**SPI PROTOCOL**

**==============================================================================**

**How to use this driver**

**==============================================================================**

**The SPI HAL driver can be used as follows:**

Declare a **SPI\_HandleTypeDef** handle structure, for example: SPI\_HandleTypeDef hspi;

Initialize the SPI low level resources by implementing the **HAL\_SPI\_MspInit()** API:

1. Enable the **SPIx** interface clock
2. SPI pins configuration: Enable the clock for the SPI GPIOs. **Configure these SPI** pins as alternate function push-pull
3. NVIC configuration if you need to use interrupt process: Configure the **SPIx interrupt priority**. Enable the **NVIC SPI IRQ** handle
4. DMA Configuration if you need to use DMA process

* Declare a **DMA\_HandleTypeDef** handle structure for the transmit or receive Stream/Channel
* Enable the **DMAx clock**
* Configure the **DMA handle parameters**
* Configure the **DMA Tx or Rx Stream/Channel**
* Associate the initialized hdma\_tx(or \_rx) handle to the hspi DMA Tx or Rx handle
* Configure the priority and enable the NVIC for the transfer complete interrupt on **the DMA Tx or Rx Stream/Channel**

1. **Program the Mode**, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
2. **Initialize the SPI registers** by calling the **HAL\_SPI\_Init()** API: This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized **HAL\_SPI\_MspInit()** API.

**Circular mode restriction:**

1. The DMA circular mode cannot be used when the SPI is configured in these modes:

* Master 2Lines RxOnly
* Master 1Line Rx

1. The CRC feature is not managed when the DMA circular mode is enabled
2. When the SPI DMA Pause/Stop features are used, we must use the following APIs the **HAL\_SPI\_DMAPause()/ HAL\_SPI\_DMAStop()** only under the SPI callbacks

**Master Receive mode restriction:**

In Master unidirectional receive-only mode (MSTR =1, BIDIMODE=0, RXONLY=1) or bidirectional receive mode (MSTR=1, BIDIMODE=1, BIDIOE=0), to ensure that the SPI does not initiate a new transfer the following procedure has to be respected:

* **HAL\_SPI\_DeInit()**
* **HAL\_SPI\_Init()**

**Callback registration:**

The compilation flag USE\_HAL\_SPI\_REGISTER\_CALLBACKS when set to 1U allows the user to configure dynamically the driver callbacks. Use Functions **HAL\_SPI\_RegisterCallback()** to register an interrupt callback.

Function **HAL\_SPI\_RegisterCallback()** allows to register following callbacks:

* TxCpltCallback: SPI Tx Completed callback
* RxCpltCallback: SPI Rx Completed callback
* TxRxCpltCallback: SPI TxRx Completed callback
* TxHalfCpltCallback: SPI Tx Half Completed callback
* RxHalfCpltCallback: SPI Rx Half Completed callback
* TxRxHalfCpltCallback: SPI TxRx Half Completed callback
* ErrorCallback: SPI Error callback
* AbortCpltCallback: SPI Abort callback
* MspInitCallback: SPI Msp Init callback
* MspDeInitCallback: SPI Msp DeInit callback

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function

Use function **HAL\_SPI\_UnRegisterCallback** to reset a callback to the default weak function.

**HAL\_SPI\_UnRegisterCallback** takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

* TxCpltCallback: SPI Tx Completed callback
* RxCpltCallback: SPI Rx Completed callback
* TxRxCpltCallback: SPI TxRx Completed callback
* TxHalfCpltCallback: SPI Tx Half Completed callback
* RxHalfCpltCallback: SPI Rx Half Completed callback
* TxRxHalfCpltCallback: SPI TxRx Half Completed callback
* ErrorCallback: SPI Error callback
* AbortCpltCallback: SPI Abort callback
* MspInitCallback: SPI Msp Init callback
* MspDeInitCallback: SPI Msp DeInit callback

[..]

By default, after the **HAL\_SPI\_Init()** and when the state is HAL\_SPI\_STATE\_RESET all callbacks are set to the corresponding weak functions: examples HAL\_SPI\_MasterTxCpltCallback(), HAL\_SPI\_MasterRxCpltCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL\_SPI\_Init()/ HAL\_SPI\_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL\_SPI\_Init()/ HAL\_SPI\_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

[..]

Callbacks can be registered/unregistered in HAL\_SPI\_STATE\_READY state only.

Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL\_SPI\_STATE\_READY or HAL\_SPI\_STATE\_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL\_SPI\_RegisterCallback() before calling HAL\_SPI\_DeInit() or HAL\_SPI\_Init() function.

[..]

When the compilation define USE\_HAL\_PPP\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

[..]

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes, the following table resume the max SPI frequency reached with data size 8bits/16bits, according to frequency of the APBx Peripheral Clock (fPCLK) used by the SPI instance.