
- **Count Up/Down Wrap** – Implements an 8-bit Up/Down Counter that wraps when it reaches either DU DATA1 (count up) or when it reaches 0 (count down). The DU output goes high when the internal DU working data register is equal to either DU DATA1 register or if it is equal to 0.
  - TR0 = reset signal that resets the working register to DU DATA0 value
  - TR1 = enable signal to start the increment operation
  - TR2 = enable signal to start the decrement operation
- **Rotate Right** – Implements a right circular shift register. The DU output is the LSB of the working register, which also gets fed back to the MSB of the working register.
  - TR0 = load signal that loads the working register with DU DATA0 value

- TR1 = enable signal to start the shift right and rotate operation

- **Shift Right** – Implements a right shift register. The DU output is the LSB of the working register.
  - TR0 = load signal that loads the working register with DU DATA0 value
  - TR1 = enable signal to start the shift right operation
  - TR2 = shift in value that gets inserted into the MSB of the working register
- **DATA0 & DATA1** – Implements a bitwise AND operation on the DU working register and DU DATA1. The DU output is high if the result of the operation is true.
  - TR0 = load signal that loads the working register with DU DATA0 value
- **Majority 3** – Implements a shift register with a majority 3 comparison. The DU output will go high if the contents of the working register is equal to 0x03, 0x05, 0x06 or 0x007.
  - TR0 = load signal that loads the working register with DU DATA0 value
  - TR1 = enable signal to start the shift right operation
  - TR2 = shift in value that gets inserted into the MSB of the working register
- **Match DATA1** – Implements a shift register with a match DU DATA1 value comparison. The DU output will go high if the contents of the working register is equal to DU DATA1.
  - TR0 = load signal that loads the working register with DU DATA0 value
  - TR1 = enable signal to start the shift right operation
  - TR2 = shift in value that gets inserted into the MSB of the working register

DATA0

Defines the DU DATA0 register source. This value is often used as the initial/reset value that is loaded into the DU working register when TR0 signal is high.

- **Constant 0** – Source is constant 0x00
- **data[7:0]** – Sourced from all 8 data terminals of the Smart I/O, allowing internal chip signals to be directly loaded into DATA0
- **gpio[7:0]** – Sourced from all 8 gpio terminals of the Smart I/O, allowing external signals to be directly loaded into DATA0
- **DU Reg** – Sourced from the DU Register, which is accessible by the CPU

DATA1

Defines the DU DATA1 register source. This value is often used as the comparison value that gets applied to the DU working register.

Note DU DATA1 is not necessary for all Opcodes.

- **Constant 0** – Source is constant 0x00
- **data[7:0]** – Sourced from all 8 data terminals of the Smart I/O, allowing internal chip signals to be directly loaded into DATA1
- **gpio[7:0]** – Sourced from all 8 gpio terminals of the Smart I/O, allowing external signals to be directly loaded into DATA1
- **DU Reg** – Sourced from the DU Register, which is accessible by the CPU

Register Value

Defines the 8-bit DU Reg value. This value is used as a source for DATA0 and/or DATA1.

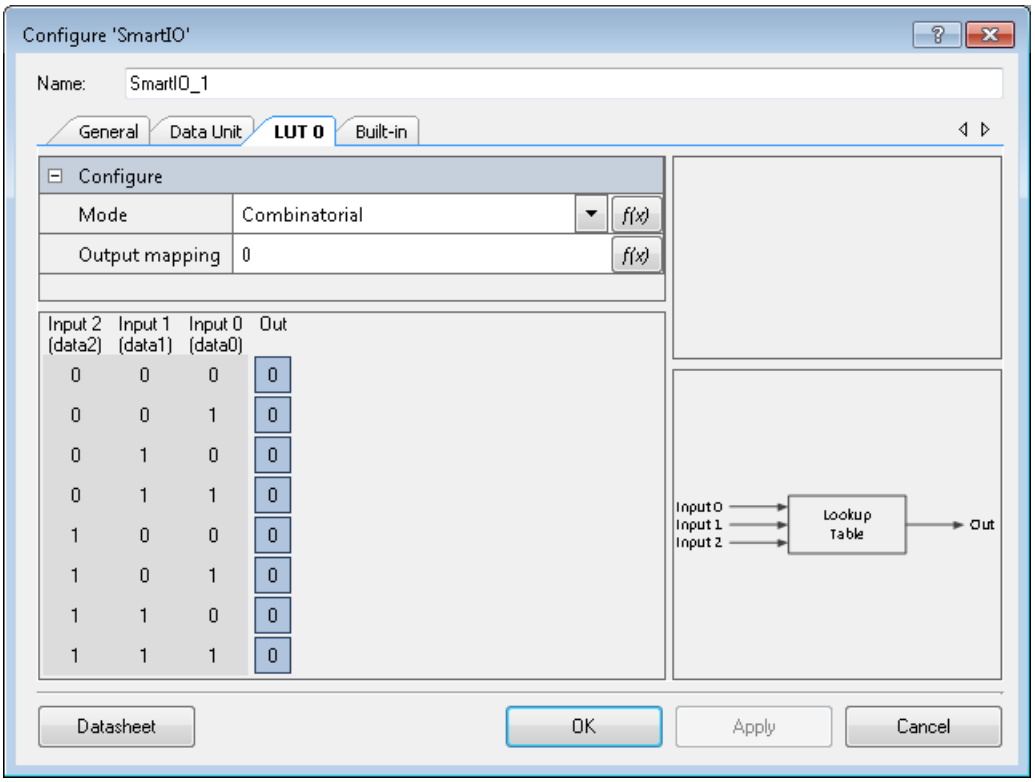
Note DU Reg is available only if either DATA0 or DATA1 are configured to be sourced from it.

Size

Defines the bit size operation to be performed by the data unit. Valid range is from 1 to 8 bits.

LUT Tabs

When a LUT in the **General** tab is configured to accept an input, the corresponding LUT configuration tab will appear.



This tab contains the following parameters:

LUT 7…0 Output mapping

Defines the lookup truth table of the 3-to-1 LUT. The state on the three inputs (input 0, 1 and 2) are translated to an output value according to this truth table.

Note If the LUT is used to operate on a single signal (e.g. to invert a signal), then that signal must be connected to all 3 inputs of the LUT.

LUT 7…0 Mode

The LUTs can be configured in one of four modes:

- **Combinatorial** – The LUT is purely combinatorial. The LUT output is the result of the LUT mapping truth table, and will only be delayed by the LUT combinatorial path.
- **Gated Input 2** – The LUT input 2 is registered. The other inputs are direct connects to the LUT. The LUT output is combinatorial. You may use the output to feed back into input 2.
- **Gated Output** – The inputs are direct connects to the LUT but the output is registered.
- **Set/reset flip-flop** – The inputs and the LUT truth table are used to control an asynchronous S/R flip-flop.

Application Programming Interface

By default, PSoC Creator assigns the instance name **SmartIO_1** to the first instance of a component in a given design. You can rename it 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 table is **SmartIO**.

Port channel selection constants

Description

Constants to be passed as "portChannel" parameter in SmartIO_SetBypass() function.

Macros

- #define SmartIO_CHANNEL_NONE 0x00UL
- #define SmartIO_CHANNEL0 0x01UL
- #define SmartIO_CHANNEL1 0x02UL
- #define SmartIO_CHANNEL2 0x04UL
- #define SmartIO_CHANNEL3 0x08UL
- #define SmartIO_CHANNEL4 0x10UL
- #define SmartIO_CHANNEL5 0x20UL
- #define SmartIO_CHANNEL6 0x40UL
- #define SmartIO_CHANNEL7 0x80UL
- #define SmartIO_CHANNEL_ALL 0xffUL

Macro Definition Documentation

#define SmartIO_CHANNEL_NONE 0x00UL

Do not bypass any channels

#define SmartIO_CHANNEL0 0x01UL

Channel 0 (data0 <-> gpio0)

#define SmartIO_CHANNEL1 0x02UL

Channel 1 (data1 <-> gpio1)

#define SmartIO_CHANNEL2 0x04UL

Channel 2 (data2 <-> gpio2)

#define SmartIO_CHANNEL3 0x08UL

Channel 3 (data3 <-> gpio3)

#define SmartIO_CHANNEL4 0x10UL

Channel 4 (data4 <-> gpio4)

#define SmartIO_CHANNEL5 0x20UL

Channel 5 (data5 <-> gpio5)

#define SmartIO_CHANNEL6 0x40UL

Channel 6 (data6 <-> gpio6)

#define SmartIO_CHANNEL7 0x80UL

Channel 7 (data7 <-> gpio7)

#define SmartIO_CHANNEL_ALL 0xffUL

Bypass all channels

Component clock selection constants

Description

Constants to be passed as "clockSrc" parameter in SmartIO_ClockSelect() function.

Macros

- #define SmartIO_CLK_GPIO0 0UL
- #define SmartIO_CLK_GPIO1 1UL
- #define SmartIO_CLK_GPIO2 2UL
- #define SmartIO_CLK_GPIO3 3UL
- #define SmartIO_CLK_GPIO4 4UL
- #define SmartIO_CLK_GPIO5 5UL
- #define SmartIO_CLK_GPIO6 6UL
- #define SmartIO_CLK_GPIO7 7UL
- #define SmartIO_CLK_DATA0 8UL
- #define SmartIO_CLK_DATA1 9UL
- #define SmartIO_CLK_DATA2 10UL
- #define SmartIO_CLK_DATA3 11UL
- #define SmartIO_CLK_DATA4 12UL
- #define SmartIO_CLK_DATA5 13UL
- #define SmartIO_CLK_DATA6 14UL
- #define SmartIO_CLK_DATA7 15UL
- #define SmartIO_DIV_CLK_ACT 16UL
- #define SmartIO_DIV_CLK_DS 17UL
- #define SmartIO_DIV_CLK_HIB 18UL
- #define SmartIO_LFCLK 19UL
- #define SmartIO_CLK_GATED 20UL
- #define SmartIO_ASYNC 31UL

Macro Definition Documentation

#define SmartIO_CLK_GPIO0 0UL

Clock sourced from signal on gpio0

#define SmartIO_CLK_GPIO1 1UL

Clock sourced from signal on gpio1

#define SmartIO_CLK_GPIO2 2UL

Clock sourced from signal on gpio2

#define SmartIO_CLK_GPIO3 3UL
Clock sourced from signal on gpio3

#define SmartIO_CLK_GPIO4 4UL
Clock sourced from signal on gpio4

#define SmartIO_CLK_GPIO5 5UL
Clock sourced from signal on gpio5

#define SmartIO_CLK_GPIO6 6UL
Clock sourced from signal on gpio6

#define SmartIO_CLK_GPIO7 7UL
Clock sourced from signal on gpio7

#define SmartIO_CLK_DATA0 8UL
Clock sourced from signal on data0

#define SmartIO_CLK_DATA1 9UL
Clock sourced from signal on data1

#define SmartIO_CLK_DATA2 10UL
Clock sourced from signal on data2

#define SmartIO_CLK_DATA3 11UL
Clock sourced from signal on data3

#define SmartIO_CLK_DATA4 12UL
Clock sourced from signal on data4

#define SmartIO_CLK_DATA5 13UL
Clock sourced from signal on data5

#define SmartIO_CLK_DATA6 14UL
Clock sourced from signal on data6

#define SmartIO_CLK_DATA7 15UL
Clock sourced from signal on data7

#define SmartIO_DIV_CLK_ACT 16UL
Clock sourced from a divided clock (Active)

#define SmartIO_DIV_CLK_DS 17UL
Clock sourced from a divided clock (Deep-Sleep)

#define SmartIO_DIV_CLK_HIB 18UL
Clock sourced from a divided clock (Hibernate)

#define SmartIO_LFCLK 19UL

Clock sourced from LFCLK

#define SmartIO_CLK_GATED 20UL

Disables the clock connection. Used when turning off the block

#define SmartIO_ASYNC 31UL

Asynchronous operation

Component hold override selection constants

Description

Constants to be passed as "ovCtrl" parameter in SmartIO_HoldOverride() function.

Macros

- #define SmartIO_OVCTRL_DISABLE 0UL
- #define SmartIO_OVCTRL_ENABLE 1UL

Macro Definition Documentation

#define SmartIO_OVCTRL_DISABLE 0UL

Controlled by HSIOM

#define SmartIO_OVCTRL_ENABLE 1UL

Controlled by SmartIO

Terminal selection constants

Description

Constants to be passed as "portTerm" parameter in SmartIO_IoSyncMode() and SmartIO_ChipSyncMode() functions.

Macros

- #define SmartIO_TERM_NONE 0x00UL
- #define SmartIO_TERM0 0x01UL
- #define SmartIO_TERM1 0x02UL
- #define SmartIO_TERM2 0x04UL
- #define SmartIO_TERM3 0x08UL
- #define SmartIO_TERM4 0x10UL
- #define SmartIO_TERM5 0x20UL
- #define SmartIO_TERM6 0x40UL
- #define SmartIO_TERM7 0x80UL
- #define SmartIO_TERM_ALL 0xffUL

Macro Definition Documentation

#define SmartIO_TERM_NONE 0x00UL

No synchronization for all data/gpio

#define SmartIO_TERM0 0x01UL
Enable synchronization for data0/gpio0

#define SmartIO_TERM1 0x02UL
Enable synchronization for data1/gpio1

#define SmartIO_TERM2 0x04UL
Enable synchronization for data2/gpio2

#define SmartIO_TERM3 0x08UL
Enable synchronization for data3/gpio3

#define SmartIO_TERM4 0x10UL
Enable synchronization for data4/gpio4

#define SmartIO_TERM5 0x20UL
Enable synchronization for data5/gpio5

#define SmartIO_TERM6 0x40UL
Enable synchronization for data6/gpio6

#define SmartIO_TERM7 0x80UL
Enable synchronization for data7/gpio7

#define SmartIO_TERM_ALL 0xffUL
Enable sycnhronization for all data/gpio

Look-up table number constants

Description

Constants to be passed as "lutNum" parameter in SmartIO_LUT_SelectInputs() and SmartIO_LUT_ConfigureMode() functions.

Macros

- #define SmartIO_LUT0 0UL
- #define SmartIO_LUT1 1UL
- #define SmartIO_LUT2 2UL
- #define SmartIO_LUT3 3UL
- #define SmartIO_LUT4 4UL
- #define SmartIO_LUT5 5UL
- #define SmartIO_LUT6 6UL
- #define SmartIO_LUT7 7UL

Macro Definition Documentation

#define SmartIO_LUT0 0UL
LUT number 0

#define SmartIO_LUT1 1UL
LUT number 1

#define SmartIO_LUT2 2UL
LUT number 2

#define SmartIO_LUT3 3UL
LUT number 3

#define SmartIO_LUT4 4UL
LUT number 4

#define SmartIO_LUT5 5UL
LUT number 5

#define SmartIO_LUT6 6UL
LUT number 6

#define SmartIO_LUT7 7UL
LUT number 7

LUT input number constants

Description

Constants to be passed as "inputNum" parameter in SmartIO_LUT_SelectInputs() function.

Macros

- #define SmartIO_LUT_INPUT0 0x01UL
- #define SmartIO_LUT_INPUT1 0x02UL
- #define SmartIO_LUT_INPUT2 0x04UL
- #define SmartIO_LUT_INPUT_ALL 0x07UL

Macro Definition Documentation

#define SmartIO_LUT_INPUT0 0x01UL
LUT input terminal 0

#define SmartIO_LUT_INPUT1 0x02UL
LUT input terminal 1

#define SmartIO_LUT_INPUT2 0x04UL
LUT input terminal 2

#define SmartIO_LUT_INPUT_ALL 0x07UL
All LUT input terminals

LUT input source constants

Description

Constants to be passed as "inputSrc" parameter in SmartIO_LUT_SelectInputs() function.

Macros

- #define SmartIO_SRC_LUT0 0UL
- #define SmartIO_SRC_LUT1 1UL
- #define SmartIO_SRC_LUT2 2UL
- #define SmartIO_SRC_LUT3 3UL
- #define SmartIO_SRC_LUT4 4UL
- #define SmartIO_SRC_LUT5 5UL
- #define SmartIO_SRC_LUT6 6UL
- #define SmartIO_SRC_LUT7 7UL
- #define SmartIO_SRC_DATA_04 8UL
- #define SmartIO_SRC_DATA_15 9UL
- #define SmartIO_SRC_DATA_26 10UL
- #define SmartIO_SRC_DATA_37 11UL
- #define SmartIO_SRC_GPIO_04 12UL
- #define SmartIO_SRC_GPIO_15 13UL
- #define SmartIO_SRC_GPIO_26 14UL
- #define SmartIO_SRC_GPIO_37 15UL
- #define SmartIO_SRC_DU 16UL

Macro Definition Documentation

#define SmartIO_SRC_LUT0 0UL

Source is LUT0 output

#define SmartIO_SRC_LUT1 1UL

Source is LUT1 output

#define SmartIO_SRC_LUT2 2UL

Source is LUT2 output

#define SmartIO_SRC_LUT3 3UL

Source is LUT3 output

#define SmartIO_SRC_LUT4 4UL

Source is LUT4 output

#define SmartIO_SRC_LUT5 5UL

Source is LUT5 output

#define SmartIO_SRC_LUT6 6UL

Source is LUT6 output

#define SmartIO_SRC_LUT7 7UL

Source is LUT7 output

#define SmartIO_SRC_DATA_04 8UL
Source is data0/data4

#define SmartIO_SRC_DATA_15 9UL
Source is data1/data5

#define SmartIO_SRC_DATA_26 10UL
Source is data2/data6

#define SmartIO_SRC_DATA_37 11UL
Source is data3/data7

#define SmartIO_SRC_GPIO_04 12UL
Source is gpio0/gpio4

#define SmartIO_SRC_GPIO_15 13UL
Source is gpio1/gpio5

#define SmartIO_SRC_GPIO_26 14UL
Source is gpio2/gpio6

#define SmartIO_SRC_GPIO_37 15UL
Source is gpio3/gpio7

#define SmartIO_SRC_DU 16UL
Source is Data Unit output

LUT mode constants

Description

Constants to be passed as "mode" parameter in SmartIO_LUT_ConfigureMode() function.

Macros

- #define SmartIO_MODE_COMB 0UL
- #define SmartIO_MODE_REGIN 1UL
- #define SmartIO_MODE_REGOUT 2UL
- #define SmartIO_MODE_SRFF 3UL

Macro Definition Documentation

#define SmartIO_MODE_COMB 0UL
Combinatorial mode

#define SmartIO_MODE_REGIN 1UL
Registered input mode

#define SmartIO_MODE_REGOUT 2UL
Registered output mode

#define SmartIO_MODE_SRFF 3UL

S/R Flip-Flop mode

Data Unit trigger input constants

Description

Constants to be passed as "tr0Sel", "tr1Sel", and "tr2Sel" parameters in SmartIO_DU_SelectTriggers() function.

Macros

- #define SmartIO_TR_CONST_ZERO 0UL
- #define SmartIO_TR_CONST_ONE 1UL
- #define SmartIO_TR_DU_OUT 2UL
- #define SmartIO_TR_LUT0 3UL
- #define SmartIO_TR_LUT1 4UL
- #define SmartIO_TR_LUT2 5UL
- #define SmartIO_TR_LUT3 6UL
- #define SmartIO_TR_LUT4 7UL
- #define SmartIO_TR_LUT5 8UL
- #define SmartIO_TR_LUT6 9UL
- #define SmartIO_TR_LUT7 10UL

Macro Definition Documentation

#define SmartIO_TR_CONST_ZERO 0UL

Constant 0

#define SmartIO_TR_CONST_ONE 1UL

Constant 1

#define SmartIO_TR_DU_OUT 2UL

Data unit output

#define SmartIO_TR_LUT0 3UL

LUT 0 output

#define SmartIO_TR_LUT1 4UL

LUT 1 output

#define SmartIO_TR_LUT2 5UL

LUT 2 output

#define SmartIO_TR_LUT3 6UL

LUT 3 output

#define SmartIO_TR_LUT4 7UL

LUT 4 output

#define SmartIO_TR_LUT5 8UL

LUT 5 output

#define SmartIO_TR_LUT6 9UL

LUT 6 output

#define SmartIO_TR_LUT7 10UL

LUT 7 output

Data Unit data register constants

Description

Constants to be passed as "dataNum" parameter in SmartIO_DU_SelectData() function.

Macros

- #define SmartIO_DATA0 1UL
- #define SmartIO_DATA1 2UL
- #define SmartIO_DATA_BOTH 3UL

Macro Definition Documentation

#define SmartIO_DATA0 1UL

DU DATA0 register

#define SmartIO_DATA1 2UL

DU DATA1 register

#define SmartIO_DATA_BOTH 3UL

Both DU DATA0 and DATA1 registers

Data Unit data register source selection constants

Description

Constants to be passed as "dataSel" parameter in SmartIO_DU_SelectData() function.

Macros

- #define SmartIO_DATA_CONST_ZERO 0UL
- #define SmartIO_DATA_TERM_DATA 1UL
- #define SmartIO_DATA_TERM_GPIO 2UL
- #define SmartIO_DATA_DU_REG 3UL

Macro Definition Documentation

#define SmartIO_DATA_CONST_ZERO 0UL

Constant 0

#define SmartIO_DATA_TERM_DATA 1UL

Terminal data[7:0]

#define SmartIO_DATA_TERM_GPIO 2UL

Terminal gpio[7:0]

#define SmartIO_DATA_DU_REG 3UL

Data Unit register

Data Unit opcode constants

Description

Constants to be passed as "opCode" parameter in SmartIO_DU_OpCode() function.

Macros

- #define SmartIO_OPC_INCR 1UL
- #define SmartIO_OPC_DECR 2UL
- #define SmartIO_OPC_INCR_WRAP 3UL
- #define SmartIO_OPC_DECR_WRAP 4UL
- #define SmartIO_OPC_INCR_DECR 5UL
- #define SmartIO_OPC_INCR_DECR_WRAP 6UL
- #define SmartIO_OPC_ROR 7UL
- #define SmartIO_OPC_SHR 8UL
- #define SmartIO_OPC_AND_OR 9UL
- #define SmartIO_OPC_SHR_MAJ3 10UL
- #define SmartIO_OPC_SHR_EQL 11UL

Macro Definition Documentation

#define SmartIO_OPC_INCR 1UL

Count Up

#define SmartIO_OPC_DECR 2UL

Count Down

#define SmartIO_OPC_INCR_WRAP 3UL

Count Up and wrap

#define SmartIO_OPC_DECR_WRAP 4UL

Count Down and wrap

#define SmartIO_OPC_INCR_DECR 5UL

Count Up/Down

#define SmartIO_OPC_INCR_DECR_WRAP 6UL

Count Up/Down and wrap

#define SmartIO_OPC_ROR 7UL

Rotate right

#define SmartIO_OPC_SHR 8UL

Shift right

#define SmartIO_OPC_AND_OR 9UL

DU Data0 AND DU Data1

#define SmartIO_OPC_SHR_MAJ3 10UL
Majority 3

#define SmartIO_OPC_SHR_EQL 11UL
Equal DU Data1

API Functions

Description

API functions are used for run-time configuration of the component during active power mode.
Use these to start/stop and to re-configure the SmartIO component during run-time.

Macros

- #define SmartIO_Enable() (SmartIO_CTL |= SmartIO_FABRIC_ENABLE)
  Enables the component.
- #define SmartIO_Disable() (SmartIO_CTL &= SmartIO_FABRIC_DISABLE)
  Disables the component.
- #define SmartIO_GetBypass()
  Returns the bypass configuration of the channels on a bit by bit basis.

Functions

- void SmartIO_Init (void)
  Initializes the component as specified in the Configuration dialog.
- void SmartIO_Start (void)
  Invokes SmartIO_Init() and SmartIO_Enable().
- void SmartIO_Stop (void)
  The routing fabric is bypassed and the block is powered down.
- void SmartIO_SetBypass (uint32 portChannel)
  Bypasses channels on a bit by bit basis.
- void SmartIO_ClockSelect (uint32 clockSrc)
  Selects the component clock source.
- void SmartIO_HoldOverride (uint32 ovCtrl)
  Controls the port's chip hibernate mode override.
- void SmartIO_IoSyncMode (uint32 portTerm)
  Controls synchronization of signals coming from I/O Pin.
- void SmartIO_ChipSyncMode (uint32 portTerm)
  Controls synchronization of signals coming from chip to the component.
- cystatus SmartIO_LUT_SelectInputs (uint32 lutNum, uint32 inputNum, uint32 inputSrc)
  This function is used to select individual inputs to the specified LUT.
- void SmartIO_LUT_ConfigureMode (uint32 lutNum, uint32 mapping, uint32 mode)
  This function sets the logical function of the selected LUT (based on the 3 inputs) and specifies its operating mode.
- cystatus SmartIO_DU_SelectTriggers (uint32 tr0Sel, uint32 tr1Sel, uint32 tr2Sel)
  Selects the trigger sources for the data unit.
- cystatus SmartIO_DU_SelectData (uint32 dataNum, uint32 dataSel)

Specifies the values for the Data0 and Data1 registers in the Data Unit.

- cystatus SmartIO_DU_RegValue (uint32 dataValue)
  Specifies the 8-bit value in the internal DU register that may be a source for Data0 and/or Data1.
- cystatus SmartIO_DU_OpCode (uint32 opCode, uint32 dataSize)
  Selects the data unit operation and the size of the data to be operated on (1 to 8 bits).

Macro Definition Documentation

#define SmartIO_Enable() (SmartIO_CTL |= SmartIO_FABRIC_ENABLE)

Enables the component.

Once enabled, it takes two component clock cycles for the fabric reset to deactivate and the fabric becomes operational. If the clock source is set to Asynchronous mode, it takes three SYSCLK cycles before becoming functional.

Function Usage

```
{
    /* Refer to the SmartIO_Init() function usage example */
}
```

#define SmartIO_Disable() (SmartIO_CTL &= SmartIO_FABRIC_DISABLE)

Disables the component.

The block is disabled, which places the channels into bypass mode and the sequential elements are reset based on the chosen clock selection.

Function Usage

```
{
    /* Refer to the SmartIO_Init() function usage example */
}
```

#define SmartIO_GetBypass()

Value: `((uint8)(SmartIO_CTL \ & ((uint32)SmartIO_CHANNEL_ALL << CYFLD_PRGIO_PRT_BYPASS__OFFSET)))`

Returns the bypass configuration of the channels on a bit by bit basis.

Bypassed channels behave like they would as if the SmartIO component was not present for those particular channels.

Returns:

uint8 Bypass state of the channels on the port.

Function Usage

```
{
    /* Refer to SmartIO_SetBypass() example */
}
```

Function Documentation

void SmartIO_Init (void)

Initializes the component as specified in the Configuration dialog.

This function can also be used to reinitialize the component to the settings as specified in the Configuration dialog.

Function Usage

```
{
    /* Start the component */
    SmartIO_Start();

    /* ... Perform some run-time changes to SmartIO ... */

    /* Reinitialize to the default configuration as specified in the GUI */
    SmartIO_Disable();
    SmartIO_Init();
    /* Enable the component */
    SmartIO_Enable();
}
```

void SmartIO_Start (void)

Invokes SmartIO_Init() and SmartIO_Enable().

After this function call, the component is initialized to specified values in the Configuration dialog and is then enabled.

If calling this function after calling SmartIO_Stop(), the fabric will set the clock configuration as specified before calling Stop().

Global Variables

SmartIO_initVar - used to check initial configuration, modified on first function call.

Function Usage

```
{
    /* First start the SmartIO component */
    SmartIO_Start();
    /* Then start the peripherals connected to SmartIO */
    /* SCB_Start(), TCPWM_Start() etc. */
}
```

void SmartIO_Stop (void)

The routing fabric is bypassed and the block is powered down.

This function saves the clock configuration before powering down. Calling Start() will enable the block and restore the clock setting.

Global Variables

SmartIO_clkConfig - used to save the clock configuration.

Function Usage

```
{
    /* Start the component */
    SmartIO_Start();
    /* Stop the component, which disables the block and all channels are bypassed */
    SmartIO_Stop();
    /* Re-start the component and pick up where you left off */
    SmartIO_Start();
}
```

void SmartIO_SetBypass (uint32 *portChannel*)

Bypasses channels on a bit by bit basis.

Bypassed channels behave as if the SmartIO component was not present for those particular channels.

Note This function impacts all channels on the port. It will write over the bypass configuration of all the channels.

Parameters:

portChannel	The channel location to be bypassed on the port. If the bit value is 0, the channel is not bypassed; if the bit value is 1, the channel is bypassed. Valid options are documented in Port channel selection constants. These should be ORed to configure multiple bypass channels.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    uint8 bypassVal = 0u;

    /* Retrieve the port bypass status */
    bypassVal = SmartIO_GetBypass();
    /* Bypass channel 7, and leave the others as is */
    SmartIO_SetBypass(bypassVal | SmartIO_CHANNEL7);
}
```

void SmartIO_ClockSelect (uint32 *clockSrc*)

Selects the component clock source.

The clock selection impacts the reset behavior of the component and low power operation. Refer to the Functional Description section of the component datasheet for more information.

Parameters:

clockSrc	The component clock source. Valid options are documented in Component clock selection constants.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Change the clock source to LFCLK in preparation for Deep-Sleep operation */
    SmartIO_ClockSelect (SmartIO_LFCLK);
    /* Put the device into deep-sleep mode */
    CySysPmDeepSleep();
    /* Once the device wakes up, change the clock source to HFCLK */
    SmartIO_ClockSelect (SmartIO_DIV_CLK_ACT);
}
```

void SmartIO_HoldOverride (uint32 *ovCtrl*)

Controls the port's chip hibernate mode override.

This function should be used when the block is desired to be operational during chip deep-sleep or hibernate mode. The override functionality allows the port with SmartIO component to be operational during these modes. When in chip active/sleep modes, the hold override functionality should be disabled.

Parameters:

ovCtrl	Hold override control. Valid options are documented in Component hold override selection constants.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Set the SmartIO component to operate in purely combinatorial fashion */
    SmartIO_ClockSelect (SmartIO_ASYNC);
    /* Enable the hold override functionality for operating in Hibernate mode */
    SmartIO_HoldOverride (SmartIO_OVCTRL_ENABLE);
    /* Put the device into hibernate mode */
    CySysPmHibernate();
    /* Once the device wakes up, turn off the hold override */
    SmartIO_HoldOverride (SmartIO_OVCTRL_DISABLE);
}
```

void SmartIO_IoSyncMode (uint32 *portTerm*)

Controls synchronization of signals coming from I/O Pin.

The synchronization is performed using the component clock. Synchronization should not be used if the component is configured to operate in asynchronous mode.

Note This function impacts all gpio terminals on the port. It will write over the synchronization configuration of all the gpio terminals.

Parameters:

portTerm	The terminal location on the port. If the bit value is 0, the terminal is not synchronized; if the bit value is 1, the terminal is synchronized. Valid options are documented in Terminal selection constants. These should be ORed to configure multiple terminals.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Set the SmartIO component to operate in purely combinatorial fashion */
    SmartIO_ClockSelect (SmartIO_ASYNC);
    /* Disable synchronization for all gpio inputs */
    SmartIO_IoSyncMode (SmartIO_TERM_NONE);
}
```

void SmartIO_ChipSyncMode (uint32 *portTerm*)

Controls synchronization of signals coming from chip to the component.

The synchronization is performed using the component clock. Synchronization should not be used if the component is configured to operate in asynchronous mode.

Note This function impacts all data terminals on the port. It will write over the synchronization configuration of all the data terminals.

Parameters:

portTerm	The terminal location on the port. If the bit value is 0, the terminal is not synchronized; if the bit value is 1, the terminal is synchronized. Valid options are documented in Terminal selection constants. These should be ORed to configure multiple terminals.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
```

```
    /* Set the SmartIO component to operate in purely combinatorial fashion */
    SmartIO_ClockSelect (SmartIO_ASYNC);
    /* Disable synchronization for all gpio inputs */
    SmartIO_ChipSyncMode (SmartIO_TERM_NONE);
}
```

cystatus SmartIO_LUT_SelectInputs (uint32 *lutNum*, uint32 *inputNum*, uint32 *inputSrc*)

This function is used to select individual inputs to the specified LUT.

This function can be used to configure multiple LUTs with the same input configuration.

Note The selections are different for the upper and lower nibbles. See the LUT input parameters and Functional Description section in the component datasheet for more information.

Parameters:

lutNum	LUT number. Valid options are documented in Look-up table number constants.
inputNum	The input number (input0, input1, input2). Valid options are documented in LUT input number constants. Note that LUT input0 does not accept LUT0 output as an input.
inputSrc	The source of the LUT input. Valid options are documented in LUT input source constants.

Returns:

Status of the LUT input selection.

Status	Description
CYRET_SUCCESS	Write successful
CYRET_BAD_PARAM	Invalid parameter

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
    {
        /* Change the "input 1" selection of LUT1 and LUT2 to gpio0 */
        SmartIO_LUT_SelectInputs (SmartIO_LUT1 | SmartIO_LUT2, SmartIO_LUT_INPUT1,
SmartIO_SRC_GPIO_04);
    }
```

void SmartIO_LUT_ConfigureMode (uint32 *lutNum*, uint32 *mapping*, uint32 *mode*)

This function sets the logical function of the selected LUT (based on the 3 inputs) and specifies its operating mode.

This function can be used to configure multiple LUTs with the same configuration. The LUT mapping (truth table) is defined as follows.

input2	input1	input0	Mapping bit
0	0	0	bit0
0	0	1	bit1
0	1	0	bit2
0	1	1	bit3
1	0	0	bit4
1	0	1	bit5
1	1	0	bit6
1	1	1	bit7

Parameters:

lutNum	LUT number. Valid options are documented in Look-up table number constants.
mapping	This is the LUT truth table. Bit0 corresponds to the LUT output when the input value is (in2=0, in1=0, in0=0). Bit7 corresponds to the LUT output when the input value is (in2=1, in1=1, in0=1).
mode	The LUT mode. Valid options are documented in LUT mode constants.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Set LUT5 and LUT7 to combinatorial mode and define its truth table to be 0xAAu */
    SmartIO_LUT_ConfigureMode (SmartIO_LUT5 | SmartIO_LUT7, 0xAAu, SmartIO_MODE_COMB);
}
```

cystatus SmartIO_DU_SelectTriggers (uint32 *tr0Sel*, uint32 *tr1Sel*, uint32 *tr2Sel*)

Selects the trigger sources for the data unit.

Parameters:

tr0Sel	Trigger 0 input selection. Valid options are documented in Data Unit trigger input constants.
tr1Sel	Trigger 1 input selection. Valid options are documented in Data Unit trigger input constants.
tr2Sel	Trigger 2 input selection. Valid options are documented in Data Unit trigger input constants.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* TR0 = 0, TR1 = 1, TR2 = LUT1 */
    SmartIO_DU_SelectTriggers (SmartIO_TR_CONST_ZERO, SmartIO_TR_CONST_ONE, SmartIO_TR_LUT1);
}
```

cystatus SmartIO_DU_SelectData (uint32 *dataNum*, uint32 *dataSel*)

Specifies the values for the Data0 and Data1 registers in the Data Unit.

Parameters:

dataNum	DU DATA register selection. Valid options are documented in Data Unit data register constants.
dataSel	DU DATA register source selection. Valid options are documented in Data Unit data register source selection constants.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* DATA0 = DU_Reg, DATA1 = DU_Reg */
    SmartIO_DU_SelectData (SmartIO_DATA_BOTH, SmartIO_DATA_DU_REG);
    /* DU_Reg = 0x7F */
    SmartIO_DU_RegValue(0x7Fu);
}
```

cystatus SmartIO_DU_RegValue (uint32 *dataValue*)

Specifies the 8-bit value in the internal DU register that may be a source for Data0 and/or Data1.

Parameters:

dataValue	8-bit value to be stored in the register.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Refer to SmartIO_DU_SelectData() function */
}
```

cystatus SmartIO_DU_OpCode (uint32 *opCode*, uint32 *dataSize*)

Selects the data unit operation and the size of the data to be operated on (1 to 8 bits).

Parameters:

opCode	Op-code for the Data Unit. Valid options are documented in Data Unit opcode constants.
dataSize	Size/width of the data unit operands. Range: 1 to 8 bits.

Side Effects

This function disables the block for reconfiguration. During this time, all signals on the port will be bypassed. All peripheral and gpio signals must be stopped or properly handled to account for the bypass state.

Function Usage

```
{
    /* Increment Wrap mode, 6-bit operation */
    SmartIO_DU_OpCode (SmartIO_OPC_INCR_WRAP, 6u);
}
```

Global Variables

Description

Global variables used in the component.

The following global variables are used in the component.

Variables

- uint8 SmartIO_initVar

Variable Documentation

uint8 SmartIO_initVar

Initialization state variable

API Constants

Description

Component API functions are designed to work with pre-defined enumeration values. These values should be used with the functions that reference them.

Modules

- Port channel selection constants
- Component clock selection constants
- Component hold override selection constants
- Terminal selection constants
- Look-up table number constants
- LUT input number constants
- LUT input source constants
- LUT mode constants
- Data Unit trigger input constants
- Data Unit data register constants
- Data Unit data register source selection constants
- Data Unit opcode constants

Group_structs

Description

Data Structures

- struct SmartIO_lut_config_struct

Data Structure Documentation

SmartIO_lut_config_struct Struct Reference

Description

LUT selection and mapping register struct

Data Fields

- uint32 lutSel [SmartIO_CHANNELS]
- uint32 lutCtl [SmartIO_CHANNELS]

Field Documentation

uint32 SmartIO_lut_config_struct::lutSel[SmartIO_CHANNELS]

LUT input selection configuration

uint32 SmartIO_lut_config_struct::lutCtl[SmartIO_CHANNELS]

LUT mode and mapping configuration

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. The project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Smart I/O Component does not have any specific deviations.

Sample Firmware Source Code

PSoC Creator provides many 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.

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 4 (GCC)	
	Flash Bytes	SRAM Bytes
Default	696	8

Functional Description

The Smart I/O Component uses the underlying PRGIO block to implement a port-wide logic array that can be used to perform routing and logic operations to peripheral and I/O signals. The following sub sections describe the block restrictions and application critical information.

Routing Fabric

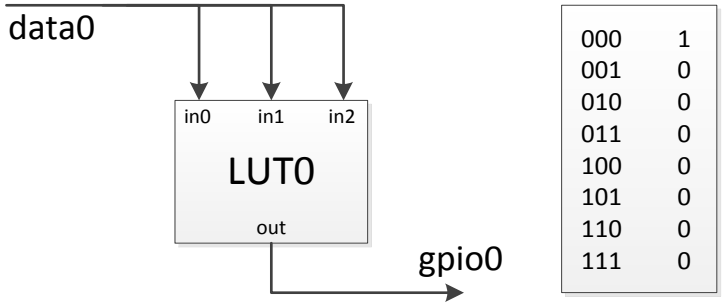
The Smart I/O routing fabric is divided into two portions, where each portion is capable of accepting half of the data or GPIO signals. The LUTs have the following structure.

- LUT 7…4 are capable of accepting signals from gpio/data 7…4 as inputs.
- LUT 3…0 are capable of accepting signals from gpio/data 3…0 as inputs.
- The LUTs can accept any LUT output as an input.
- Each LUT output is dedicated to the corresponding output gpio/data terminal. For example, LUT 0 can go to either gpio0 terminal (output type) or data0 terminal (output type). The LUT output cannot be routed to an input terminal type.

Single Source LUT Input

If a LUT is used, all three inputs to the LUT must be designated. For example, even If a LUT is used to accept a single source as its input, all three inputs must accept that same signal. The lookup truth table should then be designed such that it only changes the output value when all three inputs satisfy the same condition.

For example, consider the case where the signal on data0 must be inverted before being passed to gpio0. LUT0 accepts data0 as input 0, 1 and 2. The truth table is defined such that it outputs a logic 1 only when the inputs are all 0.



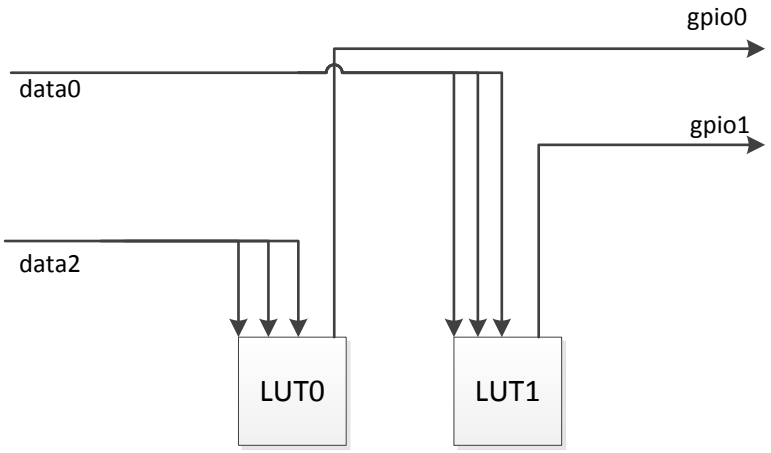
SCB Restriction

The SCB routing paths are restricted and can only be connected to the dedicated pin. They can however go through the dedicated LUT. For example, an SCB SPI slave select line on data2 may be an input to LUT2, and then the output go to gpio2. It cannot go to any other LUTs.

Buried Pins

If a **data** terminal is used, but the corresponding **gpio** terminal is not used, then that pin cannot be used for software controlled GPIO functionality or for placing analog pins.

For example, if data0 is used as an input terminal, the gpio0 terminal will be set to **None** by default and the pin location will not be useable. The only valid option for using gpio0 is to make it an output terminal, and route a signal to it from LUT0 as shown in the following diagram.



Note The instantiation of buried pins can be avoided for advanced users by declaring the opposite terminal as either **Input** or **Output** instead of **None**. This gives greater flexibility to designs that require special routing considerations.

Clock and Reset Behavior

The Smart I/O Component drives its synchronous elements using a single Component-wide clock. Depending on the clock source, the Component will have different reset behaviors, which will reset all the flip-flops in the LUTs and synchronizers to logic 0. The Component configuration registers will retain their values unless coming out of Power on Reset (POR).

Note If the Component is only disabled, the values in the LUT flip-flips and I/O synchronizers are held as long as the chip remains in a valid power mode.

Note The selected clock for the fabric's synchronous logic is not phase aligned with other synchronous logic on the chip operating on the same clock. Therefore, communication between the Smart I/O and other synchronous logic should be treated as asynchronous (just as the communication between I/O input signals and other synchronous logic should be treated as asynchronous).

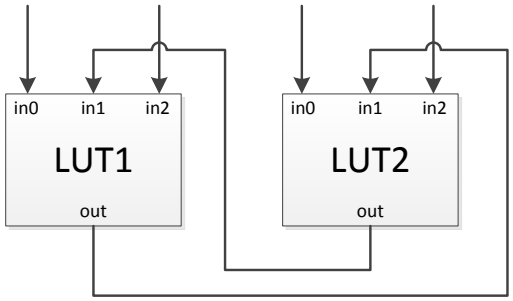
Clock Source	Reset Behavior	Enable Delay	Description
gpio 7…0	Reset on POR	2 clock edges	If chosen as the clock source, that particular signal cannot also be used as an input to a LUT as it may cause a race condition. The fabric will be enabled after 2 clock edges of the signal on the gpio terminal.
data 7…0	Reset on POR	2 clock edges	If chosen as the clock source, that particular signal cannot also be used as an input to a LUT as it may cause a race condition. The fabric will be enabled after 2 clock edges of the signal on the data terminal.
Divided Clock (Active)	Reset when going to Deep Sleep, Hibernate or POR	2 clock edges	The fabric will be enabled after 2 clock edges of the divided clock. Any synchronous logic in the LUTs will be reset to 0 when in chip deep-sleep or hibernate modes.
Divided Clock (Deep-Sleep)	Reset when going to Hibernate or POR	2 clock edges	The fabric will be enabled after 2 clock edges of the divided clock. Any synchronous logic in the LUTs will be reset to 0 when in hibernate mode.
Divided Clock (Hibernate)	Reset on POR	2 clock edges	The fabric will be enabled after 2 clock edges of the divided clock.
LFCLK	Reset when going to Hibernate and POR	2 clock edges	The fabric will be enabled after 2 clock edges of the low frequency clock (LFCLK). Any synchronous logic in the LUTs will be reset to 0 when in hibernate mode.
Asynchronous	Reset on POR	3 clock edges of SYSCLK	The fabric will be enabled after 3 clock edges of the system clock (SYSCLK).

Signal Synchronization Requirement

If any of the signals coming in through the Smart I/O Component terminals are meant to be used in sequential elements in the LUTs, the terminal synchronizer must first be used to synchronize that signal to the Component clock. For example, if the signal on gpio0 must be used in LUT0 in Sequential output mode, the synchronization for gpio0 terminal should be enabled for reliable operation.

LUT Combinatorial Feedback

Since the LUTs can be configured as purely (or partially) combinatorial elements and since they can chain to each other in any fashion, combinatorial timing loops can occur. This causes oscillations that burn power and create unpredictable behavior. If a feedback is required, the signals should always go through a flip-flop before feeding back. For example, the following is a potentially problematic design. LUT1 and LUT2 are configured in **Combinatorial** mode. This will result in oscillations. To prevent it, one of the LUTs should be configured to **Gated Output** mode.



Low Power Mode

The Smart I/O Component is capable of operating during chip Deep-Sleep and Hibernate modes. The block has the following requirements when operating in this mode:

- All sequential elements must be clocked by a valid clock in these power domains. Refer to Clock and Reset Behavior section for more details.
- All signals in the block (including the clock) must be less than 1 MHz when in Deep-Sleep and Hibernate modes.
- The hold override functionality should be enabled by using the HoldOverride() function when entering Deep-Sleep or Hibernate modes. This functionality should then be disabled when the chip is not in this mode.

Resources

Each Smart I/O Component consumes an entire SmartIO hardware block for the chosen port.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

Parameter	Description	Conditions	Min	Typ	Max	Units
Bypass Delay	Delay through the fabric when the line is bypassed	Max. delay added in I/O path when bypassed	–	–	1.6	ns
LUT Delay	Delay per LUT	Total delay should be estimated as n*8.5ns, where n is the number of LUTs in the path	–	–	8.5	ns
F_{max}	Maximum signal frequency in the Component (including clock)	Chip active, and sleep modes	–	–	48	MHz
		Chip deep-sleep and hibernate modes	–	–	1	MHz

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.a	Updated datasheet to correct title.	It is Smart I/O™.
1.10	Removed Errata	Updated the configure dialog to work on high DPI environments.
	Added data unit support	The Component provides additional API functions to configure the Smart I/O data unit.
	Added advanced information with regards to buried pins	Buried pins are necessary for many designs. However it can be avoided for advanced applications.
	Updated SmartIO_SetBypass() description	Added clarifying statement that the function configures all pins to be not passed except those passed in through the function parameter.
1.0.a	Updated SmartIO_IoSyncMode() and SmartIO_ChipSyncMode() API function descriptions.	Added usage scenario information.
	Added Errata section.	To document a potential usability defect in the Smart I/O Component.
1.0	Initial release	

© Cypress Semiconductor Corporation, 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.

