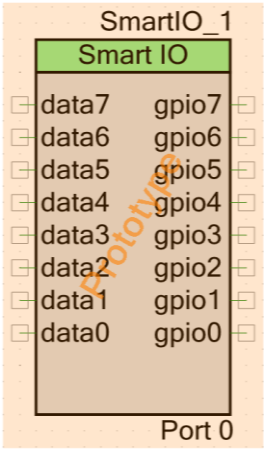
Smart IO™ 1.0



Features

- Provides glue logic functionality at I/O ports
- Low-power mode (Deep Sleep and Hibernate) operation
- Flexible array for user-defined logic functions
- Combinatorial and clocked (registered) operation
- Low-latency deterministic delays
- Simple user interface for routing signals in the fabric

General Description

The Cypress Smart IO 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 IO 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 IO component is associated with a particular GPIO port and consumes the port entirely. If the component is not used, then the Smart IO functionality for that port is bypassed.

The component can operate in low-power modes (Deep Sleep and Hibernate) and can wake up the chip using the port interrupt, when required.

Note See Component Errata section for details about a usability defect in the Smart IO component customizer that causes the PSoC Creator window and fonts to be automatically resized.

When to Use a Smart IO Component

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

Change routing to/from Pins

Signals from fixed-function peripherals, such as a TCPWM, can be rerouted to non-dedicated pins on the same port by using the Smart IO 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 IO component.

Polarity inverter

Peripheral signals, such as the SPI select of an SPI Master (SCB) component, 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 IO 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.

Strobe in an input vector

The Smart IO LUTs may be configured to register a field of inputs with an external clock.

Logic elements

The Smart IO 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 IO 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 I/Os of a Smart IO component will differ based on which device and which port is used. To use the Smart IO 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 IO 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 IO 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 IO 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 IO is located and avoid assigning pins to the component.

Input/Output Connections

This section describes the various input and output connections for the Smart IO component. 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.

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 GPIO porton 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

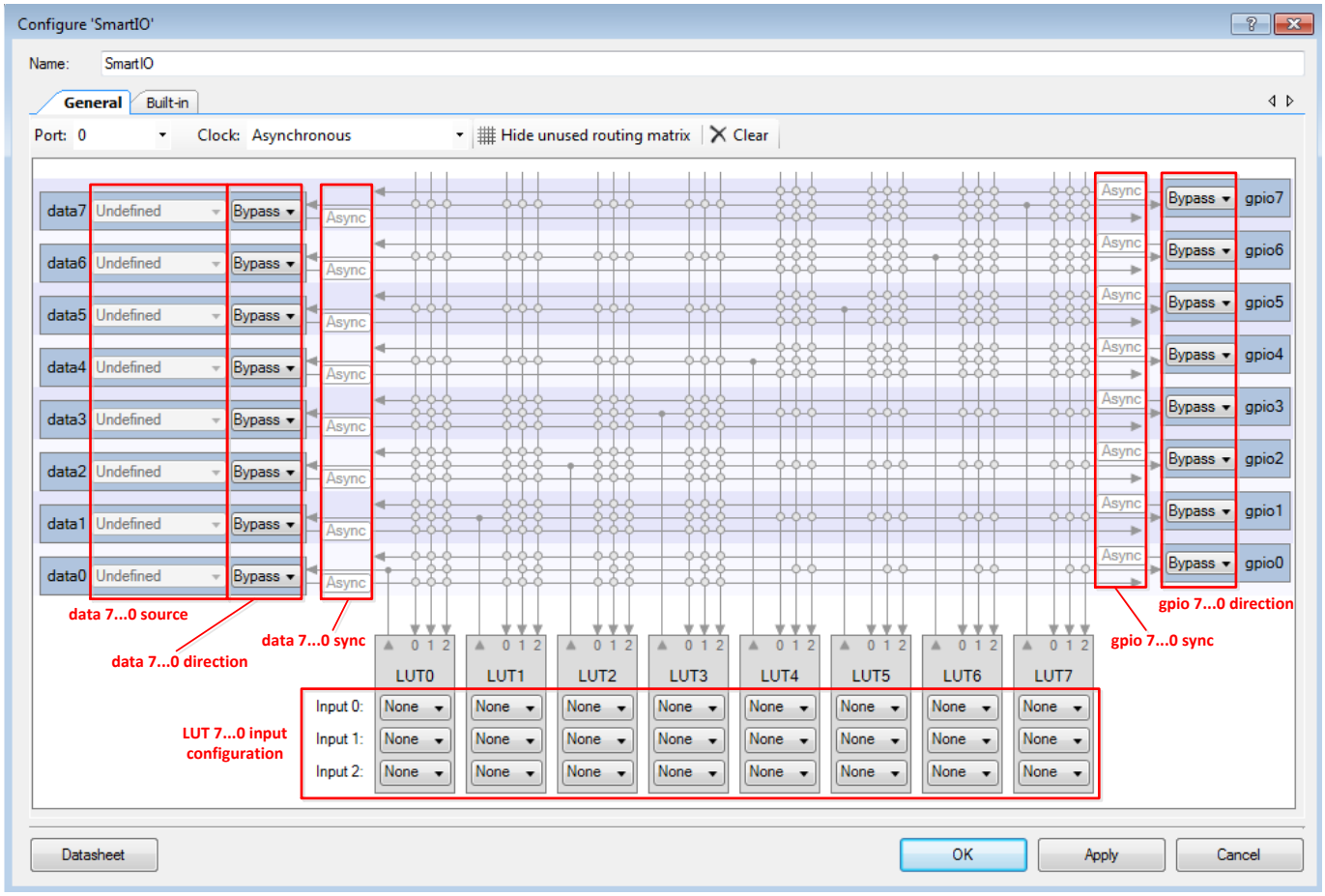
This terminal is available if the component **Clock** parameter is set to **Divided Clock (Active), (Deep-Sleep), or (Hibernate)**. Only a clock component may be connected to this terminal. All other **Clock** parameter selections automatically configure the source internally in the component.

Component Parameters

Drag a Smart IO 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 IO component is port and device specific. This parameter allows selecting a port that supports Smart IO.

Clock

Selects the clock source used to drive all sequential logic in the block. This clock is global within the Smart IO 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 IO sequential logic to be clocked during those power modes.
- **Asynchronous** – If the Smart IO 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 Unused Routing Matrix / Show Routing Matrix

This button hides or displays the Smart IO 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 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 IO, 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** – 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.
- **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 IO 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 IO 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 gpio is **Output**).

data 7…0 sync

Synchronizers on the terminal allow the signals from the chip to be synchronized to the component clock. This is essential if the signal is used in sequential logic in the LUTs. The synchronization is disabled by default, and must be manually enabled.

GPIO Configuration

gpio 7…0 direction

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

- **Input** – Changes the direction of the terminal to be input type and allows connecting an input Pin to it. Input signals can be used as inputs to the LUTs.
- **Output** – Changes the direction of the terminal to be output type and allows connecting an output 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 IO 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 IO fabric.

  None – The gpio terminal is consumed and cannot be used for connecting Pins 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 data is **Output**).

gpio 7…0 sync

Synchronizers on the terminal allow the signals from the gpio to be synchronized to the component clock. This is essential if the signal is used in sequential logic in the LUTs. The synchronization is disabled by default.

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 input 0 may accept any LUT outputs as an input **with the exception of LUT 0**.

- 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. All LUT outputs are valid inputs.

- 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.

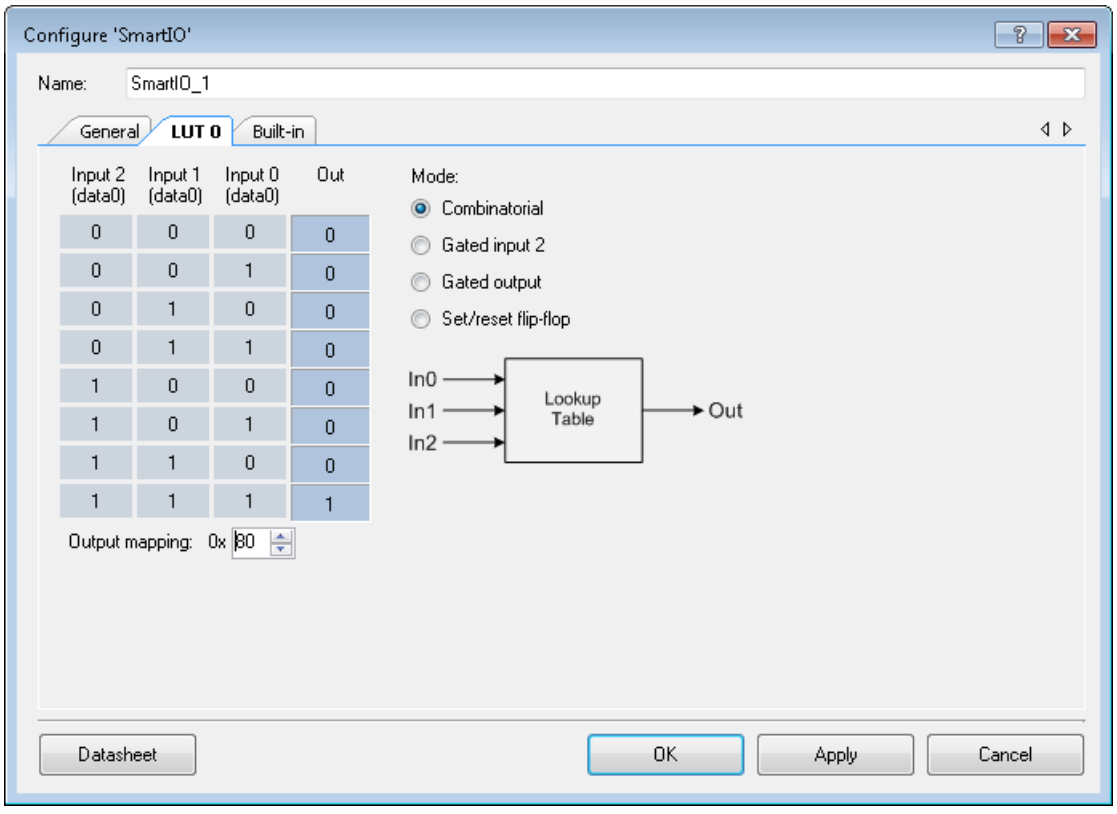
LUT 7…0 input 2

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

- 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 Tabs

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



This tab contains the following parameters:

LUT 7…0 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** 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** .

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 deep-sleep/hibernate mode override.
- void SmartIO_IoSyncMode(uint32 portTerm)
  Controls synchronization of signals coming from I/O Pins. The synchronization is performed using the component clock.
- void SmartIO_ChipSyncMode(uint32 portTerm)
  Controls synchronization of signals coming from the chip (data0..7) to the component. The synchronization is performed using the component clock.
- 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.

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

```
{
   /* Start the component */
   SmartIO_Start();
   /* Disable the component, which places the channels into bypassed mode. */
   SmartIO_Disable();
   /* Re-enable the component */
   SmartIO_Enable();

}
```

#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_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 deep-sleep/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 Deep-Sleep mode */
    SmartIO_HoldOverride (SmartIO_OVCTRL_ENABLE);
    /* Put the device into Deep-Sleep mode */
    CySysPmDeepSleep();
    /* 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 Pins. The synchronization is performed using the component clock.

Signals coming from I/O port pins (gpio 0..7) into the Smart IO block can be asynchronous with respect to clock domains inside the chip. Thus, signals from the port pins need to be synchronized to prevent meta-stability if they are to interact with signals that are synchronous to clock domains inside the chip. This synchronization is performed at an individual port pin level using a two-flop synchronizer using the clock selected for the component. I/O port pin signals that need synchronization are specified by the lower 8 bits of the "portTerm" parameter (bit 0 corresponds to gpio0 etc.).

Synchronization is not required if the incoming signals are logically processed asynchronously without needing to interact with signals in a different clock domain; for instance, if the user wants gpio3 (output pin) to be the logical AND of gpio 0, 1, and 2 (input pins) and the LUT used is not clocked then synchronization is not required.

If the Clock Select option chosen is Asynchronous then the Smart IO block is purely combinatorial and synchronization should not be used.

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);

    /* Set the SmartIO component to operate synchronously wrt LFCLK */
    SmartIO_ClockSelect (SmartIO_LFCLK);
    /* Enable synchronization for gpio3 and gpio4 inputs */
    SmartIO_IoSyncMode (SmartIO_TERM3| SmartIO_TERM4);
}
```

void SmartIO_ChipSyncMode (uint32 *portTerm*)

Controls synchronization of signals coming from the chip (data0..7) to the component. The synchronization is performed using the component clock.

Signals coming from internal blocks of the PSoC (data 0..7) into the Smart IO block can be asynchronous with respect to the clock source chosen for the Smart IO. Thus, signals from the internal blocks of the chip that are not synchronous to, but need to interact with, signals that are synchronous to the Smart IO clock domain need to be synchronized to prevent meta-stability. This synchronization is done at an individual data signal level using a two-flop synchronizer using the clock selected for the component. Chip data signals that need synchronization are specified by the lower 8 bits of the "portTerm" parameter (bit 0 corresponds to data0 etc.).

Synchronization is not required if the data signals are logically processed asynchronously without needing to interact with signals in a different clock domain; thus if the user wishes to generate the logical AND of data 0, 1, and 2 using a LUT and produce it on gpio 4, no synchronization is required if the LUT is not registered.

If the Clock Select option chosen is Asynchronous then the Smart IO block is purely combinatorial and synchronization should not be used.

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 chip(data) inputs */
   SmartIO_ChipSyncMode (SmartIO_TERM_NONE);

   /* Set the SmartIO component to operate synchronously wrt divided (HFCLK) clock */
   SmartIO_ClockSelect (SmartIO_DIV_CLK_ACT);
   /* Enable synchronization for data0 and data1 inputs */
   SmartIO_ChipSyncMode (SmartIO_TERM0| SmartIO_TERM1);
}
```

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 to gpio0 */
   SmartIO_LUT_SelectInputs (SmartIO_LUT1, 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 Look-up table number 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 to combinatorial mode and define its truth table to be 0xAAu */
   SmartIO_LUT_ConfigureMode (SmartIO_LUT5, 0xAAu, SmartIO_MODE_COMB);
}
```

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
  Constants to be passed as "portChannel" parameter in SmartIO_SetBypass() function.
- Component clock selection constants
  Constants to be passed as "clockSrc" parameter in SmartIO_ClockSelect() function.
- Component hold override selection constants
  Constants to be passed as "ovCtrl" parameter in SmartIO_HoldOverride() function.
- Terminal selection constants
  Constants to be passed as "portTerm" parameter in SmartIO_IoSyncMode() and SmartIO_ChipSyncMode() functions.
- Look-up table number constants
  Constants to be passed as "lutNum" parameter in SmartIO_LUT_SelectInputs() and SmartIO_LUT_ConfigureMode() functions.
- LUT input number constants
  Constants to be passed as "inputNum" parameter in SmartIO_LUT_SelectInputs() function.
- LUT input source constants
  Constants to be passed as "inputSrc" parameter in SmartIO_LUT_SelectInputs() function.
- LUT mode constants
  Constants to be passed as "mode" parameter in SmartIO_LUT_ConfigureMode() function.

Port channel selection constants

Description

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

Macros

- #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
  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
  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
  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
  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
  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
  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.

LUT mode constants

Description

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

Macros

- #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.

Group_structs

Description

Data Structures

- struct SmartIO_lut_config_struct

Data Structure Documentation

SmartIO_lut_config_struct Struct Reference

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 IO 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 IO 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

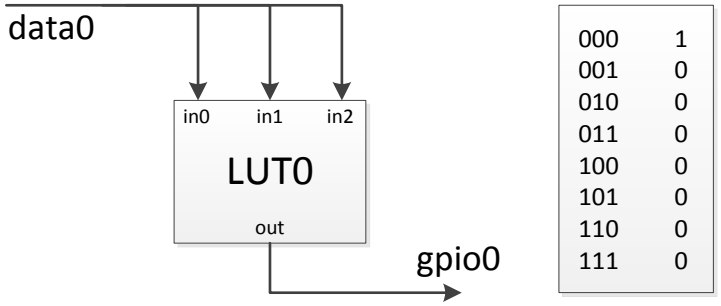
The Smart IO routing fabric is divided into two portions, where each portion is capable of accepting half of the data or GPIO signals. The LUTs have the following structure.

- 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.



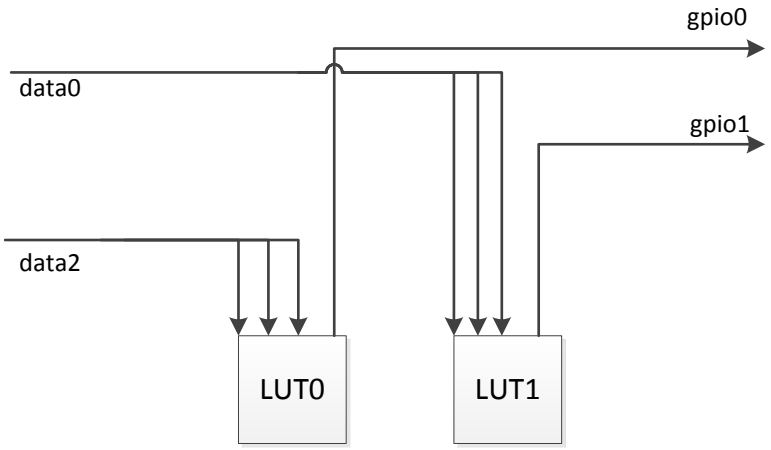
SCB Restriction

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.



Clock and Reset Behavior

The Smart IO 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 IO 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.

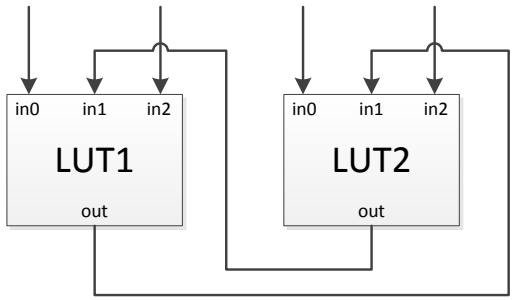
Clock Source	Reset Behavior	Enable Delay	Description
Divided Clock (Active)	Reset when going to Deep Sleep, Hibernate or POR	2 clock edges	The fabric will be enabled after 2 clock edges of the divided clock. Any synchronous logic in the LUTs will be reset to 0 when in chip deep-sleep or hibernate modes.
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 IO 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 IO component is capable of operating during chip Deep-Sleep and Hibernate modes. The block has the following requirements when operating in these modes:

- 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 these modes.

Resources

Each Smart IO component consumes an entire PRGIO 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 Errata

This section lists known problems with the Smart IO component.

Cypress ID	Component Version	Problem	Workaround
239495	1.0	The Smart IO v1.0 component is marked as a prototype because of a defect in the Configure dialog. This defect causes the PSoC Creator window and fonts to automatically resize (or, in extreme cases, disappear) when the Configure dialog is opened. This makes interaction with the tool difficult until you save the workspace and restart PSoC Creator. The problem occurs when the Windows system desktop scaling factor is set above 100%. On Windows 10 machines, this happens automatically when one or more high resolution monitors are connected to the PC.	Change the Windows desktop scaling factor by entering the Windows 10 Settings dialog. Under **Display Settings**, slide the **Change the size of text, apps and other items:** setting down to 100% and click **Apply**. Windows 10 will ask you to sign out of the current session in order for these changes to take effect. It is important to save all work before doing so. The component implementation is unaffected by this defect. So, if you do not experience the resizing problem, it is safe to use Smart IO in your projects. Cypress is working on a fix to the component and will offer a new version in a subsequent component pack.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
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 IO component.
1.0	Initial release	

© Cypress Semiconductor Corporation, 2016. 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, 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.

