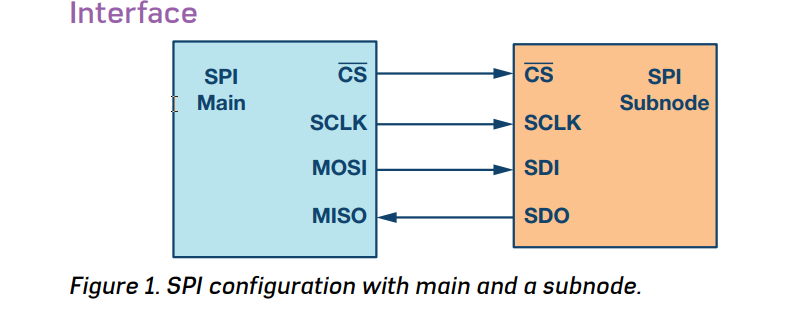
**THE INFORMATIONS ABOUT SPI OF ESP32 C3**

1. **Overview about SPI This informations are summary of SPI so it can wrong with ESP32 C3**

With SPI communication, any number of bits can be sent or received in a continuous stream.

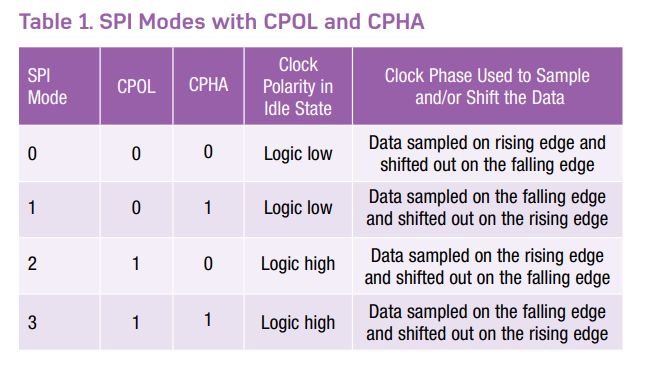
****

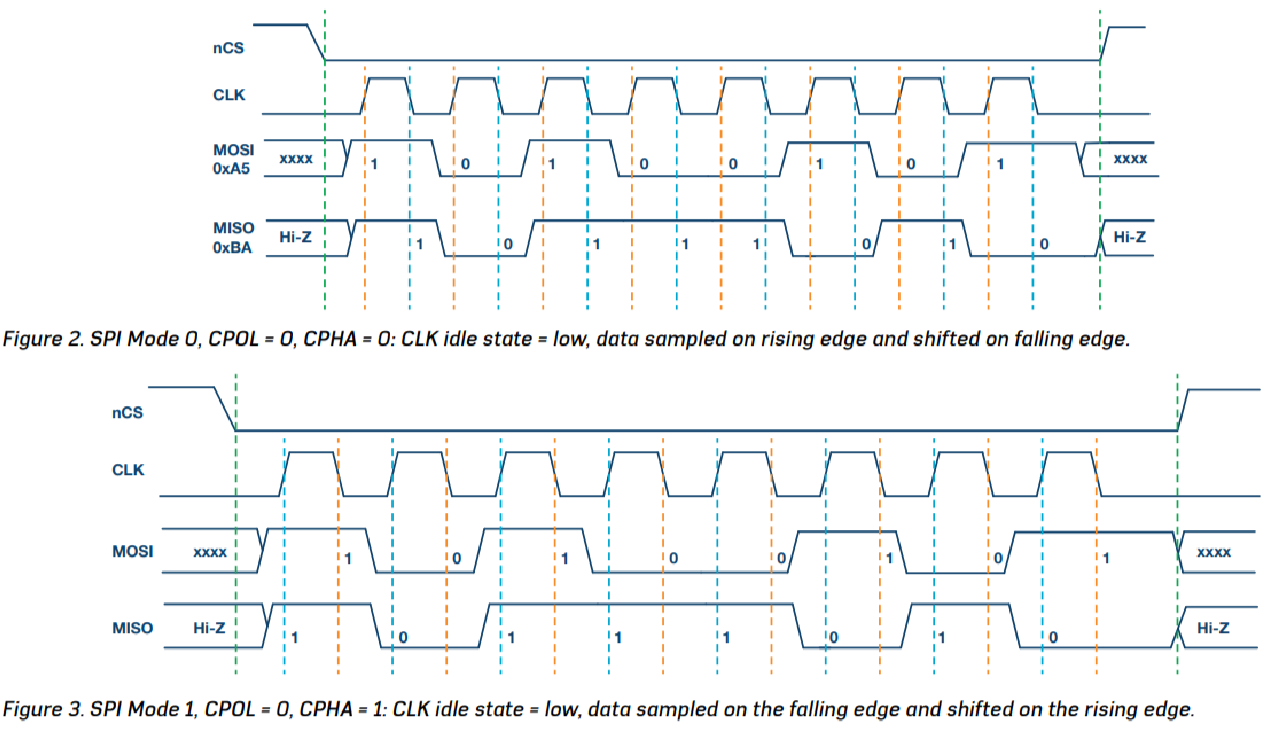
* ***4-wire SPI devices have four signals:***
* Clock (SPI CLK, SCLK)
* Chip select (CS)
* Main out, subnode in (MOSI)
* Main in, subnode out (MISO)

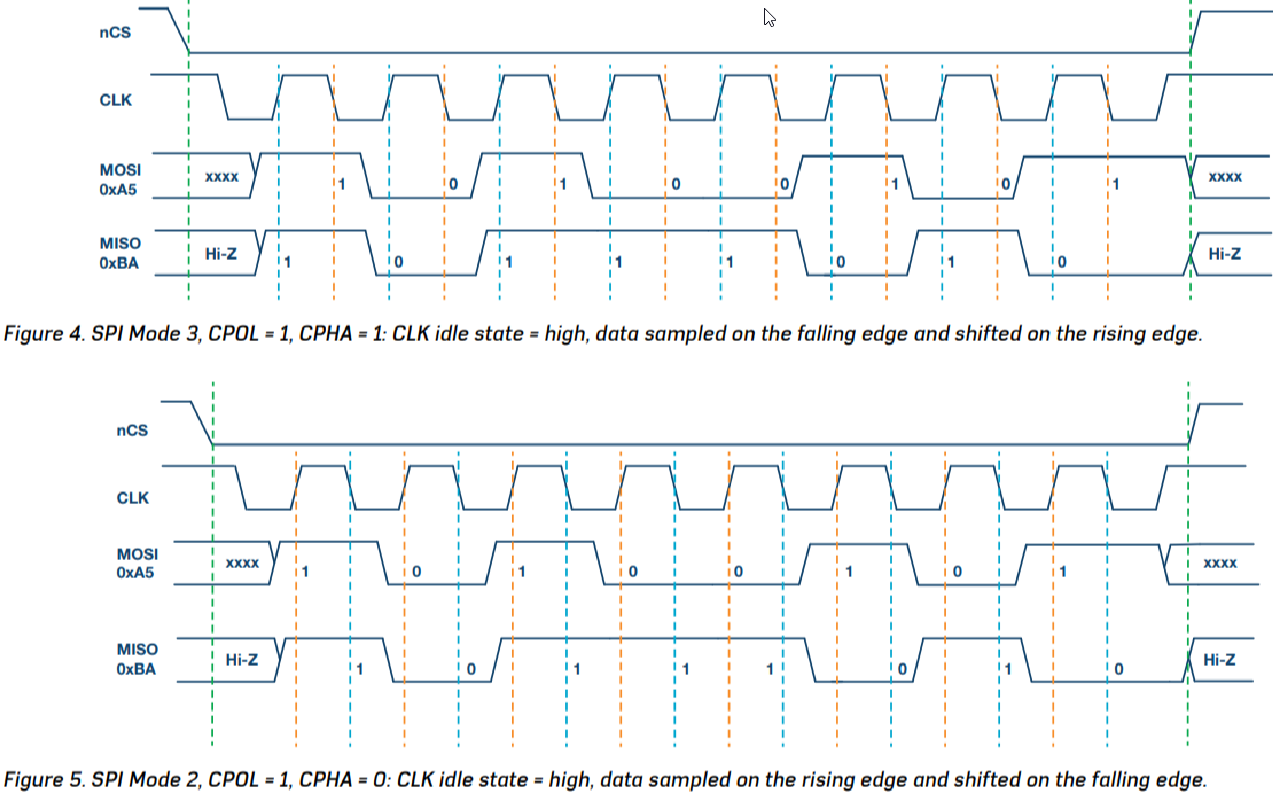
The device that generates the clock signal is called the main. Data transmitted between the main and the subnode is synchronized to the clock generated by the main. The chip select signal from the main is used to select the subnode. This is normally an active low signal and is pulled high to disconnect the subnode from the SPI bus. MOSI and MISO are the data lines. MOSI transmits data from the main to the subnode and MISO transmits data from the subnode to the main.

* ***Data transmission***
* Begin SPI communication, the main must send the clock signal and select the subnode by enabling the CS signal (usually is an active low signal).
* Data is simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus (MISO/SDI) is sampled or read in).
* ***Clock Polarity and clock phase***
* The CPOL bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission.
* Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or

shift the data. The main must select the clock polarity and clock phase, as per the requirement of the subnode.



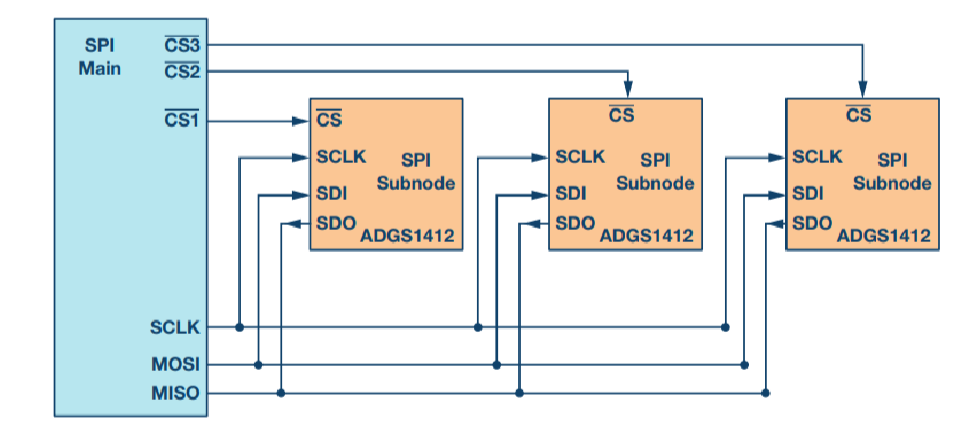




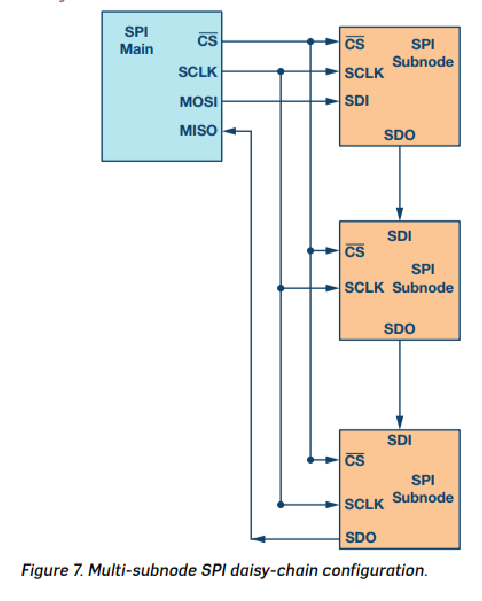
* ***Multi- Subnode configuration***

The subnodes can be connected in regular mode or daisy-chain mode.

* ***Regular SPI Mode***

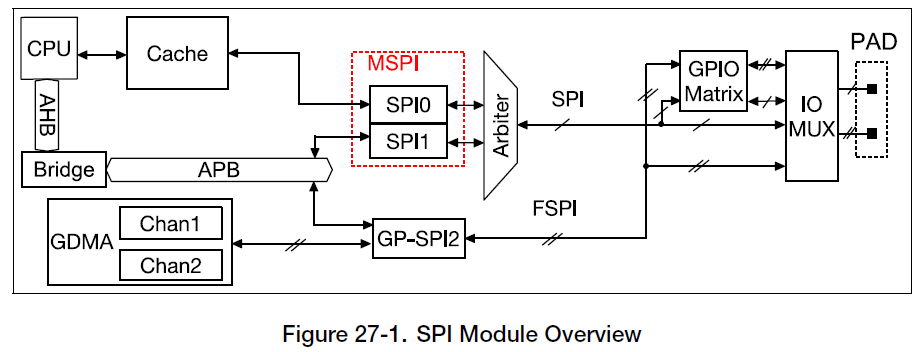
******

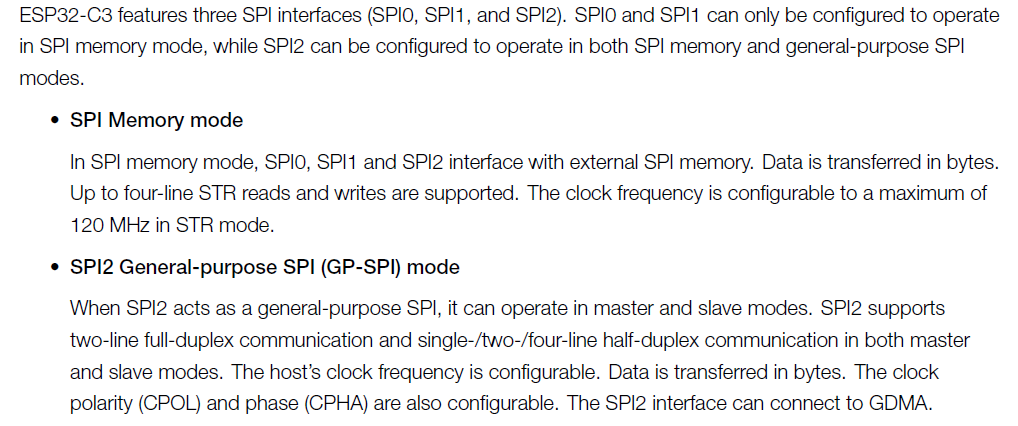
* ***Daisy- Chain Method***

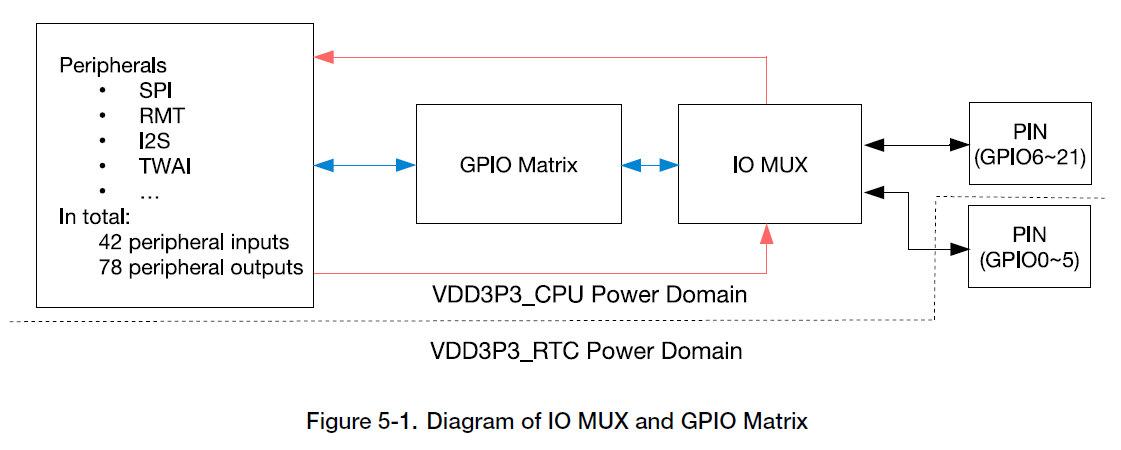


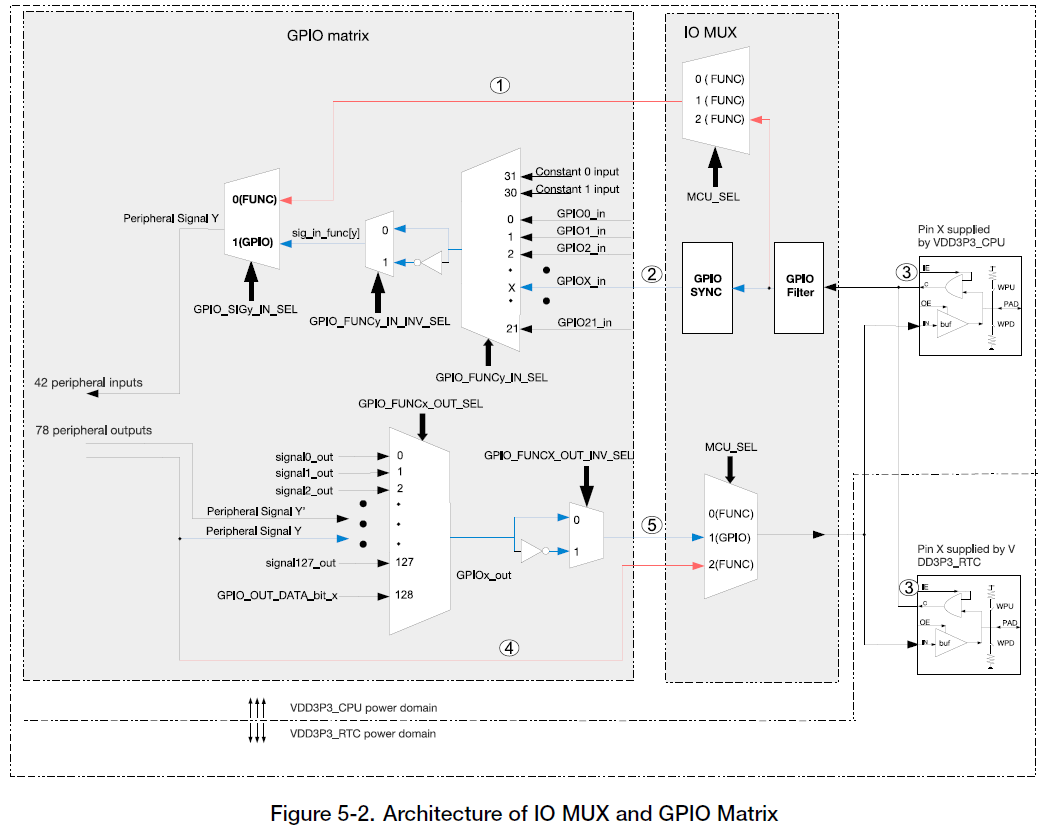
The chip select signal for all subnodes is tied together and data propagates from one subnode to the next. In this configuration, all subnodes receive the same SPI clock at the same time. The data from the main is directly connected to the first subnode and that subnode provides data to the next subnode and so on.

1. **Architecture overview**

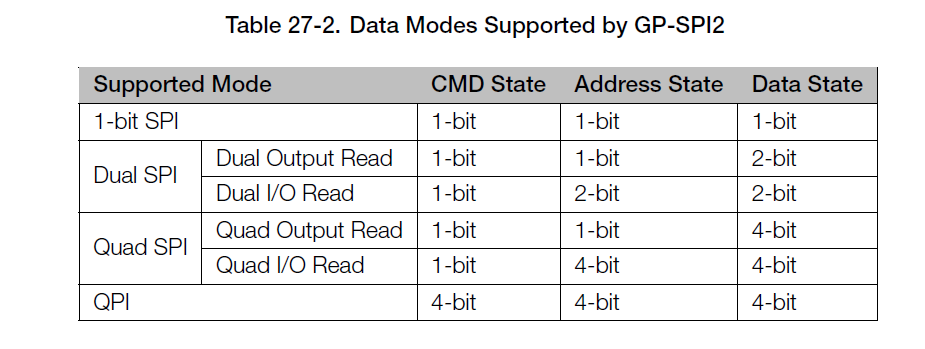




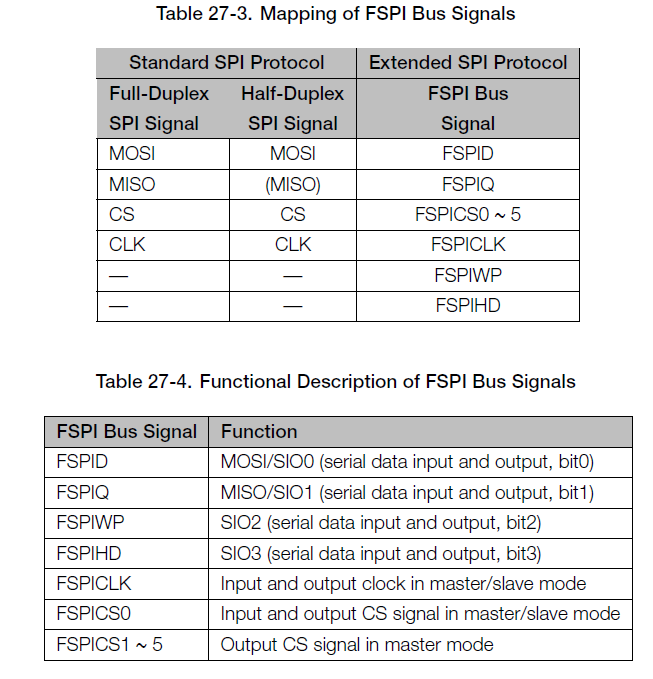




* ***Data mode:***

****

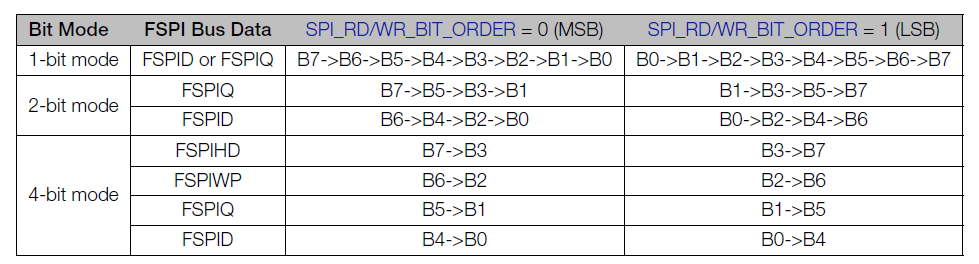
* ***FSPI Bus Signal Mapping***

****

* ***Bit Read/Write Order Control***

***SPI\_WR\_BIT\_ORDER***

***SPI\_RD\_BIT\_ORDER***

****

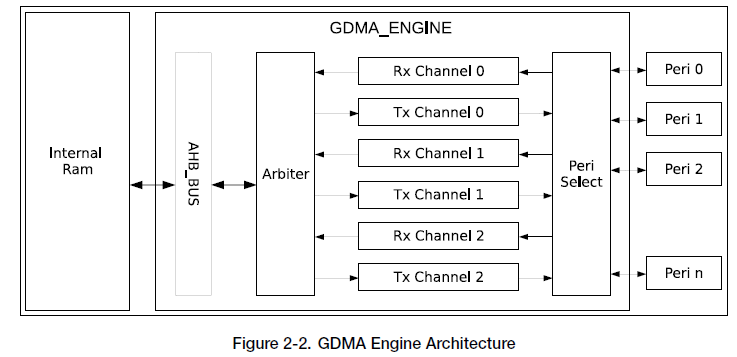
1. **DMA-Controlled Data Transfer**

A DMA-controlled transfer only needs to be triggered once by CPU,

* **GDMA Configuration**

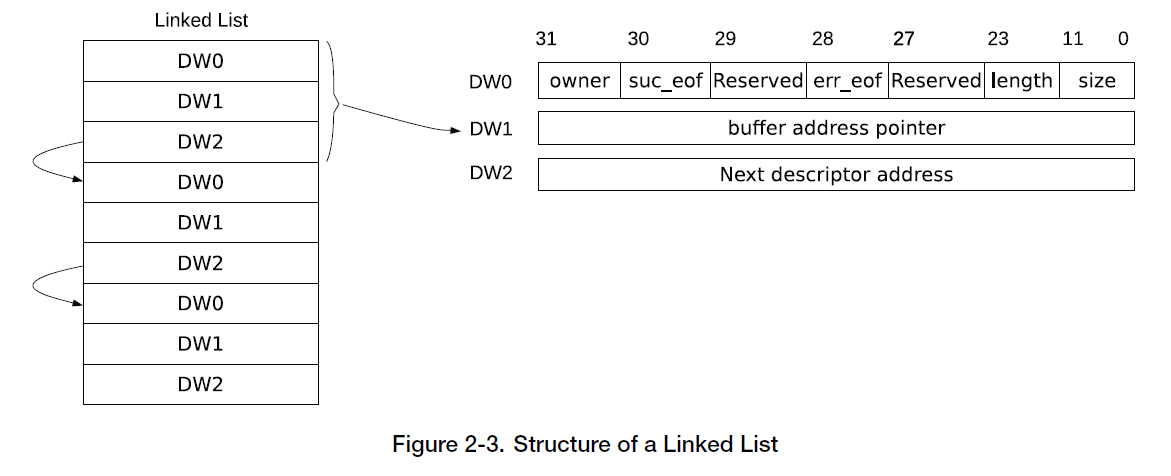
General Direct Memory Access (GDMA) is a feature that allows peripheral-to-memory, memory-to-peripheral, and memory-to-memory data transfer at a high speed.

The GDMA controller in ESP32-C3 has six independent channels.



The GDMA engine reads data from or writes data to internal RAM via the AHB\_BUS. Before this, the GDMA controller uses fixed-priority arbitration scheme for channels requesting read or write access. Software can use the GDMA engine through **linked lists**. These linked lists, stored in internal RAM, consist of outlinkn and inlinkn, where n indicates the channel number (ranging from 0 to 2).

* **Link List**



* **Owner:**

0: CPU can access the buffer.

1: The GDMA controller can access the buffer.

* **Suc\_eof:**

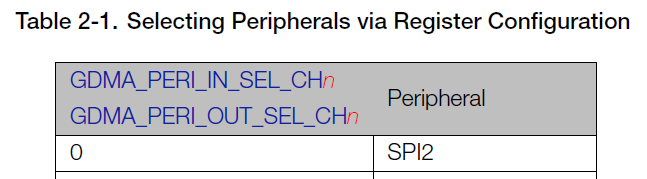
0: This descriptor is not the last one.

1: This descriptor is the last one.

* **Err\_eof**: When an error is detected in the received frame or packet, this bit in the receive descriptor is set to 1 by hardware
* **Length:** indicates how many bytes can be read from the buffer.
* **Size:** Specifies the size of the buffer that this descriptor points to.
* **Buffer address pointer (DW1):** Address of the buffer. This field can only point to internal RAM
* **Next descriptor address (DW2)**: Address of the next descriptor.

The GDMA controller can transfer data from memory to peripheral (transmit) and from peripheral to memory (receive). A transmit channel transfers data in the specified memory location to a peripheral’s transmitter via an outlinkn, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlinkn.

Select chanel peripherals



* **Enabling GDMA**
* When the GDMA controller receives data, software loads an inlink, configures GDMA\_INLINK\_ADDR\_CHn field with address of the first receive descriptor, and sets GDMA\_INLINK\_START\_CHn bit to enable GDMA.
* When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures GDMA\_OUTLINK\_ADDR\_CHn field with address of the first transmit descriptor, and sets GDMA\_OUTLINK\_START\_CHn bit to enable GDMA.
* **Check descriptors**
* ***Owner bit check***:

GDMA\_IN\_CHECK\_OWNER\_CHn or GDMA\_OUT\_CHECK\_OWNER\_CHnis set to 1, it’s pass.

* ***Buffer address pointer (DW1) check***: If the buffer address pointer points to 0x3FC80000 ~ 0x3FCDFFFF (please refer to Section 2.4.7), it passes the check.
* **EOF**

The GDMA controller uses EOF (end of frame) flags to indicate the end of data frame or packet transmission.

* **Accessing Internal RAM**

GDMA can send data in burst mode, which is disabled by default. This mode is enabled for receive channels by setting GDMA\_IN\_DATA\_BURST\_EN\_CHn,and enabled for transmit channels by setting GDMA\_OUT\_DATA\_BURST\_EN\_CHn.

* **Arbitration**

Each channel can be assigned a priority from 0 ~ 9. When several channels are assigned the same priority, the GDMA controller adopts a round-robin arbitration scheme.

1. **Data flow control in gp­spi2 master**

The two transfer modes to control transfers:

+ CPU-controlled : data transfers between registers *SPI\_W0\_REG ~ SPI\_W15\_REG* and the SPI device

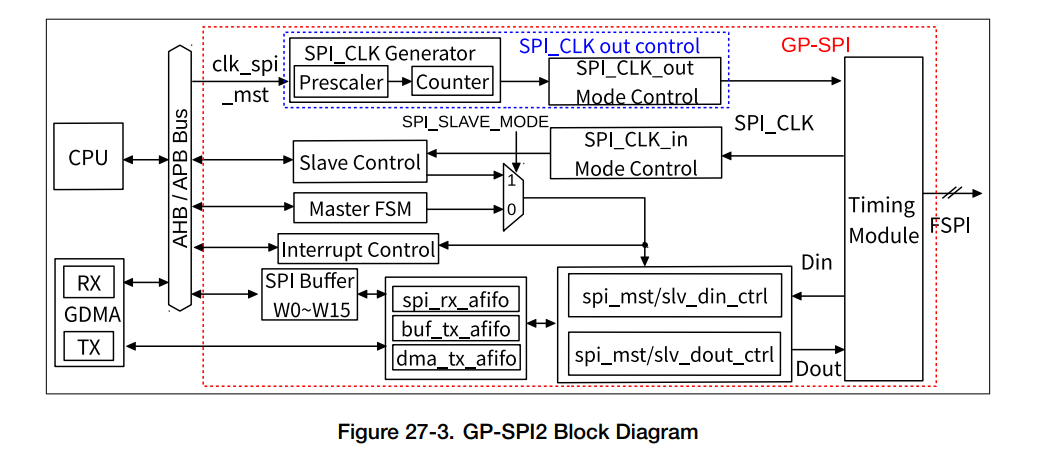
+ DMA-controlled : data transfers between the configured GDMA TX/RX buffer and the

SPI device.

**Configure:**

*SPI\_DMA\_RX\_ENA*

*SPI\_DMA\_TX\_ENA*



***SPI Buffer***: The data transferred in CPU-controlled mode is prepared in this buffer.

***spi\_mst/slv\_din/dout\_ctrl***: convert the TX/RX data into bytes.

**spi\_rx\_afifo**: store the received data

**buf\_tx\_afifo:** store the data to send from CPU

***dma\_tx\_afifo***: store the data from GDMA.

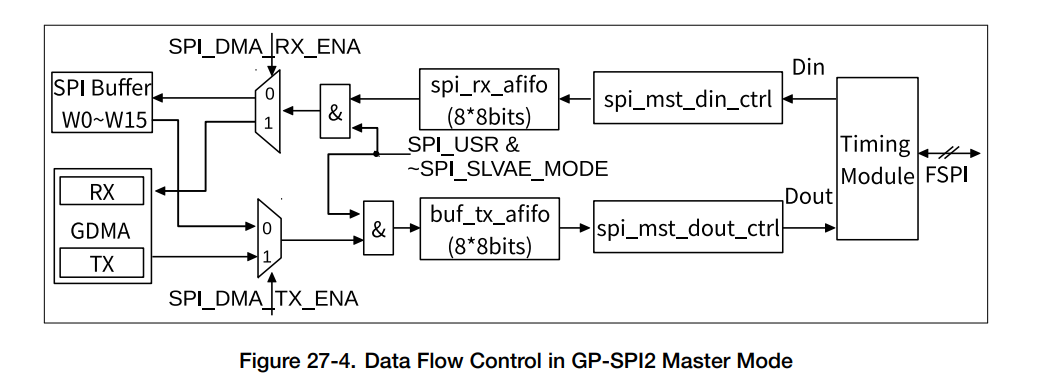
***clk\_spi\_mst***: this clock is the module clock of GP-SPI2 and derived from PLL\_CLK. It is used in GP-SPI2 master mode, to generate SPI\_CLK signal for data transfer and for slaves.

***SPI\_CLK Generator***: generate SPI\_CLK by dividing clk\_spi\_mst. The divider is determined by SPI\_CLKCNT\_N and SPI\_CLKDIV\_PRE.

***SPI\_CLK\_out Mode Control***: output the SPI\_CLK signal for data transfer and for slaves.

***SPI\_CLK\_in Mode Control***: capture the SPI\_CLK signal from SPI master when GP-SPI2 works as a slave.

***Data flow Control in Master Mode***



**RX data**: data in ***FSPI*** ***bus*** is captured by ***Timing Module***, converted in units of bytes by ***spi\_mst\_din\_ctrl*** module, and then stored in corresponding addresses according to the transfer modes.

SPI\_SLAVE\_MODE : 1: slave mode , 0: master mode

– CPU-controlled transfer: the data is stored to registers SPI\_W0\_REG ~ SPI\_W15\_REG.

– DMA-controlled transfer: the data is stored to GDMA RX buffer.

**TX data**: the TX data is from corresponding addresses according to transfer modes and is saved to ***buf\_tx\_afifo*** or **dma\_tx\_afifo.**

– CPU-controlled transfer: TX data is from ***SPI\_W0\_REG ~ SPI\_W15\_REG***.

– DMA-controlled transfer: TX data is from GDMA TX buffer.

The data in **buf\_tx\_afifo** is sent out to Timing Module in 1/2/4-bit modes, controlled by GP-SPI2 state machine. The Timing Module can be used for timing compensation.

1. **State Machine**

When GP-SPI2 works as a master, the state machine controls its various states during data transfer, including:

1. ***IDLE***: GP-SPI2 is not active or is in slave mode.

2. ***CONF***: only used in DMA-controlled configurable segmented transfer. Set SPI\_USR and SPI\_USR\_CONF to enable this state. If this state is not enabled, it means the current transfer is a single transfer.

SPI\_USR: 1: enable; 0: disable

SPI\_USR\_CONF: 1: use DMA-controlled configurable segmented transfer.

0: transfer is a single transfer.

3. ***PREP***: prepare an SPI transaction and control SPI CS setup time. Set SPI\_USR and SPI\_CS\_SETUP to enable this state.

SPI\_CS\_SETUP: 1: enable SPI CS

0: disable

4. ***CMD***: send command sequence. Set SPI\_USR and SPI\_USR\_COMMAND to enable this state.

5. ***ADDR***: send address sequence. Set SPI\_USR and SPI\_USR\_ADDR to enable this state.

6. ***DUMMY*** (wait cycle): send dummy sequence. Set SPI\_USR and SPI\_USR\_DUMMY to enable this state.

7. ***DATA***: transfer data.

• DOUT: send data sequence. Set SPI\_USR and SPI\_USR\_MOSI to enable this state.

• DIN: receive data sequence. Set SPI\_USR and SPI\_USR\_MISO to enable this state.

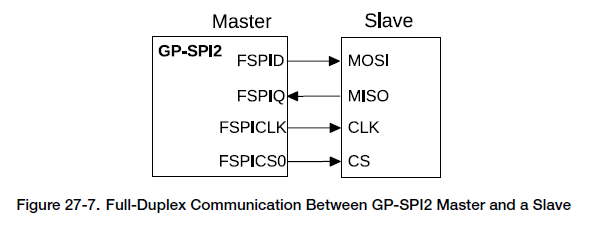
8. ***DONE***: control SPI CS hold time. Set SPI\_USR to enable this state.

* **DMA-Controlled Configurable Segmented Transfer**

A DMA-controlled transfer in master mode can configurable segmented transfer, consisting of several transactions (segments). In a configurable segmented transfer, the registers of its each single transaction (segment) are configurable. When this whole DMA-controlled transfer (consisting of several segments) has finished, a GP-SPI2 interrupt, SPI\_DMA\_SEG\_TRANS\_DONE\_INT, is triggered.

**V. Full- Duplex: (1-bit Mode Only)**

Master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode via MOSI (FSPID, sending) and MISO (FSPIQ, receiving) at the same time.

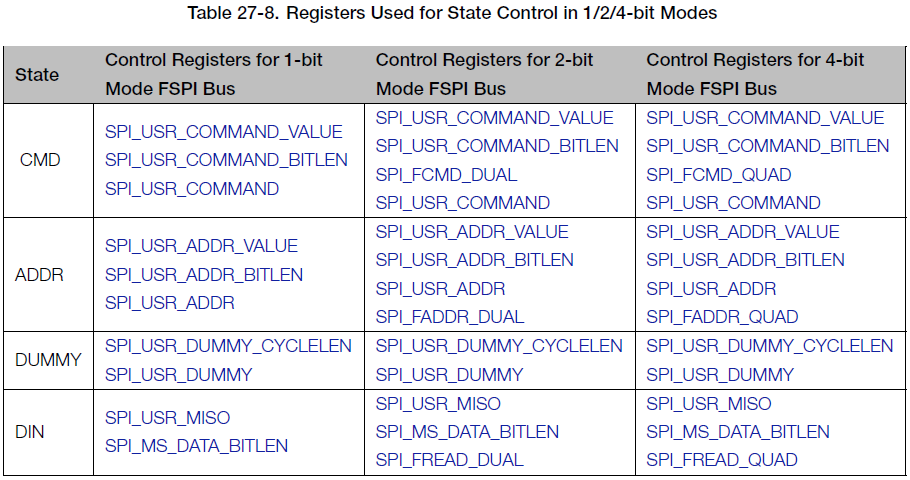
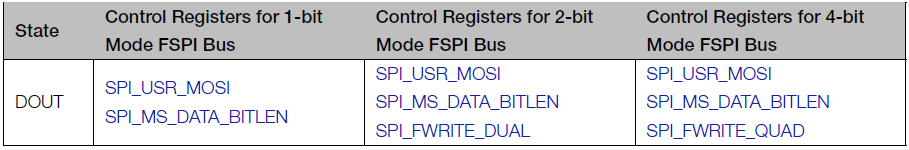


In full-duplex communication, usually, the states CMD, ADDR and DUMMY are not used in this communication.

The bit length of transferred data is configured in SPI\_MS\_DATA\_BITLEN. The actual bit length used in communication equals to (SPI\_MS\_DATA\_BITLEN + 1).

**Configuration:**

1. Configure the IO path via IO MUX and GPIO matrix
2. Configure APB clock and module clock (clk\_spi\_mst)
3. Set SPI\_DOUTDIN and clear SPI\_SLAVE\_MODE to enable full-duplex communication.
4. Configure GP-SPI2 registers listed in table 27-8. (State machine)



1. Configure SPI CS setup time and hold time according to Section 27.6

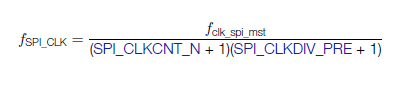
SPI CS setup time: SPI\_CS\_SETUP in SPI\_USER\_REG and

SPI\_CS\_SETUP\_TIME in SPI\_USER1\_REG

SPI CS hold time: SPI\_CS\_HOLD in SPI\_USER\_REG and

SPI\_CS\_HOLD\_TIME in SPI\_USER1\_REG:

1. Set the property of FSPICLK



Use clock divisions, set SPI\_CLK\_EQU\_SYSCLK is 0

1. Prepare data according to the selected transfer mode:

– In CPU-controlled MOSI mode, prepare data in registers SPI\_W0\_REG ~ SPI\_W15\_REG.

– In DMA-controlled mode,

* configure SPI\_DMA\_TX\_ENA/SPI\_DMA\_RX\_ENA
* configure GDMA TX/RX link
* start GDMA TX/RX engine

Set the bit GDMA\_INLINK\_START\_CHn or GDMA\_OUTLINK\_START\_CHn to start GDMA RX/TX engine.

1. Configure interrupts and wait for SPI slave to get ready for transfer.
2. Set SPI\_DMA\_AFIFO\_RST, SPI\_BUF\_AFIFO\_RST, and SPI\_RX\_AFIFO\_RST to reset these buffers.
3. Set SPI\_USR in register SPI\_CMD\_REG to start the transfer and wait for the configured interrupts
4. **Half-Duplex Communication.**

Only one side (SPI master or slave) can send data at a time, while the other side receives the data. To enable this communication mode, clear the bit SPI\_DOUTDIN in register SPI\_USER\_REG.

**Clock Control in Master Mode**

The polarity and phase of GP-SPI2 clock are controlled by the bit SPI\_CK\_IDLE\_EDGE in register SPI\_MISC\_REG and the bit SPI\_CK\_OUT\_EDGE in register SPI\_USER\_REG.

**GPSPI2 Timing Compensation**

**?????????????????????????????**

**Interrupts**

When an SPI transfer ends, an interrupt is generated in GP-SPI2.

**REFERENCES**

[1]. Piyu Dhaker, *Introduction to SPI Interface*,

[**https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#:~:text=SPI%20is%20a%20synchronous%2C%20full,%2Dwire%20or%204%2Dwire**](https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#:~:text=SPI%20is%20a%20synchronous%2C%20full,%2Dwire%20or%204%2Dwire)**.**