Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN220203

Smart I/O Usage Setup in Traveo II Family

Author: Myungsoo Cho

Associated Part Family: Traveo™ II Family CYT2/CYT3/CYT4 Series

Related Documents: see Related Documents

AN220203 describes how to use Smart I/O in Traveo™ II MCUs. Smart I/O adds a programmable logic circuitry between a peripheral and a GPIO port, thereby integrating board-level glue logic.

Contents

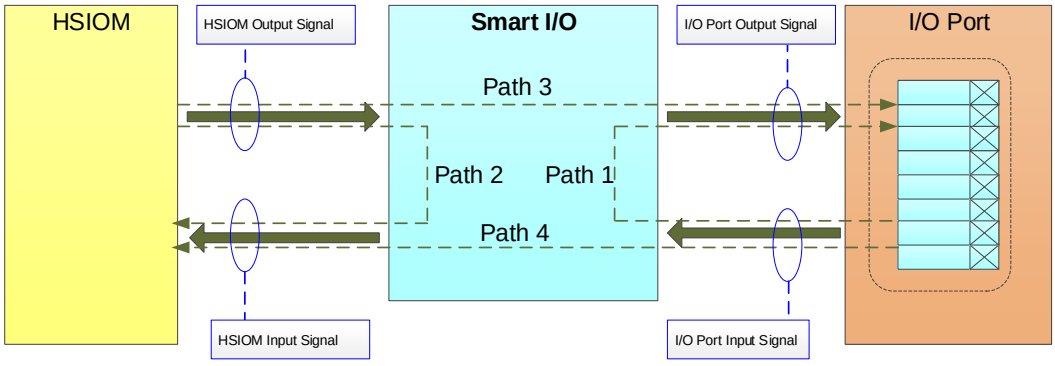
1 Introduction

This application note describes how to use and setup Smart I/O in Cypress Traveo™ II family CYT2/CYT3/CYT4 series MCUs.

Smart I/O adds programmable logic to an I/O port. Smart I/O integrates Boolean logic functionality such as AND, OR, and XOR into a port. It also pre- or post-processes the signals between high-speed I/O matrix (HSIOM) and I/O port. For example, Smart I/O can enable digital glue logic for input signals using multiple flip-flops without CPU intervention. HSIOM multiplexes GPIOs sharing multiple functions into peripheral devices selected by the user. See the Architecture Technical Reference Manual (TRM) for details of HSIOM.

To understand the functionality described and terminology used in this application note, see the Smart I/O chapter in the Architecture TRM. Figure 1 shows examples of typical signal paths.

Figure 1. Smart I/O Interface



Path 1: Implements self-contained logic functions that directly operate on I/O port signals

Path 2: Implements self-contained logic functions that operate on HSIOM signals

Path 3: Logic conversed HSIOM output signals route to I/O port

Path 4: Logic conversed I/O port input signals route to HSIOM

For each signal path, the Smart I/O function gives an option for a programmable output. This application note shows the example usage and configuration of the Smart I/O function.

1.1 Applications of Smart I/O

Smart I/O can be used whenever simple logic operations and routing are required to be performed on signals to or from the I/O pins. Typical applications include the following:

- **Change routing to/from pins:** This function allows rerouting signals from the fixed-function peripherals to non-dedicated pins on the same port.
- **Invert the polarity of signal:** This function inverts the polarity of output signals, such as the SPI signal, before it goes out from a pin.
- **Clock or signal buffer:** This function drives a GPIO input signal, which has to drive a heavier load for one pin, through two GPIO buffers.
- **Detect a pattern on pins:** This function detects the patterns of several signal inputs and outputs the programmable signal depending on the result of detection.

These applications of Smart I/O can work in low-power mode (DeepSleep), therefore can be used as a wakeup interrupt.

1.2 Bypass of Smart I/O

When the Smart I/O function is not used, it will be automatically bypassed by setting the `SMARTIO_PRTx_CTL.ENABLE`[1] bit to “0”: Disabled. It is also possible to bypass any I/O pin in the Port group using the `SMARTIO_PRTx_CTL.BYPASS` bits. When BYPASS bits are set to “1”: Bypass, HSIOM and I/O port are connected directly.

Note that the bypass setting must be configured before enabling the Smart I/O. (`SMARTIO_PRTx_CTL.ENABLE` set to “1”: Enabled)

Table 1 shows the description of the `SMARTIO_PRTx_CTL` register for bypass setting. See the Registers TRM for details.

Table 1. Register for Bypass Setting

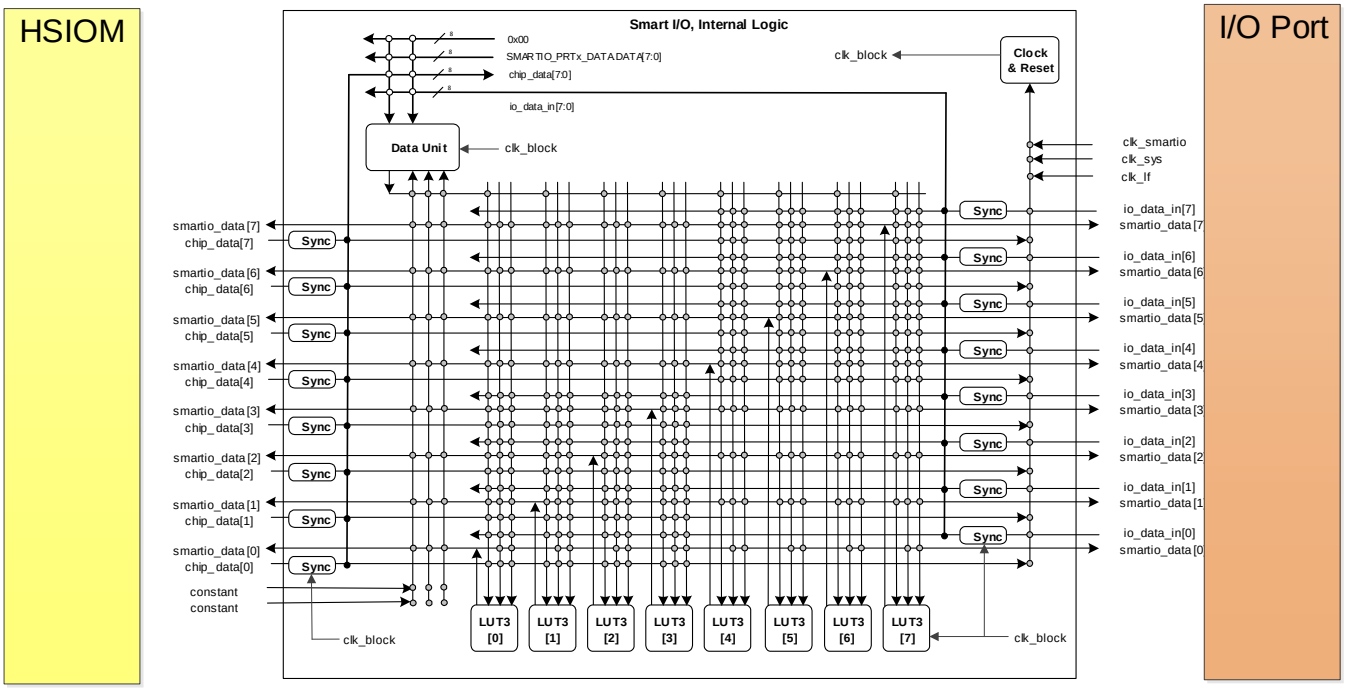
Register	Bit Field	Setting
`SMARTIO_PRTx_CTL`	BYPASS [7:0]	Bypass Smart I/O '0': No bypass (Smart I/O is present in the signal path) '1': Bypass (Smart I/O is absent in the signal path)
	ENABLED [31]	Enable Smart I/O 0: Disabled (Signals are bypassed: default) 1: Enabled (Should only be set to '1' when Smart I/O is completely configured.)

[1] Subscripts x in register names used in this sentence are Port number.

2 Structure of Smart I/O

Figure 2 shows the block diagram of Smart I/O. Smart I/O is positioned in the signal path between the HSIOM and the I/O port.

Figure 2. Block Diagram of Smart I/O



The Smart I/O consists of the following components:

- Clock and reset
- Synchronizer (Sync)
- 3-input lookup tables (LUT3 [x]): x = 0 to 7
- Data unit (DU)

Smart I/O is implemented for the specified I/O cell. Smart I/O can provide programmable signals to HSIOM and I/O port with a combination of these components. See the Package Pin List and Alternate Functions of Device Datasheet for details on the I/O port that can be used as Smart I/O.

The io_data_in [7:0] is the input signal from the I/O port, while the chip_data [7:0] is the input signal from HSIOM. These signals are input to Smart I/O via the Sync components (synchronizer). The smartio_data [7:0] is the output signal from Smart I/O. These signals are routed or modified by Smart I/O and output to the I/O port or HSIOM.

The clk_block is used for all components in Smart I/O. The clk_block can be selected from the I/O port input signals (io_data_in [7:0]), HSIOM input signals (chip_data [7:0]), clk_smartio, and clk_lf. The clk_smartio is derived from the system clock (clk_sys/CLK_HF) using a peripheral clock divider, and the clk_smartio is input in the Clock and Reset block. See the Clocking system chapter of Architecture TRM for details on clk_smartio and clk_lf.

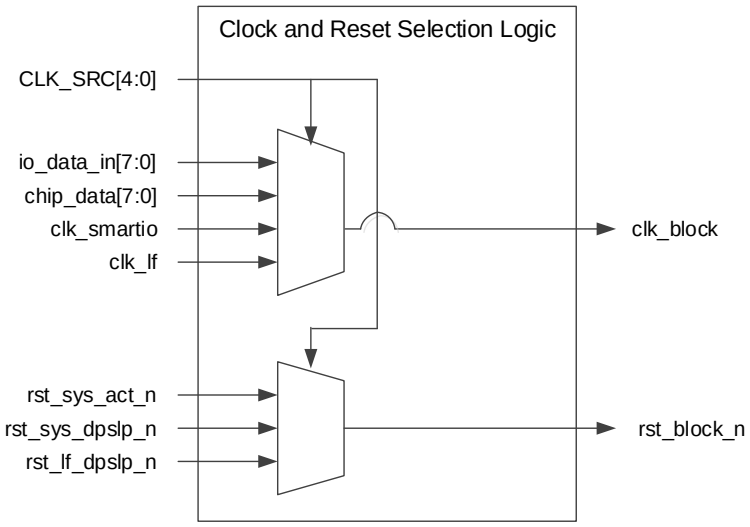
Eight lookup tables (LUT3 [x]) are implemented for each Smart I/O unit. LUT3 [x] can provide the programmable output, and it can decide the connection of signals between HSIOM and the I/O port. This means that the eight lookup tables offer a flexible routing combination of input channel and output.

The data unit can provide enhanced functionality for the output signal. Clock and Reset block is used to synchronize the signals of HSIOM, I/O port, and each block in Smart I/O. Synchronizer controls the synchronization/asynchronization of the HSIOM input and the I/O port input.

2.1 Clock and Reset

Smart I/O can provide reset signals and clock selection. Figure 3 shows the selection logic of the configuration of clock and reset.

Figure 3. Functional Image of Clock and Reset Setting



When io_data_in [7:0] and chip_data [7:0] are selected as the clock source, the clocks have no associated reset. When clk_smartio is selected as the clock source, either rst_sys_act_n or rst_sys_dpslp_n can be used depending on the operating power mode: Active or DeepSleep. When clk_lf is selected as the clock source, rst_lf_dpslp_n can be used. The clock (clk_block) and reset (rst_block_n) can be set by the `SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]` register.

The following clock sources are available for selection:

- io_data_in [7:0]: These are I/O port input signals.
- chip_data [7:0]: These are HSIOM input signals.
- clk_smartio: This clock is derived from the system clock clk_sys/CLK_HF.
- clk_lf: This clock is a low-frequency system clock. This clock is only available in DeepSleep mode.

The following reset sources are available for selection:

- rst_sys_act_n: Smart I/O is active only in Active power mode with the clock from the peripheral divider.
- rst_sys_dpslp_n: Smart I/O is active in all power modes except in DeepSleep mode with the clock from the peripheral divider.
- rst_lf_dpslp_n: Smart I/O is active in all power modes with the clock from ILO.

Table 2 shows the configuration of the `SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]` register. See the Registers TRM for details.

Table 2. Register for Clock and Reset Setting

Register	Bit Field	Setting
SMARTIO_PRTx_CTL	CLOCK_SRC [12:8]	Clock (clk_block)/Reset (rst_block_n) source selection: • 0 ... 7: io_data_in[0]/1 ... io_data_in[7]/'1' • 8... 15: chip_data[0]/1 ...chip_data[7]/'1' • 16: clk_smartio/rst_sys_act_n • 17: clk_smartio/rst_sys_dpslp_n • 19: clk_lf/rst_lf_dpslp_n • 20... 30: Clock source is a constant '0'. • 31: asynchronous mode/"1". Select this when a clockless operation is configured.

2.2 Synchronizer

Each input signal at the I/O port and HSIOM can be used either in synchronous or asynchronous mode. The synchronizer synchronizes the input signal with the Smart I/O clock (clk_block).

Table 3 shows synchronizer setting register and configuration. See the Registers TRM for details.

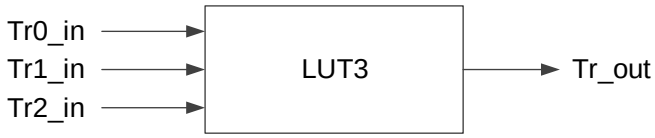
Table 3. Register for Synchronizer Setting

Register	Bit Field	Setting
SMARTIO_PRTx_SYNC_CTL	IO_SYNC_EN [7:0]	Synchronization of the io_data_in [7:0] signals with clk_block 0: No synchronization 1: Synchronization
	CHIP_SYNC_EN [15:8]	Synchronization of the chip_data [7:0] signals with clk_block 0: No synchronization 1: Synchronization

2.3 3-Inputs Lookup Tables (LUT3 [x])

Each LUT3 [x] has three inputs and one output. All inputs (Tr0_in, Tr1_in, Tr2_in) of each LUT3 [x] block should be selected. If there is only one input operation, provide the input to all three input sources (Tr0_in, Tr1_in, Tr2_in). Each LUT3 [x] takes three input signals and generates an output based on the configuration set in register. Figure 4 shows the basic block diagram of each LUT3 [x]. The output pattern can be set by the register.

Figure 4. LUT3 [x] Block Diagram



2.3.1 LUT3 [x] Output Configuration

Output signal (Tr_out) of LUT3 [x] can be programmed using `SMARTIO_PRTx_LUT_CTLy.LUT[7:0]`[2] based on three input sources (Tr0_in, Tr1_in, Tr2_in). Table 4 shows example of setting each LUT3 [x].

Table 4. LUT3 [x] Output Setting

Tr 0_in	Tr 1_in	Tr 2_in	Tr_out	Tr_out (Example 1)	Tr_out (Example 2)
0	0	0	A	0	0
0	0	1	B	0	0
0	1	0	C	0	1
0	1	1	D	0	0
1	0	0	E	1	1
1	0	1	F	1	0
1	1	0	G	1	0
1	1	1	H	1	0

Eight output patterns (A to H) are generated for three input signals. Each output from A to H is a Boolean value of 0 or 1. This output pattern value [H, G, F, E, D, C, B, A] is set in the LUT [7:0].

In case of example 1, the output pattern is [H, G, F, E, D, C, B, A] = [1, 1, 1, 1, 0, 0, 0, 0]. Therefore, the value "0xF0" is set to LUT [7:0]. Also, in the case of example 2, the output pattern is [H, G, F, E, D, C, B, A] = [0, 0, 0, 1, 0, 1, 0, 0]. Therefore, the set value is "0x14" to LUT [7:0].

Table 5 shows the `SMARTIO_PRTx_LUT_CTLy.LUT [7:0]` register for LUT3 [x] output setting. See the Registers TRM for details.

Table 5. Register for Setting the Output from LUT3 [x]

Register	Bit Field	Setting
`SMARTIO_PRTx_LUT_CTLy`	LUT [7:0]	LUT3 [x] configuration. Depending on the LUT opcode (LUT_OPC), internal state and LUT3 [x] input signals tr0_in, tr1_in, and tr2_in, the LUT3 [x] configuration is used to determine the LUT3[x] output signal and the next sequential state.

2.3.2 LUT3 [x] Input Selection

The input sources (Tr0_in, Tr1_in, Tr2_in) of each LUT3 [x] can be selected from the following:

- Data unit output
- Other LUT3 [x] output signal (Tr_out)
- Input signal from HSIOM (chip_data [7:0])
- Input signal from I/O port (io_data_in [7:0])

LUT3[7] to LUT3[4] operate on io_data/chip_data[7] to io_data/chip_data[4], whereas LUT3[3] to LUT3[0] operate on io_data/chip_data[3] to io_data/chip_data[0].

The input sources can be configured with LUT_TR0_SEL [3:0], LUT_TR1_SEL [11:8], and LUT_TR2_SEL [19:16] in the `SMARTIO_PRTx_LUT_SELy` register. Table 6 shows the `SMARTIO_PRTx_LUT_SELy` register and input selection setting. Note that Data Unit output can only be input to tr0_in. See the Registers TRM for details.

[2] Subscripts y in register names used in this sentence are LUT3 number.

Table 6. Register for LUT3 [x] Input Source Setting

Register	Bit Field	Setting
SMARTIO_PRTx_LUT_SELy	LUT_TR0_SEL [3:0]	LUT3 [x] input signal tr0_in source selection: 0: Data unit output 1: LUT3 [1] output 2: LUT3 [2] output 3: LUT3 [3] output 4: LUT3 [4] output 5: LUT3 [5] output 6: LUT3 [6] output 7: LUT3 [7] output 8: chip_data [0] (for LUT3 [0], [1], [2], [3]); chip_data [4] (for LUT3 [4], [5], [6], [7]) 9: chip_data [1] (for LUT3 [0], [1], [2], [3]); chip_data [5] (for LUT3 [4], [5], [6], [7]) 10: chip_data [2] (for LUT3 [0], [1], [2], [3]); chip_data [6] (for LUT3 [4], [5], [6], [7]) 11: chip_data [3] (for LUT3 [0], [1], [2], [3]); chip_data [7] (for LUT3 [4], [5], [6], [7]) 12: io_data_in [0] (for LUT3 [0], [1], [2], [3]); io_data_in [4] (for LUT3 [4], [5], [6], [7]) 13: io_data_in [1] (for LUT3 [0], [1], [2], [3]); io_data_in [5] (for LUT3 [4], [5], [6], [7]) 14: io_data_in [2] (for LUT3 [0], [1], [2], [3]); io_data_in [6] (for LUT3 [4], [5], [6], [7]) 15: io_data_in [3] (for LUT3 [0], [1], [2], [3]); io_data_in [7] (for LUT3 [4], [5], [6], [7])
	LUT_TR1_SEL [11:8] / LUT_TR2_SEL [19:16]	LUT3 [x] input signal tr1_in / tr2_in source selection: 0: LUT3 [0] output 1: LUT3 [1] output 2: LUT3 [2] output 3: LUT3 [3] output 4: LUT3 [4] output 5: LUT3 [5] output 6: LUT3 [6] output 7: LUT3 [7] output 8: chip_data [0] (for LUT3 [0], [1], [2], [3]); chip_data [4] (for LUT3 [4], [5], [6], [7]) 9: chip_data [1] (for LUT3 [0], [1], [2], [3]); chip_data [5] (for LUT3 [4], [5], [6], [7]) 10: chip_data [2] (for LUT3 [0], [1], [2], [3]); chip_data [6] (for LUT3 [4], [5], [6], [7]) 11: chip_data [3] (for LUT3 [0], [1], [2], [3]); chip_data [7] (for LUT3 [4], [5], [6], [7]) 12: io_data_in [0] (for LUT3 [0], [1], [2], [3]); io_data_in [4] (for LUT3 [4], [5], [6], [7]) 13: io_data_in [1] (for LUT3 [0], [1], [2], [3]); io_data_in [5] (for LUT3 [4], [5], [6], [7]) 14: io_data_in [2] (for LUT3 [0], [1], [2], [3]); io_data_in [6] (for LUT3 [4], [5], [6], [7]) 15: io_data_in [3] (for LUT3 [0], [1], [2], [3]); io_data_in [7] (for LUT3 [4], [5], [6], [7])

Each LUT3 [x] has limited connections with input/output of HSIOM and I/O port signals. Sometimes, multiple LUT3 [x] are necessary for a complete flexible routing.

The LUT3 [x] and data unit do not include any combinatorial loops. However, when one LUT3 [x] interacts with the other or to the data unit, inadvertent combinatorial loops are possible. To overcome this limitation, the SMARTIO_PRTx_CTL.PIPELINE_EN bit is used. When set, all outputs (LUT3 [x] and data unit) are registered before branching out to other components. Table 7 shows PIPELINE_EN setting. This bit is set to "1" (Enabled) to ensure low power consumption, if Smart I/O is not used. See the Registers TRM for details.

Table 7. PIPELINE_EN Setting

Register	Bit Field	Setting
SMARTIO_PRTx_CTL	PIPELINE_EN [25]	Enable for pipeline register: 0: Disabled (Register is bypassed) 1: Enabled (Default value)

2.3.3 LUT3 [x] Operation

Each LUT3 [x] has the following four operations selected by a 2-bit Op Code field. The four operations are:

- **Combinatorial**
  LUT3 [x] is purely combinatorial. Each LUT3 [x] output is the result of the LUT mapping truth table, and will only be delayed by the LUT3 [x] combinatorial path (Basic mode).

Figure 5. Combinatorial

- **Gated Input 2**
  LUT3 [x] input 2 is registered. Other inputs are directly connected to LUT3 [x]. The output is combinatorial (Input synchronization).

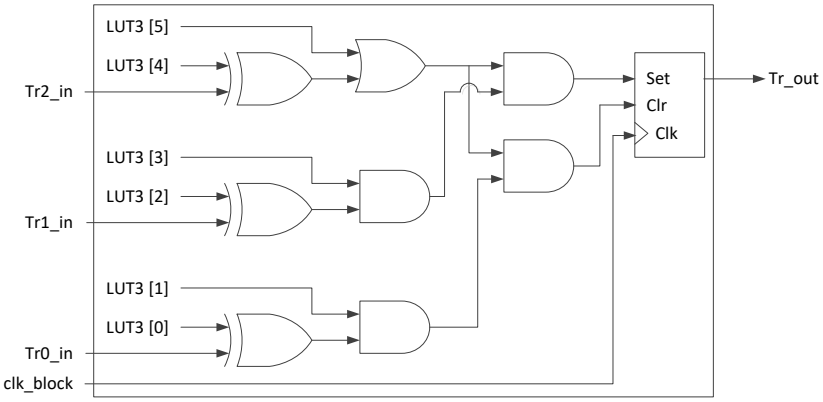
Figure 6. Gated Input2

- **Gated Output**
  Inputs are directly connected to LUT3 [x] and the output is registered (Output synchronization).

Figure 7. Gated Output

- **Set/reset flip-flop**
  Input signals are used to control an S/R flip-flop.

Figure 8. S/R Flip-Flop Enable



These four operations can be set with the register shown in Table 8. See the Registers TRM for details.

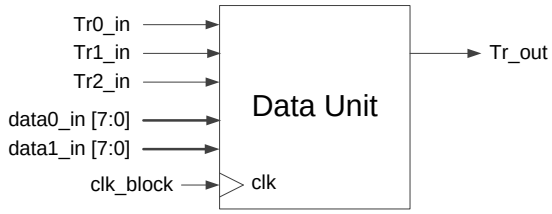
Table 8. Register for LUT3 [x] Mode Setting

Register	Bit Field	Setting
SMARTIO_PRTx_LUT_CTLy	LUT_OPC [9:8]	0: Combinatorial 1: Gated Input 2 2: Gated Output 3: Set/reset flip-flop

2.4 Data Unit (DU)

Each Smart I/O block includes a data unit (DU) component. DU consists of a simple 8-bit data path. It is capable of performing simple increment, decrement, increment/decrement, shift, and AND/OR operations. DU can generate a programmable output (Tr_out) signal based on two 8-bit data inputs that DATA0 (data0_in [7:0]) and DATA1 (data1_in [7:0]). The internal state is captured in flip-flops. The DU behavior can be controlled by up to three input signals (Tr0_in, Tr1_in, Tr2_in). Figure 9 shows the basic block diagram of Data Unit.

Figure 9. Data Unit Block Diagram



2.4.1 Input Selection

DU has up to three control input signals. These signals can be selected as input from the following.

- Constant “0”
- Constant “1”
- DU output
- LUT3 [x] outputs

The number of control signals required depends on the DU operation code.

These inputs can be configured with the SMARTIO_PRTx_DU_SEL register. Table 9 shows the SMARTIO_PRTx_DU_SEL register and input selection setting. See the Registers TRM for details.

Table 9. Register for DU Inputs Source Setting

Register	Bit Field	Setting
SMARTIO_PRTx_DU_SEL	DU_TR0_SEL [3:0] / DU_TR1_SEL [11:8] / DU_TR2_SEL [19:16]	Data unit input signal “tr0_in” / “tr1_in” / “tr2_in” source selection: 0: Constant '0' 1: Constant '1' 2: Data unit output 3- 10: LUT3 [x] outputs Otherwise: Undefined

DATA 0 and DATA 1 use input data for DU logic to be initialized. These data can be selected from the following:

- Constant 0x00
- io_data_in [7:0]
- chip_data_in [7:0]
- DATA [7:0] bits of SMARTIO_PRTx_DATA register

The data width handled by the data unit can be changed between 1 bit and 8 bits. Table 10 shows the configuration registers for input data to DU.

Table 10. Register for DU Data Setting

Register	Bit Field	Setting
SMARTIO_PRTx_DU_SEL	DU_DATA0_SEL [25:24] / DU_DATA1_SEL [29:28]	Data unit input data “data0_in” / “data1_in” source selection: 0: 0x00 1: chip_data [7:0]. 2: io_data_in [7:0]. 3: SMARTIO_PRTx_DATA.DATA [7:0] MMIO register field.
SMARTIO_PRTx_DATA	DATA [7:0]	Data unit input data source
SMARTIO_PRTx_DU_CTL	DU_SIZE [2:0]	Size/width of the data unit (in bits) is DU_SIZE+1.

2.4.2 Operation of Data Unit

The DU operation is defined by SMARTIO_PRTx_DU_CTL.DU_OPC [11:8]. Table 11 shows the configuration registers for DU operation code setting. See the Registers TRM for details.

Table 11. DU Operation Code Configuration

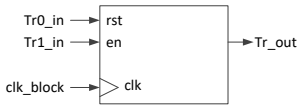
Register	Bit Field	Setting
SMARTIO_PRTx_DU_CTL	DU_OPC [11:8]	Data unit opcode specifies the data unit operation: "1": INCR "2": DECR "3": INCR_WRAP "4": DECR_WRAP "5": INCR_DECR "6": INCR_DECR_WRAP "7": ROR "8": SHR "9": AND_OR "10": SHR_MAJ3 "11": SHR_EQL Otherwise: Undefined Default Value: Undefined

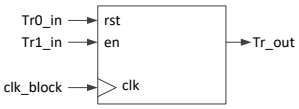
Table 12 shows each DU operation.

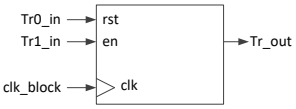
The ‘Operation’ column in Table 12 shows the operation outline and the pseudo code. In the pseudo code, “Combinational:” indicates that the operations are independent of previous output states. "Registered:" indicates that data operates on inputs and previous output states (registered using flip-flops).

Table 12. DU Operation

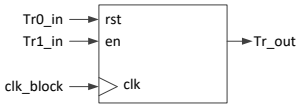
Operation Code	Operation

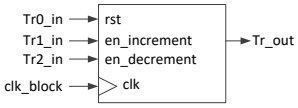
DU_OPC [11:8] = 1: INCR	INCR increments data by 1 from an initial value (DATA 0) until it reaches a final value (DATA 1).
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? data: (data + 1) & mask;`

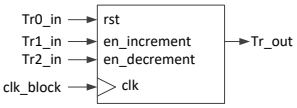
DU_OPC [11:8] = 2: DECR	DECR decrements data from an initial value (DATA 0) until it reaches '0'.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	
	Combinational:
	`Tr_out = data_eql_0`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_0 ? data: (data - 1) & mask;`

DU_OPC [11:8] = 3: INCR_WRAP	INCR_WRAP operates similar to INCR, but instead of stopping at DATA 1, it wraps around to DATA 0.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? DATA0 & mask: (data + 1) & mask;`

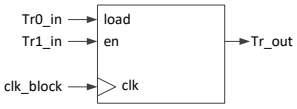
Operation Code	Operation

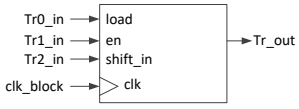
DU_OPC [11:8] = 4: DECR_WRAP	DECR_WRAP works similar to DECR. Instead of stopping at '0', it wraps around to DATA0.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	
	Combinational:
	`Tr_out = data_eql_0`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_0? DATA0 & mask: (data + 1) & mask;`

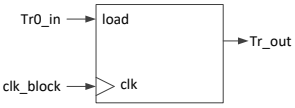
DU_OPC [11:8] = 5: INCR_DECR	INCR_DECR is a combination of INCR and DECR. Depending on the trigger signals, it either starts incrementing or decrementing. Increment stops at DATA 1 and decrement stops at '0'.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1 | data_eql_0`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? data: (data + 1) & mask;`
	`else if (Tr2_in)`
	`data <= data_eql_0? data: (data - 1) & mask;`

DU_OPC [11:8] = 6: INCR_DECR_WRAP	INCR_DECR_WRAP has the same functionality as INCR_DECR with wrap around to DATA 0 on reaching the limits (DATA 1 or '0').
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1 | data_eql_0`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? DATA0 & mask: (data + 1) & mask;`
	`else if (Tr2_in)`
	`data <= data_eql_0 ? DATA0 & mask: (data - 1) & mask;`

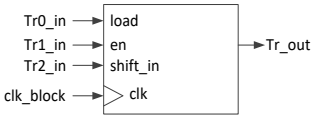
Operation Code	Operation

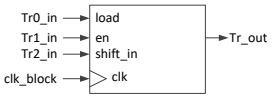
DU_OPC [11:8] = 7: ROR	POR rotates the data right and the LSB is sent out. The data for rotation is taken from DATA 0.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	
	Combinational:
	`Tr_out = data [0]`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= data [7:1] & mask;`
	`data [du_size] <= data [0]`
	`}`
	 

DU_OPC [11:8] = 8: SHIR	SHIR performs the shift register operation. Initial data (DATA 0) is shifted out and data on tr2_in is shifted in.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	
	Combinational:
	`Tr_out = data [0]`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= data [7:1] & mask;`
	`data [du_size] <= Tr2`
	`}`
	 

DU_OPC [11:8] = 9: AND_OR	ANDs data1 and data0 along with mask; then, ORs all bits of the ANDed output
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	
	Combinational:
	`Tr_out =| (data & DATA1 & mask)`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	 

Operation Code	Operation

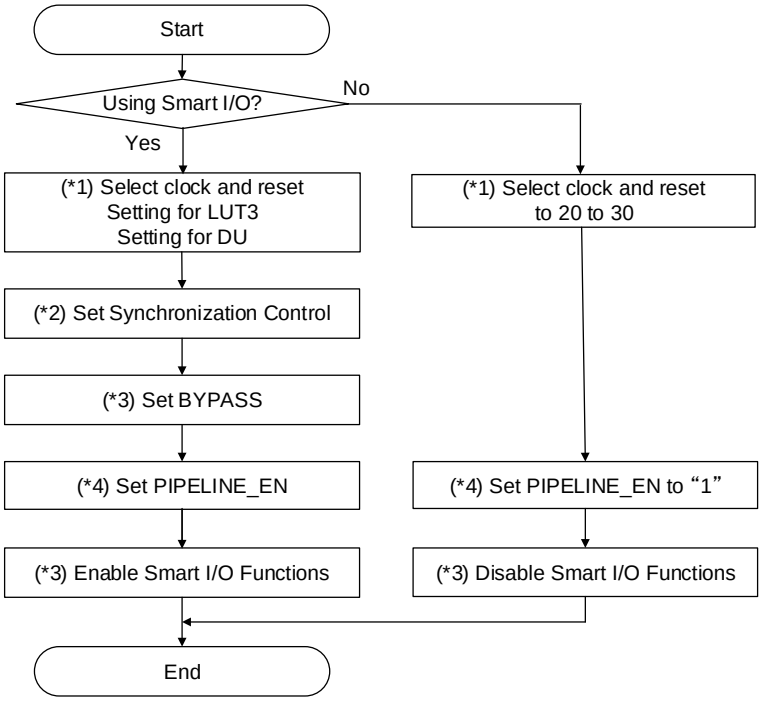
DU_OPC [11:8] = 10: SHR_MAJ3 (Majority 3)	SHR_MAJ3 performs the same functionality as SHR. Instead of sending the shifted-out value, it sends a '1', if at least two samples are high in the last three samples/shifted-out values of data [0]. Otherwise, it sends a '0'. This function sends out the majority of the last three samples.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	
	Combinational:
	`Tr_out = data == 0x03 | data == 0x05 | data == 0x06 | data == 0x07`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= (0, data [7:1]) & mask;`
	`data [du_size] <= Tr2_in`
	`}`
	 

DU_OPC [11:8] = 11: SHR_EQL (Match DATA1)	SHR_EQL performs the same operation as SHR. Instead of shift-out, the output is the comparison result (DATA 0 == DATA 1).
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= (0, data [7:1]) & mask;`
	`data [du_size] <= Tr2_in`
	`}`
	 

3 Smart I/O Configuration

Figure 10 shows an example of the configuration flow of Smart I/O.

Figure 10. Smart I/O Configuration Flow



When configuring Smart I/O, first initialize each component such as clock and reset, synchronizer, LUT3 [x] and DU. Before enabling Smart I/O (`SMARTIO_PRTx_CTL.ENABLE` set to “1”: Enabled), all components and routing should be configured.

If Smart I/O is not used, clock selection in clock and reset component should be set to a value between 20 to 30, and PIPELINE_EN should be set to “1”, to ensure low power consumption.

Note: (*1) See Structure of for ports, source and clock setting, see 3-Inputs Lookup Tables (LUT3 [x]) for LUT3 [x] setting, and see Data Unit (DU) for DU setting.
(*2) See Table 3 for Synchronization setting.
(*3) See Table 1 for bypass and Smart I/O enable setting.
(*4) See Table 7 for PIPELINE_EN setting.

4 Example Configuration

Smart I/O can be useful for an application that involves simple logic operations for input/output signal, or the internal routing between internal HSIOM port and the I/O port. No CPU is required for these operations. This section explains how to use Smart I/O according to the use case.

4.1 Change Routing from I/O Pins to HSIOM by Inverting Polarity

This section explains an example of routing and simple logic operations by using Smart I/O.

In this use case, routing is changed to connect the input from pin 7 of Port 0 (io_data_in [7]) to pin 1 of HSIOM (smartio_data [1]). In addition, the polarity of io_data_in [7] is inverted, and the inverted io_data_in [7] signal is output to smartio_data [1]. See the Package Pin List and Alternate Functions of Device Datasheet for I/O port which can used Smart I/O.

Figure 11 shows the connection from the I/O port to HSIOM with signal inverting. LUT3 [1] and LUT3 [7] are used for this use case.

Figure 11. Signal Inverting Image

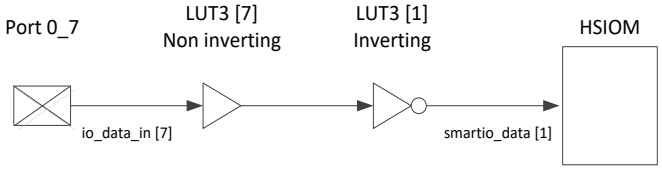
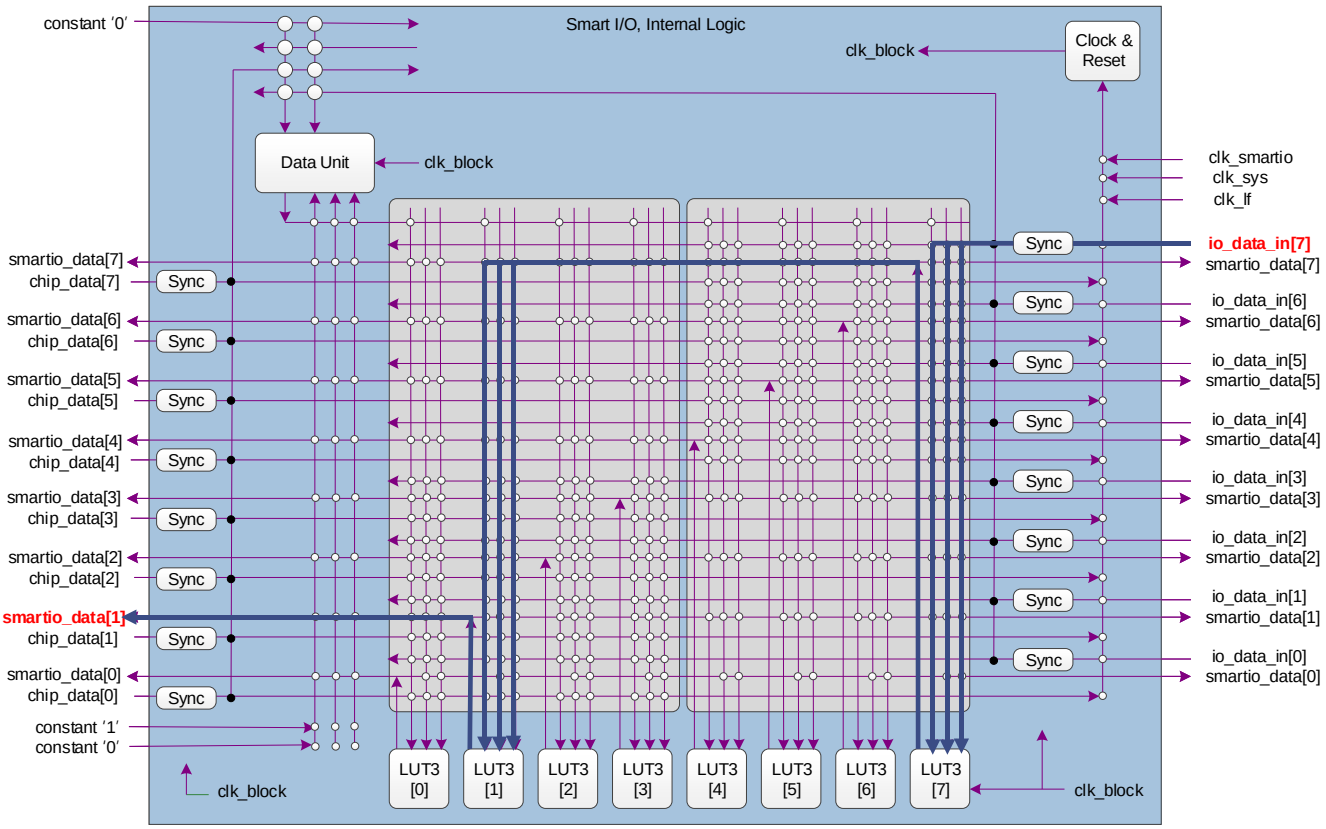


Figure 12 shows the signal path of this example.

Figure 12. Example of the Overview of Routing



Note that LUT3 [1] and LUT3 [7] are used. LUT3[7:4] that can use io_data [7] as input cannot be routed to smartio_data [1] directly. Therefore, output of LUT3 [7] must go through LTU3 [1] which can be routed to smartio_data [1]. In this use case, LUT3 [1] inverts the input signal from LUT3 [7] and outputs it to smartio_data [1].

Table 13 shows the truth table of LUT3 [7] and Table 14 shows the truth table of LUT3 [1]. The blue highlights in the tables indicate an invalid combination pattern.

Table 13. Look Up Table LUT3 [7]

Tr0_in	Tr1_in	Tr2_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 14. Look Up Table LUT3 [1]

Tr0_in	Tt1_in	Tr2_in	Tr_out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

The three inputs of LUT3 [1] are input same signal, Similarly, LUT3 [7] are input same signal. Therefore, the input pattern of LTU3s is [Tr0_in, Tr1_in, Tr2_in] = [0, 0, 0] or [1, 1, 1].

LUT3 [7] does not change polarity. That is, Tr_out is “1”, when [Tr0_in, Tr1_in, Tr2_in] = [1, 1, 1], and otherwise, Tr_out = “0”.

LUT3 [1] reverses polarity. That is, Tr_out is “0”, when [Tr0_in, Tr1_in, Tr2_in] = [1, 1, 1], and otherwise, Tr_out = “1”.

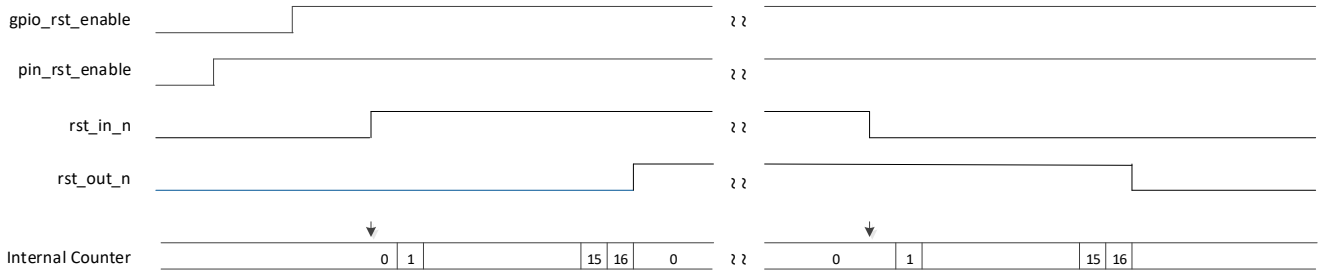
The setting procedure of this use case is as follows:

1. Clock and reset component setting
   - `SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]` = 0x10 (clk_smartio/rst_sys_act_n)
2. LUT3 [1] setting
   - `SMARTIO_PRTx_LUT_CTL1.LUT [7:0]` = 0x7F (see the Table 14)
   - `SMARTIO_PRTx_LUT_CTL1.LUT_OPC [9:8]` = 0 (Combinatorial)
   - `SMARTIO_PRTx_LUT_SEL1.LUT_TR0_SEL [3:0]` = 7 (LUT3 [7] output)
   - `SMARTIO_PRTx_LUT_SEL1.LUT_TR1_SEL [11:8]` = 7 (LUT3 [7] output)
   - `SMARTIO_PRTx_LUT_SEL1.LUT_TR2_SEL [19:16]` = 7 (LUT3 [7] output)
3. LUT3 [7] setting
   - `SMARTIO_PRTx_LUT_CTL7.LUT [7:0]` = 0x80 (See the Table 13)
   - `SMARTIO_PRTx_LUT_CTL7.LUT_OPC [9:8]` = 0 (Combinatorial)
   - `SMARTIO_PRTx_LUT_SEL7.LUT_TR0_SEL [3:0]` = 15 (io_data_in [7])
   - `SMARTIO_PRTx_LUT_SEL7.LUT_TR1_SEL [11:8]` = 15 (io_data_in [7])
   - `SMARTIO_PRTx_LUT_SEL7.LUT_TR2_SEL [19:16]` = 15 (io_data_in [7])
4. Synchronizer setting
   - `SMARTIO_PRTx_SYNC_CTL.IO_SYNC_EN [7:0]` = 0x00 (No synchronization for io_data_in [7])
5. BYPASS setting
   - `SMARTIO_PRTx_CTL.BYPASS [7:0]` = 0x7D (io_data_in [7] and smartio_data [1] are not bypassed)
6. PIPELINE setting
   - `SMARTIO_PRTx_CTL.PIPELINE_EN [25]` = 0 (Disabled)
7. Smart I/O enabling
   - `SMARTIO_PRTx_CTL.ENABLED [31]` = 1 (Enabled after configuration is done)

4.2 Reset Detection/Stability Circuitry

This section explains how to implement a reset detection/stability circuitry on the Smart I/O. Figure 13 shows the operation of reset detection/stability.

Figure 13. Operation of Reset Detection/Stability Circuity



In this use case, circuitry has two enable signals; pin_rst_enable and gpio_rst_enable. The pin_rst_enable is an enable signal from external circuitry and the gpio_rst_enable is an enable control signal by software. When both signals are enabled, the circuitry is active.

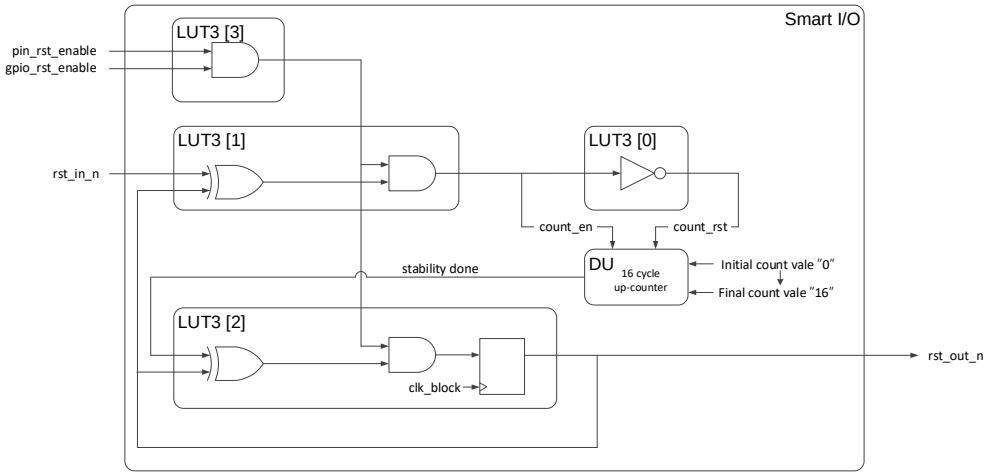
The rst_in_n is an external reset input with active low, and rst_out_n is a reset output with active low. The circuitry monitors rst_in_n. When rst_in_n is activated for a specific number of continuous cycles, the rst_out_n is output. A reset will be activated or released, when the operation clock selected by CLOCK_SRC [12:8] is input continuously for 16 cycles.

The following I/O port and HSIOM signals are used:

- io_data_in [0] = pin_rst_enable; (from I/O port)
- io_data_in [1] = rst_in_n; (from I/O port)
- smartio_data [2] = rst_out_n; (to I/O port)
- chip_data [0] = gpio_rst_enable; (from HSIOM)

Figure 14 shows the connection and functional logic of each LUT3 [3:0] and DU in this circuity.

Figure 14. Logical Example of a Reset Detection/Stability Circuitry



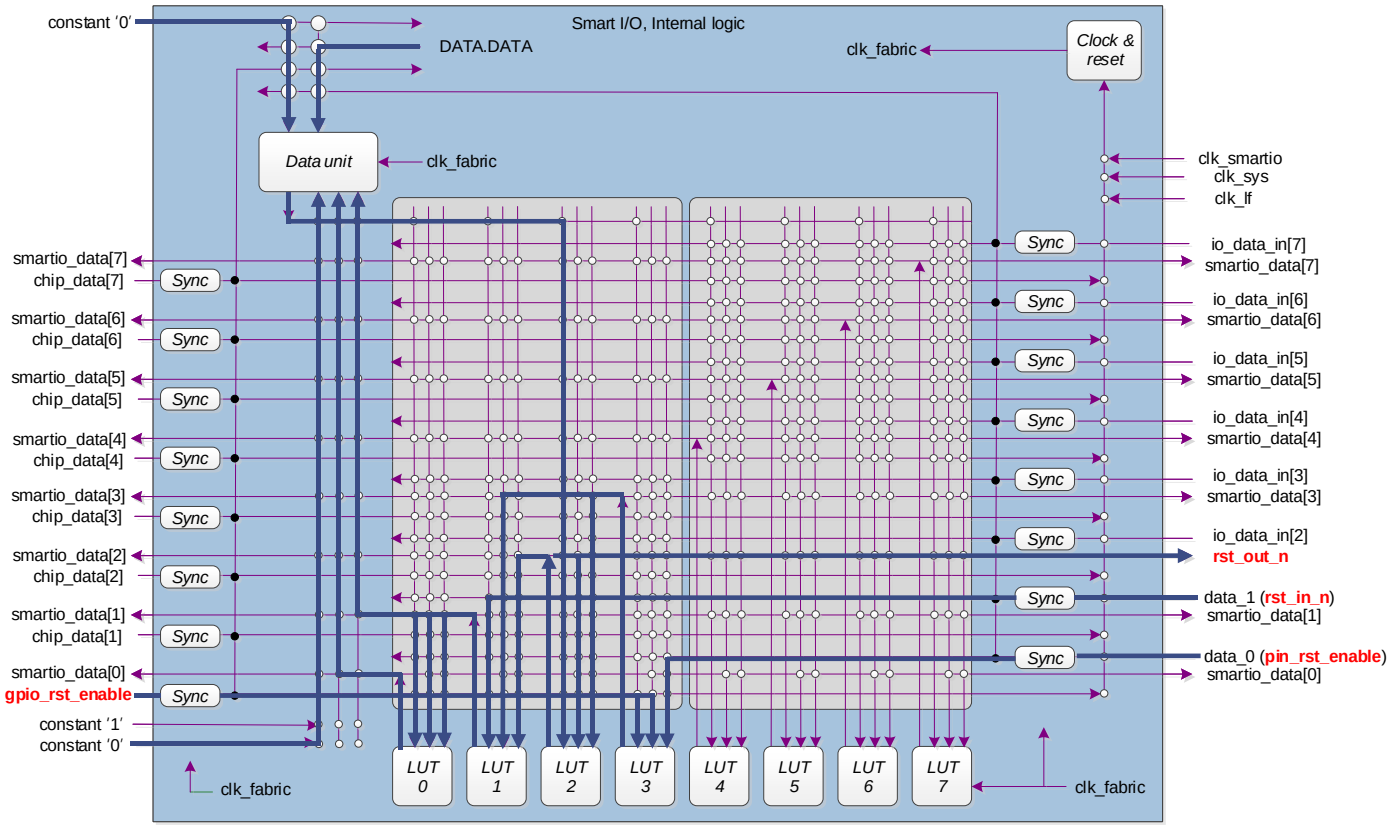
In this use case, four LUT3s and one DU are used.

LUT3 [3] is used to generate the activation signal of this circuitry from two enable signals (pin_rst_enable and gpio_rst_en). LUT 3 [0] and LUT 3 [1] are used to monitor the rst_in_n state and to start the counter of the DU. LUT 3 [2] detects the stabilization wait completion and outputs rst_out_n.

DU is used to generate reset stability wait time, and the Tr_out of LUT3 [2] is output synchronously by gated output mode.

Figure 15 shows the signal path of this use case.

Figure 15. Signal Path of Reset Detection/Stability Circuitry



In this use case, io_data_in [1:0], chip_data [0] and smartio_data [2] are used as input or output signals. Therefore, it can be configured with four LUT3s (LUT3 [3:0]). If smartio_data [4] is used for rst_out_n, it is necessary to go through LUT3 [4]. That is, five LUT3 [x] are required for this case.

Table 15, Table 16, Table 17, and Table 18 show truth table of each LUT3. The blue highlights in the tables indicate an invalid combination pattern.

Table 15. Look Up Table LUT3 [0]

Tr0_in	Tr1_in	Tr2_in	Tr_out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 16. Look Up Table LUT3 [1]

Tr0_in	Tr1_in	Tr2_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Table 17. Look Up Table LUT3 [2]

Tr0_in	Tr1_in	Tr2_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Table 18. Look Up Table LUT3 [3]

Tr0_in	Tr1_in	Tr2_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

LUT3 [0] is an inverter circuit with one input and one output, and each input of Tr0_in, Tr1_in and Tr2_in is the same signal. Therefore, valid combination pattern of [Tr0_in, Tr1_in, Tr2_in, Tr_out] is [0, 0, 0, 1] or [1, 1, 1, 0]. If an invalid pattern occurs, the counter circuit is reset and rst_out_n keeps the current value.

LUT3 [1] has three different inputs. If the enable signal from LUT3 [3] is valid (=”1”) and the rst_in_n state is different from rst_out_n state, “1” is output.

LUT3 [2] generates rst_out_n signal. When the enable signal from LUT3 [3] is valid (=”1”) and the stability done signal from DU is detected (stabilization wait time has passed), the current rst_out_n signal is reversed.

LUT3 [3] generates the enable signal for this circuitry. It has two inputs; pin_rst_enable and gpio_rst_enable. gpio_rst_enable is input to Tr0_in and Tr1_in. Therefore, different value combination of Tr0_in and Tr1_in are invalid pattern. If an invalid pattern occurs, the circuitry is disabled (Tr_out = "0").

DU operates in the INCR_WRAP mode. This mode increments data by 1 from an initial value (DATA 0) until it reaches a final value (DATA 1). When the count value matches the final value, it wraps around to DATA 0. If rst is "1", the counter value is set to initial value.

In this mode, DU has two control signal input (Tr0_in and Tr1_in), two counter control registers (DATA 0 and DATA 1) and one output signal (Tr_out).

Table 19 shows DU configuration and input/output operation.

Table 19. DU Operation

Tr0_in (rst)	Tr1_in (en)	Operation	DATA 0 (Initial value)	DATA 1 (Final value)	Tr_out
1	0	INCR_WRAP: Increments data by 1 from an initial value (DATA 0) until it reaches a final value (DATA 1). When count value matches the final value, it wraps around to DATA 0.	0	16	0 (It is the reset state)
0	1				A single clock pulse is output when the count value is equal to the final value.

Tr0_in operates as “rst” and Tr1_in operates as “en”. Tr0_in is connected to the output of LUT3 [0] and Tr1_in is connected to the input of LUT3 [0]. When "1" is input to en, the DU starts a counter. Then, outputs the single pulse, when counter value reaches the final value.

The setting procedure of this use case is as follows:

1. Clock and reset component setting
   - `SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]` = 0x10 (clk_smartio/rst_sys_act_n)
2. LUT3 [0] setting
   - `SMARTIO_PRTx_LUT_CTL0.LUT [7:0]` = 0xEF (see Table 15)
   - `SMARTIO_PRTx_LUT_CTL0.LUT_OPC [9:8]` = 0 (Combinatorial)

- SMARTIO_PRTx_LUT_SEL0.LUT_TR0_SEL [3:0] = 1 (LUT3 [1] output)
- SMARTIO_PRTx_LUT_SEL0.LUT_TR1_SEL [11:8] = 1 (LUT3 [1] output)
- SMARTIO_PRTx_LUT_SEL0.LUT_TR2_SEL [19:16] = 1 (LUT3 [1] output)

3. LUT3 [1] setting
   - SMARTIO_PRTx_LUT_CTL1.LUT [7:0] = 0x28 (see Table 16)
   - SMARTIO_PRTx_LUT_CTL1.LUT_OPC [9:8] = 0 (Combinatorial)
   - SMARTIO_PRTx_LUT_SEL1.LUT_TR0_SEL [3:0] = 13 (io_data_in [1])
   - SMARTIO_PRTx_LUT_SEL1.LUT_TR1_SEL [11:8] = 2 (LUT3 [2] output)
   - SMARTIO_PRTx_LUT_SEL1.LUT_TR2_SEL [19:16] = 3 (LUT3 [3] output)
4. LUT3 [2] setting
   - SMARTIO_PRTx_LUT_CTL2.LUT [7:0] = 0x28 (see Table 17)
   - SMARTIO_PRTx_LUT_CTL2.LUT_OPC [9:8] = 2 (Gated output)
   - SMARTIO_PRTx_LUT_SEL2.LUT_TR0_SEL [3:0] = 0 (DU output)
   - SMARTIO_PRTx_LUT_SEL2.LUT_TR1_SEL [11:8] = 2 (LUT3 [2] output)
   - SMARTIO_PRTx_LUT_SEL2.LUT_TR2_SEL [19:16] = 3 (LUT3 [3] output)
5. LUT3 [3] setting
   - SMARTIO_PRTx_LUT_CTL3.LUT [7:0] = 0x80 (see Table 18)
   - SMARTIO_PRTx_LUT_CTL3.LUT_OPC [9:8] = 0 (Combinatorial)
   - SMARTIO_PRTx_LUT_SEL3.LUT_TR0_SEL [3:0] = 8 (chip_data [0])
   - SMARTIO_PRTx_LUT_SEL3.LUT_TR1_SEL [11:8] = 8 (chip_data [0])
   - SMARTIO_PRTx_LUT_SEL3.LUT_TR2_SEL [19:16] = 12 (io_data_in [0])
6. DU setting
   - SMARTIO_PRTx_DU_CTL.DU_OPC [11:8] = 3 (INCR_WRAP mode)
   - SMARTIO_PRTx_DU_CTL.DU_SIZE [2:0] = 7 (DU width is 8)
   - SMARTIO_PRTx_DU_SEL.DU_TR0_SEL [3:0] = 3 (LUT3 [0])
   - SMARTIO_PRTx_DU_SEL.DU_TR1_SEL [11:8] = 4 (LUT3 [1])
   - SMARTIO_PRTx_DU_SEL.DU_TR2_SEL [19:16] = 0 (Constant “0”)
   - SMARTIO_PRTx_DU_SEL.DU_DATA0_SEL [25:24] = 0 (Fixed “0x00”)
   - SMARTIO_PRTx_DU_SEL.DU_DATA1_SEL [29:28] = 3 (SMARTIO_PRTx_DATA.DATA [7:0])
   - SMARTIO_PRTx_DATA.DATA [7:0] = 16
7. Synchronizer setting
   - SMARTIO_PRTx_SYNC_CTL.IO_SYNC_EN [7:0] = 0x00 (No synchronization for io_data_in [7])
8. BYPASS setting
   - SMARTIO_PRTx_CTL.BYPASS [7:0] = 0xF0 (io_data_in [3:0] is no bypass)
9. PIPELINE setting
   - SMARTIO_PRTx_CTL.PIPELINE_EN [25] = 0 (Disabled)
10. Smart I/O enabling
    - SMARTIO_PRTx_CTL.ENABLED [31] = 1 (Enabled after configuration is done)

5 Glossary

Terms	Description
HSIOM	High Speed I/O Matrix. See the High-Speed I/O Matrix section in the I/O System chapter of the Architecture TRM for details.
GPIO	General-purpose input/output
I/O Port	I/O Port provides the interface between the CPU core and peripheral components to the outside world. See the I/O System chapter of the Architecture TRM for details.
LUT3 [x]	3-input Lookup Tables. LUT3 [x] block takes three input signals and generates an output based on the configuration set in register. See the Smart I/O - LUT3 section in the I/O System chapter of the Architecture TRM for details.
DU	Data Unit. DU performs simple increment, decrement, increment/decrement, shift, and AND/OR operations based on opcode configuration in register. See the Smart I/O - Data Unit section in the I/O System chapter of the Architecture TRM for details.
DeepSleep	Power mode that only low-frequency peripherals are available. See the DeepSleep Mode section in the Device Power Modes chapter of the Architecture TRM for details.
io_data_in	Input signals from I/O port
chip_data	Input signals from HSIOM
smartio_data	Output signals from Smart I/O
Clk_sys/CLK_HF	This is derived from the system clock using a peripheral clock divider. See the Clocking System chapter of the Architecture TRM for details.

6 Related Documents

The following are the Traveo II family series datasheets and Technical Reference Manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- Body Controller Entry Family
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- Body Controller High Family
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT4BF
  - Traveo II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D Family
  - Traveo™ II Automotive Cluster 2d Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2d Registers Technical Reference Manual (TRM)

Document History

Document Title: AN220203 – Smart I/O Usage Setup in Traveo II Family

Document Number: 002-20203

Revision	ECN	Submission Date	Description of Change
**	6100786	09/27/2018	New Application Note.
*A	6597050	06/17/2019	Added target part number (CYT4B series)
*B	6733892	11/20/2019	Added target part number (CYT4D series)
*C	6908342	06/29/2020	Changed target parts number (CYT2/CYT4 series) Added target part number (CYT3 series)

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709