Mode	Clock polarity (CPOL)	Clock phase (CPHA)
SPI_MODE0	0 (Active `High`)	0 (Leading)
SPI_MODE1	0 (Active `High`)	1 (Trailing)
SPI_MODE2	1 (Active `Low`)	0 (Leading)
SPI_MODE3	1 (Active `Low`)	1 (Trailing)

Table 59: SPI modes

8.19.5 Shared resources

The SPIM peripheral shares registers and other resources with peripherals that have the same ID as SPIM. Before SPIM can be configured and used, all peripherals that have the same ID as SPIM must be disabled.

Disabling a peripheral with the same ID as SPIM will not reset any shared SPIM registers. Configure all SPIM registers to ensure they operate correctly.

See the Instantiation table in Instantiation on page 216 for details on peripherals and their IDs.

8.19.6 EasyDMA

SPIM uses EasyDMA to fetch data to transmit from RAM or store received data in RAM.

SPIM implements the following EasyDMA channels.

Channel	Type	Register Cluster
TXD	READER	DMA.TX.PTR on page 609
RXD	WRITER	DMA.RX.PTR on page 606

Table 60: SPIM EasyDMA channels

The .PTR and .MAXCNT registers are double-buffered. After receiving the STARTED event, the registers can be written to before the next transmission.

SPIM automatically stops transmitting after TXD.MAXCNT bytes have been transmitted and RXD.MAXCNT bytes have been received. If RXD.MAXCNT is larger than TXD.MAXCNT, the remaining transmitted bytes will contain the value defined in the ORC register. If TXD.MAXCNT is larger than RXD.MAXCNT, the additional received bytes will be discarded.

The RX.END and TX.END events indicate that EasyDMA has finished accessing buffers in RAM. Both RX and TX must be finished before the END event is generated.

If several AHB bus masters try to access the same AHB slave at the same time, AHB bus congestion can occur. In this case, the EasyDMA channel behavior will depend on the SPIM instance. Refer to Instances on page 588 for information about what behavior is expected in each instance.

See EasyDMA for more detailed information.

8.19.7 Low power

When the peripheral is not needed, stop and disable SPIM to ensure lowest possible power consumption.

When the STOP task is sent, the software must wait until the STOPPED event is received before disabling the peripheral through the ENABLE register. If the peripheral is already stopped, the STOP task is not needed to ensure data is not lost.