Datasheet

CC-I2C_MST-APB Documentation

1.2

Scope

This document describes the CC-I2C_MST-APB IP core. Module features and configuration registers are described. The document contains integration guide that covers synthesis options and instantiation example for easy implementation in customer's environment.

Contents

1. I2C Master Controller

1.1 Functionality

- work in a Master mode,
- multi-master configuration support (automatic I2C bus arbitration),
- 7 and 10 bit addressing,
- configurable baud rate,
- configurable interrupt signaling the status of transmission and events on the bus,
- automatic clock-stretching in anticipation of reading or writing data,
- work with DMA,
- configurable automatic transmission (automatic Start, Stop, Repeated Start, sending and interpreting of ACK),
- four data transfer modes.

1.2 Overview

The I2C (Inter IC), is a synchronous interface utilizing two lines: a bidirectional SDA and a unidirectional SCL (clock line). The drivers of both lines are of the open collector type (drain), which through the resistors are connected to the power supply. At least two devices participate in the data exchange, one of which is a Master and the other one is a Slave. The Master device initiates and controls the exchange of data and generates a clock signal for the bus. A typical way of connecting between devices on the I2C bus is shown in Figure 1.1.

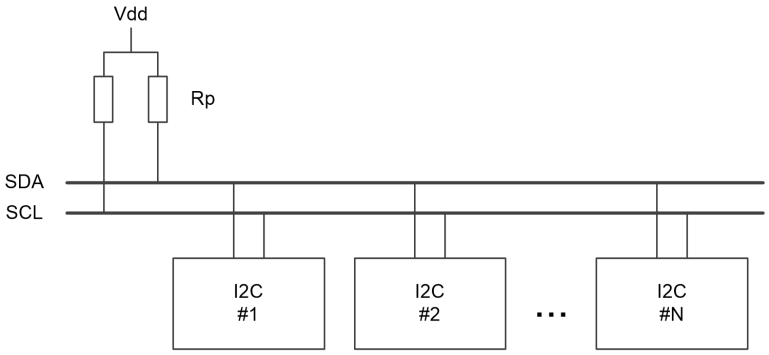


Figure 1.1. I2C bus topology

Each Slave device has a unique address, which is used by a Master device to connect to it. The bus topology allows multiple Master and Slave devices to be attached simultaneously. The collision detection mechanism allows arbitrage of access to the bus. The Master device initiates the transaction (Figure 1.2) by sending the start bit (S), the address packet (ADDRESS), and the transmission direction bit (R/W). Then the data packets (DATA) are sent or received, and the reception of each of them must be acknowledged (ACK) or rejected (NACK) by the receiver. The transmission terminates the stop bit sent by the Master. The SCL signal is generated by the Master, but the Slave can hold down the SCL line.

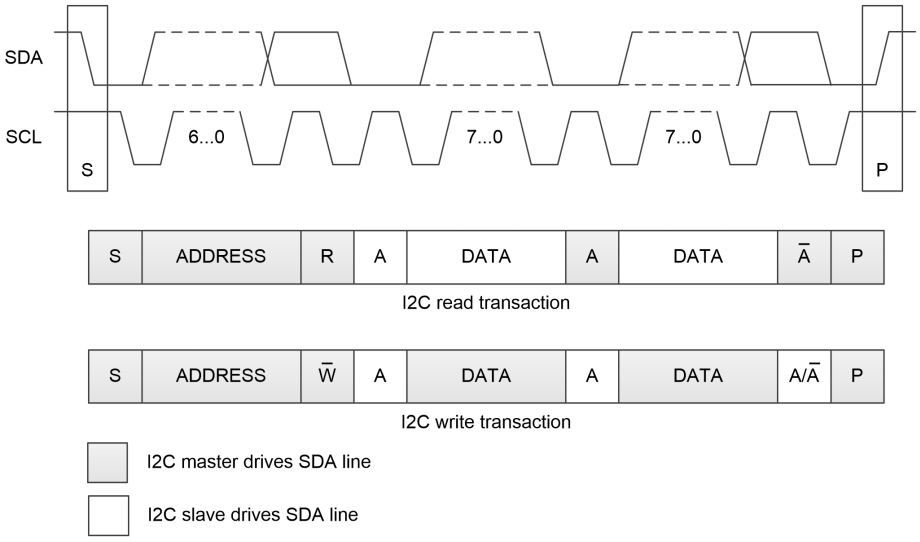


Figure 1.2. Sample transmission on the I2C bus.

1.3 Block Diagram

The I2C controller enables two-way communication in the Master mode via two lines. Transmission and receiving paths are buffered, thus eliminating delays between subsequent frames. Program-level communication may be based on the interrupts provided by the module. The use of a DMA channel (along with automatic transmission control modes) enables further unloading of the processor. Figure 1.3 shows the block diagram of the I2C module. The basic components are:

- the host bus interface, DMA and interrupt lines,
- configuration and status registers,
- I2C Master controller.

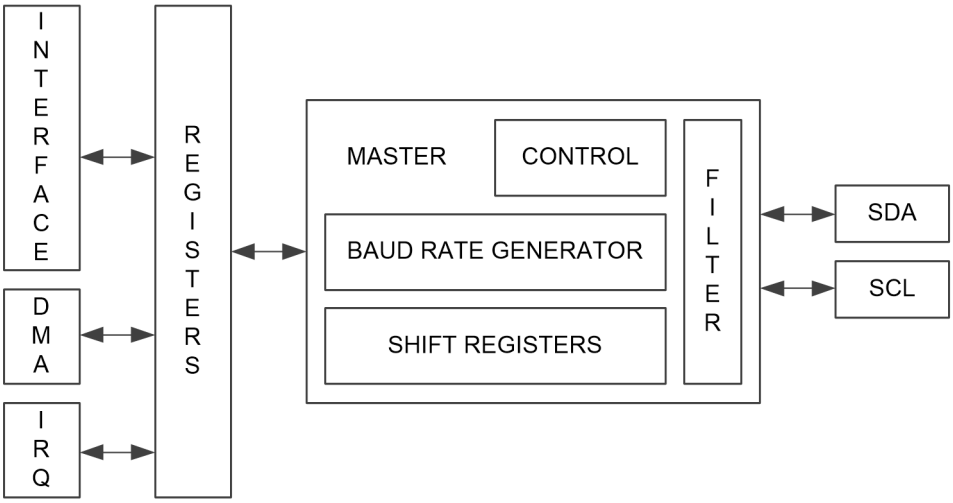


Figure 1.3. Block diagram of the I2C controller

1.4 Data Transmission

1.4.1 START/STOP Bits

The data transmission begins with the start bit (S) and ends with the stop bit (P) (Figure 1.4). The MASTER sends a start bit (S) by changing the state of the SDA line from high to low while the SCL line is high. The stop bit (P) signals the change of the SDA line level from low to high while the SCL line is high. During data transmission it is possible to send several start bits. This allows the MASTER device to address several Slave devices (or change the direction of the transmission) without releasing the bus. The start bit without the stop bit is called the repeat start bit (Sr).

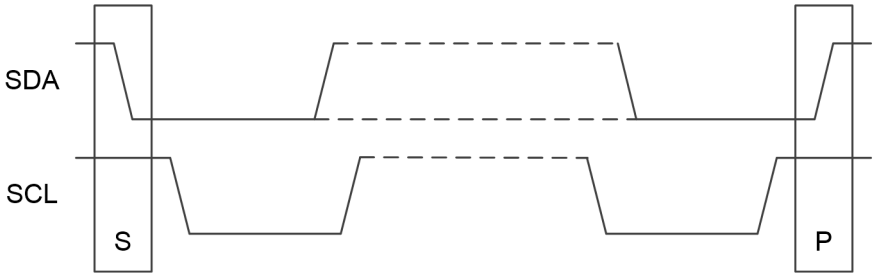


Figure 1.4. Start (S) and Stop (P) bits.

1.4.2 Data Bits

As shown in Figure 1.5, the signal level on the SDA line must be stable while the SCL line is high. Consequently, a change in the SDA line level may occur only when the SCL line is low. Data bits forms packets (data and address packets). Each packet is made up of 8 data bits (or address bits) transferred in order from the oldest bit. After each byte, the acknowledge bit is transmitted - ACK (when SDA = 0) or NACK (when SDA = 1).

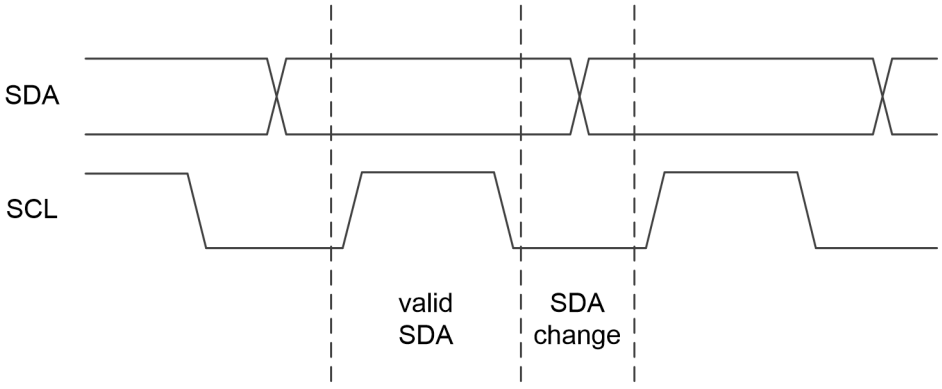


Figure 1.5. Data bit.

1.4.3 Address Phase

After the start bit transmission (S), the Master sends an address (7 or 10 bit) with a write/read (R/W) bit defining the direction of the transmission. A Slave device that recognizes its address confirms this by setting the SDA line to low state during the transmission of the ACK / NACK bit. Other Slave devices should keep the SDA line high for the duration of the transaction and wait for the next start bit (S). The seven- or ten-bit address along with the ACK/NACK bits represent the address packet. After each start bit, only one address packet can be sent. The R/W bit specifies the direction of the transaction. When it is set to 0, the MASTER will send data to the Slave device (write mode). A value of 1 means that the Slave should send data to the MASTER device immediately after acknowledging receiving its address.

1.4.4 Data Phase

Data packets consist of 9 bits: 8 bits of data (1 byte) and acknowledgment bit (the ACK/NACK bit). The directional bit (the R/W bit) contained in the address packet defines which device (Master or Slave) will control the SDA line while sending the data byte and which will send the acknowledge bit.

1.4.5 Serial Clock Signal

The timing parameters of the generated SCL signal (and ultimately also the SDA signal) may be precisely defined by the CWGR register (Clock Waveform Generator Register). Time parameters are defined as a function of the frequency F_P determined by the formula:

$$F_P[Hz] = \frac{F_{PCLK}[Hz]}{(PRESCALER + 1)}$$

The PRESCALER value is configurable through the PRESCALER (1.7.5) register. It is possible to configure the following times (Figure 1.6):

- $t_{STA/STO}$ - time between rising / falling edges of SCL and SDA signals when generating S, P and Sr bits,
- t_{LOW} - duration of low signal level on the SCL line,
- t_{HIGH} - duration of high signal level on the SCL line,
- $t_{SETUP/HOLD}$ - time ahead of the rising edge of the SCL line/time following the slope falling on the SCL line (t_{SETUP} follows the state of the SDA line, t_{HOLD} must precede this change).

Nominally, the duration of the SCL line in low state is $t_{LOW} + 2 * t_{SETUP/HOLD}$. Each Master or Slave device participating in data transfer can prolong this time by holding down the SCL line in a low state. The module monitors the state of the SCL line and starts the t_{HIGH} counter at the SCL high line state detection (not when the driver releases the line). Counters for the low state of the SCL line ($t_{SETUP/HOLD}$) are started when the SCL line is triggered (common collector/drain configuration will force the SCL line into the low state independently of other devices). In addition, to take into account the external electrical parameters, the timer t_{LOW} turns on when the SCL monitor confirms that the line actually went low. The SCL signal takes two clock cycles (input synchronizer) to reach monitor input and the internal state machine takes another two clock cycle to start appriopriate timer. This time can be further prolonged by the number of activated input glitch filter stages.

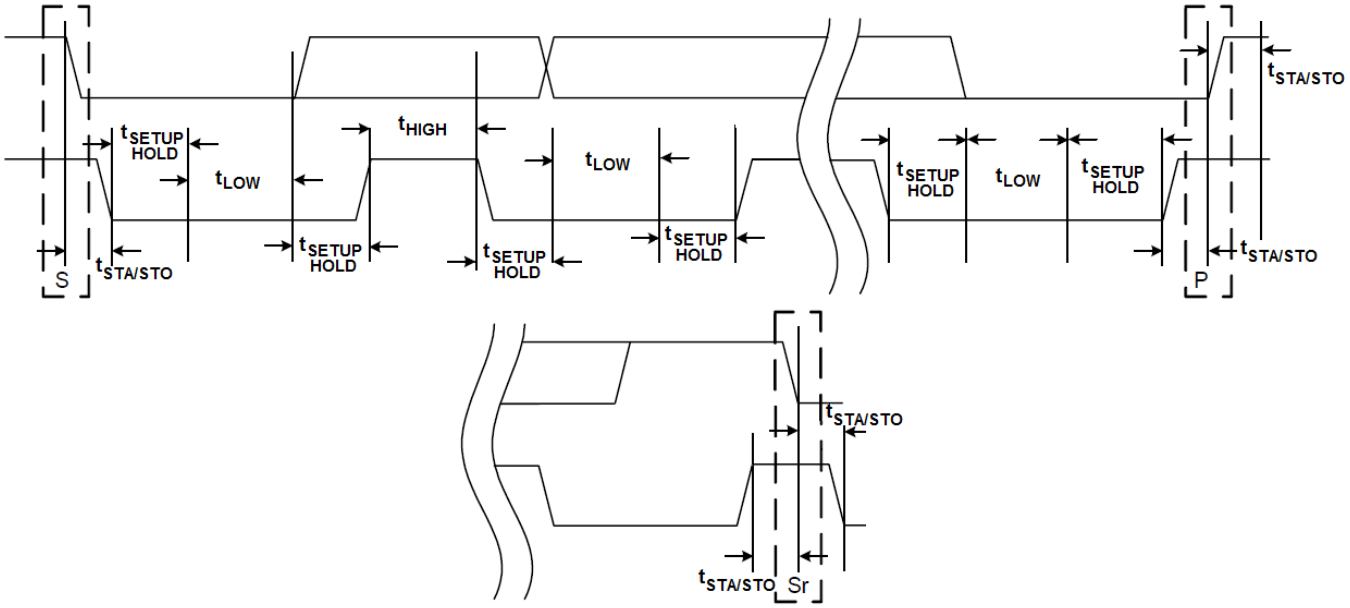


Figure 1.6. Parameters of the signal generated on the SCL and SDA lines.

1.4.6 Bus State Graph

When enabled, the module continually monitors the I2C interface lines. The I2C controller detects the START and STOP bits and collisions and determines the current state of the I2C bus (Figure 1.7). The software can check the current bus status by reading the STATUS register. The bus can be in one of four states:

- UNKNOWN - bus status is uknown,
- IDLE - no transmission on the bus,
- BUSY - on the bus is running data transmission initiated by another Master device,
- OWNER - the module transmites data.

After the reset, the bus status is unknown (UNKNOWN). It can be changed by the software (by writing a new state to the STATUS register). If the software does not write a new value, the state will automatically be changed to IDLE upon detection of the stop bit (P), or BUSY after the start bit (S) is detected. Only resetting or deactivating the module can re-enter the bus state into an unknown state (UNKNOWN). Transmission can only be started in the IDLE state. Detecting a start bit generated by another Master device causes passing into the BUSY state. The controller will remain in this state until the stop bit is detected (transmission initiated by the software will be started only after the BUSY has been removed). If the start bit was generated by the I2C controller, the bus enters the OWNER state and stays there until the transmission stops and the stop bit (P) is generated. The bus state will change back to IDLE again. Detection of a collision during transmission is a sign of arbitration loss over the bus and results in the BUSY state.

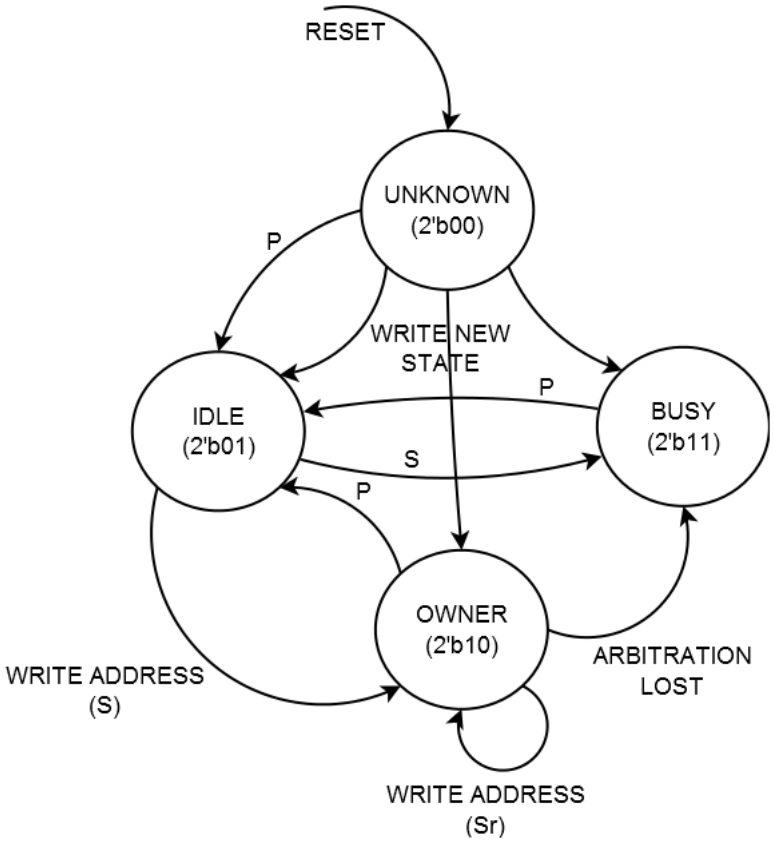


Figure 1.7. Bus status graph.

1.4.7 Commands

The module provides three commands that allow the software to influence the transmission progress and the internal state of the module. Issue of commands is to write their code in COMMAND (1.7.4) register. Apart from the RESET command that is executed immediately, it may happen that the module can not immediately execute commands because of its internal state. Then the command will be paused and then executed as soon as possible. Suspending a command is signaled by reporting its code in the STATUS register.

RESET The RESET command allows to restore the initial values in the module configuration registers (as well as the initial state of its state machine). The command is executed immediately, regardless of the transmission stage, so it can trigger behaviour of the SDA and SCL lines not according to the I2C protocol. When the RESET command is executed, the bus state is unknown (UNKNOWN).

ACK The ACK command has a different meaning, depending on the configuration of the module and its current state (Figure 1.10, Figure 1.14). At the address sending stage and when the module is in data send mode (data writing) to the Slave, the ACK command is used to resume operation of the module (continuation of the operation) upon detection of unexpected situation (such as unexpected reception of the NACK bit). In read mode from the Slave device, the ACK command is used when the AutoACK = 0 bit and is used to send an ACK bit (also configured in the COMMAND register) to the Slave device after receiving each byte of data.

STOP The STOP command is used to immediately (within the I2C protocol) terminate the transmission by sending the stop bit. In the mode of sending (writing) data to the Slave, the STOP command will send the STOP bit at the first possible (from the point of view of the I2C protocol) opportunity. Even if the TDR register contains current data, it will be ignored. In data read mode from the Slave, the STOP command will send the LAST_ACK bit (configurable via the COMMAND (1.7.4) register and the stop bit and then terminate the transmission.

1.4.8 Data Frame Transmission

Figure 1.8 shows the frame transmitted on the I2C bus in write mode. The transaction starts with the start bit, after which the address packet is sent (along with the directional bit equal to 0 - write). After acknowledging the address byte received by the Slave device (ACK = 0), the Master sends another data byte. Each of them must be acknowledged by the Slave (ACK = 0) except for the last after which the Slave responds by the ACK = 1. The transaction ends the stop bit.

Figure 1.8. Data transmission in write mode.

Transmission is analogous in 10-bit addressing mode (1.9). In this case, the address packet consists of two bytes. The first one transmits 2 bits of the configured address, while the next one is the other 8 bits.

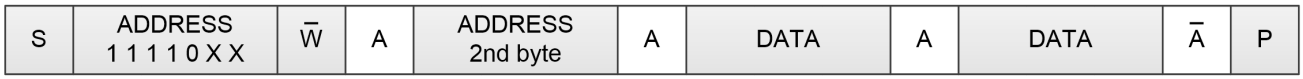


Figure 1.9. Data transmission in write mode (10-bit addressing mode).

Figure 1.10 shows a status diagram for the write operation, including possible module configuration variants, and Figure 1.11 shows how the transmission status is reported when sending a single byte.

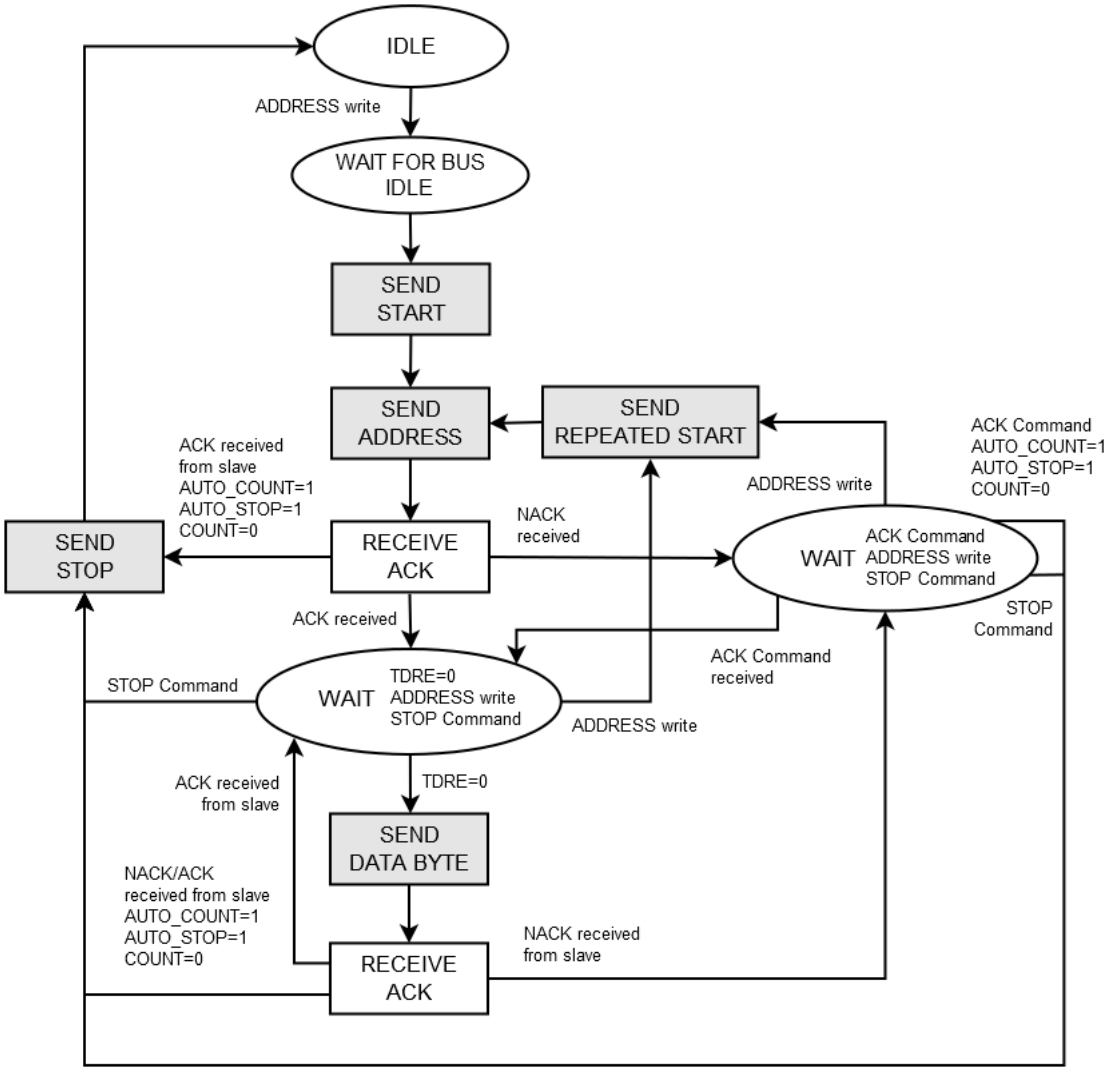


Figure 1.10. Data transmission algorithm in write mode.

Transfer configuration Before transmitting, the I2C controller must be configured.

- Run the module by activating the CTRL (1.7.3) register. Activate selected transmission automation modes and addressing mode (10 or 7 bit).
- The run-time parameters are configured using the PRESCALER (1.7.5) and the CWGR (1.7.6) registers.
- The interrupt mask register (1.7.11) must be activated to activate the selected interrupt source.
- If the STATUS (1.7.2) register signals the UNKNOWN bus state, change it to the IDLE by writing 2'b01.
- If automatic counting of sent bytes (bit AUTO_COUNT in the CTRL register) is activated, program the number of bytes sent in the COUNT register (1.7.7).

Address sending The software initiates the transaction by writing the Slave device address in the ADDRESS registry (1.7.8). The module then waits for the bus to be in the IDLE state. When this state is reached, the start bit (Figure 1.11) is transmitted, followed by the address packet transmission. After sending the address, the controller waits for acknowledgment (ACK bit) from the Slave. The value of the received ACK bit is reported in the STATUS register (Figure 1.11). If the ACK bit is received, the module expects data to be written to the TDR register. If the NACK bit is received, the controller waits for the decision to continue the transmission (via the ACK command), terminate the transmission (via the STOP command) or resend the address (by writing the address to the ADDRESS register) - Figure 1.10.

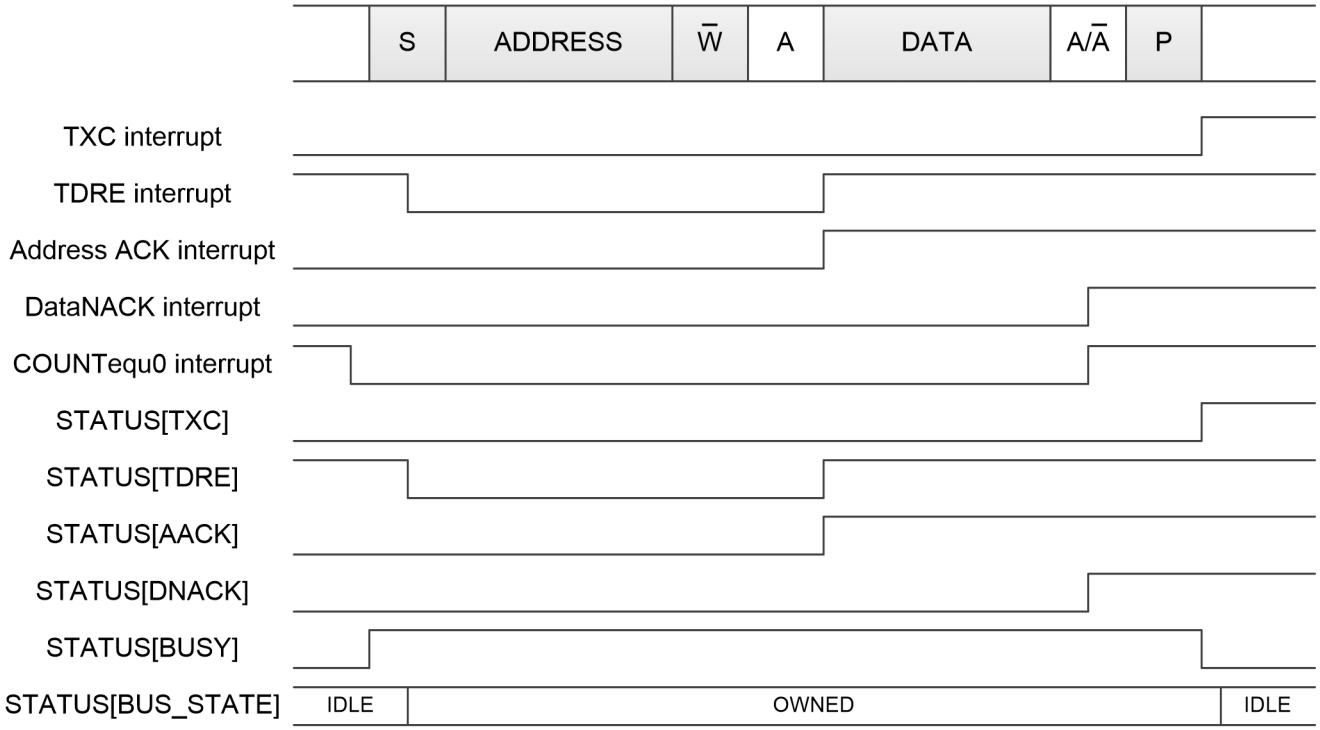


Figure 1.11. Transmission of a single byte of data (AUTO_STOP = 1, AUTO_COUNT = 1).

With 10-bit addressing mode after sending the first byte of address, the module automatically handles the ACK bit received from the Slave (and does not report it in any way). If no Slave has acknowledged (ACK = 1) receiving of the address, the frame transmission will be interrupted and the module will signal the reception of the NACK bit in response to the address packet.

Data sending The module expects to write successive bytes to be sent by holding down the SCL line in a low state so that no other Master module can access the bus. Once data is saved to the TDR register (Figure 1.11), they are transmitted immediately when the address is sent. At the same time, the TDR register is emptied so that another data byte can be written. The module then waits for the ACK bit from the Slave device. The received ACK/NACK bit is reported in the STATUS register (Figure 1.11). If the ACK bit (0) is received, the controller increments (or decrements) the COUNT register and then waits for the next data byte to be written to the TDR register (1.7.9). If the NACK bit is received (1), the module will expect the software to:

- terminate the transmission and send the STOP bit (STOP command),
- continuation of the transmission (ACK command),
- sending the repetitive start bit and an address frame (write to ADDRESS register - 1.7.8).

Exception from the above is a transmission with automatic byte counting and stop bit (AUTO_COUNT = 1 and AUTO_STOP = 1). According to the I2C protocol, the NACK bit is always transmitted after the last data byte, so in this case the NACK will not pause the transmission but send the stop bit (Figure 1.11) and terminate the transaction signaled by the TXC bit set in the STATUS register.

Transmission ending After sending the programmed number of bytes, the I2C controller can automatically send the stop bit (Figure 1.11) and thus terminate the transmission if automatic counting and termination is enabled (by AUTO_COUNT and AUTO_STOP bits in CTRL register). Transmission automatics allows the module to be used in conjunction with the DMA controller with minimal software intervention. The software can also terminate the transmission at any time by sending the STOP command. If the stop command was issued during the transmission, it will be delayed until the data/address has been sent and the ACK/NACK has been terminated.

Short frame transmission The module allows the transmission of short frames consisting only of start bit, address and stop bit. This is possible both in automatic mode (AUTO_COUNT = 1, AUTO_STOP = 1, COUNT = 0) and without using it (by sending the STOP command directly after address transmission).

Repeated start transmission The I2C controller enables sending of combined packets (when the repeated start bit and the address frame are transmitted instead of the stop bit). Sending a repeated start bit is enforced by writing to the ADDRESS (1.7.8) register during data transmission. The module detects that the bus is in the OWNED state and then transmits the repeated start bit (Sr) and the new address. If auto-count of sent bytes is active, it will continue (the COUNT register will be decremented from the current value). Otherwise, after the successful transmission of the address, the COUNT register will be reset and the count of transmitted bytes will start from 0.

1.4.9 Data Frame Reception

Figure 1.12 represents a frame that is transmitted on the I2C bus in read mode. The transaction starts with the start bit followed by an address packet (including the directional bit equal to 1 - read mode). After acknowledgment of the address packet received by the Slave device (ACK = 0), the Master waits for the next data bytes from the Slave and sends the acknowledgment during the ACK bit. The transaction ends with the stop bit.

Figure 1.12. Data transmission in read mode

In 10-bit addressing mode (Figure 1.13), after sending the first and second byte addresses, the module automatically handles the ACK bit received from the Slave. If no Slave has acknowledged (ACK = 1) that the address is received, the frame transmission will be interrupted and the module will signal the reception of the NACK bit in response to the address packet.

Figure 1.13. Read mode transmission (10-bit addressing mode).

Figure 1.14 represents a status diagram for read operations, while Figure 1.15 shows how to report a transmission status when receiving a single byte.

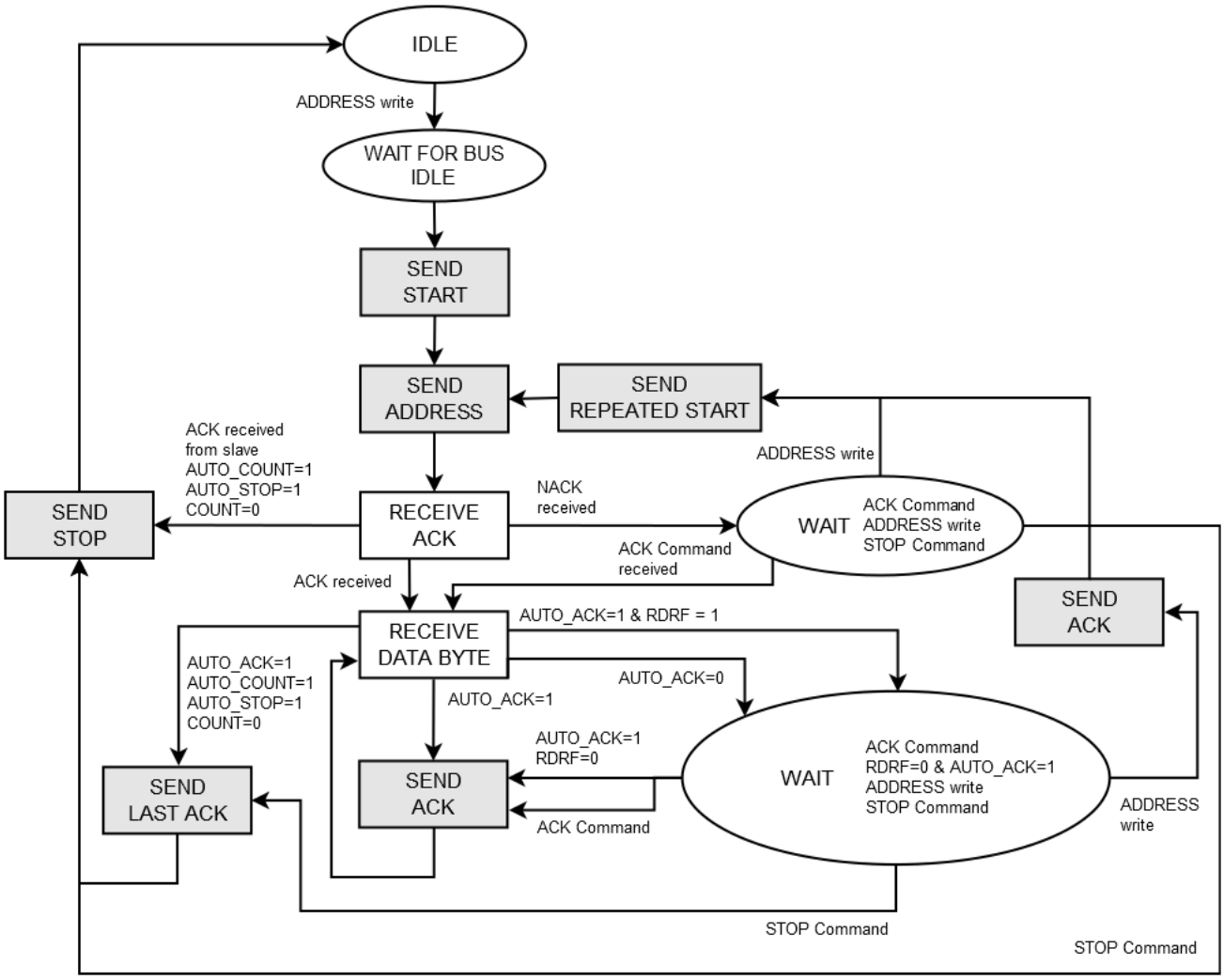


Figure 1.14. Data transmission algorythm in read mode.

Transfer configuration Before starting a transmission the module needs to be configured.

- Run the module by activating CTRL (1.7.3) register. Activate selected transmission automation modes and addressing mode (10 or 7 bit).
- The run-time parameters are configured using the PRESCALER (1.7.5) and the CWGR (1.7.6) registers.
- The interrupt mask register (1.7.11) must be activated for the selected interrupt.
- If STATUS (1.7.2) register signals the UNKNOWN bus state, change it to the IDLE by writing 2'b01.
- If automatic counting of sent bytes (bit AUTO_COUNT in the CTRL register) is activated, program the number of bytes sent in the COUNT register (1.7.7).
- If the ACK auto-send mode (AUTO_ACK = 1) is active, by the COMMAND (1.7.4) register configure the ACK bit that will be sent after each received byte. In addition, in the automatic byte counting and stop bit sending (AUTO_COUNT = 1, AUTO_STOP = 1) mode, configure the LAST_ACK bit that will be sent last - just before the stop bit is sent.

Address sending The software initiates the transaction by writing the Slave address in the ADDRESS (1.7.8) register. The module then waits for the bus to be in the IDLE state. When this state is reached, the start bit is transmitted (Figure 1.15) followed by the address packet transmission. After sending the address, the module waits for acknowledgment (ACK bit) from the Slave. The value of the received ACK bit is reported in the STATUS register (Figure 1.15). If the ACK bit is received, the module automatically starts receiving the first byte of the data. If the NACK bit is received, the module waits for the decision to continue the transmission (via the ACK command), terminate the transmission (via the STOP command), or resend the address (by writing the address to the ADDRESS register).

Figure 1.15. Receive a single byte of data

In 10-bit addressing mode after sending the first byte of address, the module automatically handles the ACK bit received from the Slave (and does not report it in any way). If no Slave has acknowledged (ACK = 1) that the address is received, the frame transmission will be paused and the module will signal the reception of the NACK bit.

Receiving data As soon as the Slave acknowledges receipt of the address, the module starts receiving the first byte of the data. The reception is signaled by setting the RDRF bit in the STATUS register (Figure 1.15). The next actions of the module depend on the configuration of the automatic data transmission modes:

AUTO_ACK=1, AUTO_COUNT=1, AUTO_STOP=1 If this is not the last received byte, the module sends the ACK bit configured previously through the COMMAND (1.7.4) register, and then waits for reading data from the RDR (1.7.10) register while holding the SCL line in low state to prevent other Master devices from taking control of the bus. When the RDR register is read, the module receives another byte of data. If the last byte (COUNT = 0) was received, the module sends a LAST_ACK bit previously set in the COMMAND register followed by the stop bit (Figure 1.15), and signals the end of the transmission by setting the TXC bit in the STATUS register.

AUTO_ACK=1 The module sends the ACK bit configured in the COMMAND register, and then waits for the RDR data to read, keeping the SCL line low. When the RDR register is read, the module receives another byte of data.

AUTO_ACK=0 Module expects to intervene software for further operations. The intervention is to send a command (ACK or STOP) or to write into the ADDRESS register.

Transmission ending Upon the reception of the programmed byte count, the module can automatically send the stop bit (1.15) and terminate the transmission, provided that automatic counting, acknowledgment, and termination (AUTO_COUNT, AUTO_ACK, AUTO_STOP in the CTRL register) are activated. Transmission automatics allows the module to be used in conjunction with a DMA controller with minimal software intervention (only in case of special problems such as bus collisions or addressing of the Slave device). The software can also terminate the transmission at any time by sending the STOP command. If the stop command was issued during the transmission, execution of the Stop command will be delayed until final receiving of the data. As with automatic transmission, the LastACK bit is sent with the stop bit configured in the COMMAND register.

Short frame transmission The module allows the transmission of short frames consisting only of the start bit, address, and stop bit in the Slave read mode. However, this is only possible in the automatic mode (AUTO_COUNT = 1, AUTO_STOP = 1, COUNT = 0).

Repeated start transmission The module enables consecutive sending of packets (when the repeated start bit and the address frame are transmitted instead of the stop bit) also in the receive mode. Sending the repeated start bit is enforced by writing the ADDRESS (1.7.8) register during frame transmission. The module detects that the bus is in the OWNED state and then transmits the Sr (repeated start) bit and the new address. If automatic count of sent bytes is active, it will continue sending and the COUNT register will be decremented from the current value. Otherwise, after the successful transmission of the address, the COUNT register will be reset and the count of transmitted bytes will start from 0.

1.5 Interrupts

The I2C controller can report an interrupt in response to a selection of 9 configurable events.

1.5.1 TXC Interrupt

The interrupt line is set high when the STOP bit transmission is complete. The line is set to low when the STATUS register is read (1.7.2).

1.5.2 TDRE Interrupt

The interrupt line is set high when the data from the TDR register is transferred to the shift register of the transmitter. The line is set low when data is written to the TDR register (1.7.9).

1.5.3 RDRF Interrupt

The interrupt line is set high when the selected data is written to the RDR register. The line is set low when the RDR register is read (1.7.10).

1.5.4 ARBITRATION LOST Interrupt

The interrupt line is set high at the time of collision detection on the I2C bus. The line is set low when the STATUS register is read.

1.5.5 ADDRESS NACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 1 (NACK) in response to the address packet. The line is set low when the STATUS register is read.

1.5.6 ADDRESS ACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 0 (ACK) in response to the address packet. The line is set low when the STATUS register is read.

1.5.7 DATA NACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 1 (NACK) in response to the data packet (in write mode only). The line is set low when the STATUS register is read.

1.5.8 DATA ACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 0 (ACK) in response to the data packet (in write mode only). The line is set low when the STATUS register is read.

1.5.9 COUNT EQUALS 0 Interrupt

The interrupt line is set high when the COUNT register reaches 0 (only if AUTO_COUNT = 1). The line is set low when the STATUS register is read.

1.6 Input Filter

Both SDA and SCL inputs are equipped with input glitch filters. Its construction is shown in Figure 1.16. The input filter is composed of a flip-flop chain and the function that determines if the output value should be updated or should remain unchanged. Signal that leaves the input synchronizer must remain stable for FLTVAL clock cycles to be recognized as valid.

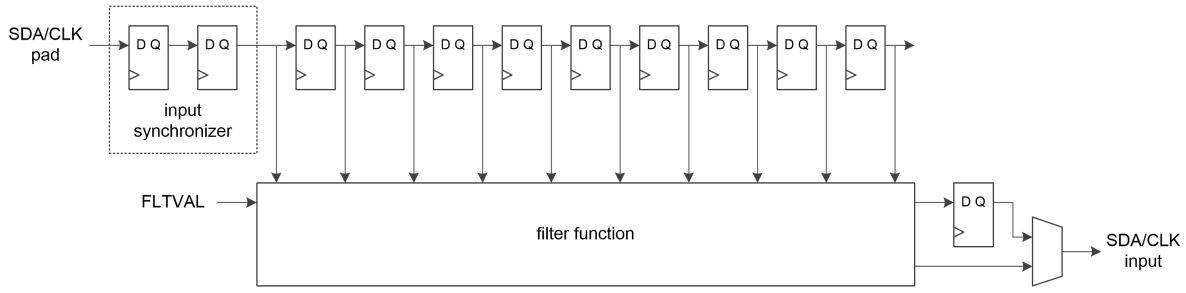


Figure 1.16. Block diagram of SDA/SCL input filter

Figure 1.17 presents the exemplary filter wafeform for FLTVAL = 5. It is shown that the actual width of filtered glitches depend on their occurrence in time related to the input synchronizer clock rising edge. Synchronized signal that is stable five and more clock cycles will be passed to the output otherwise it will be suppressed.

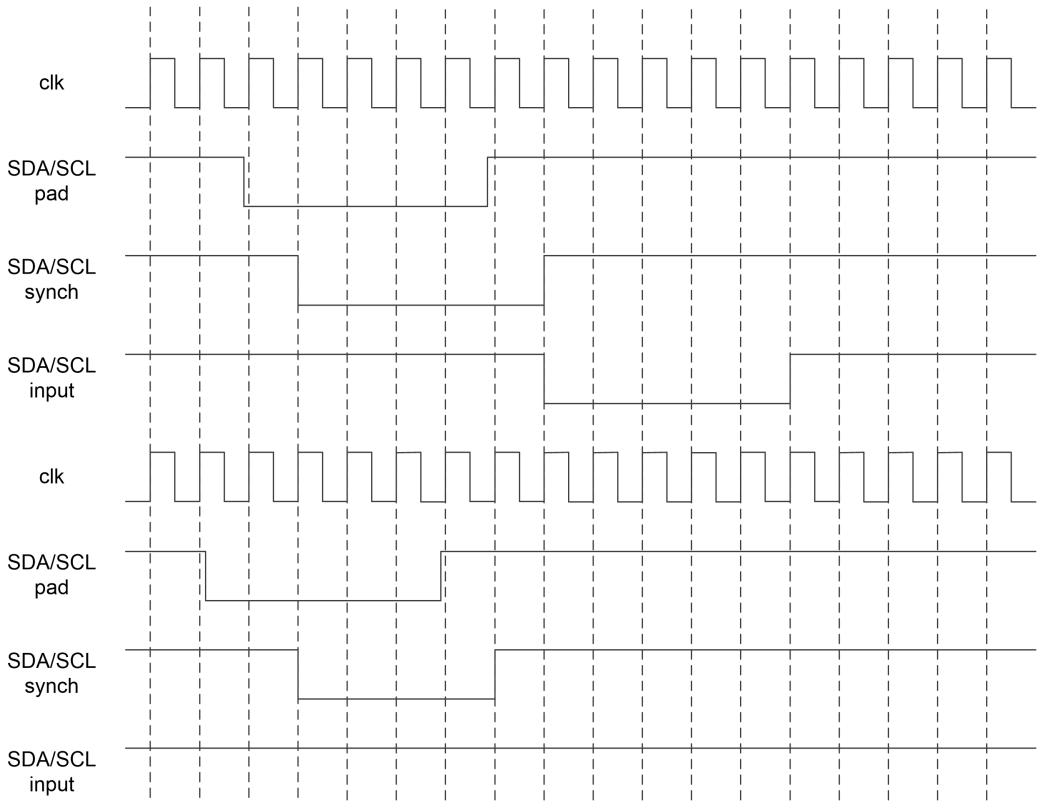


Figure 1.17. Input filter waveform for FLTVAL = 5.

1.7 Configuration Registers

1.7.1 Registers List

Address Offset	Register	Name
0x00	STATUS	Status Register
0x04	CTRL	Control Register
0x08	CMD	Command Register
0x0C	PRES	Prescaler Register
0x10	CWGR	Clock Waveform Generator Register
0x14	COUNT	Count Register
0x18	ADDR	Address Register
0x1C	TDR	Transmit Data Register
0x20	RDR	Receive Data Register
0x24	IRQM	Interrupt Mask Register
0x28	IRQMAP	Interrupt Mapping Register
0x2C	FILTER	Filter Register

1.7.2 Status Register

Address: 0x00

31	30	...	...	...	...	17	16
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8
	DNACK	ANACK	DACK	AACK	ACK	CURRENT_CMD[1:0]	
R	R	R	R	R	R	R	
0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0
BUS_HOLD	ARB_LOST	BUSY	RDRF	TDRE	TXC	BUS_STATE[1:0]	
R	R	R	R	R	R	R/W	
0	0	0	0	1	0	0	

BUS_STATE[1:0] *Current bus status*

When BUS_STATE is UNKNOWN, it can be changed by writing a new value to the STATUS register.

BUS_STATE[1:0]	Bus status
00	UNKNOWN
01	IDLE
10	OWNED
11	BUSY

TXC *Transmission Complete*

The TXC bit is set to “1” when the STOP bit transmission ends. The bit is set to “0” when the STATUS register is read.

TDRE *Transmit Data Register Empty*

0 The TDR register contains data haven’t been send yet.

1 The register is empty.

The TDRE bit is set to “1” when the TDR register data is transferred to the transmitter shift register and is set to “0” when data is written to the TDR register.

RDRF *Receive Data Register Full*

The RDRF bit is set to “1” at the end of frame transmission and loading into the RDR register. The register is set to “0” when data is read from the RDR register.

BUSY *Transmission in progress*

0 The module does not perform any operations.

1 The module transmits data.

ARB_LOST *Bus Arbitration Lost*

The value “1” means that the module detected a collision on the I2C bus. The ARB_LOST bit is set to “0” when the STATUS register is read.

BUS_HOLD *Holding the BUS*

The value “1” means that the module is waiting for data or command and the SCL line is kept in low state.

CURRENT_CMD[1:0] *Current Command*

Pending command code. After executing the command, the bit field will read as 0 (NO COMMAND):

CURRENT_CMD[1:0]	Executed command
00	NO COMMAND
01	ACK
10	STOP
11	RESET

ACK *Received ACK bit*

Last received transmission acknowledgment bit (0 - ACK, 1 - NACK). The bit has the current value when the address or data transmission is complete.

AACK *Address ACK-ed by Slave*

The value “1” means that the Slave device has acknowledged receiving the address by the ACK bit. The AACK bit is set to “0” when the STATUS register is read.

DACK *Data ACK-ed by Slave*

The value “1” means that the Slave device has acknowledged receiving the data by the ACK bit (the bit value is valid only during transmission in write mode). The DACK bit is set to “0” when the STATUS register is read.

ANACK *Address NACK-ed by Slave*

The value “1” means that the Slave device has acknowledged receiving the address by the NACK bit. The ANACK bit is set to “0” when the STATUS register is read.

DNACK *Receive Data Register Full*

The value “1” means that the Slave device has acknowledged receiving the data by the NACK bit (the bit value is valid only during transmission in write mode). The DNACK bit is set to “0” when the STATUS register is read.

1.7.3 Control Register

Address: 0x04

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
			AUTO_STOP	AUTO_ACK	AUTO_CNT	10BIT_ADDR	ENABLE
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

ENABLE *I2C Enable*

0 The I2C module is disabled.

1 The I2C module is enabled.

10BIT_ADDR *10bit Addressing Mode Enable*

0 7-bit addressing mode.

1 10-bit addressing mode.

AUTO_CNT *Automatic Byte Counting Enable*

0 Automatic data byte countdown is disabled. In this mode, the address packet transmission resets the COUNT register, which is then incremented after each data byte transmission. The COUNT register has an informational function and does not affect automatic transmission control or the generation of interruptions.

1 Automatic data byte countdown is enabled. In this mode, the transmission of each byte of data causes a decrease in the value previously stored in the COUNT register. When the COUNT register reaches 0, an assigned interrupt is generated. The COUNT register value is also used for automatic transmission control:

- In the write mode and when the AUTO_STOP bit is set to “1” also, after receiving the acknowledgment for the last byte (COUNT = “0”), the stop bit is automatically sent.

- In the read mode and when the both AUTO_ACK and AUTO_STOP bits are set to “1” also, after receiving the last byte (COUNT = “0”), the LAST_ACK bit (1.7.4) and the stop bit are automatically sent.

AUTO_ACK *Automatic ACK bit Transmission Enable*

0 Automatic acknowledgement transmission bit is disabled.

1 Automatic acknowledgement transmission bit is enabled.

The AUTO_ACK bit is only valid in read mode. When it is equal to “1”, after each received byte the module will automatically send a acknowledgment bit equal to the ACK bit (1.7.4). In addition, when AUTO_NCT = “1” and AUTO_STOP = “1”, after the last received byte, the module will send an acknowledgment bit equal to LAST_ACK bit (1.7.4)

AUTO_STOP *Automatic STOP bit Transmission Enable*

0 Automatic stop bit transmission is disabled.

1 Automatic stop bit transmission is enabled.

Bit AutoStop obowiązuje jedynie, gdy: The AUTO_STOP bit is valid:

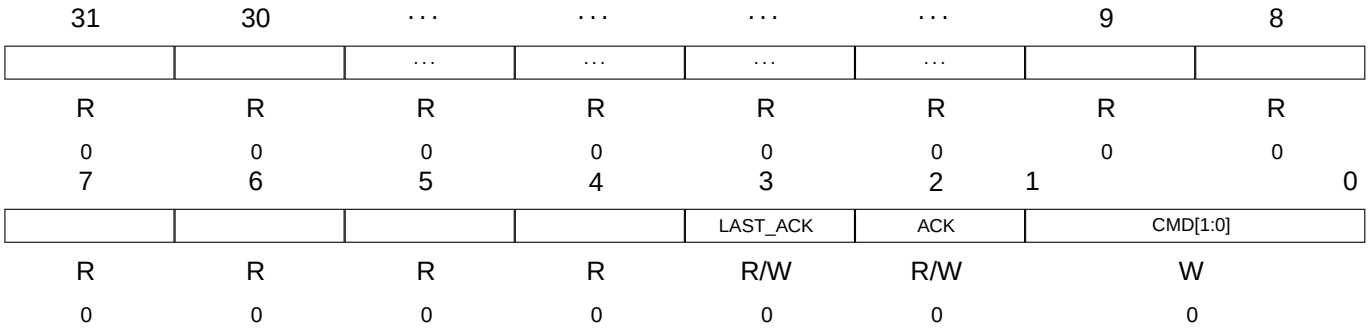
- in the write mode only when AUTO_CNT = 1,

- in the read mode only when AUTO_ACK = 1 and AUTO_CNT = 1.

When the above conditions are met, after sending/receiving the last byte, the module automatically (without software intervention) will send the stop bit and end the transmission.

1.7.4 Command Register

Address: 0x08



CMD[1:0] *Command*

The next command code for the module:

CMD[1:0]	Command code
00	NO COMMAND
01	ACK
10	STOP
11	RESET

ACK *ACK bit*

The value of the ACK bit sent after each received byte (or with the ACK command):

0 ACK.

1 NACK.

LAST_ACK *Last ACK bit*

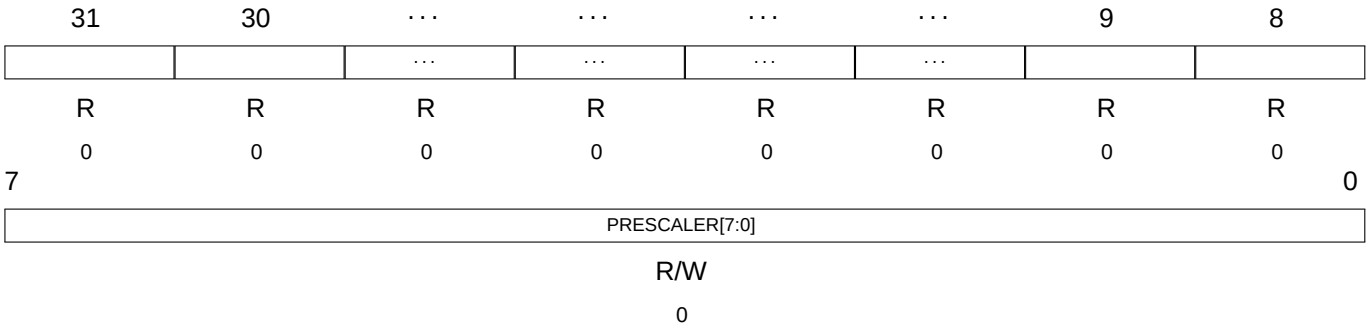
The value of the ACK bit sent after the last received byte (or with the STOP command):

0 ACK.

1 NACK.

1.7.5 Prescaler Register

Address: 0x0C



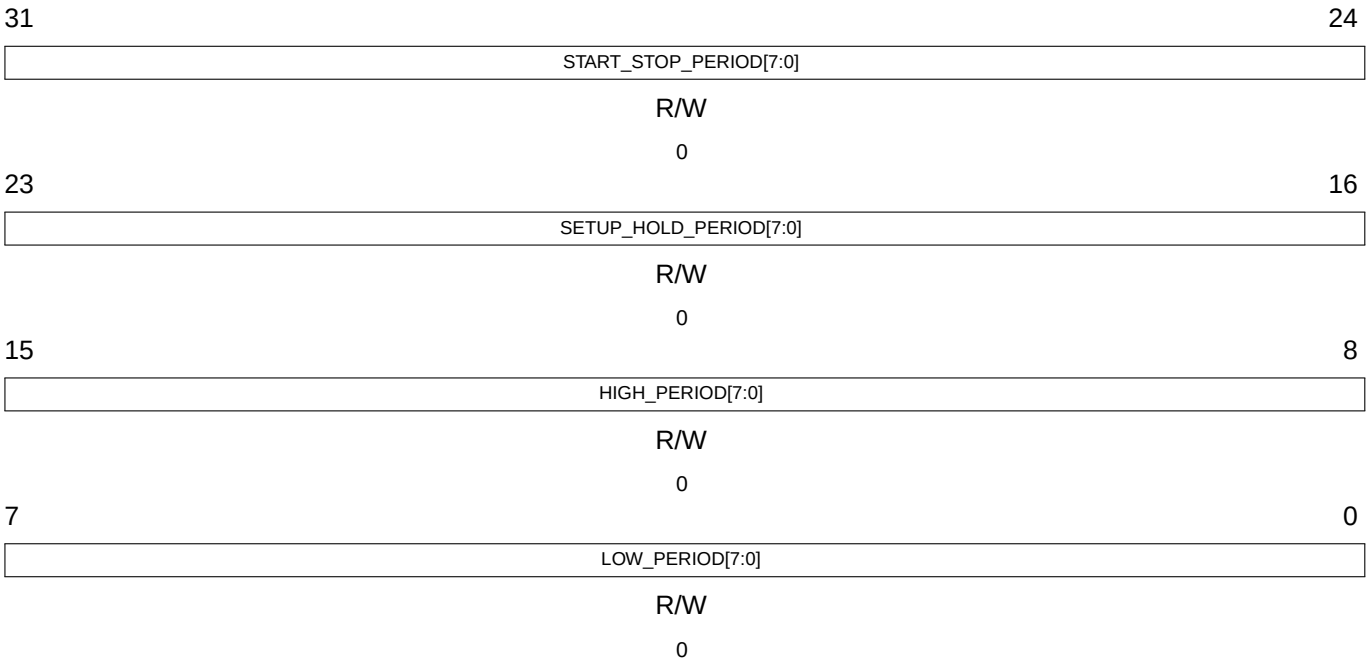
PRESCALER[7:0] *Prescaler value*

Prescaler for the time base of the SCL signal generator:

$$F_P[Hz] = \frac{F_{PCLK}[Hz]}{(PRESCALER + 1)}$$

1.7.6 Clock Waveform Generator Register

Address: 0x10



LOW_PERIOD[7:0] *Low Level SCL period*

Duration of the low level of the SCL signal:

$$t_{LOW}[s] = \frac{(LOW\_PERIOD + 1)}{F_{PRESCALER}[Hz]}$$

HIGH_PERIOD[7:0] *High Level SCL period*

Duration of the high level of the SCL signal:

$$t_{HIGH}[s] = \frac{(HIGH\_PERIOD + 1)}{F_{PRESCALER}[Hz]}$$

SETUP_HOLD_PERIOD[7:0] *Setup/Hold time*

The time preceding the rising edge on the SCL line / time following the falling edge on the SCL line (t_{SETUP} follows the state of the SDA line, t_{HOLD} must precede the change):

$$t_{SETUP/HOLD}[s] = \frac{(SETUP\_HOLD\_PERIOD + 1)}{F_{PRESCALER}[Hz]}$$

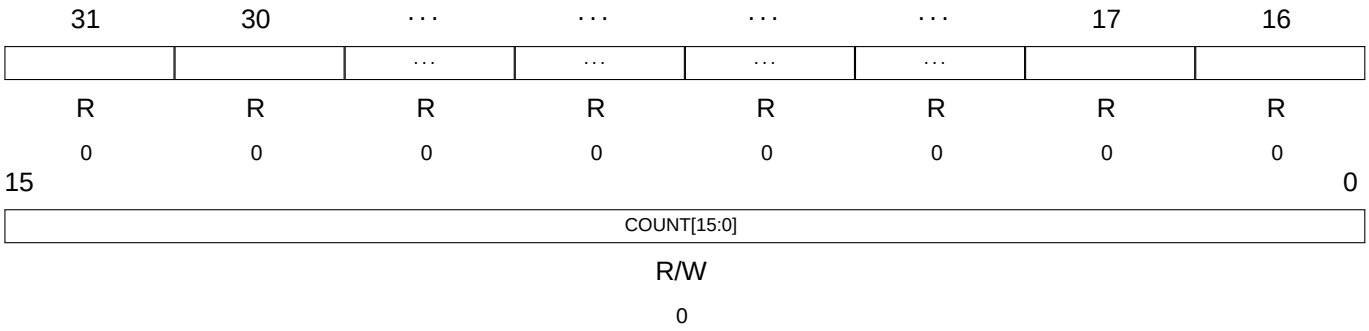
START_STOP_PERIOD[7:0] *Start/Stop time*

Time between rising / falling edges of SCL and SDA signals when generating start (S), stop (P), and repeated start (S_r) bits:

$$t_{START/STOP}[s] = \frac{(START\_STOP\_PERIOD + 1)}{F_{PRESCALER}[Hz]}$$

1.7.7 Count Register

Address: 0x14

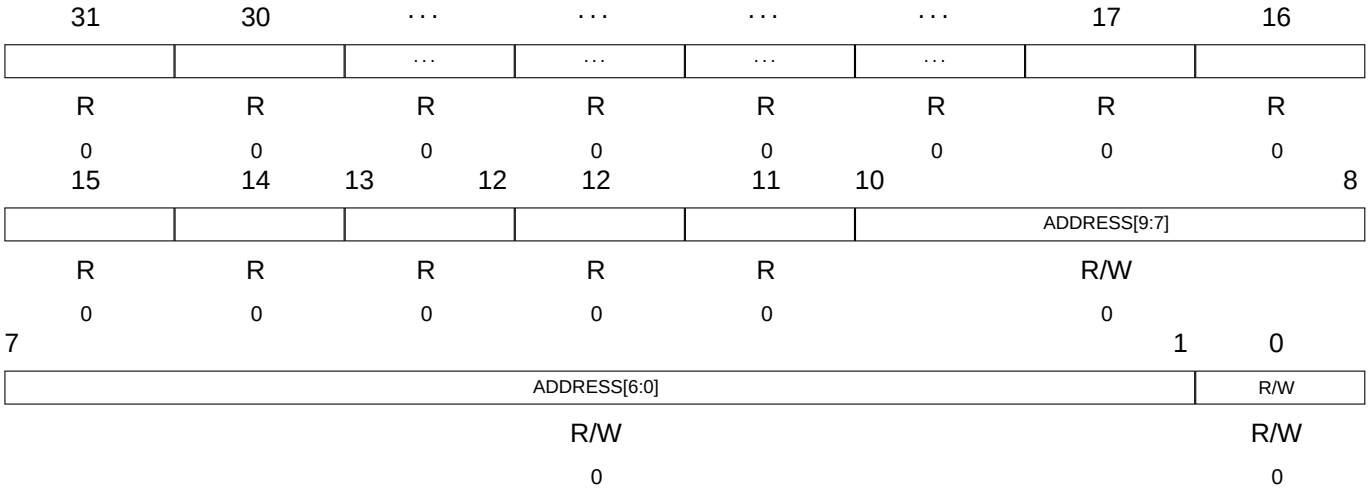


COUNT[15:0] *The counter of sent bytes via I2C interface*

In the automatic byte counting mode (the AUTO_CNT bit in the CTRL register equal to 1), this register must be programmed with a value equal to the number of bytes sent/received. With subsequent frames transmitted, the contents of the register is decrement until 0. With the automatic counting mode disabled, the COUNT register is cleared after the address frame has been successfully sent and then incremented after each data frame that has been sent / received.

1.7.8 Address Register

Address: 0x18



R/W *Transmission Direction*

Selecting the transmission direction (according to the I2C protocol):

0 Write.

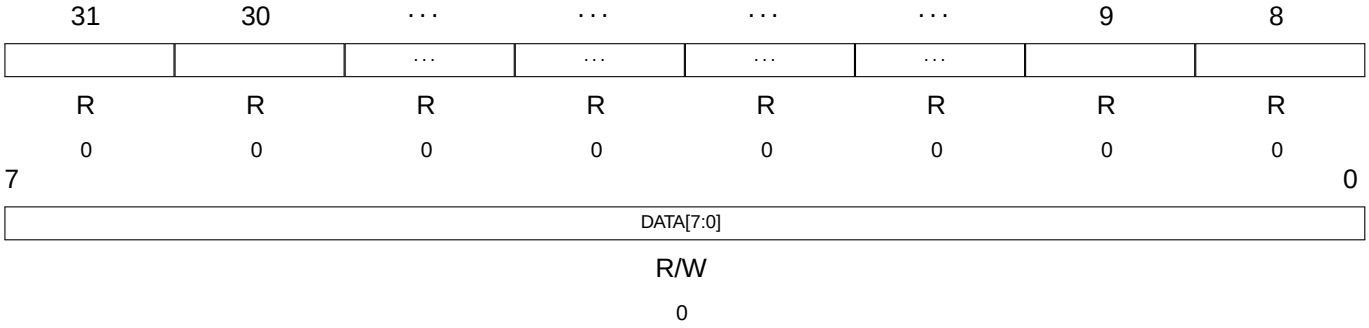
1 Read.

ADDRESS[9:0] *Slave Address*

Writing to the ADDRESS register causes a new transmission to start (sending a start bit or a repeated start bit).

1.7.9 Transmission Data Register

Address: 0x1C

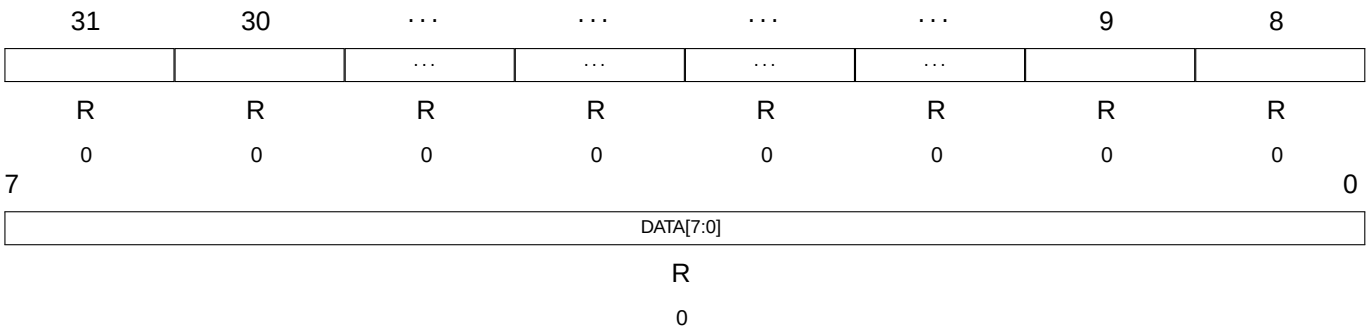


DATA[7:0] *Transmission Buffer*

When the TDRE flag from the STATUS (1.7.2) register is set to “0”, the TDR register contains data that has not yet been sent (any write will causes data be lost). If the address packet (in write mode) has already been sent and accepted by the Slave device, writing data into the TDR register will start the transmission of the data frame.

1.7.10 Reception Data Register

Address: 0x20



DATA[7:0] *Reception Buffer*

The data received from the Slave is written to the RDR buffer. The data in the register is valid if the RDRF flag of the STATUS register (1.7.2) is set to “ 1 ”. The module does not allow overflow of the RDR register by pausing the reception of consecutive bytes of data until the RDR register is read.

1.7.11 Interrupt Mask Register

Address: 0x24

31	30	...	...	...	...	9	8
		...	...	...	...		CNT0IE
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
DACKIE	DNACKIE	AACKIE	ANACKIE	ARBLOSTIE	RDRFIE	TDREIE	TXCIE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

TXCIE *Transmission Completed Interrupt Enable*

0 Interrupt after the transmission of the STOP bit is disabled.

1 Interrupt after the transmission of the STOP bit is enabled.

TDREIE *TX Data Register Empty Interrupt Enable*

0 Interrupt signaling lack of data in the TDR register is disabled.

1 Interrupt signaling lack of data in the TDR register is enabled.

RDRFIE *RX Data Register Full Interrupt Enable*

0 Interrupt after receiving a data byte is disabled.

1 Interrupt after receiving a data byte is enabled.

ARBLOSTIE *Arbitration Lost Interrupt Enable*

0 Interrupt signaling collision on the I2C bus is disabled.

1 Interrupt signaling collision on the I2C bus is enabled.

ANACKIE *Address NACK Received Interrupt Enable*

0 Interrupt signaling reception of the NACK bit in response to the address packet is disabled.

1 Interrupt signaling reception of the NACK bit in response to the address packet is enabled.

AACKIE *Address ACK Received Interrupt Enable*

0 Interrupt signaling reception of the ACK bit in response to the address packet is disabled.

1 Interrupt signaling reception of the ACK bit in response to the address packet is enabled.

DNACKIE *Data NACK Received Interrupt Enable*

0 Interrupt signaling reception of the NACK bit in response to the data packet is disabled (in the write mode only).

1 Interrupt signaling reception of the NACK bit in response to the data packet is enabled (in the write mode only).

DACKIE *Data ACK Received Interrupt Enable*

0 Interrupt signaling reception of the ACK bit in response to the data packet is disabled (in the write mode only).

1 Interrupt signaling reception of the ACK bit in response to the address data is enabled (in the write mode only).

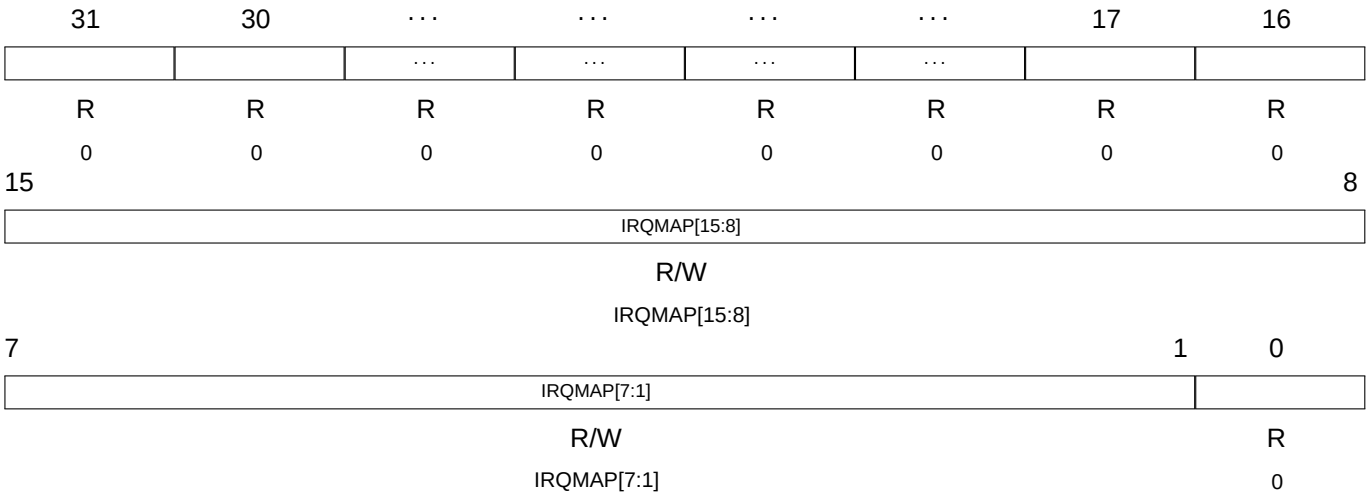
CNT0IE *COUNT Register Equals 0 Interrupt Enable*

0 Interrupt signaling the presence of a “0” value in the COUNT (1.7.7) registrer is disabled.

1 Interrupt signaling the presence of a “0” value in the COUNT (1.7.7) registrer is enabled.

1.7.12 Interrupt Mapping Register

Address: 0x28

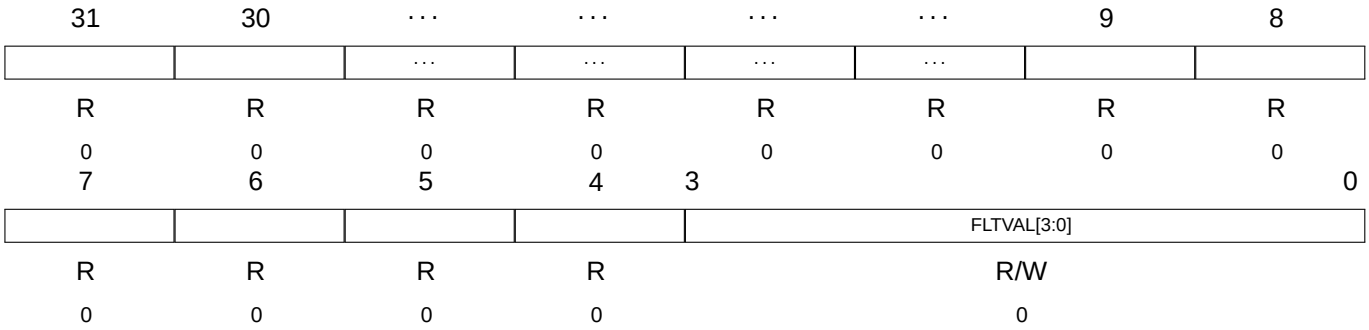


IRQMAP[15:1] *Interrupt Mapping*

Each set bit represents the interrupt number that will be passed to interupt controller. It is allowed to set more than one bit.

1.7.13 Filter Register

Address: 0x2C



FLTVAL[3:0] *Input Filter Value*

Input filter configuration. Number of clock cycles the SDA or SCL line must remain stable to recognize its value. Maximum allowed value i 10. FLTVAL = 0 means that the filter will be bypassed. FLTVAL = 1 means that the filter will delay input signal by 1 clock cycle. Value above the maximum will be treated as FLTVAL = 10.

1.8 Implementation

1.8.1 Design Structure

The synthesible RTL IP core part (*COMMON/rtl* and *I2C_MST/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*I2C_MST/tb* folder) uses SystemVerilog language.

```
COMMON
├── rtl
│   ├── DFF_en.v
│   ├── edge_detector.v
│   ├── input_input.v
│   └── synchronizer.v
I2C_MST
├── beh
├── rtl
│   ├── APB_I2C_MST.v
│   ├── I2C_MST_config.v
│   ├── I2C_MST_defines.v
│   └── I2C_MST_PDMA_stream_interface.v
└── tb
    ├── APB
    │   ├── tb_APB_I2C_init.v
    │   └── tb_APB_I2C_reg_access_tasks.v
    ├── common
    │   ├── tb_I2C_config_tasks.v
    │   ├── tb_I2C_data_transfer_tasks.v
    │   ├── tb_I2C_interrupt_tasks.v
    │   └── timescale.v
    ├── i2c
    │   └── virtual_I2C_slave.sv
    ├── run
    │   └── irun_apb_i2c_mst.sh
    └── tests
        ├── tb_interrupt_MAPPING_test.sv
        ├── tb_RX_0byte_incorrect_adress_INTERRUPT_test.v
        ├── tb_RX_0byte_incorrect_adress_test.v
        ├── tb_RX_0byte_INTERRUPT_test.v
        ├── tb_RX_0byte_test.v
        ├── tb_RX_1byte_INTERRUPT_test.v
        ├── tb_RX_1byte_test.v
        ├── tb_RX_32bytes_incorrect_adress_INTERRUPT_test.v
        ├── tb_RX_32bytes_incorrect_adress_test.v
        ├── tb_RX_32bytes_INTERRUPT_test.v
        ├── tb_RX_32bytes_Sr_INTERRUPT_test.v
        ├── tb_RX_32bytes_Sr_test.v
        ├── tb_RX_32bytes_STOPPED_INTERRUPT_test.v
        ├── tb_RX_32bytes_STOPPED_test.v
        ├── tb_RX_32bytes_test.v
        ├── tb_RX_PDMA_1byte_test.v
        ├── tb_RX_PDMA_32bytes_test.v
        ├── tb_TX_0byte_incorrect_adress_INTERRUPT_test.v
        ├── tb_TX_0byte_incorrect_adress_test.v
        ├── tb_TX_0byte_INTERRUPT_test.v
        └── tb_TX_0byte_test.v
```

```
    ├── tb_TX_1byte_INTERRUPT_test.v
    ├── tb_TX_1byte_test.v
    ├── tb_TX_32bytes_DATA_NACK_INTERRUPT_test.v
    ├── tb_TX_32bytes_DATA_NACK_test.v
    ├── tb_TX_32byte_incorrect_address_INTERRUPT_test.v
    ├── tb_TX_32byte_incorrect_address_test.v
    ├── tb_TX_32byte_INTERRUPT_test.v
    ├── tb_TX_32bytes_Sr_INTERRUPT_test.v
    ├── tb_TX_32bytes_Sr_test.v
    ├── tb_TX_32bytes_STOPPED_INTERRUPT_test.v
    ├── tb_TX_32bytes_STOPPED_test.v
    ├── tb_TX_32bytes_test.v
    ├── tb_TX_PDMA_1byte_test.v
    └── tb_TX_PDMA_32bytes_test.v
  └── tb_APB_I2C_MST.sv
└── compile.list
```

1.8.2 Simulation Flow

The IP Core is provided with self-checking testbench to verify the correct operation of the IP prior to use in a design. To run the simulation using Cadence® Incisive® Enterprise Simulator run *irun_apb_i2c_mst.sh* script located in *I2C_MST/tb/run* folder. The simulation should end with reporting no errors.

1.8.3 Clock and Reset

The CC-I2C_MST-APB utilizes a fully synchronous design with one positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion.

1.8.4 Constraints

In most cases only module output ports are registered. Therefore, it is a good practice to reserve the entire clock cycle for module inputs combinational logic and set minimal input delay (*set_input_delay* command). Registered outputs leave the entire clock cycle for external logic (*set_output_delay* command).

By default module SCK and SDA inputs are synchronized using Synchronizer3 module located in the synchronizer.v file. If possible, they should be replaced with integrated 2FF synchronizers from the target technology library. Otherwise, max delay (*set_max_delay* command) of 10% to 20% of one destination clock cycle should be set between synchronizer stages. Do not use dynamic FFs to implement synchronizer module.

1.8.5 Configuration Options

The table below shows the generic parameters of the core.

Generic name	Description	Range	Default
PDMA_support	Configure PDMA interface support	0,1	1
i2cPrescalerWidth	Configure clock prescaler width	1:32	8
i2cCountWidth	Configure data counter width	1:32	16
default_interrupt_MAPPING	Reset value of interrupt_MAPPING register	1:32767	0

The table below shows the define parameters of the core (I2C_MST_config.v file).

Define name	Description	Default
I2C_MST_INTERFACE_INPUTS_SYNCHRONIZATION	Comment to remove SDA and SCL synchronizing flip flops for I2C master	defined
I2C_MST_INTERFACE_INPUTS_FILTER	Comment to remove SDA and SCL input filter	defined

1.8.6 Signals Description

Signal name	Description	I/O	Active	Type
PCLK	Synchronous clock	I	rising	clock
PRESETn	Asynchronous reset	I	low	reset
PSEL	APB peripheral select	I	high	comb.
PENABLE	APB bus enable	I	high	comb.
PADDR[5:2]	APB bus address	I	data	comb.
PWRITE	APB bus write	I	high	comb.
PWDATA[31:0]	APB bus write data	I	data	comb.
PREADY	APB bus ready	O	data	reg.
PRDATA[31:0]	APB bus read data	O	data	reg.
interrupt_TXC	Transmission complete interrupt	O	high	reg.
interrupt_TDRE	Transmit data register empty interrupt	O	high	reg.
interrupt_RDRF	Read data register full interrupt	O	high	reg.
interrupt_ArbitrationLost	Arbitration lost interrupt	O	high	reg.
interrupt_AddressNACK	Address NACK interrupt	O	high	reg.
interrupt_AddressACK	Address ACK interrupt	O	high	reg.
interrupt_CountEqu0	Count equals zero interrupt	O	high	reg.
interrupt_MAPPING[15:1]	Interrupt mapping vector	O	data	reg.
Downstream_enable	PDMA downstream enable signal	I	high	comb.
Downstream_busy	PDMA downstream busy signal	O	high	reg.
Downstream_request	PDMA downstream request signal	O	high	reg.
Downstream_ack	PDMA downstream ack signal	I	high	comb.
Downstream_data[31:0]	PDMA downstream data	I	data	comb.
Upstream_enable	PDMA upstream enable signal	I	high	comb.
Upstream_busy	PDMA upstream busy signal	O	high	reg.
Upstream_request	PDMA upstream request signal	O	high	reg.
Upstream_ack	PDMA upstream ack signal	I	high	comb.
Upstream_data[31:0]	PDMA upstream data	O	data	reg.
clock_request	Clock request signal	O	high	reg.
SDA_pad_input	I2C data pin input	I	As I2C spec.	reg./comb[1]
SCL_pad_input	I2C clock pin input	I	As I2C spec.	reg./comb[1]
SDA_pad_output	I2C data pin output	O	As I2C spec.	reg.
SCL_pad_output	I2C clock pin output	O	As I2C spec.	reg.

[1] Depending on I2C_MST_INTERFACE_INPUTS_SYNCHRONIZATION setting. Defined value will insert input synchronization flip-flops.

1.8.7 Instantiation

```
icg
icg_i2c (
    .E(i2c_PSEL|i2c_clock_request),
    .clk(PCLK),
    .gclk(i2c_clk),
    .scan_enable(scan_enable));

APB_I2C_MST #(
    .PDMA_support(CFG_DMA_EN),
    .default_interrupt_MAPPING(CFG_DEF_INT_MAPPING))
APB_I2C_MST_u (
    .PCLK(i2c_clk),
    .PRESETn(rst),
    .PSEL(i2c_PSEL),
    .PENABLE(i2c_PENABLE),
    .PADDR(PADDR[5:2]),
    .PWRITE(PWRITE),
    .PWDATA(PWDATA),
    .PREADY(i2c_PREADY),
    .PRDATA(i2c_PRDATA),
    .SDA_pad_input(SDA_input),
    .SCL_pad_input(SCL_input),
    .SDA_pad_output(SDA_output),
    .SCL_pad_output(SCL_output),
    .interrupt_TXC(i2c_interrupt_TXC),
    .interrupt_TDRE(i2c_interrupt_TDRE),
    .interrupt_RDRF(i2c_interrupt_RDRF),
    .interrupt_ArbitrationLost(i2c_interrupt_ArbitrationLost),
    .interrupt_AddressNACK(i2c_interrupt_AddressNACK),
    .interrupt_AddressACK(i2c_interrupt_AddressACK),
    .interrupt_DataNACK(i2c_interrupt_DataNACK),
    .interrupt_DataACK(i2c_interrupt_DataACK),
    .interrupt_CountEqu0(i2c_interrupt_CountEqu0),
    .interrupt_MAPPING(i2c_interrupt_MAPPING),
    .Downstream_enable(i2c_downstream_enable),
    .Downstream_busy(i2c_downstream_busy),
    .Downstream_request(i2c_downstream_request),
    .Downstream_ack(i2c_downstream_ack),
    .Downstream_data(downstream_data),
```

```
    .Upstream_enable(i2c_upstream_enable),
    .Upstream_busy(i2c_upstream_busy),
    .Upstream_request(i2c_upstream_request),
    .Upstream_ack(i2c_upstream_ack),
    .Upstream_data(i2c_upstream_data),
    .clock_request(i2c_clock_request));

assign i2c_irq        = i2c_interrupt_TXC | i2c_interrupt_TDRE |
                        i2c_interrupt_RDRF | i2c_interrupt_ArbitrationLost |
                        i2c_interrupt_AddressNACK | i2c_interrupt_AddressACK |
                        i2c_interrupt_DataNACK | i2c_interrupt_DataACK |
                        i2c_interrupt_CountEqu0;

assign i2c_irq_vector = i2c_interrupt_MAPPING & {15{i2c_irq}};

io_pad_model #(
    .IO_NUM(1))
scl_pad_model (
    .core_input(SCL_input),
    .core_output(1'b0),
    .IO_pad(SCL_pad),
    .output_enable(~SCL_output));

io_pad_model #(
    .IO_NUM(1))
sda_pad_model (
    .core_input(SDA_input),
    .core_output(1'b0),
    .IO_pad(SDA_pad),
    .output_enable(~SDA_output));
```

1.9 Revision History

Doc. Rev.	Date	Comments
1.2	11-2018	Editorial corrections in 1.8.7 Instantiation section.
1.1	08-2017	Corrected F_P, $t_{STA/STO}$, t_{LOW}, t_{HIGH} and $t_{SETUP/HOLD}$ formulas. Updated 1.8.4 Constraints section. Added description of synchronizers implementation.
1.0	05-2017	First Issue.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

 CC-I2C_MST-APB-Doc_112018.