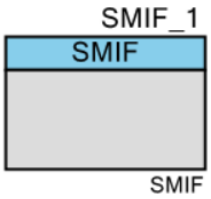
Serial Memory Interface (SMIF_PDL)

1.10



Features

- Standard SPI Master interface
- Supports Single/Dual/Quad/Octal SPI Memories
- Supports Dual-Quad SPI mode
- Design-time configurable support for multiple (up to 4) external serial memory devices
- eXecute-In-Place (XIP) operation mode for both read and write accesses with 4KB XIP read cache and on-the-fly encryption and decryption
- Supports external serial memory initialization via Serial Flash Discoverable Parameters (SFDP) standard

General Description

The SMIF_PDL Component is a multifunction hardware block that implements the SPI communication to external serial memory devices, including the NOR Flash, SRAM, and non-volatile SRAM.

The SMIF_PDL Component is a graphical configuration entity built on top of the cy_smif driver available in the Peripheral Driver Library (PDL). It allows schematic-based connections and hardware configuration as defined by the Component Configure dialog.

When to Use a SMIF_PDL Component

Use a SMIF_PDL Component to increase the total memory capacity of a system. It includes using external flash memory as a code space, data storage or data logging, using external SRAM memory as a data storage, etc.

Definitions

- **SMIF**: Serial Memory Interface: This IP block implements a SPI-based communication interface for interfacing external memory devices to a PSoC. SMIF supports Octal-SPI, Dual Quad-SPI, Quad-SPI, DSPI and SPI.
- **DSPI**: DSPI supports 2 bits/cycle throughput. This is a mode of communication using the SMIF block.

- **Quad-SPI**: Quad SPI supports 4 bits/cycle throughput. This is a mode of communication using the SMIF block.
- **Dual Quad-SPI**: Two Quad-SPI slaves that implement two nibbles and thus support a byte transfer/cycle. This is a mode of communication using the SMIF block.
- **Octal SPI**: Octal SPI supports 8 bits/cycle throughput. This is a mode of communication using the SMIF block.
- **XIP**: eXecute In Place: XIP is a mode of operation where read or write commands to the memory device are directed through the SMIF without any use of API function calls. In this mode, the SMIF block maps the AHB bus-accesses to external memory device addresses to make it behave similar to internal memory. This allows the CPU to execute code directly from external memory. This mode is not limited to code and is suitable also for data read and write accesses.
- **MMIO**: Memory Mapped I/O mode. The MMIO mode is used for special commands like program/erase of FLASH, device configuration etc. The MMIO mode can be used for read/write operations too, but that is much less common.
- **Memory mode**: This is same as the XIP mode of operation of the SMIF. The memory mode nomenclature is followed, since it is more user-friendly than a specific name like XIP.
- **Normal mode**: This is same as the MMIO mode of operation.
- **Memory device**: A physical memory device. A user connects one or more memory devices to the SMIF.
- **Slave Slot**: Slave Slot refers to the individual slave select lines in the SMIF interface. The SMIF interface will have 4 Slave Slots. Each Slave Slot can have a memory device or non-memory device or can be left unused.
- **Memory Configuration Data structure**: This is a data structure array which holds the parameters that correspond to the individual memory device slots of the SMIF. The array is of length 4 corresponding to the number of slave slots supported in the SMIF. Each structure holds all the information required to configure the memory slave slot associated with it. The content of the structure is populated based on the memory device being interfaced to and the memory configuration of it.

Quick Start

1. Drag a SMIF_PDL Component from the Component Catalog Memory Interface folder onto your schematic (the placed instance takes the name SMIF_1).
2. Double-click to open the Configure dialog.
3. Select the Datalines to be used.

4. Select the Slave Select lines of the external serial memory devices.
5. Make sure the SMIF clock (HFCLK2) is set up in the PSoC Creator Clock Editor.
6. Build the project in order to verify the correctness of your design, add the required PDL modules to the Workspace Explorer, and generate the configuration data for the SMIF_1 instance.
7. In *main.c*, initialize the peripheral and start the application.
   a. Declare the SMIF memory configuration structure (stc_smif_mem_config_t** memConfigs) or use SMIF Configuration Tool to generate the configuration structure. Right-click on the SMIF_PDL Component and select **SMIF Configuration Tool** from the context menu.
   b. Add the code.

```
cy_stc_smif_context_t smifContext;
void ExtMemInterrupt(void);
void RxCmpltCallback(uint32_t event);
uint8_t extMemAddress[3u] = {0x00, 0x00, 0x00};
uint8_t transferData[3u] = {0x01, 0x0F, 0xAA};
uint8_t rxBuffer;

int main(void)
{
    /* SMIF interrupt setup */
    cy_stc_sysint_t smifIntConfig =
    {
        /* SMIF interrupt */
        .intrSrc = smif_interrupt_IRQn,
        /* Mapping the SMIF M4 core interrupt to
         * the 10th M0+ core interrupt
         */
        .intrCm0p = NvicMux10,
        /* SMIF interrupt priority */
        .intrPriority = 1u
    };
    Cy_SysInt_Init(&smifIntConfig, ExtMemInterrupt);

    /* SMIF configuration parameters */
    cy_stc_smif_config_t tst_psvpSmifConfig =
    {
    /* The mode of operation */
        .mode = CY_SMIF_NORMAL,
        /* The minimum duration of SPI deselection */
        .deselectDelay = SMIF_1_DESELECT_DELAY,
        /* The clock source for the receiver clock */
        .rxClockSel = SMIF_1_RX_CLOCK_SELECT,
        /* What happens when there is a Read
         * to an empty RX FIFO or Write to a full TX FIFO
         */
        .blockEvent = SMIF_1_AHB_BUS_ERROR,
    };
```

```
    /* SMIF initialization */
    Cy_SMIF_Init(SMIF0, &tst_psvpSmifConfig, &smifContext, 1000ul);
    Cy_SMIF_SetDataSelect(SMIF0, 1u, CY_SMIF_DATA_SEL0);
    Cy_SMIF_Enable(SMIF0, &smifContext);

    __enable_irq(); /* Enable global interrupts. */

    /* Enable the SMIF interrupt */
    NVIC_EnableIRQ((uint32_t)smif_interrupt_IRQn);

    /* Enable the Write bit in the status register.
     * Send the WREN 0x06 command to the external memory.
     */
    Cy_SMIF_TransmitCommand(SMIF0,
                            &smifContext,
                            0x06,
                            CY_SMIF_WIDTH_SINGLE,
                            0u,
                            0u,
                            CY_SMIF_WIDTH_SINGLE,
                            1u,
                            1u);

    /* Check if the SMIF IP is busy */
    while(Cy_SMIF_BusyCheck(SMIF0))
    {
        /* Wait until the SMIF IP operation is completed. */
    }

    /* Writes data to the external memory in the single mode.
     * Sends the PP 0x02 command to the external memory.
     */
    Cy_SMIF_TransmitCommand(SMIF0,
                            &smifContext,
                            0x02,
                            CY_SMIF_WIDTH_SINGLE,
                            extMemAddress,
                            3u,
                            CY_SMIF_WIDTH_SINGLE,
                            1u,
                            0u);

    Cy_SMIF_TransmitData(SMIF0,
                            &smifContext,
                            transferData,
                            0x03,
                            CY_SMIF_WIDTH_SINGLE,
                            &RxCmpltCallback);
    {
        /* Wait until the SMIF IP operation is completed */
    }
}
```

```
/* The ISR for the SMIF interrupt. All Read/Write transfers to/from
 * the external memory are processed inside the SMIF ISR.
 */
void ExtMemInterrupt(void)
{
    Cy_SMIF_Interrupt(SMIF0, &smifContext);
}

/* The callback called after the transfer completion. */
void RxCmpltCallback (uint32_t event)
{
}
```

8. Build and program the device.

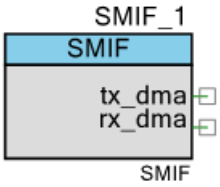
   The SMIF API is divided into low level functions and memory slot functions.

   - The low level API is used for SMIF block initialization and for implementing a generic SPI communication interface using the SMIF block.
   - The Memory slot API has functions that implement basic memory operations like program, read, erase etc. These functions are implemented using the memory parameters in the memory device config data structure. Also Memory slot initialization API will initialize all the memory slots based on the settings in the array.

9. The driver is not responsible for external memory persistence. You cannot edit the buffer during read/write operations. In case of a memory error, the SMIF IP block could get stuck and need to be reset. Check the SMIF IP busy status by using the appropriate API call and implementing a timeout. Reset the IP block by toggling CTL.ENABLED. Then reconfigure the SMIF block.
10. For write operations, check that the SMIF finished transferring data by calling SMIF_1_BusyCheck(). Also check that memory finished operation using SMIF_1_Mem_IsBusy() before proceeding.
11. For read operations before accessing to the read buffer, check that SMIF operation finished by calling SMIF_1_GetTxfrStatus().
12. Changing of SMIF from memory to normal mode does not invalidate cache. As a result, a read after the second write in memory will return cached outdated values. It is recommended to the SMIF_CacheInvalidate() function to invalidate the cache, if values in external memory were updated in memory mode. See the PDL API reference guide for the details.

Input/Output Connections

This section describes the various input and output connections for the SMIF_PDL Component. An asterisk (*) in the following list indicates that it may not be shown on the Component symbol for the conditions listed in the description of that I/O.



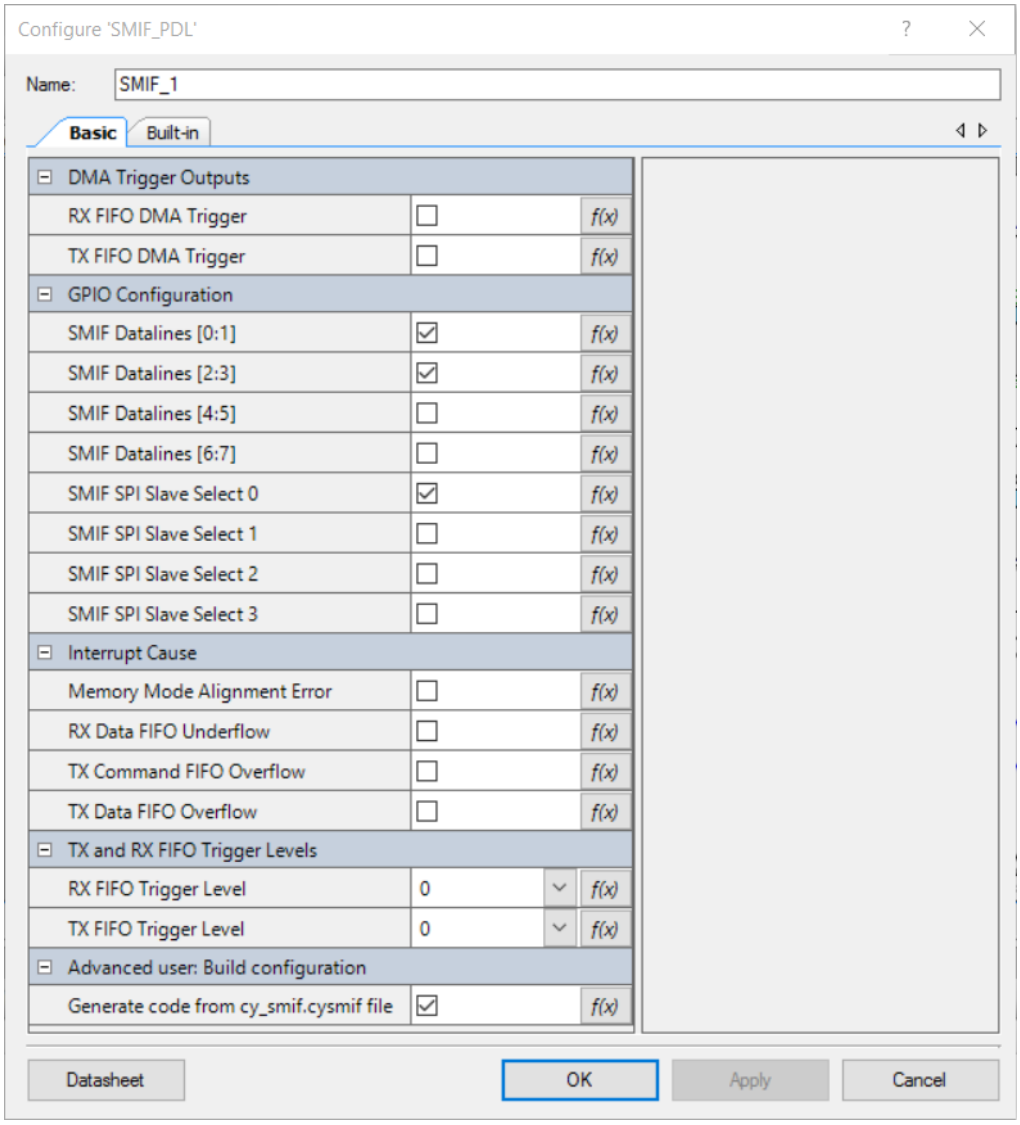
Terminal Name	I/O Type	DescrIPtion
rx_dma *	Digital Output	This signal can only be connected to a DMA channel trigger input. This signal is used to trigger a DMA transaction. The output of this terminal is controlled by the RX FIFO level. The presence of this terminal varies, depending on the RX FIFO DMA Trigger parameter. It is an optional terminal because the DMA usage is optional.
tx_dma *	Digital Output	This signal can only be connected to a DMA channel trigger input. This signal is used to trigger a DMA transaction. The output of this terminal is controlled by the TX FIFO level. The presence of this terminal varies, depending on the TX FIFO DMA Trigger parameter. It is an optional terminal because the DMA usage is optional.

Component Parameters

The SMIF_PDL Component Configure dialog allows you to edit the configuration parameters for the Component instance.

Basic Tab

This tab contains the Component parameters used in the general peripheral initialization settings.



Parameter Name	DescrIPtion
RX FIFO DMA Trigger	DMA trigger enable for the RX FIFO trigger.
TX FIFO DMA Trigger	DMA trigger enable for the TX FIFO trigger.
SMIF Datalines	When selected, these fields enable the corresponding data lines to be used by the SMIF block. The selections have no impact on generated code/constants for the SMIF driver.
SMIF SPI Slave Select	When selected these fields enable the corresponding slave select lines to be used by the SMIF block. The selections have no impact on generated code/constants for the SMIF driver.
Memory mode alignment error	Alignment error in the memory mode (XIP mode) is set as an interrupt cause.
RX Data FIFO Underflow	The RX Data FIFO underflow condition is set as an interrupt cause.
TX Command FIFO Overflow	The TX command FIFO overflow condition is set as an interrupt cause.
TX Data FIFO Overflow	The TX data FIFO overflow condition is set as an interrupt cause.
RX FIFO Trigger Level	This sets up the level that would trigger the RX FIFO trigger which could trigger an interrupt or a DMA request.
TX FIFO Trigger Level	This sets up the level that would trigger the TX FIFO trigger which could trigger an interrupt or a DMA request.
Generate code from cy_smif.cysmif file	This option calls the SMIF header generator during the build process. If you de-select this check box, the SMIF Configuration Tool menu item will be disabled. For more information, refer to the SMIF Configuration Tool User Guide.

Application Programming Interface

The Application Programming Interface (API) is provided by the cy_smif driver module from the PDL. The driver is copied into the “pdl\drivers\peripheral\smif\” directory of the application project after a successful build.

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open PDL Documentation…**” option in the drop-down menu.

The Component generates the configuration structures and base address described in the Global Variables and Preprocessor Macros sections. Pass the generated data structure and the base address to the associated cy_smif driver function in the application initialization code to configure the peripheral. Once the peripheral is initialized, the application code can perform run-time changes by referencing the provided base address in the driver API functions.

The SMIF_PDL Component provides an instance-based reference API that provide wrappers around the PDL API. These are provided as reference only.

By default, PSoC Creator assigns the instance name SMIF_1 to the first instance of a Component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol.

Global Variables

The SMIF_PDL Component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the Component (e.g., SMIF_1.c).

cy_stc_smif_mem_config_t* SMIF_1_memSlotConfigs[((uint8_t) SMIF_DEVICE_NR)]

Allocate space memory configurations.

cy_stc_smif_config_t SMIF_1_config

Allocate space device configuration.

cy_stc_smif_context_t SMIF_1_context

Allocate space for context.

uint8 SMIF_1_initVar

SMIF_1_initVar indicates whether the SMIF_1 Component has been initialized. The variable is initialized to 0 and set to 1 the first time SMIF_1_Start() is called. This allows the Component to restart without re-initialization after the first call to the SMIF_1_Start() routine.

Preprocessor Macros

The SMIF_PDL Component generates the following preprocessor macro(s).

#define SMIF_1_DESELECT_DELAY (7u)

Minimum duration of SPI de-selection

#define SMIF_1_RX_CLOCK_SELECT (1u)

Clock source for the receiver clock

#define SMIF_1_AHB_BUS_ERROR (0u)

What happens when there is a Read to an empty RX FIFO or a Write to a full TX FIFO.

#define SMIF_1_HW (SMIF_1_SMIF__HW)

Pointer to the base address of the SMIF instance

Macro defined by the Component parameter.

In the following table, there is a list of macros that reflect values of Component parameters selected in Component GUI. See parameter details in the Component Parameters section.

SMIF Component parameter name	Macro generated in SMIF_1.h
Memory mode alignment error	SMIF_1_MEMORY_MODE_ALIGMENT_ERORR
RX Data FIFO Underflow	SMIF_1_RX_DATA_FIFO_UNDERFLOW
TX Command FIFO Overflow	SMIF_1_TX_COMMAND_FIFO_OVERFLOW
TX Data FIFO Overflow	SMIF_1_TX_DATA_FIFO_OVERFLOW
RX FIFO Trigger Level	SMIF_1_RX_FIFO_TRIGEER_LEVEL
TX FIFO Trigger Level	SMIF_1_TX_FIFO_TRIGEER_LEVEL

Component Functions

This Component also includes a set of Component-specific functions that provide additional functionality available through PSoC Creator. These functions are generated during the build process and are all prefixed with the name of the Component instance.

void SMIF_1_Start (cy_stc_smif_block_config_t *configStruct, uint32_t timeout)

This function starts the SMIF block, allocating and configuring its interrupt for Normal mode. This function initializes all the memory slots, sets the trigger level, and enables Memory mode cache with prefetching. The SMIF HW block is configured according to the SMIF_1_config values.

Note:

Changing SMIF mode does not invalidate cache. You should invalidate cache after changing from Normal to Memory mode. This will prevent reading outdated values from cache in Memory mode.

Parameters:

configStruct	Define configuration of the external memories connected to the SMIF.
timeout	Timeout in microseconds for blocking APIs in use.

Global Variables

SMIF_1_initVar - Checks the initial configuration modified on the first function call.

SMIF_1_memSlotConfigs - Allocates array of external memory configuration structures.

Interrupt Service Routine

The SMIF_PDL Component has a single ISR. This ISR has six interrupt causes. The driver provides corresponding macro definitions to enable each interrupt trigger (refer to the PDL API Reference Guide).:

- TX FIFO Trigger. This interrupt cause is activated in MMIO mode, when the TX data FIFO has TX FIFO Trigger Level free entries.
- RX FIFO Trigger. This interrupt cause is activated in MMIO mode, when the RX data FIFO has RX FIFO Trigger Level used entries.
- Memory mode alignment error. This interrupt cause is activated in XIP mode, when erroneous behavior in dual-quad SPI mode is identified.
- TX Data FIFO Overflow. This interrupt cause is activated in MMIO mode, when the TX data FIFO is overflowed - not enough free entries available.
- RX Data FIFO Underflow. This interrupt cause is activated in MMIO mode, when the RX data FIFO is underflowed - no data available.
- TX Command FIFO Overflow. This interrupt cause is activated in MMIO mode when the TX command FIFO is overflowed - not enough free entries available.

Functional Description

The SMIF_PDL Component initializes the SMIF hardware block in normal mode with enabled cache and prefetching features. Cache is used in memory mode only. Initialization configuration of the SMIF block is defined in the SMIF_1_config structure. This structure is initialized with default values (see Preprocessor Macros section for details).

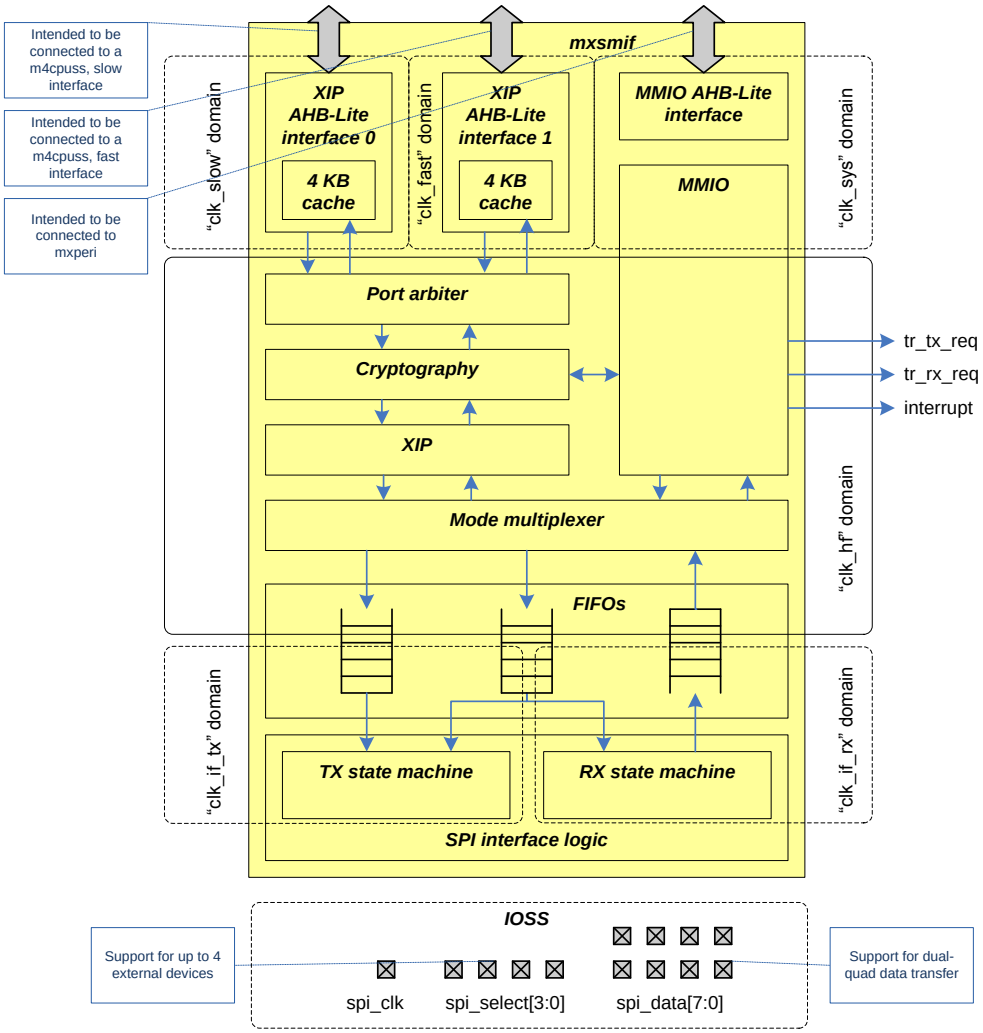
Note The driver is not responsible for external memory persistence. User cannot edit buffer during read/write operations. In case of a memory error, the SMIF IP block could get stuck and need to be reset. Check the SMIF IP busy status using the API call, and implement a timeout. You could also reset the IP block by toggling CTL.ENABLED. Then reconfigure the SMIF block.

For write operations, check that the SMIF finished transferring data by calling SMIF_1_BusyCheck() and check that memory finished operation using SMIF_1_Mem_IsBusy() before proceeding. For read operations before accessing to the read buffer, check that the SMIF operation finished by calling SMIF_1_GetTxfrStatus().

Changing the SMIF from memory to normal mode does not invalidate cache. As a result, the read after the second write in memory will return outdated cached values. You should call the SMIF_CacheInvalidate() function to invalidate cache if values in external memory were updated in memory mode. See the PDL API reference guide for the details.

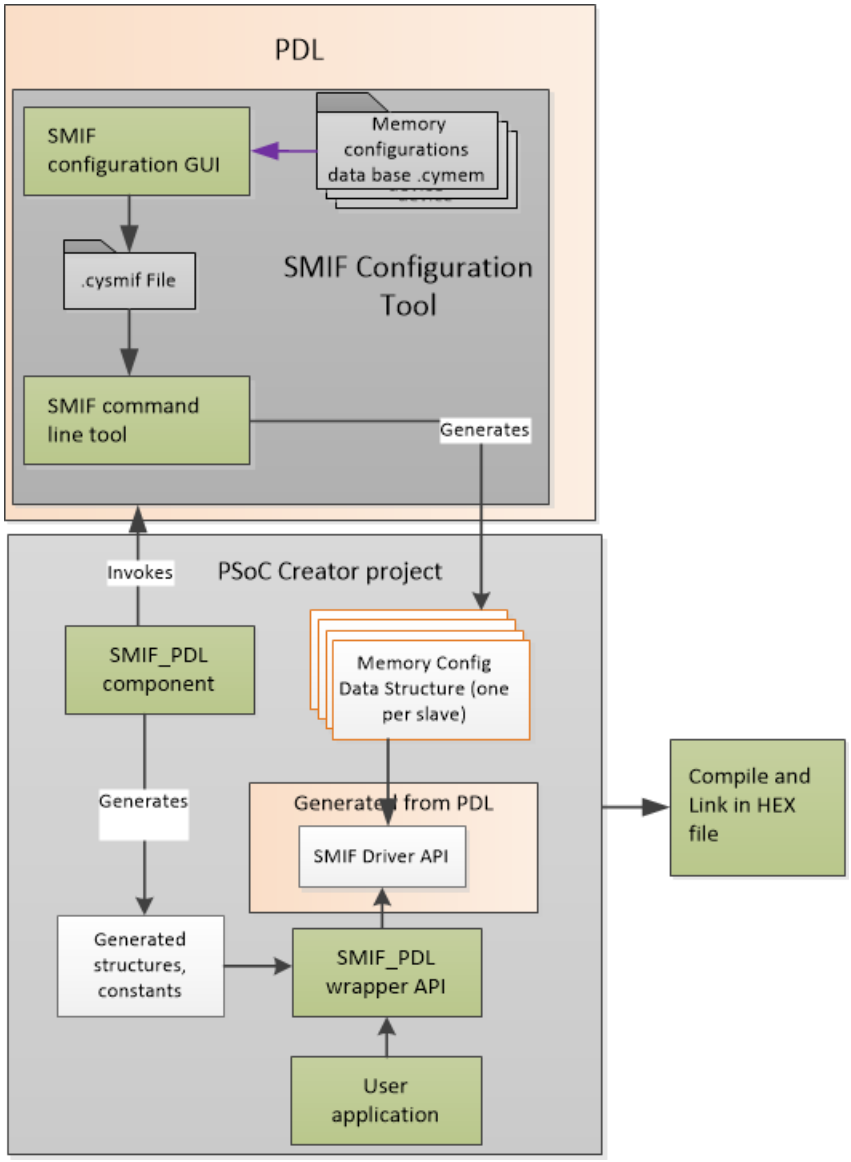
Block Diagram and Configuration

The following shows a simplified diagram of the SMIF hardware:



See the Serial Memory Interface (SMIF) section in the device *Technical Reference Manual (TRM)* for more information about the hardware block description and configuration values.

The following shows the general SMIF_PDL Component usage flow. The Component API is a wrapper over the cy_smif driver API, which is part of the PDL. The Component API needs external memory configuration structures as input parameters. These configuration parameters could be generated using the SMIF Configuration Tool. This tool is part of the PDL and it can be invoked directly from the Component context menu (see SMIF Configuration Tool user guide for details).



Clock Selection

The internal clock source for SPI interface logic of SMIF_PDL is the HFClk2 clock (which is configurable in the PSoC Creator Clock Editor). This clock needs to be configured and enabled for the SMIF to function properly.

DMA Support

The SMIF_PDL Component supports Direct Memory Access (DMA) transfers. The Component may transfer to/from the following sources.

Name of DMA Source	Length	Direction	DMA Req Signal	DMA Req Type	DescrIPtion
TRIG13_IN_SMIF_TR_RX_REQ	Word / Byte or Halfword	Source Destination	rx_dma_tr	Level sensitive	Receive FIFO
TRIG13_IN_SMIF_TR_TX_REQ	Word / Byte or Halfword	Destination	tx_dma_tr	Level sensitive	Transmit FIFO

Industry Standards

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The SMIF_PDL Component has the following specific deviations:

Rule	Rule Class	Rule DescrIPtion	DescrIPtion of Deviation(s)
1.1	R	This rule states that code shall conform to C ISO/IEC 9899:1990 standard.	PDL v3.0.0 supports ISO:C99 standard.
11.3	A	A cast should not be performed between a pointer to volatile object and an integral type.	The cast from unsigned integer to pointer does not have any unintended effect, as it is a consequence of the definition of a structure based on hardware registers.

This Component uses firmware drivers from the cy_smif PDL module. Refer to the PDL documentation for information on their MISRA compliance and specific deviations.

Registers

See the Serial Memory Interface (SMIF) Registers section in the device *Technical Reference Manual (TRM)* for more information about the registers.

Resources

The SMIF_PDL Component uses the mxsmif peripheral block.

DC and AC Electrical Characteristics

Note Final characterization data for PSoC 6 devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site.

References

- PDL API Reference Manual (<PDL installation folder>/doc/pdl_api_reference_manual.html)
- PDL User Guide (<PDL installation folder>/doc/pdl_user_guide.pdf)
- SMIF Configuration Tool User Guide (<PDL installation folder>/doc/smif_config_tool_user_guide.pdf)
- *Technical Reference Manual (TRM)*

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.a	Minor datasheet edits.	
1.10	Updated the Component version.	Support of driver changes.
1.0.b	Updated the Component symbol Updated Configure dialog Updated datasheet	Removed “Prototype” label Added/removed parameters Updated References section Updated MISRA section Added Clock Section section Added usage flow text and diagram Updated API section Updated Parameters section
1.0.a	Updated datasheet	Added driver initialization and data transmit example code in the Quick Start section. Added explanation why the SMIF API is divided in the low level API and the Memory slot API in the Quick Start section. Added user responsibility notes in the Quick Start section. Updated the Basic tab and its description table. Removed wrapper functions from the Component Functions section.
1.0	Initial Version	