Smart I/O usage setup in XMC7000 family

About this document

Scope and purpose

This document describes how to use Smart I/O in XMC7000 MCUs. Smart I/O adds a programmable logic circuitry between a peripheral and a GPIO port, thereby integrating board-level glue logic.

Associated part family

XMC7000 family of XMC™ industrial microcontrollers.

Table of contents

About this document **1**
Table of contents **2**
1 Introduction **3**
1.1 Applications of Smart I/O 3
1.2 Bypass of Smart I/O 4
2 Structure of Smart I/O **5**
2.1 Clock and reset 6
2.2 Synchronizer 7
2.3 3-inputs lookup tables (LUT [x]) 7
2.3.1 LUT [x] output configuration 8
2.3.2 LUT [x] input selection 8
2.3.3 LUT [x] operation 10
2.4 Data unit (DU) 12
2.4.1 Input selection 12
2.4.2 Operation of data unit 13
3 Smart I/O configuration **19**
4 Example **22**
4.1 Pulse counting 22
4.1.1 Configuration in Device Configurator 22
4.1.2 Reference code 25
4.2 Ramping LED using Smart I/O 26
4.2.1 Configuration in Device Configurator 26
4.2.2 Reference code 30
5 Glossary **33**
References **34**
Revision history **35**
Disclaimer **36**

1 Introduction

This application note describes how to use and set up Smart I/O in Infineon XMC7000 family XMC7100/XMC7200 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 the 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 reference manual 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 reference manual. Figure 1 shows examples of typical signal paths.

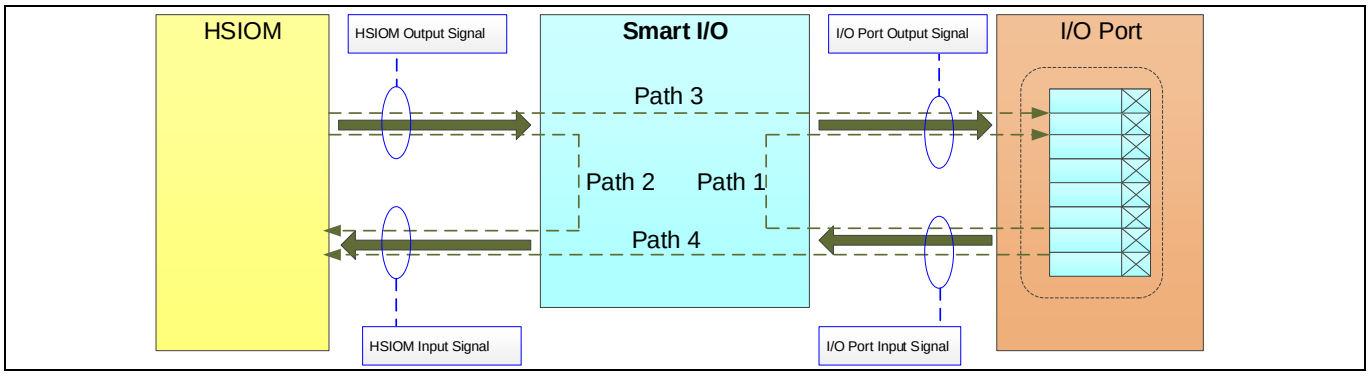


Figure 1 Smart I/O interface

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 the 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 the 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 the 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 ports are connected directly.

Note: 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 reference manual 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.

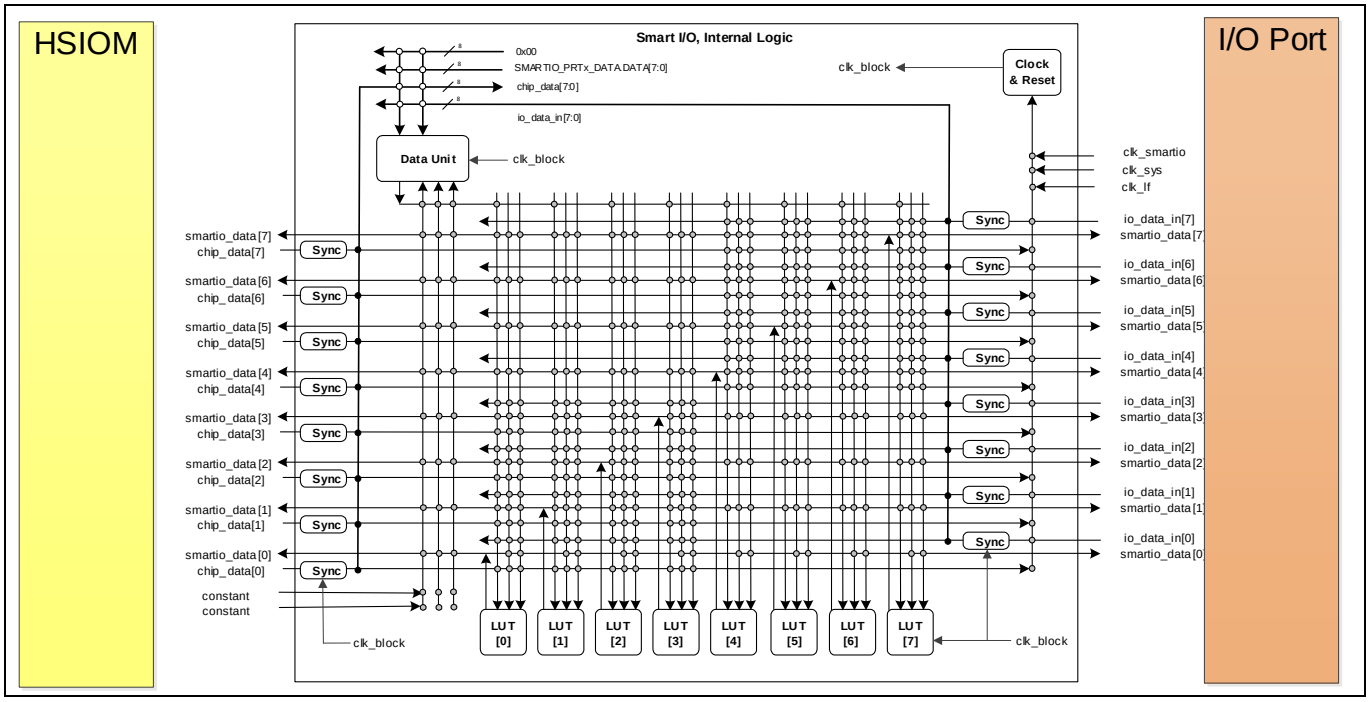


Figure 2 Block diagram of Smart I/O

The Smart I/O consists of the following components:

- Clock and reset
- Synchronizer (Sync)
- 3-input lookup tables (LUT [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 ports 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 the architecture reference manual for details on clk_smartio and clk_lf.

Eight lookup tables (LUT [x]) are implemented for each Smart I/O unit. LUT [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.

Structure of Smart I/O

The data unit can provide enhanced functionality for the output signal. Clock and reset blocks are used to synchronize the signals of HSIOM, I/O port, and each block in Smart I/O. The 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.

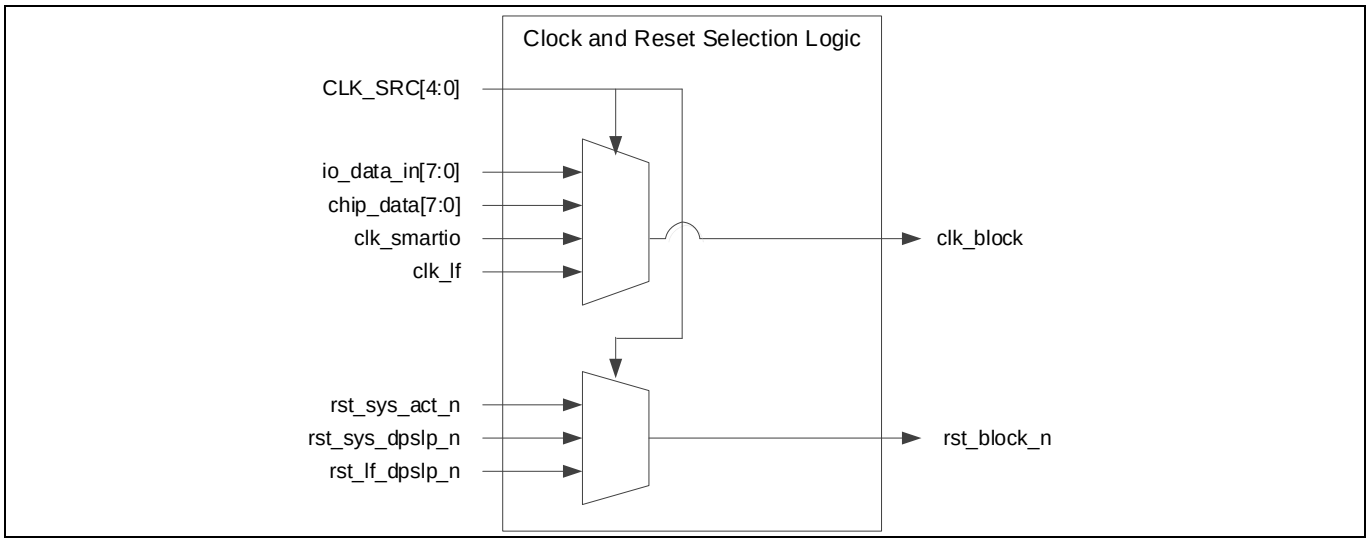


Figure 3 Functional image of clock and reset setting

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.

Structure of Smart I/O

Table 2 shows the configuration of the `SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]` register. See the registers reference manual 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 the synchronizer setting register and configuration. See the registers reference manual 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 (LUT [x])

Each LUT [x] has three inputs and one output. All inputs (Tr0_in, Tr1_in, Tr2_in) of each LUT [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 LUT [x] takes three input signals and generates an output based on the configuration set in the register. Figure 4 shows the basic block diagram of each LUT [x]. The output pattern can be set by the register.

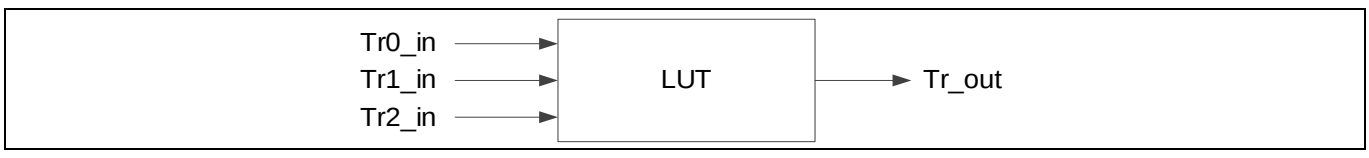


Figure 4 LUT [x] block diagram

2.3.1 LUT [x] output configuration

The output signal (Tr_out) of LUT [x] can be programmed using `SMARTIO_PRTx_LUT_CTLy.LUT[7:0]`[1] is based on three input sources (Tr2_in, Tr1_in, Tr0_in). Table 4 shows an example of setting each LUT [x].

Table 4 LUT [x] output setting

Tr 2_in	Tr 1_in	Tr 0_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 the 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 LUT [x] output setting. See the registers reference manual for details.

Table 5 Register for setting the output from LUT [x]

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

2.3.2 LUT [x] input selection

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

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

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

[1] Subscripts y in register names used in this sentence are LUT number.

Structure of Smart I/O

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 reference manual for details.

Table 6 Register for LUT [x] input source setting

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

Structure of Smart I/O

Register	Bit field	Setting
		10: chip_data [2] (for LUT [0], [1], [2], [3]); chip_data [6] (for LUT [4], [5], [6], [7]) 11: chip_data [3] (for LUT [0], [1], [2], [3]); chip_data [7] (for LUT [4], [5], [6], [7]) 12: io_data_in [0] (for LUT [0], [1], [2], [3]); io_data_in [4] (for LUT [4], [5], [6], [7]) 13: io_data_in [1] (for LUT [0], [1], [2], [3]); io_data_in [5] (for LUT [4], [5], [6], [7]) 14: io_data_in [2] (for LUT [0], [1], [2], [3]); io_data_in [6] (for LUT [4], [5], [6], [7]) 15: io_data_in [3] (for LUT [0], [1], [2], [3]); io_data_in [7] (for LUT [4], [5], [6], [7])

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

The LUT [x] and data unit do not include any combinatorial loops. However, when one LUT [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 (LUT [x] and data unit) are registered before branching out to other components. Table 7 shows the 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 reference manual 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 LUT [x] operation

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

Combinatorial

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

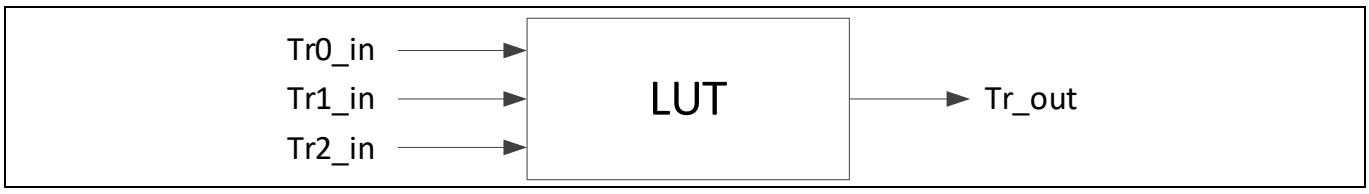


Figure 5 Combinatorial

Structure of Smart I/O

Gated input 2

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

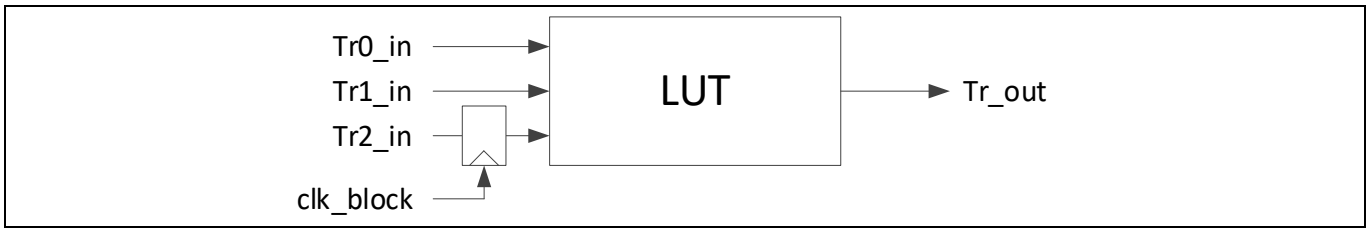


Figure 6 Gated input 2

Gated output

Inputs are directly connected to LUT [x], and the output is registered (Output synchronization).

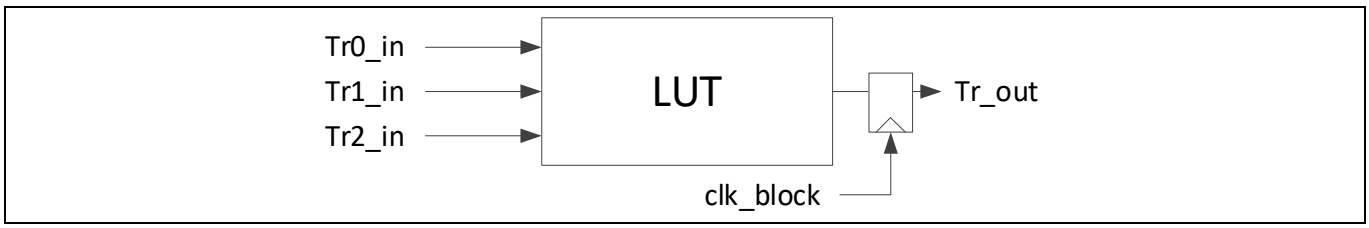


Figure 7 Gated output

Set/reset flip-flop

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

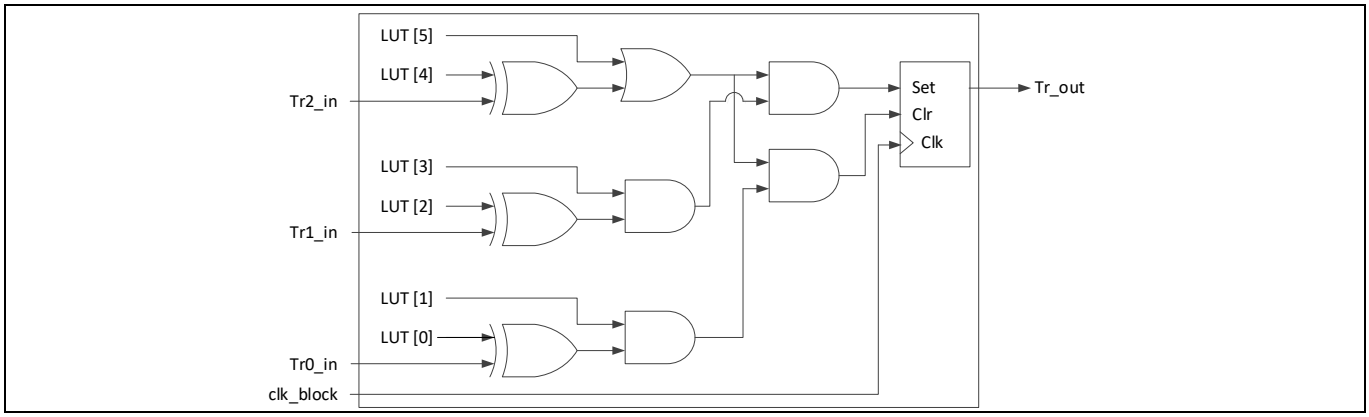


Figure 8 S/R flip-flop enable

These four operations can be set with the register shown in Table 8. See the registers reference manual for details.

Table 8 Register for LUT [x] mode setting

Register	Bit field	Setting
SMARTIO_PRTx_LUT_CTLy	LUT_OPC [9:8]	0: Combinatorial 1: Gated Input 2

Register	Bit field	Setting
		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 can perform 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 the data unit.

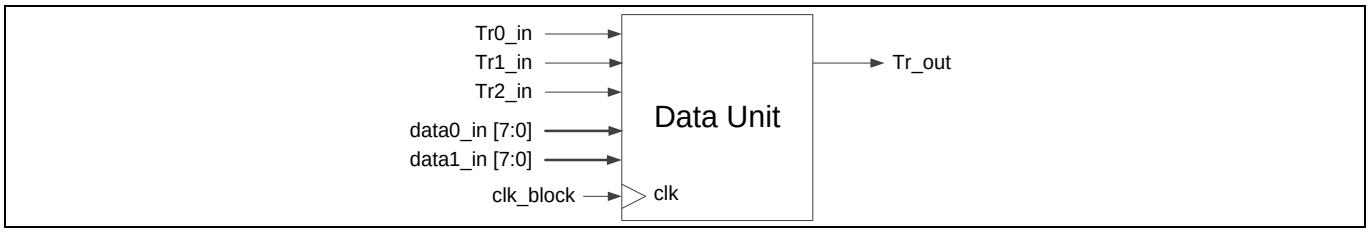


Figure 9 Data unit block diagram

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
- LUT [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 reference manual 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: LUT [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]

Structure of Smart I/O

- 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 the DU operation code setting. See the registers reference manual 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

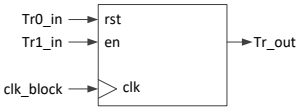
Table 12 shows each DU operation.

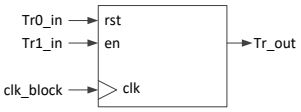
The ‘Operation’ column in Table 12 shows the operation outline and the pseudo-code. In the pseudo-code, “Combinational:” indicates that the operations are independent of previous output states. "Registered:" indicates that data operates on inputs and previous output states (registered using flip-flops).

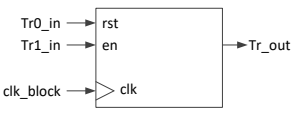
Structure of Smart I/O

Table 12 DU operation

Operation code	Operation

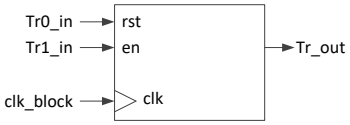
DU_OPC [11:8] = 1: INCR	INCR increments data by 1 from an initial value (DATA 0) until it reaches a final value (DATA 1).
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	`Tr_out = data_eql_data1`
	
	 
	
	Registered:
	`data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

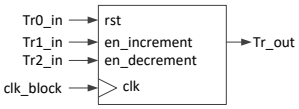
DU_OPC [11:8] = 2: DECR	DECR decrements data from an initial value (DATA 0) until it reaches ‘0’.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	
	Combinational:
	`Tr_out = data_eql_0`
	
	 
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 &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

Structure of Smart I/O

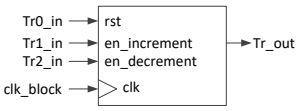
Operation code	Operation
	`data <= data_eql_data1? DATA0 & mask: (data + 1) & mask;`

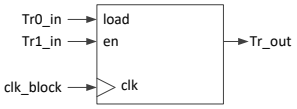
`DU_OPC [11:8] = 4: DECR_WRAP`	DECR_WRAP works similar to DECR. Instead of stopping at '0', it wraps around to DATA0.
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	
	 
	
	Combinational:
	`Tr_out = data_eql_0`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`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

Structure of Smart I/O

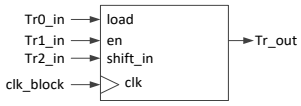
Operation code	Operation

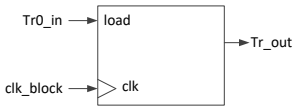
DU_OPC [11:8] = 6: INCR_DECR_WRAP	INCR_DECR_WRAP has the same functionality as INCR_DECR with wrap around to DATA 0 on reaching the limits (DATA 1 or '0').
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
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

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]`
	`}`

Structure of Smart I/O

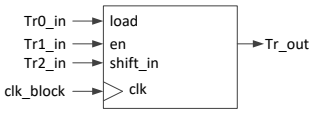
Operation code	Operation

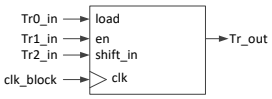
DU_OPC [11:8] = 8: SHIR	SHIR performs the shift register operation. Initial data (DATA 0) is shifted out and data on tr2_in is shifted in.
	
	du_size = Size - 1
	mask = (1 << (DU_SIZE+1)) – 1
	
	Combinational:
	Tr_out = data [0]
	
	Registered:
	data <= data;
	if (Tr0_in)
	data <= DATA0 & mask;
	else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

Structure of Smart I/O

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]) &`
	`data [du_size] <= Tr2_in`
	`}`
	 

3 Smart I/O configuration

Table 13 lists all functions of the Smart I/O configuration.

Table 13 List of Smart I/O configuration Functions

Functions	Description	Remarks
`Cy_SmartIO_Init()`	Initializes the Smart I/O	–
`Cy_SmartIO_Deinit()`	Resets the Smart I/O to default values	–
`Cy_SmartIO_Enable()`	Enables the Smart I/O	–
`Cy_SmartIO_Disable()`	Disables the Smart I/O	–
`Cy_SmartIO_HoldOverride()`	Configures the hold override mode	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetChBypass ()`	Gets the bypass/route state of all 8 channels in the Smart I/O	–
`Cy_SmartIO_SetChBypass ()`	Sets the bypass/route state of all 8 channels in the Smart I/O	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetClock ()`	Gets the clock source of the Smart I/O	–
`Cy_SmartIO_SetClock ()`	Sets the clock source of the Smart I/O	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetIoSync ()`	Gets the synchronization mode of the 8 I/O terminals	–
`Cy_SmartIO_SetIoSync ()`	Sets the synchronization mode of the 8 I/O terminals	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetChipSync ()`	Gets the synchronization mode of the 8 chip-side terminals	–
`Cy_SmartIO_SetChipSync ()`	Sets the synchronization mode of the 8 chip-side terminals	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetLutTr()`	Gets the specified LUT input trigger source	–
`Cy_SmartIO_SetLutTr()`	Sets the specified LUT input trigger source	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_SetLutTrAll()`	Sets all LUT input triggers to the same source	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetLutOpcode()`	Gets the opcode of the specified LUT	–

Smart I/O configuration

Functions	Description	Remarks
`Cy_SmartIO_SetLutOpcode()`	Sets the opcode of the specified LUT	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetLutMap()`	Gets the 3:1 truth table of the specified LUT	–
`Cy_SmartIO_SetLutMap()`	Sets the 3 bit input to 1 bit output mapping of the specified LUT	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetDuTr ()`	Gets the data unit input trigger source	–
`Cy_SmartIO_SetDuTr ()`	Sets the data unit input trigger source	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_SetDuTrAll()`	Sets all the data unit input trigger sources	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetDuData()`	Gets the data unit's input "data" source	–
`Cy_SmartIO_SetDuData()`	Sets the data unit's input "data" source	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetDuOpc()`	Gets the data unit's opcode	–
`Cy_SmartIO_GetDuSize()`	Gets the data unit's opcode operand bit-width	–
`Cy_SmartIO_SetDuOperation()`	Sets the data unit's opcode and operand bit-width	The Smart I/O block must be disabled before calling this function
`Cy_SmartIO_GetDataReg()`	Gets the data unit's DATA register value	–
`Cy_SmartIO_SetDataReg`	Sets the data unit's DATA register value	The Smart I/O block must be disabled before calling this function

Smart I/O configuration

Figure 10 shows the flow of configuring Smart I/O.

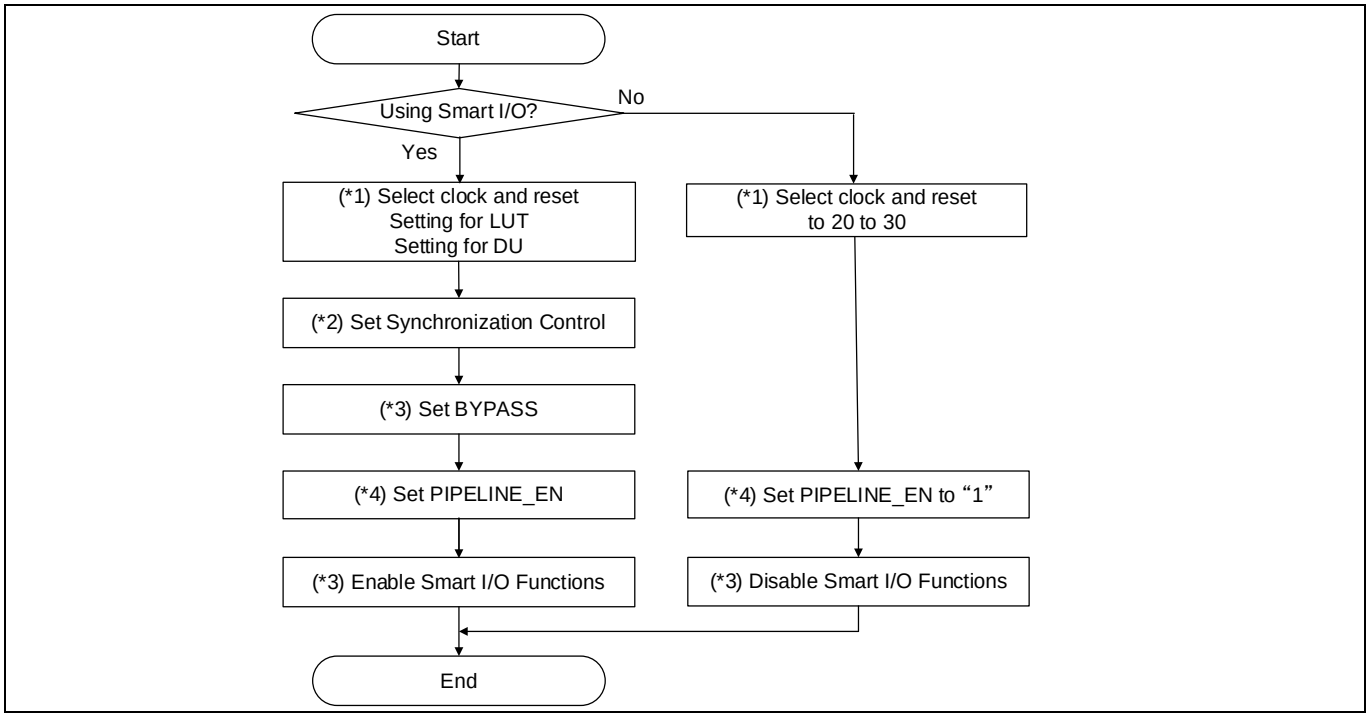


Figure 10 Flow of Smart I/O configuration

When configuring Smart I/O, first initialize each component such as clock and reset, synchronizer, LUT [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 the 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 2 Structure of Smart I/O for ports, source and clock setting, see 2.3 3-inputs lookup tables (LUT [x]) for LUT [x] setting, and see 2.4 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

This section describes how to use Smart I/O with the Peripheral Driver Library (PDL).

The Smart I/O example has a configuration part that helps in setting the parameter values for the desired operation. The PDL driver configures each register based on the parameter values in the configuration part. The configuration parts can be set according to your system.

Smart I/O is used for applications that involve simple logic operations for the input/output signal, or the internal routing between the internal HSIOM port and the I/O port. No CPU is required for these operations.

4.1 Pulse counting

In some cases, product size limitations eliminate the use of physical buttons, requiring a reliable system restoration mechanism. Smart I/O provides an ideal solution that operates only on hardware-based logical operations without needing CPU intervention. Additionally, Smart I/O continues to function effectively in deep sleep mode.

Figure 11 shows the block diagram for this example:

- A pulse is generated when plugging/unplugging the charging cable
- The P17.3 acts as Smart I/O input that connects to this pulse
- When the number of pulse reaches the set value, the Smart I/O will output a low level to reset the system

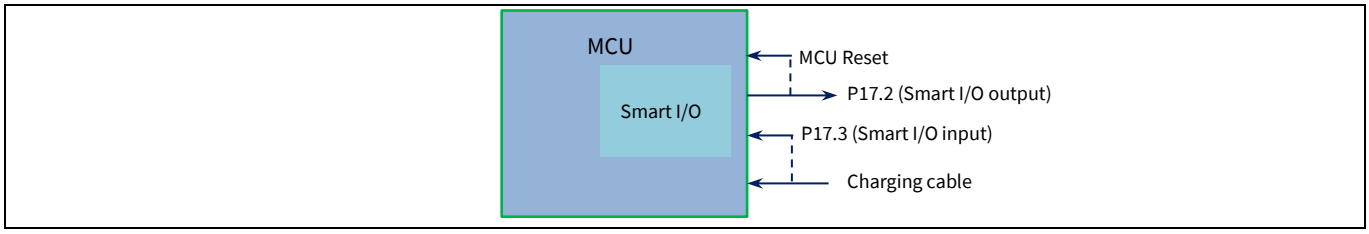


Figure 11 Pulse counting block diagram

Figure 12 shows the Smart I/O input and output waveform diagram.

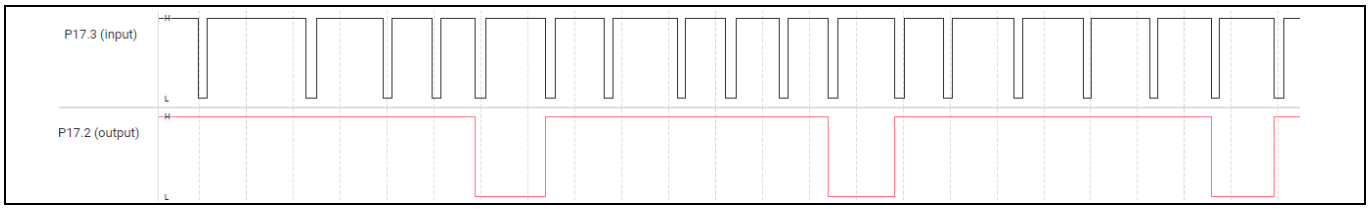


Figure 12 Input and output waveform diagram

4.1.1 Configuration in Device Configurator

Figure 13 shows the overall configuration of Smart I/O in the ModusToolbox™ Device Configurator tool.

Example

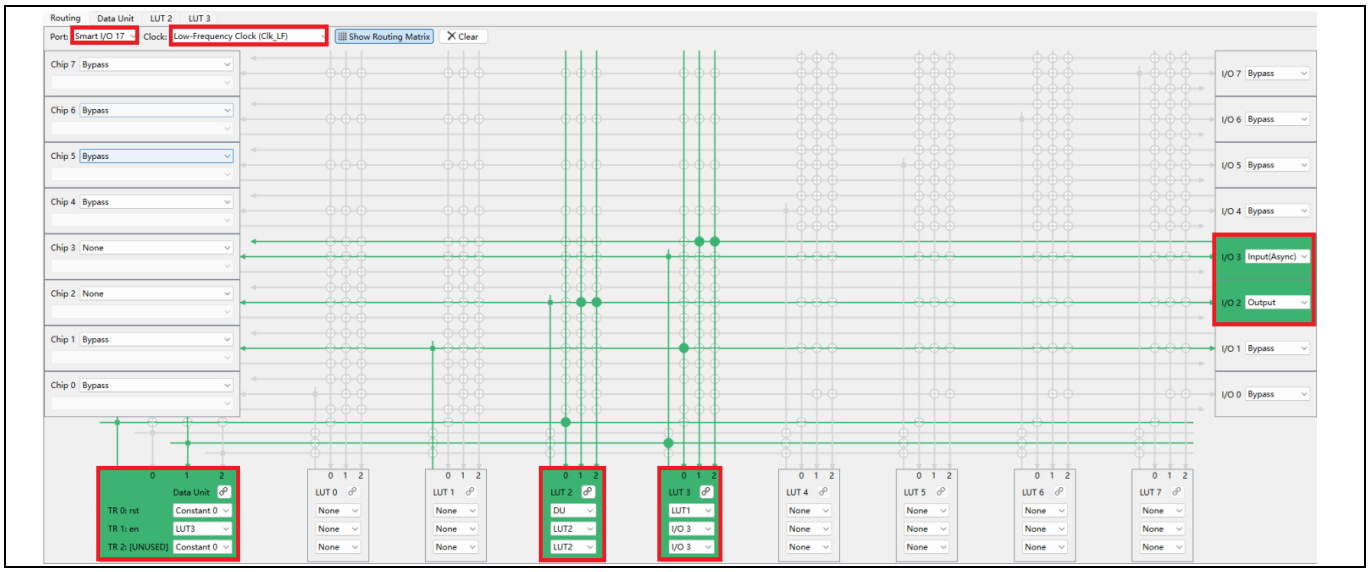


Figure 13 Overview of Smart I/O routing

Figure 14 shows the LUT3 configuration. LUT3 uses TR2 gated mode where the value of TR2 is latched during the rising edge of the clock. In other words, the TR_Out of LU3 becomes '1' only when there is a falling edge on IO3 (P17.3).

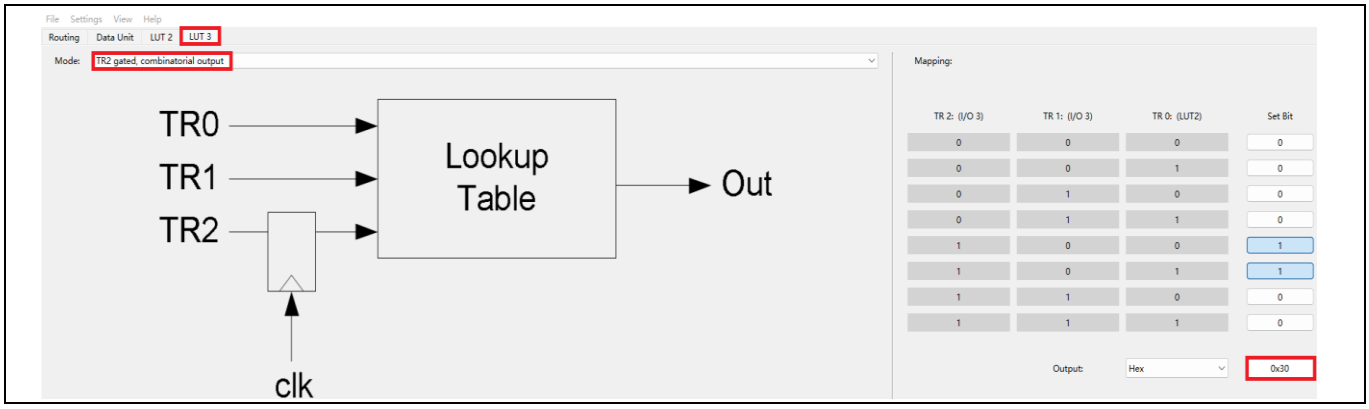


Figure 14 LUT3 configuration

Figure 15 shows the LUT2 configuration. LUT2 uses the combinatorial output mode.

The truth table yields the logical formula:

$$TR_Out = A'B'C' + A'BC' + AB'C' + ABC', \text{ here } A=B=LUT2, C=DU$$

Equation 1

Example

Simplified, it becomes:

TR_Out = A'C' + AC'=C'

Equation 2

This configuration implements a "NOT" function, with the output solely relying on the DU value.

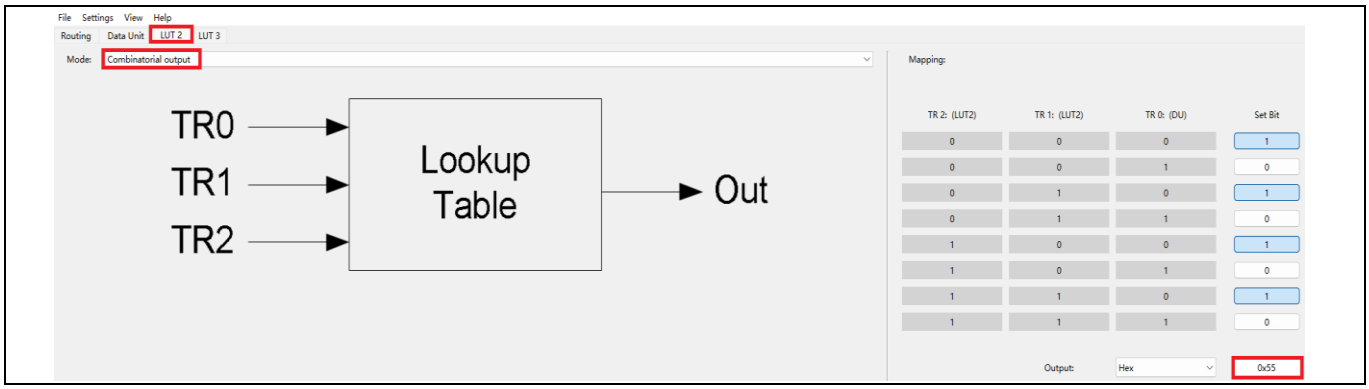


Figure 15 LUT2 configuration

Figure 16 shows the DU configuration. The pulse from LUT3 is counted, and once it is equal to 5 times, the DU output '1'.

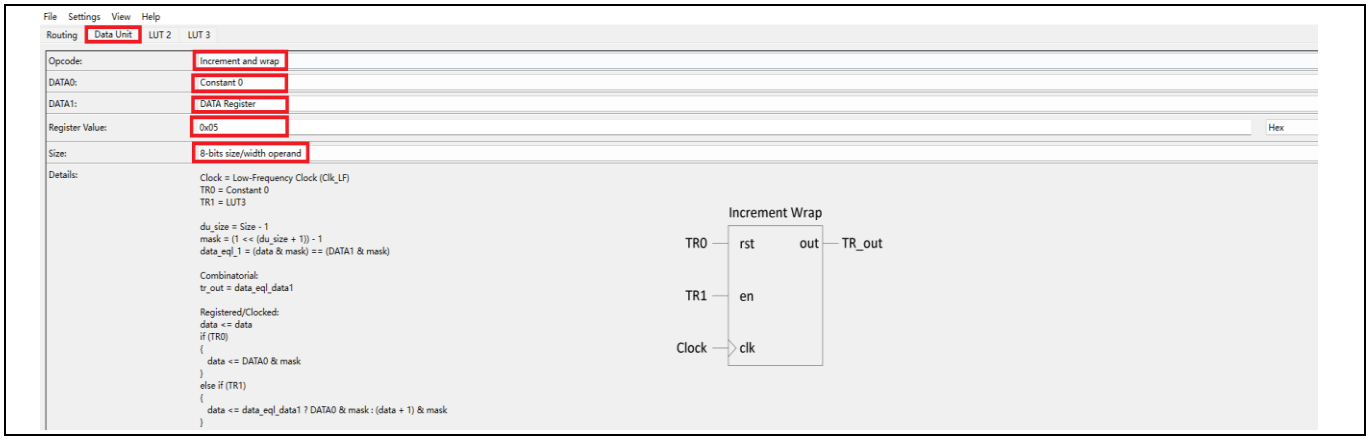


Figure 16 DU configuration

Figure 17 shows the Pin configuration, setting Drive mode is required.

Example

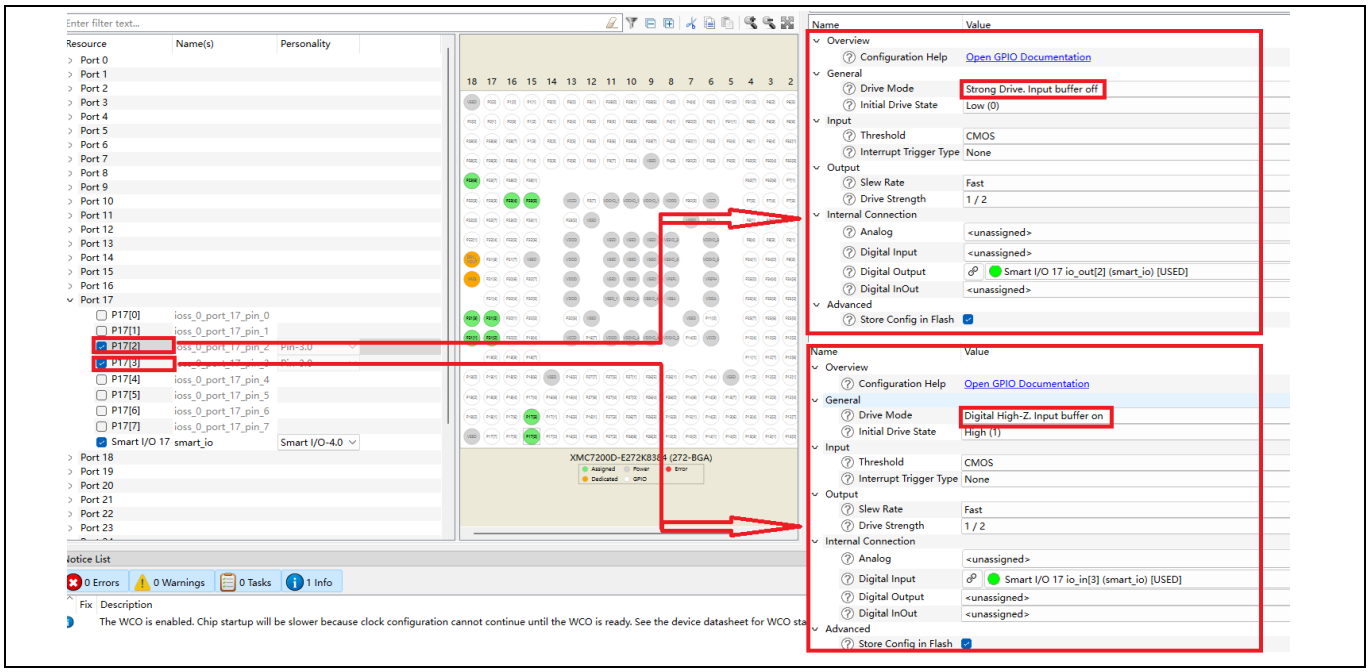


Figure 17 Pins drive mode configuration

4.1.2 Reference code

Code Listing 1 shows the source code of this example.

Code Listing 1 Source code of Pulse Counting with Smart I/O

```
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
      CY_ASSERT(0);
    }

    /* Enable interrupts */
    __enable_irq();

    /* Initialize the SmartIO block */
    Cy_SmartIO_Init(smart_io_HW, &smart_io_config);

    /* Enable the Smart I/O */
    Cy_SmartIO_Enable(smart_io_HW);

    for(;;)
    {
      ;
    }
}
```

4.2 Ramping LED using Smart I/O

This section explains how to utilize a PWM resource and Smart I/O for implementing a ramping LED that gradually cycles through increasing and decreasing brightness levels.

The Smart I/O, clocked at 99 Hz, uses its internal LUT2 and LUT3 components to divide the clock by 4 and generate 99/4 Hz square waves. Simultaneously, the PWM produces 25 Hz square waves. The system routes these two square waves to an exclusive-OR (XOR) gate within the Smart I/O component, which results in a signal with a duty cycle that changes gradually. The rate at which the duty cycle changes is directly proportional to the differences in frequency between the two square waves.

Figure 18 shows the logical block diagram. LUT1, LUT2, and LUT3 are used for this example.

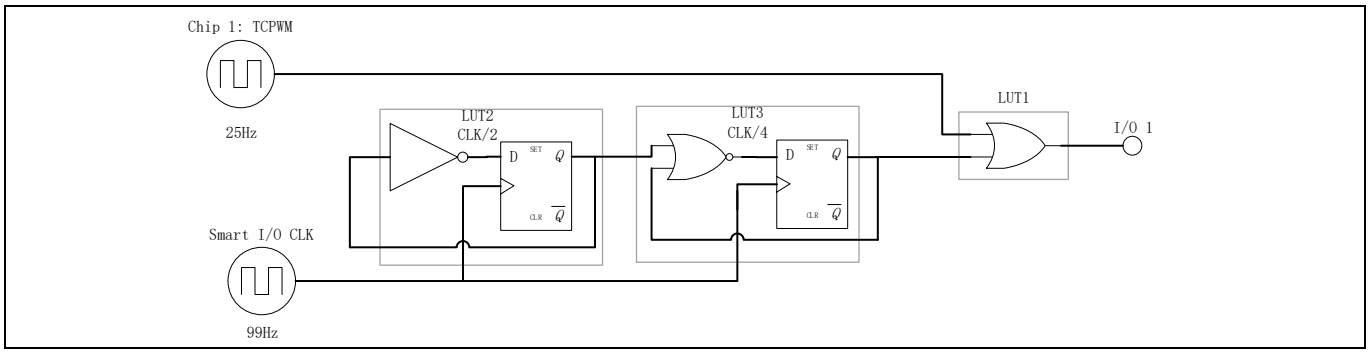


Figure 18 Logical block diagram

Figure 19 shows the timing diagram of the square waves.

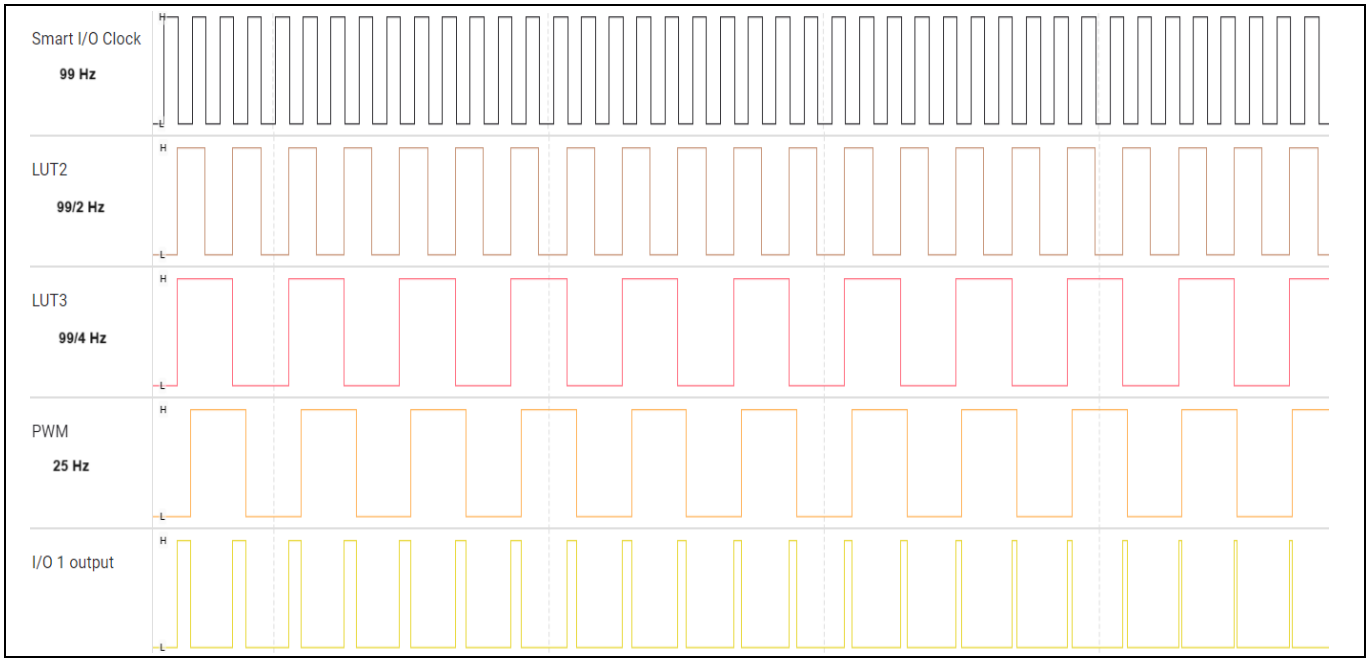


Figure 19 Timing diagram

4.2.1 Configuration in Device Configurator

Figure 20 shows the overall configuration of Smart I/O within the ModusToolbox™ device configurator tool. TCPWM generates one Smart I/O input signal from HISOM. The final Smart I/O signal routes to the I/O interface.

Example

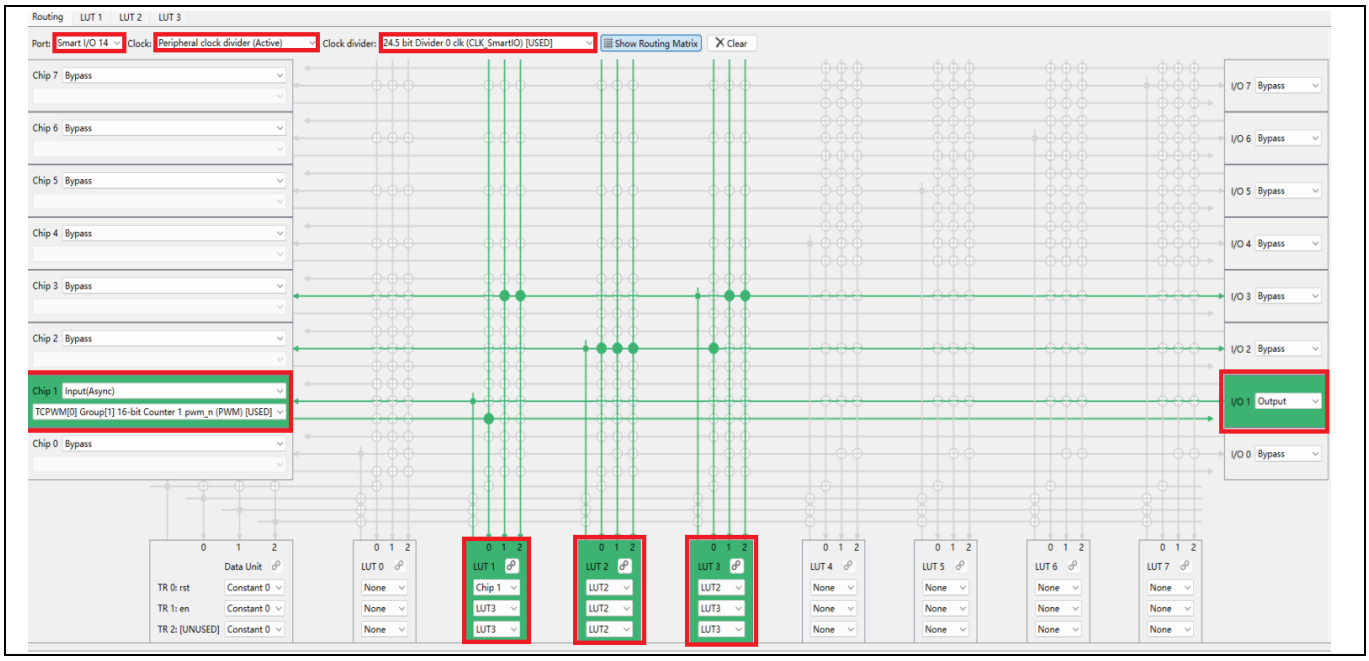


Figure 20 Overview of Smart I/O routing

Figure 21 shows the LUT2 configuration. LUT2 uses the sequential (gated) output mode, the three input signals are the same.

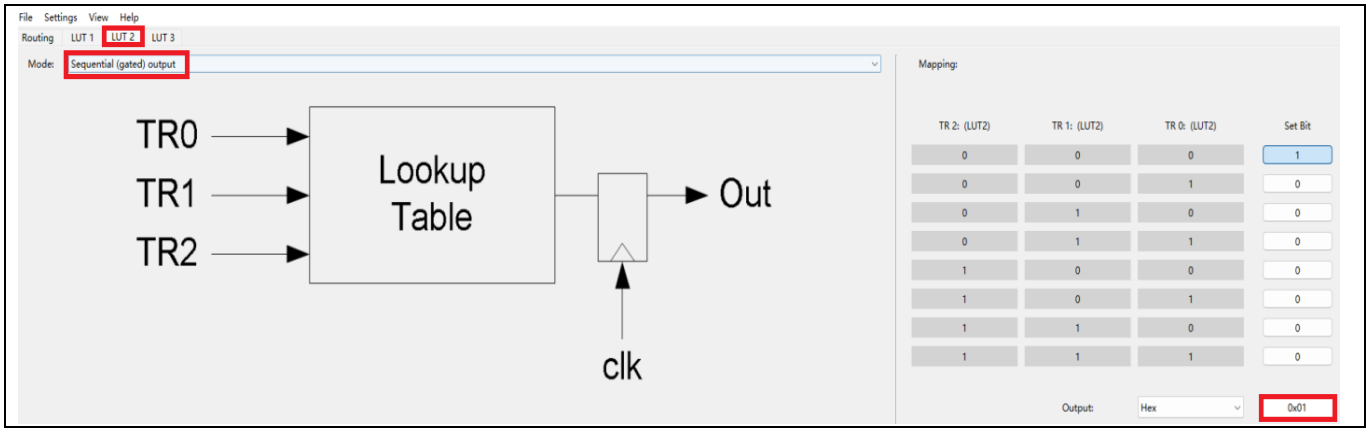


Figure 21 LUT2 Configuration

When the initial value of TR_Out is '0', TR0/TR1/TR2 become '1'. We assign the truth table with 0x01, corresponding to the NOT-Gate function. As the clock's rising edge arrives, TR_Out switches to '1'. With the next rising clock edge, TR_Out reverts to '0'. As Shown in Figure 22, the LUT2 implements a divide-by-2 circuit from the 90 Hz of the Smart I/O clock.

Example

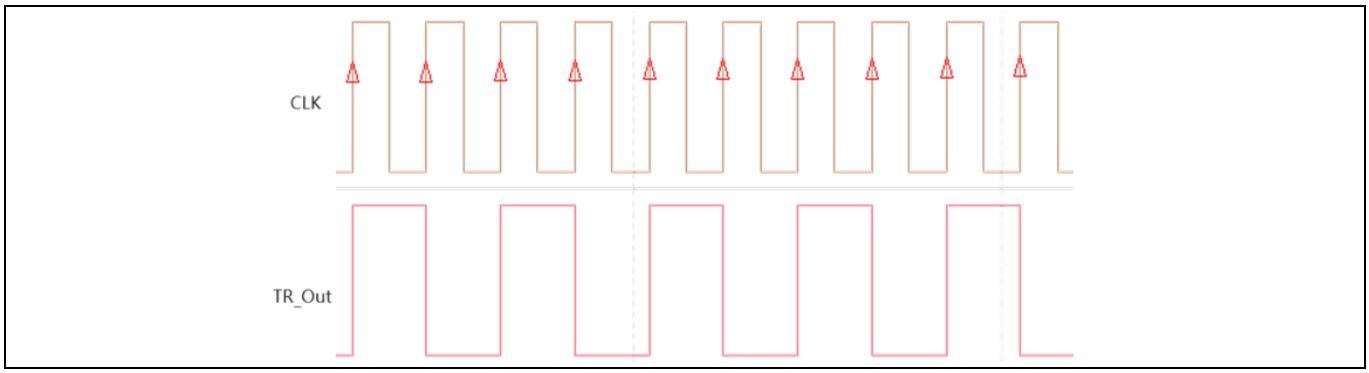


Figure 22 Divide-by-two diagram

Figure 23 shows the LUT3 configuration. LUT2 also uses the sequential (gated) output mode.

The truth table yields the logical formula:

$$TR_Out = A'B'C + ABC, \text{ here } A=B=LUT3, C=LUT2$$

Equation 3

Simplified, it becomes:

$$TR_Out = A'C' + AC$$

Equation 4

It implements the XNOR function. Similar with LUT2, LUT3 divides the square wave frequency of LUT2 output by 2, that means the TR_Out of LUT3 is 99/4 Hz.

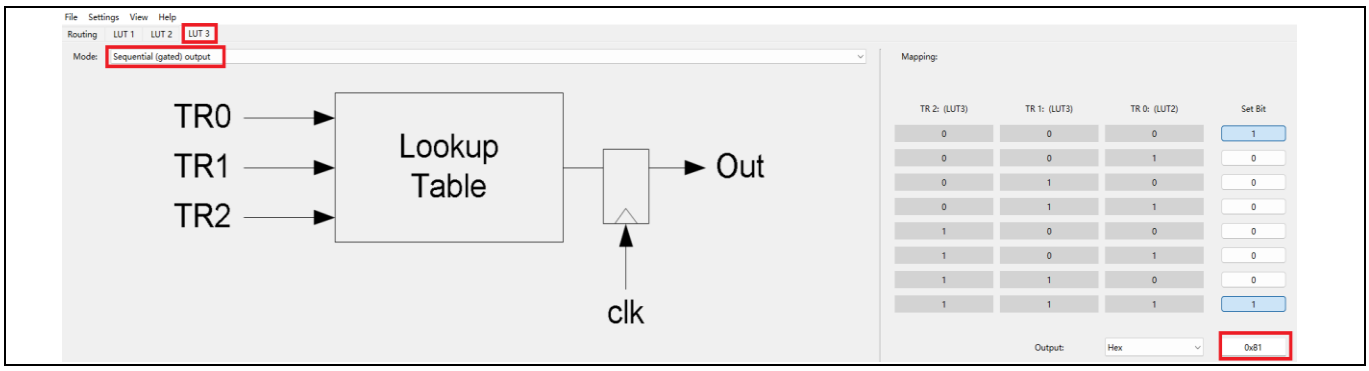


Figure 23 LUT3 configuration

Figure 24 shows the LUT1 configuration, LUT1 uses the combinatorial output mode and drives the GPIO output.

The truth table yields the logical formula:

$$TR_Out = A'B'C + ABC', \text{ here } A=B=LUT3, C=Chip\ 1\ (PWM)$$

Equation 5

Simplified, it becomes:

Example

$$TR_Out = A'C + AC'$$

Equation 6

It implements the XOR function.

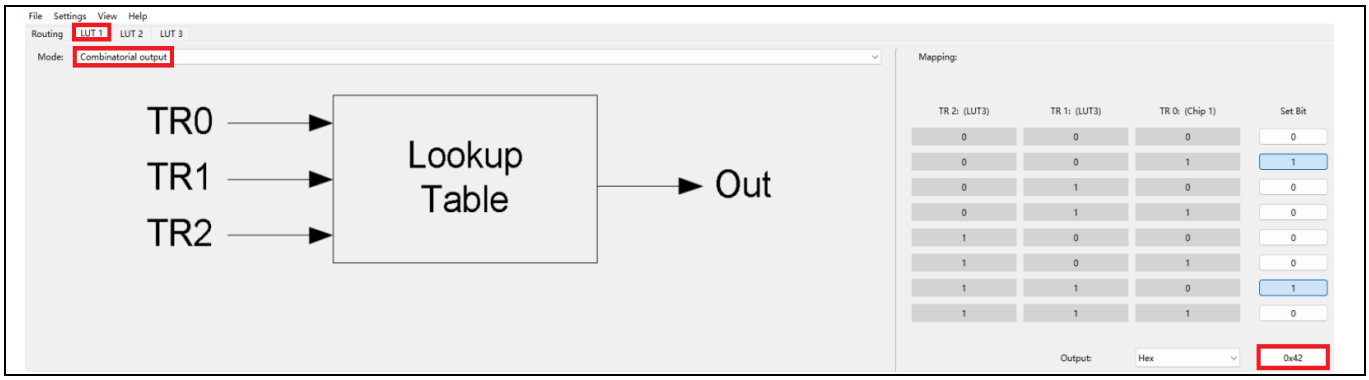


Figure 24 LUT1 configuration

Figure 25 shows the pin configuration, the P14_1 is the output of Smart I/O. In this case, setting the pin drive mode is required.

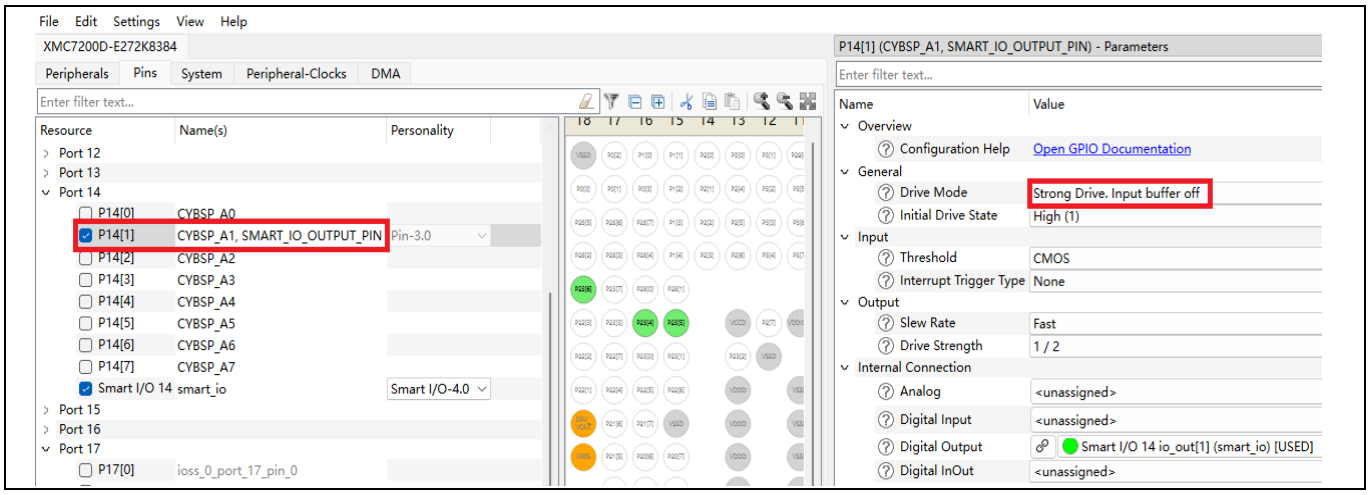


Figure 25 Pins drive mode configuration

Figure 26 shows the TCPWM configuration, the TCPWM clock is configurated as 10 kHz, the Period value set to 399 (400-1) to get 25 Hz PWM output.

Example

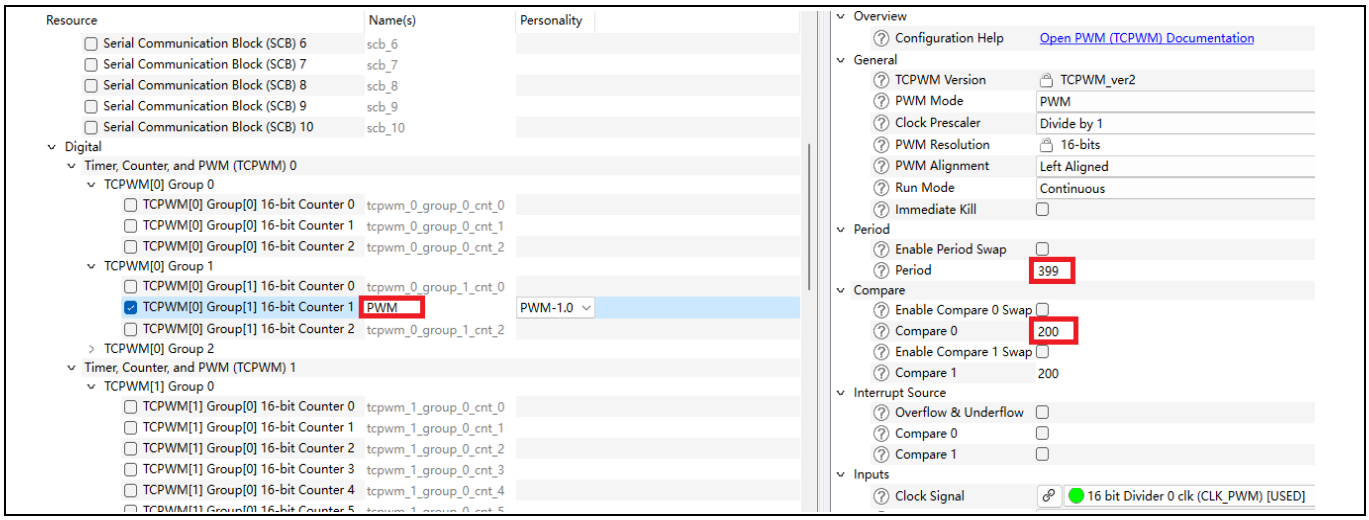


Figure 26 TCPWM configuration

Figure 27 shows the clock configuration for TCPWM and Smart I/O.

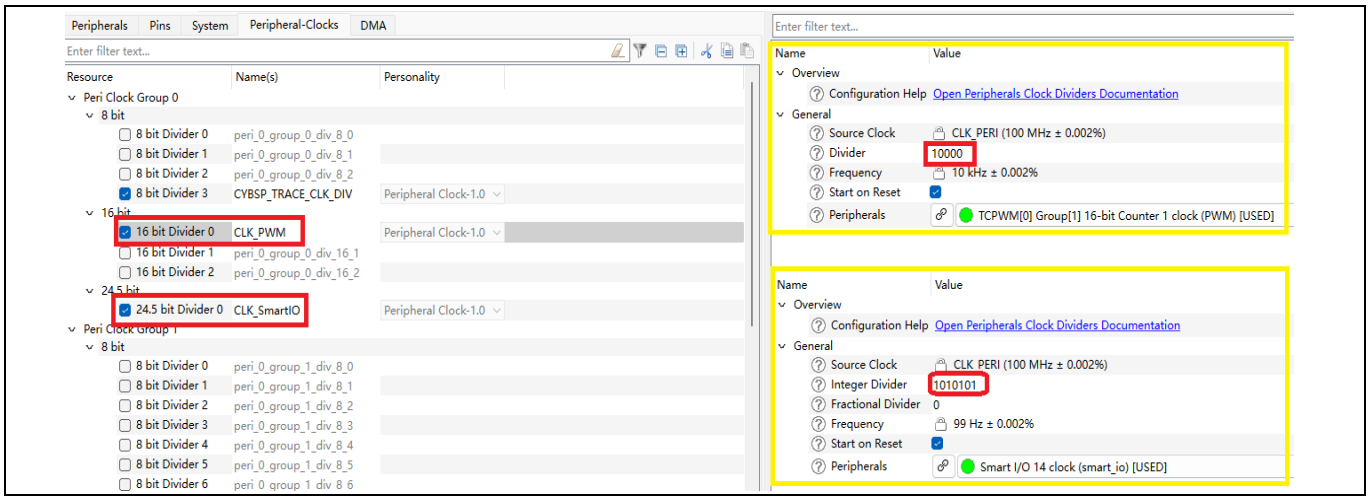


Figure 27 TCPWM and Smart I/O clock configuration

4.2.2 Reference code

Figure 28 shows the operation flow of this example.

Example

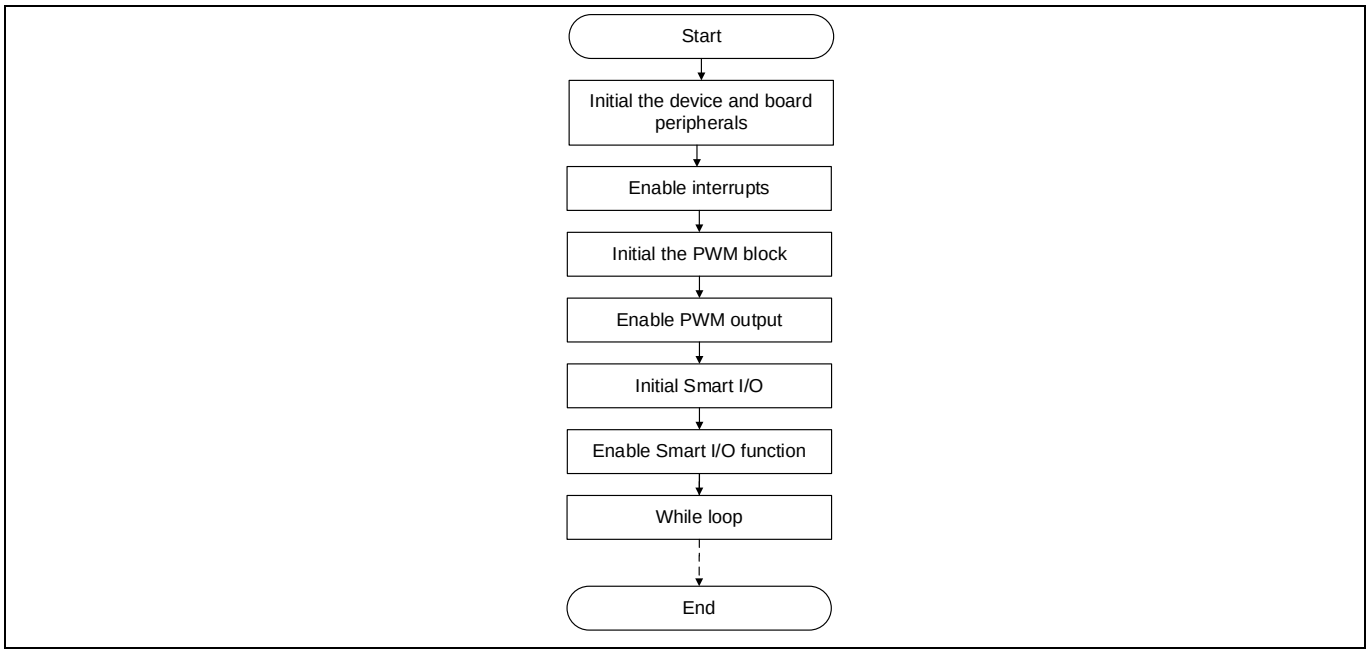


Figure 28 Setting procedure for Smart I/O

Code Listing 2 demonstrates an example program to use Smart I/O and PWM to yield a signal with a gradually changing duty cycle.

Code Listing 2 Source code of Ramping LED using Smart I/O

```
int main(void)
{
    cy_rslt_t result;
    cy_en_smartio_status_t status;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
        {
            CY_ASSERT(0);
        }

    /* Enable interrupts */
    __enable_irq();

    /* Initialize the counter in the TCPWM block for the PWM operation.
     * The PWM is configured to generate a 25 Hz square wave signal
     * with 50% duty cycle */
    Cy_TCPWM_PWM_Init(PWM_HW, PWM_NUM, &PWM_config);
    /* Enable the TCPWM for PWM mode of operation */

    Cy_TCPWM_PWM_Enable(PWM_HW, PWM_NUM);
    /* Start the TCPWM block */

    Cy_TCPWM_TriggerStart_Single(PWM_HW, PWM_NUM);

    /* Initialize the SmartIO block. The Smart I/O implements a sequential
     * circuit to generate a square wave signal with time varying duty cycle.
     * This signal drives an LED creating visual perception of breathing LED */

    status = Cy_SmartIO_Init(smart_io_HW, &smart_io_config);
```

Example

Code Listing 2 Source code of Ramping LED using Smart I/O

```
    if(CY_SMARTIO_SUCCESS != status)
    {
        CY_ASSERT(0);
    }
    /* Enable the Smart I/O */
    Cy_SmartIO_Enable(smart_io_HW);

    /* LED breathing effect is generated using PWM and Smart I/O
     * hardware peripherals - no CPU is used */
    for(;;)
    {
        ;
    }
}
```

5 Glossary

Table 14 Glossary

Terms	Description
chip_data	Input signals from HSIOM
Clk_sys/CLK_HF	This is derived from the system clock using a peripheral clock divider. See the Clocking System chapter of the architecture reference manual 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 reference manual 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 reference manual for details.
GPIO	General-purpose input/output
HSIOM	High Speed I/O Matrix. See the High-Speed I/O Matrix section in the I/O System chapter of the architecture reference manual for details.
io_data_in	Input signals from I/O port
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 reference manual for details.
LUT [x]	3-input Lookup Tables. LUT [x] block takes three input signals and generates an output based on the configuration set in register. See the Smart I/O - LUT section in the I/O System chapter of the architecture reference manual for details.
smartio_data	Output signals from Smart I/O

References

The following are the XMC7000 family series datasheets and technical reference manuals. Contact Technical support to obtain more documents.

[1] Device datasheet

- XMC7200 Datasheet 32-Bit Arm® Cortex®-M7 microcontroller XMC7000 family
- XMC7100 Datasheet 32-Bit Arm® Cortex®-M7 microcontroller XMC7000 family

[2] Technical reference manual

- XMC7000 MCU family architecture technical reference manual
- XMC7200 registers reference manual
- XMC7100 registers reference manual

[3] Application notes

- AN234334 - Getting started with XMC7000 MCU on ModusToolbox™ software

Revision history

Document revision	Date	Description of changes
**	2021-12-02	Initial release
*A	2024-06-03	Updated Table 12 Added Table 13 Modified the Example section Updated the References section

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-06-03
Published by

Infineon Technologies AG
81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
002-34023 Rev. *A

Important notice
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.