Datasheet

CC-I2C_SLV-AXI Documentation

1.0

Scope

This document describes the CC-I2C_SLV-AXI 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 Slave Controller

1.1 Functionality

- work in a Slave mode,
- dual addressing capability to acknowledge 2 slave addresses,
- general call address recognition,
- support for 7 and 10 bit addressing,
- configurable interrupt signaling the status of transmission and events on the bus,
- automatic clock-stretching if transmit or receive buffers not ready for data transfer,
- DMA interface for reducing CPU load,
- programmable input filter for noise suppression,
- configurable automatic transmission (automatic sending of ACK/NACK).

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.

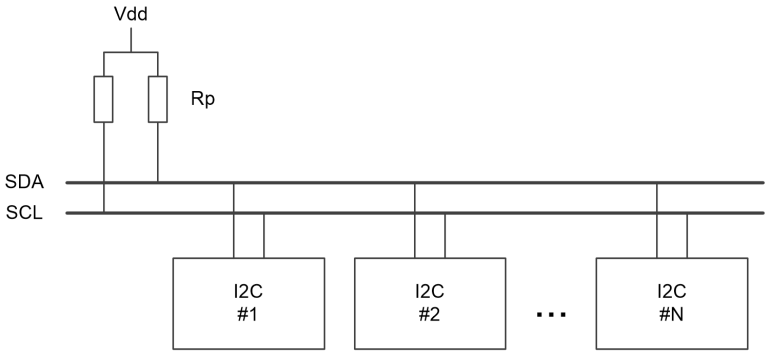


Figure 1.1. I2C bus topology

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. Transmission terminates when the Master sends stop bit. SCL signal is generated by a Master device, but the Slave can hold down the SCL line thus streching the bus clock signal.

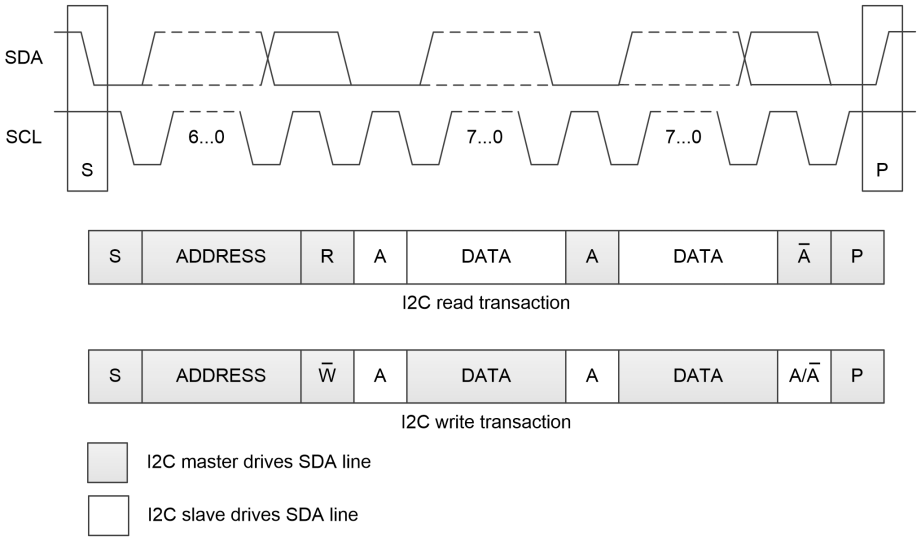


Figure 1.2. Sample transmission on the I2C bus.

1.3 Block Diagram

The I2C controller enables two-way communication in the Slave 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 Slave module. The basic components are:

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

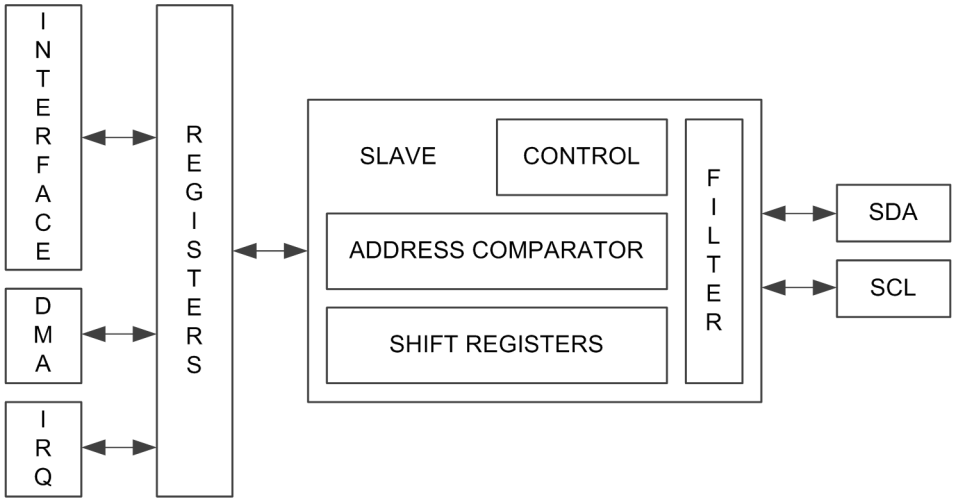


Figure 1.3. Block diagram of the I2C controller

1.4 Data Transmission

1.4.1 START/STOP Bits

Data transmission begins with a start bit (S) and ends with a 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 a 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 not preceded by stop bit is called the repeat start bit (Sr).

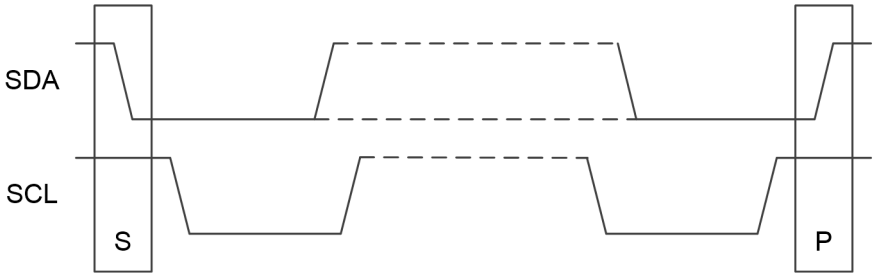


Figure 1.4. Start (S) and Stop (P) bits.

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).

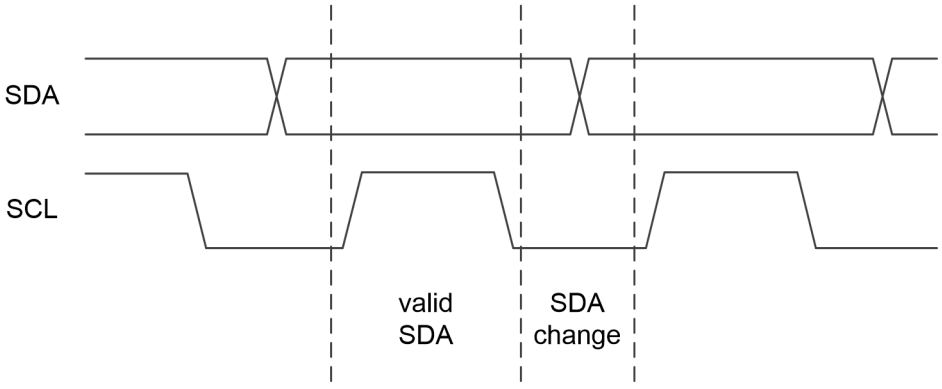


Figure 1.5. Data bit.

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 (Slave reception mode). A value of 1 means that the Slave should send data to the Master device immediately after acknowledging receiving its address (Slave transmission mode).

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 Bus Timing

In order to achieve a proper bus signals timings when transmitting data, Slave module will stretch (hold low) SCL signal while the new data bit value is set on SDA line. For this purpose Timings (1.7.6) Register provides parameters controlling the following times (Figure 1.6):

- t_{SETUP} - duration of time SCL line will be held low after setting new SDA output state,
- t_{HOLD} - time following falling edge of the SCL line, for which SCL will be held low and SDA will keep its current value. After this time, SCL line will be released (SCL output set to high), and SDA output will be set to the next data (Slave transmission mode) or ACK/NACK (after receiving address or data byte in Slave reception mode) bit value.

Above timing parameters define a number of $PCLK$ clock cycles during which SCL will be streched. Note that dependend on the configuration, the actual t_{HOLD} time may differ due to the latency introduced by SCL input synchronization (two clock cycles) and by the number of activated input glitch filter stages. As far as t_{SETUP} is concerned, Slave module will not wait for the SDA line state to stabilize, so the t_{SETUP} value must be chosen accordingly to the external electrical parameters.

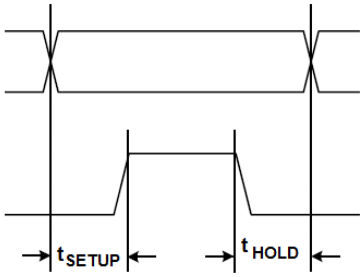


Figure 1.6. Parameters of the signal generated on the SCL and SDA lines.

1.4.6 Commands

The module provides three commands that allow the software to influence the transmission progress and the internal state of the module. Commands are issued by writing their code into the 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 executed as soon as possible. Suspending a command is signaled by reporting its code in the Status (1.7.2) Register. Suspended command can be cancelled by writing NO COMMAND code into the Command (1.7.4) Register.

RESET The RESET command allows to restore initial values in the module configuration registers. This 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.

ACK The ACK command has a different meaning, depending on the configuration of the module and its current state (Figure 1.9, Figure 1.13). At the address sending stage, when automatic address acknowledge is disabled (ADDR_ACK = 0), ACK command is used to trigger sending of ACK bit to the Master. At the data transfer stage (Slave reception mode), when automatic data acknowledge is disabled (AUTO_ACK = 0), ACK command is used to send ACK bit to the Master. In all other cases, ACK command has no effect.

STOP The STOP command is used to immediately (within the I2C protocol) terminate the transmission. Bus is released i.e. high level is set on SDA and SCL outputs at the first possible (from the point of view of the I2C protocol) opportunity and the module awaits to be addressed again. In Slave reception mode, this results in NACK bit being sent to the Master. In Slave transmission mode, even if the TDR register contains current data, it will be ignored (which will result in 0xFF bytes being received by the Master).

CLR TDR The CLR TDR command is used to immediately invalidate and clear the content of Transmission Data (1.7.9) Register. Please advise, that due to the mechanics of PDMA transaction one must take caution when the command is issued with downstream transfer operational as it could invalidate data read from memory, but not transferred to the I2C master device yet. It is suggested to disable associated PDMA channel before issuing CLR TDR command.

CLR RDR The CLR RDR command is used to immediately invalidate and clear the content of Reception Data (1.7.10) Register. Note that due to the mechanics of PDMA transaction, using CLR RDR command with PDMA upstream operational may cause undetermined (timing sensitive) results potentially leading to miscompares. It is strongly advised to successfully disable associated PDMA upstream channel before issuing CLR RDR command.

1.4.7 Slave Data Reception

Figure 1.7 shows the frame transmitted on the I2C bus in slave reception mode. Master device starts the transaction with the start bit, after which the address packet is sent (along with the directional bit equal to 0 - write). Slave device acknowledges the address (ACK = 0), after which the Master sends data bytes. Each of them must be acknowledged by the Slave (ACK = 0) except for the last after which the Slave responds with ACK = 1. Master ends the transaction by sending a stop bit.

Figure 1.7. Slave data reception.

Transmission is analogous in 10-bit addressing mode (1.8). 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.

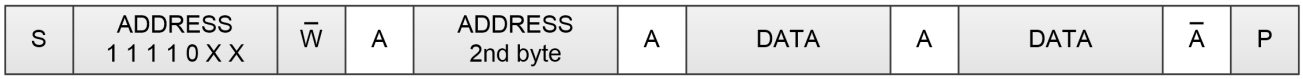


Figure 1.8. Slave data reception (10-bit addressing mode).

Figure 1.9 shows a status diagram for the slave reception operation, including possible module configuration variants, and Figure 1.10 shows how the transmission status is reported when receiving a single byte.

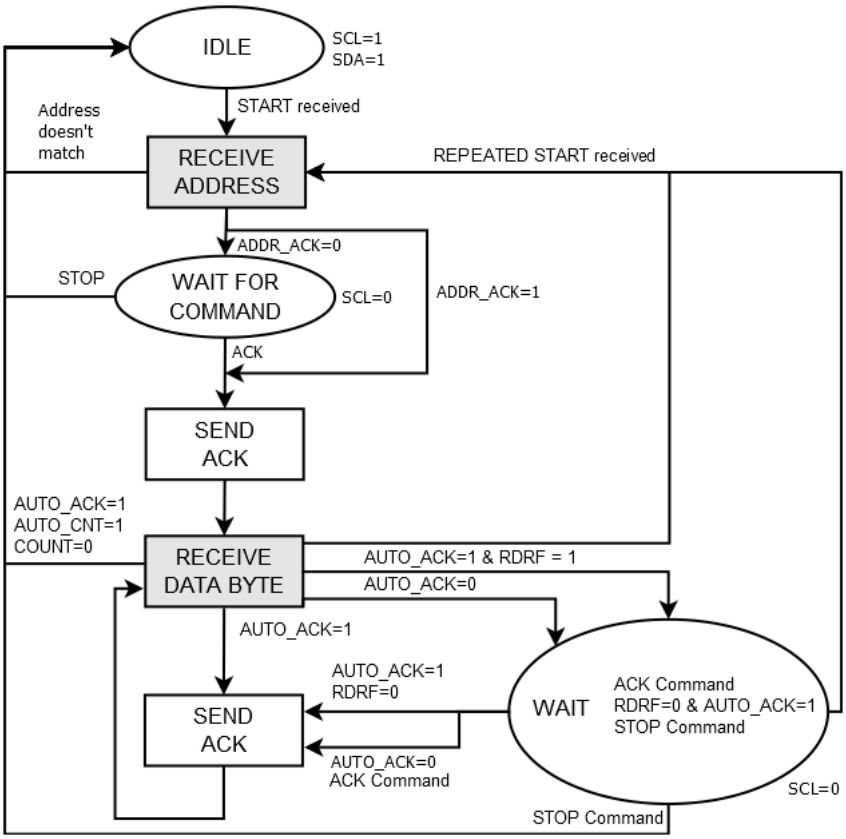


Figure 1.9. Slave reception algorithm.

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

- Enable module via Control (1.7.3) Register. Activate selected transmission automation modes, recognized addresses and addressing mode (10 or 7 bit).
- Addresses recognized by the Slave module must be configured via Address (1.7.8) Register.
- The run-time parameters are configured using the Timings (1.7.6) Register.
- The Interrupt Mask register (1.7.11) must be configured to activate the selected interrupt source.
- If automatic counting of bytes (bit AUTO_CNT in the Control (1.7.3) Register) is activated, program the number of bytes sent in the Count Register (1.7.7).

Address reception After configuration, module monitors SDA and SCL inputs to detect start condition. When start condition is detected (indicated in Status (1.7.2) Register), Slave module receives address byte and compares it to addresses set in the Address (1.7.8) Register (1.7.8). If matching address is detected (Figure 1.10), module indicates this fact through Slave Addressed interrupt. If ADDR_ACK = 1, Slave responds by sending ACK bit (Figure 1.9). Otherwise module awaits for ACK or STOP command from the firmware. If received address doesn't match, module releases the bus (and thus effectively sends NACK).

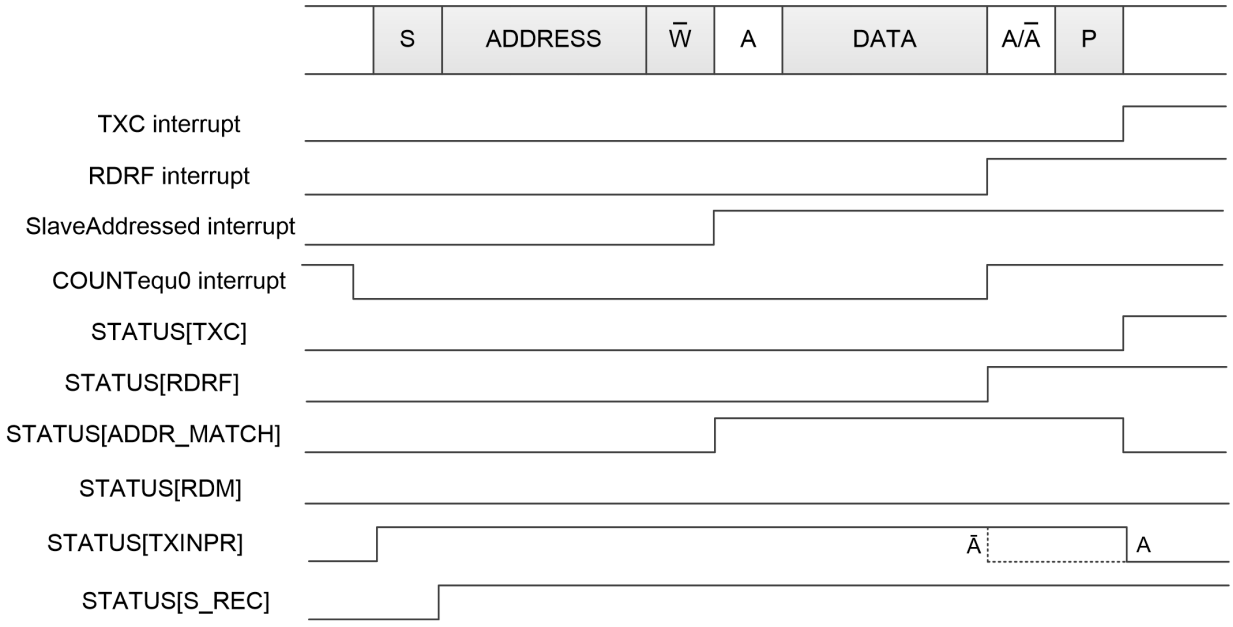


Figure 1.10. Reception of a single byte of data (ADDR_ACK = 1, AUTO_ACK = 1, AUTO_CNT = 1).

In 10-bit addressing mode, after receiving first matching byte of address, module automatically transmits ACK bit and does not report it in any way until a second matching addres byte is received.

Receiving data As soon as Slave acknowledges receipt of the address, Master starts transmitting the first byte of the data. The reception is signaled by setting the RDRF bit in the Status (Figure 1.14) Register. The next actions of the module depend on the configuration of the automatic data transmission modes:

AUTO_ACK=1, AUTO_CNT=1 If this is not the last received byte, module sends the ACK bit and then waits for reading data from the RDR (1.7.10) register while holding the SCL line in low state to prevent Master device from transmitting another byte. When the RDR register is read, module loads received data into the RDR register, decrements COUNT and starts receiving another byte of data. If the last byte (COUNT = 0) was received, Slave module sends the NACK bit and releases the bus awaiting Stop or repeated start (Sr) bit reception.

AUTO_ACK=1 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, module loads received data into the RDR register, increments COUNT and starts receiving another byte of data.

AUTO_ACK=0 Module expects firmware firmware intervention for further operations. This action is to send an ACK (to continue) or STOP command.

Reception ending Upon the reception of the programmed byte count, the module can automatically terminate the transmission (by releasing the bus), provided that automatic counting and acknowledgment (AUTO_CNT, AUTO_-ACK, in the Control (1.7.3) 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 errors). 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 bus is released thus sending NACK to the Master.

Short frame reception The module allows the reception of short frames consisting only of the start bit, address, and stop bit.

Repeated start reception The module supports reception of consecutive packets (when the repeated start bit and the address frame are transmitted instead of the stop bit). Reception of repeated start is indicated in the Status (1.7.2) Register. 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 reception of the address, the Count Register will be reset and the count of transmitted bytes will start from 0.

1.4.8 Slave Data Transmission

Figure 1.11 represents a frame that is transmitted on the I2C bus in Slave reception mode. Master device starts the transaction with the start bit, after which the address packet is sent (along with the directional bit equal to 1 - read mode from the Master point of view). Slave device acknowledges the address (ACK = 0), after which the Master waits for the next data bytes from the Slave and sends the acknowledgment during the ACK bit. Master ends the transaction by sending NACK bit followed by a stop bit.

Figure 1.11. Slave data transmission

In 10-bit addressing mode (Figure 1.12), Slave module is first addressed for write by the Master. Afterwards, Master transmits repeated start bit followed by the first address byte with READ bit set. If there's been no data transmitted from the Master in between, the Slave module will change it's address mode from write to read.

Figure 1.12. Slave data transmission (10-bit addressing mode).

Figure 1.13 represents a status diagram for Slave transmission, while Figure 1.14 shows how transmission status is reported when transmitting a single byte.

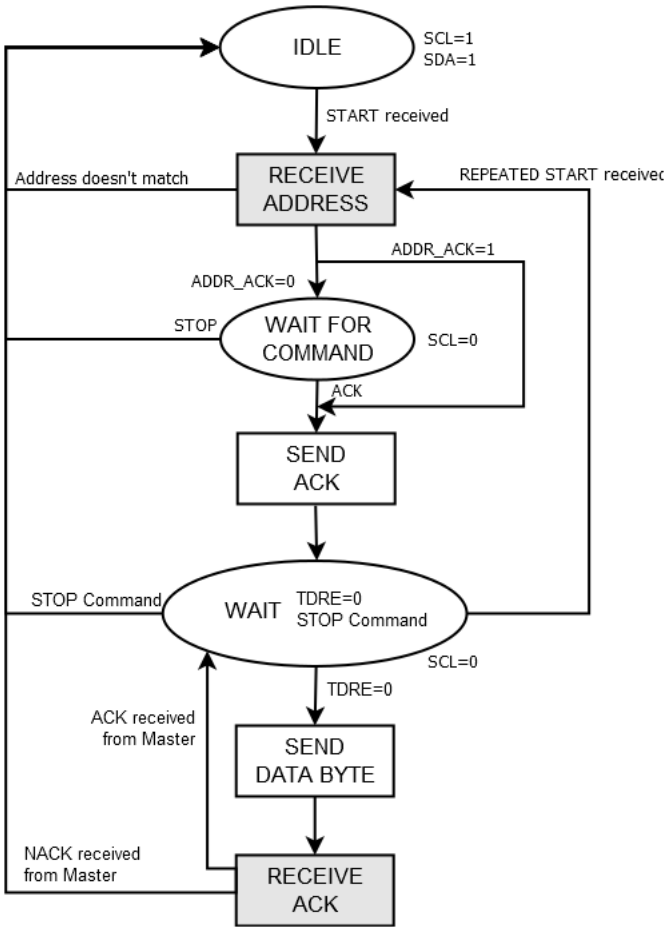


Figure 1.13. Slave data transmission algorithm.

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

- Enable module via Control (1.7.3) Register. Activate selected transmission automation modes and addressing mode (10 or 7 bit).
- Addresses recognized by the Slave module must be configured via Address (1.7.8) Register.
- The run-time parameters are configured using the Timings (1.7.6) Register.
- The interrupt mask register (1.7.11) must be configured to activate the selected interrupt source.
- If automatic counting of bytes (bit AUTO_CNT in the Control (1.7.3) Register) is activated, program the number of bytes sent in the Count Register (1.7.7).

Address reception After configuration, module monitors SDA and SCL inputs to detect start condition. When start condition is detected (indicated in Status (1.7.2) Register), Slave module receives address byte and compares it to addresses set in the Address (1.7.8) Register. If matching address is detected (Figure 1.14), module indicates this fact through Slave Addressed interrupt. If ADDR_ACK = 1, Slave responds by sending ACK bit (Figure 1.13). Otherwise module awaits for ACK or STOP command from the firmware. If the received address doesn't match, module releases the bus (and thus effectively sends NACK). There's no indication (other than the START bit in Status Register) of the address mismatch.

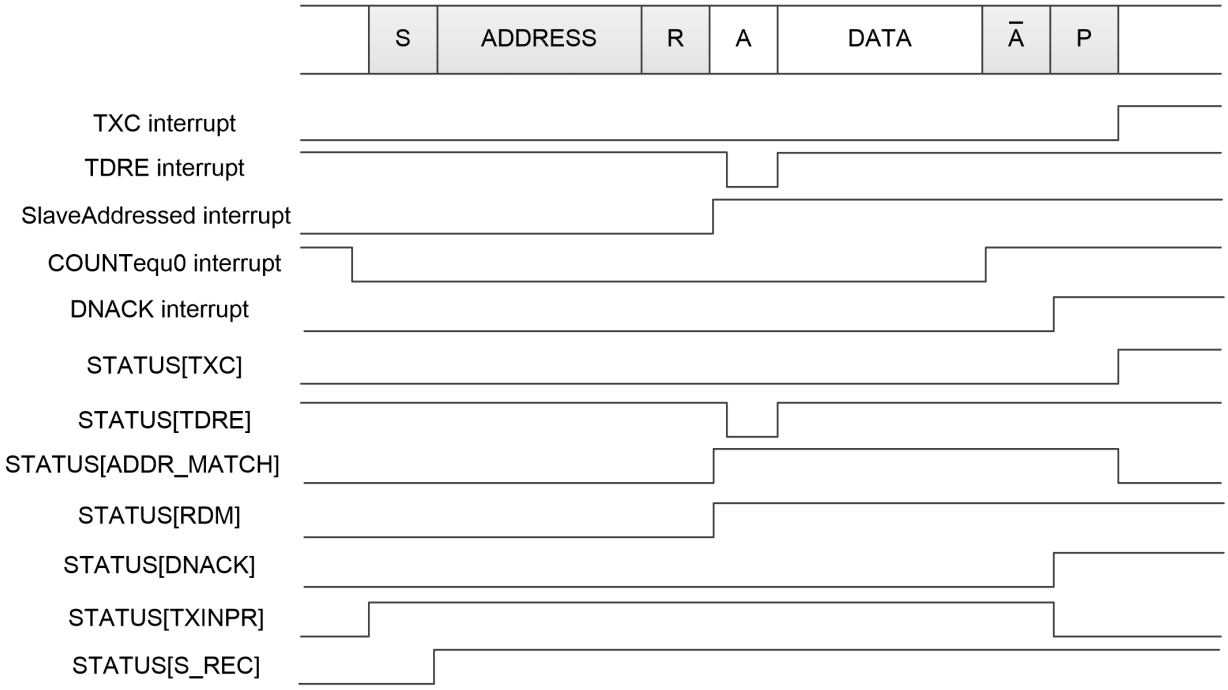


Figure 1.14. Send a single byte of data

In 10-bit addressing mode, Slave module must be addressed for WRITE first. After that, Master sends repeated start bit followed by the first byte of address with READ bit set.

Data Transmition After acknowledging received address, module awaits data bytes to be written into the TDR (1.7.9) register. Throughout this time, SCL line is held in a low state, to prevent the master from starting byte reception. This fact is indicated by SCL HOLD bit being set in Status (1.7.2) Register. Once the data is saved to the TDR register (Figure 1.14), it is transmitted immediately. At the same time, the TDR register is emptied so that another data byte can be written. Module then increments (or decrements) the Count Register and waits for the ACK bit from the Master device. The received ACK/NACK bit is reported in the Status Register (Figure 1.14). If the ACK bit (0) is received, the controller 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 release the bus (by setting high level on SCL and SDA outputs) and wait for Start or Stop bit.

Transmission ending Software can terminate the transmission at any time by sending the STOP command. If the STOP command was issued during the transmission, execution will be delayed until final receiving of data.

Short frame transmission Slave module can handle reception of short frames consisting only of the start bit, address, and stop bit. However in order to perform such transfer in Slave transmission mode, TDR register must be preset with 0xFF data byte. Otherwise Master device will not be able to transmit the stop bit.

Repeated start transmission The module supports reception of consecutive packets (when the repeated start bit and the address frame are transmitted instead of the stop bit). Reception of repeated start is indicated in the Status (1.7.2) Register. If automatic count of sent bytes is active (bit AUTO_CNT=1), 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 Slave controller can report an interrupt in response to a selection of 8 configurable events.

1.5.1 TXC Interrupt

The interrupt line is set high when the module has been addressed and a STOP bit has been detected. The line is set to low when the Status Register is read (1.7.2) or TDR register is written (1.7.9).

1.5.2 TDRE Interrupt

The interrupt line is set high when there's no valid data in the TDR (1.7.9) register (either hasn't been written or the data from the TDR register has been transferred to the shift register of the transmitter). The line is set low when data is written into the TDR register (1.7.9).

1.5.3 RDRF Interrupt

The interrupt line is set high when data has been received from Master and written to the RDR register. The line is set low when the RDR register is read (1.7.10).

1.5.4 DATA NACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 1 (NACK) in response to the data packet (valid in Slave transmission mode only). The line is set low when the Status Register is read.

1.5.5 DATA ACK Interrupt

The interrupt line is set high when the received acknowledgment bit is 0 (ACK) in response to the data packet (valid in Slave transmission mode only). The line is set low when the Status Register is read.

1.5.6 COUNT EQUALS 0 Interrupt

The interrupt line is set high when the Count Register equals 0.

1.5.7 SLAVE ADDRESSED Interrupt

The interrupt line is set high when the Slave module receives an address frame matching one of its configured addresses. The line is set low when the Status Register is read.

1.5.8 BUS ERROR Interrupt

The interrupt line is set high at when the Slave module detects protocol error (such as unexpected start or stop bit, SDA input value not matching the SDA output in Slave transmission mode or during NACK bit transmission). 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 1.15. 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.

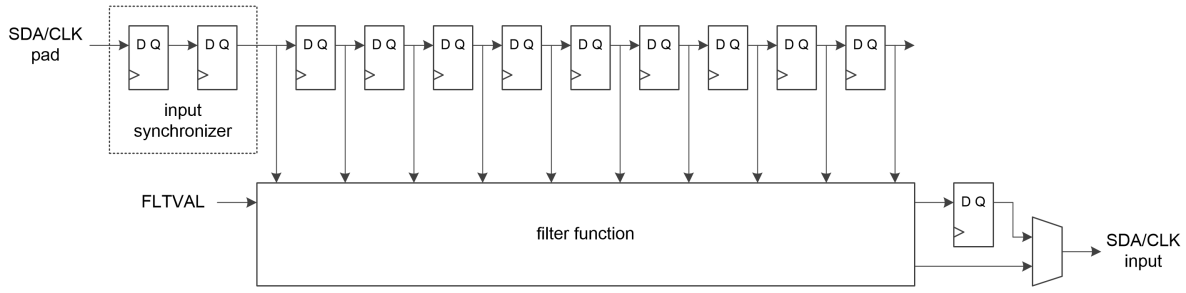


Figure 1.15. Block diagram of SDA/SCL input filter

Figure 1.16 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.

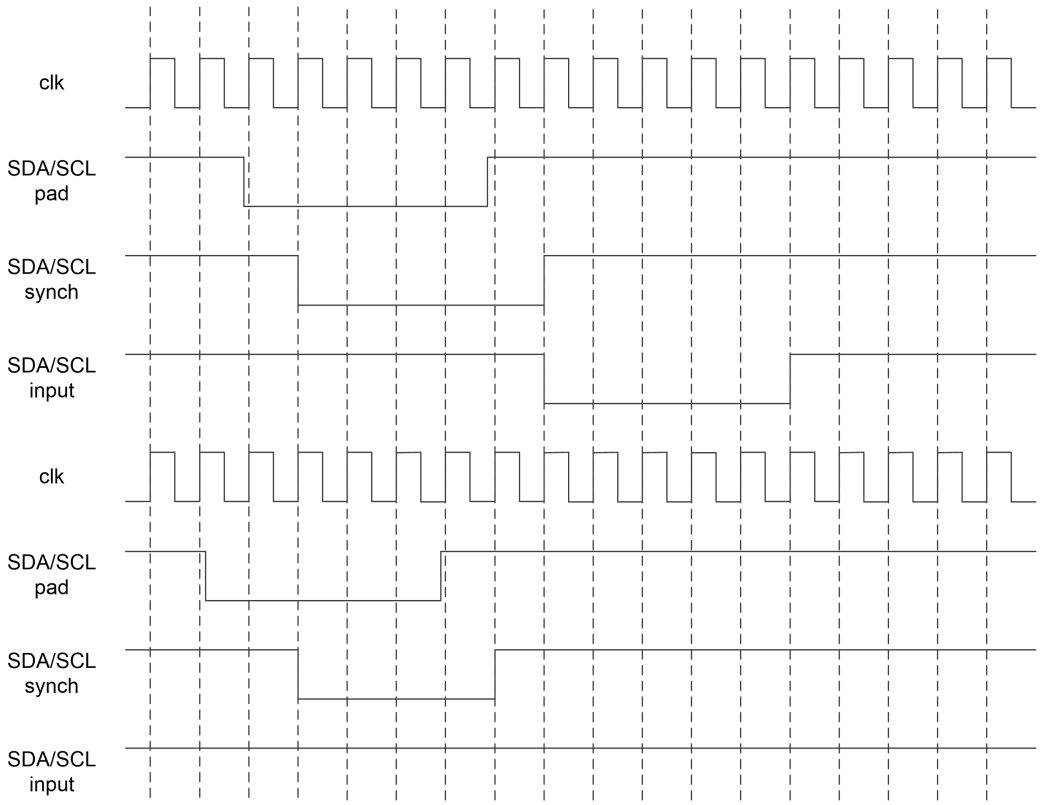


Figure 1.16. Input filter waveform for FLTVAL = 5.

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	FILTER	Filter Register
0x10	TMNG	Timings 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

1.7.2 Status Register

Address: 0x00

31	30	...	...	19	18	17	16
		...	...	SEC_MATCH	PRI_MATCH	GC_MATCH	RDM
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8
ADDR_MATCH	RS_REC	S_REC	DNACK	DACK	PACK	CURRENT_CMD[3:2]	
R	R	R	R	R	R	R	
0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0
CURRENT_CMD[1:0]		HOLD	TXINPR	RDRF	TDRE	TXC	BS_ERR
R		R	R	R	R	R	R
0		0	0	0	1	0	0

BS_ERR *Bus Error*

The BS_ERR bit is set to “1” when illegal bus condition occurs during transfer (unexpected START or STOP during data transfer etc...).

TXC *Transmission Complete*

The TXC bit is set to “1” when the STOP bit has been received (module has to be addressed first). The bit is set to “0” when the Status Register is read or TDR register is written.

TDRE *Transmit Data Register Empty*

0 The TDR register contains data that hasn't been sent yet.

1 The TDR 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" when data frame has been received and loaded into the RDR register. It is set to "0" when data is read from the RDR register.

TXINPR *Transmission in progress* The TXINPR bit is set to "1" when the Slave module is receiving or transmitting data (that also counts address reception even if it's unknown whether it will match). It is set to "0" when stop bit is detected, or module receives and executes STOP command.

HOLD *Bus hold*

The value "1" means that the module holds the I2C bus by keeping SCL signal low.

CURRENT_CMD[3:0] *Current Command*

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

CURRENT_CMD[3:0]	Executed command
0000	NO COMMAND
0001	ACK
0010	STOP
0011	RESET
0100	CLR TDR
0101	CLR RDR

PACK *Received ACK bit*

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

DACK *Data ACK-ed by Master*

The value "1" means that the Master device has acknowledged receiving the data by the ACK bit (the bit value is valid only during transmission in Slave transmission mode). The DACK bit is set to "0" when the Status Register is read.

DNACK *Data NACK-ed by Master*

The value "1" means that the Master device has acknowledged receiving the data by the NACK bit (the bit value is valid only during transmission in Slave transmission mode). The DNACK bit is set to "0" when the Status Register is read.

S_REC *Start received*

The value “1” means that the start bit has been received (doesn’t cover repeated start). The S_REC bit is set to “0” when the Status Register is read.

SR_REC *Repeated Start received*

The value “1” means that the repeated start bit has been received. The SR_REC bit is set to “0” when the Status Register is read.

ADDR_MATCH *Slave addressed*

The ADDR_MATCH bit is set to “1” when the Slave module receives address, that matches one of its own. It is set to “0” when stop bit is detected, or module receives and executes STOP command.

RDM *Read mode*

The value “1” means that the Slave module has been addressed for read (a.k.a. Slave transmission mode). It’s valid only if ADDR_MATCH bit is set. The value “0” means that the Slave module has been addressed for write (a.k.a. Slave reception mode). It’s valid only if ADDR_MATCH bit is set.

GC_MATCH *General Call Address matched*

The value “1” means that the Slave module has been addressed with General Call address. It’s valid only if ADDR_MATCH bit is set.

PRI_MATCH *Primary Address matched*

The value “1” means that the Slave module has been addressed with its primary address. It’s valid only if ADDR_-MATCH bit is set.

SEC_MATCH *Secondary Address matched*

The value “1” means that the Slave module has been addressed with its secondary address. It’s valid only if ADDR_MATCH bit is set.

1.7.3 Control Register

Address: 0x04

31	30	...	...	...	...	9	8
		...	...	...	...		SEC_10B
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
SEC_MATCH	PRI_10B	PRI_MATCH	GC_MATCH	ADDR_ACK	AUTO_ACK	AUTO_CNT	EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EN *I2C Enable*

0 The I2C module is disabled.

1 The I2C module is enabled.

AUTO_CNT *Automatic Byte Counting Enable*

0 Automatic data byte countdown is disabled. In this mode, the matching 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 (Slave reception mode) when AUTO_ACK bit is set to “1” also, after receiving the last byte of data (COUNT = “0”), the NACK is automatically sent and the I2C bus is released by the slave module.

- In the read mode (Slave transmission mode) after receiving the last byte (COUNT = “0”), COUNT EQUALS 0 Interrupt is asserted. If further data bytes are transmitted, the COUNT value will remain at “0”.

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 Slave reception mode. When it is equal to “1”, after each byte received from the Master device, module will automatically send an acknowledgment bit equal to “0”. In addition, when AUTO_CNT = “1”, after the last received byte, module will send a not-acknowledgment bit equal to “1” followed by bus release.

ADDR_ACK *Automatic Address ACK bit Transmission Enable*

0 Automatic address acknowledgement transmission bit is disabled.

1 Automatic address acknowledgement transmission bit is enabled.

When ADDR_ACK is equal to “1”, module will automatically send an acknowledgment bit (equal to “0”) when a matching device address is received.

GC_MATCH *General Call Address match Enable*

0 General call address recognition is disabled.

1 General call address recognition is enabled.

When GC_MATCH is equal to “1”, module will recognize and respond to General Call Address (0x00).

PRI_MATCH *Primary Address match Enable*

0 Primary address recognition is disabled.

1 Primary address recognition is enabled.

When PRI_MATCH is equal to “1”, module will recognize and respond to Primary Address (configured in ADDR 1.7.8 register).

PRI_10B *Primary Address 10bit Enable*

0 Primary Address is 7 bits long.

1 Primary Address is 10 bits long.

SEC_MATCH *Secondary Address match Enable*

0 Secondary address recognition is disabled.

1 Secondary address recognition is enabled.

When SEC_MATCH is equal to “1”, module will recognize and respond to Secondary Address (configured in ADDR 1.7.8 register).

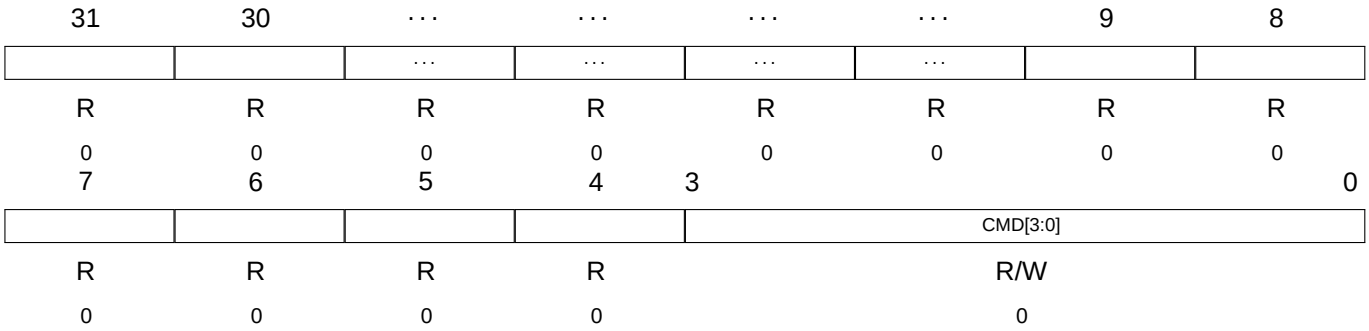
SEC_10B *Secondary Address 10bit Enable*

0 Secondary Address is 7 bits long.

1 Secondary Address is 10 bits long.

1.7.4 Command Register

Address: 0x08



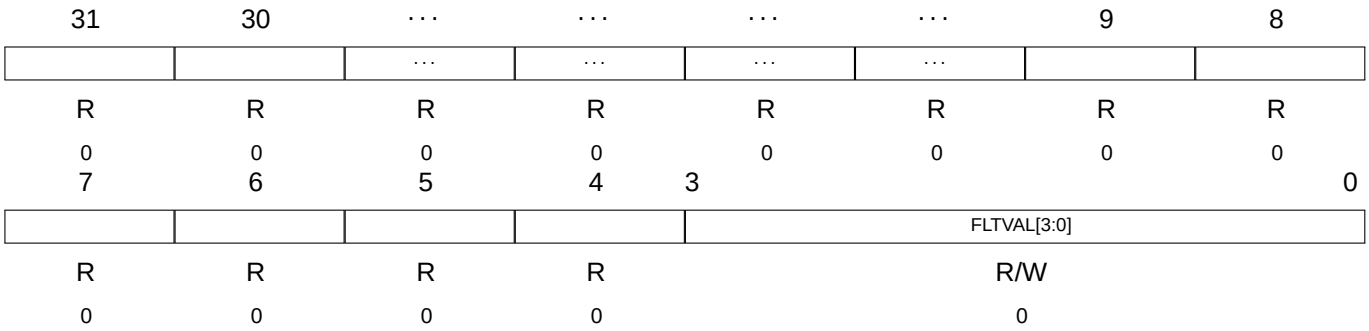
CMD[3:0] *Command*

The next command code for the module (unsupported codes shall be ignored):

CMD[3:0]	Command code
0000	NO COMMAND
0001	ACK
0010	STOP
0011	RESET
0100	CLR TDR
0101	CLR RDR

1.7.5 Filter Register

Address: 0x0C

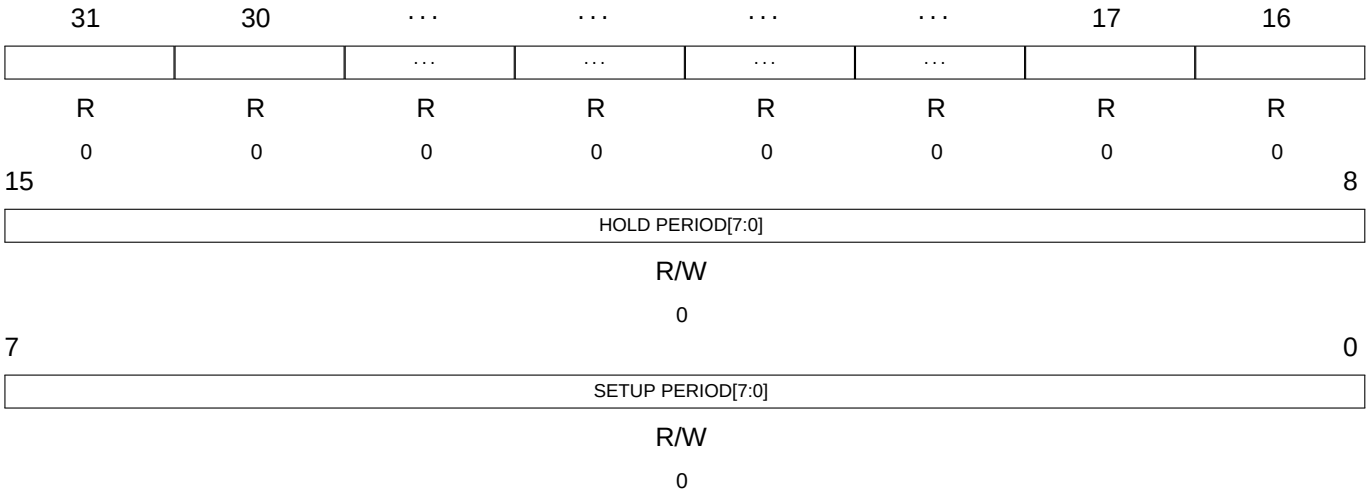


FLTVAL[3:0] *Input Filter Value*

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.

1.7.6 Timings Register

Address: 0x10



SETUP PERIOD[7:0] *Setup time*

Setup period defines the number of PCLK clock cycles, during which SCL line will be held low after setting SDA output state (applicable to data and ack bits transmission performed by the Slave module).

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

HOLD PERIOD[7:0] *Hold time*

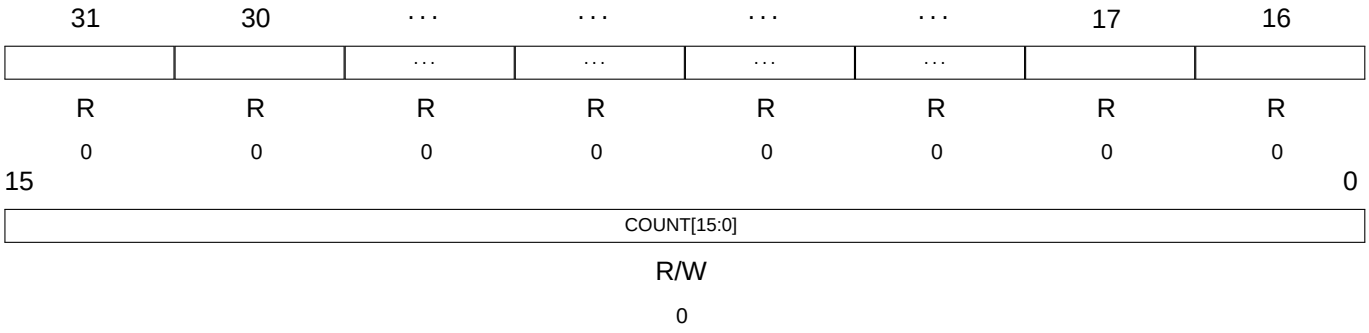
Hold period defines the number of PCLK clock cycles, during which SDA line will hold its last value, after low level of SCL input is detected (SCL output is also held low during this time). This applies to data and ack transmission performed by the Slave module.

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

Note, that all periods defined in Timings Register are nominal. The actual timings will depend on whether inputs synchronization/filter is enabled, and the filter configuration.

1.7.7 Count Register

Address: 0x14

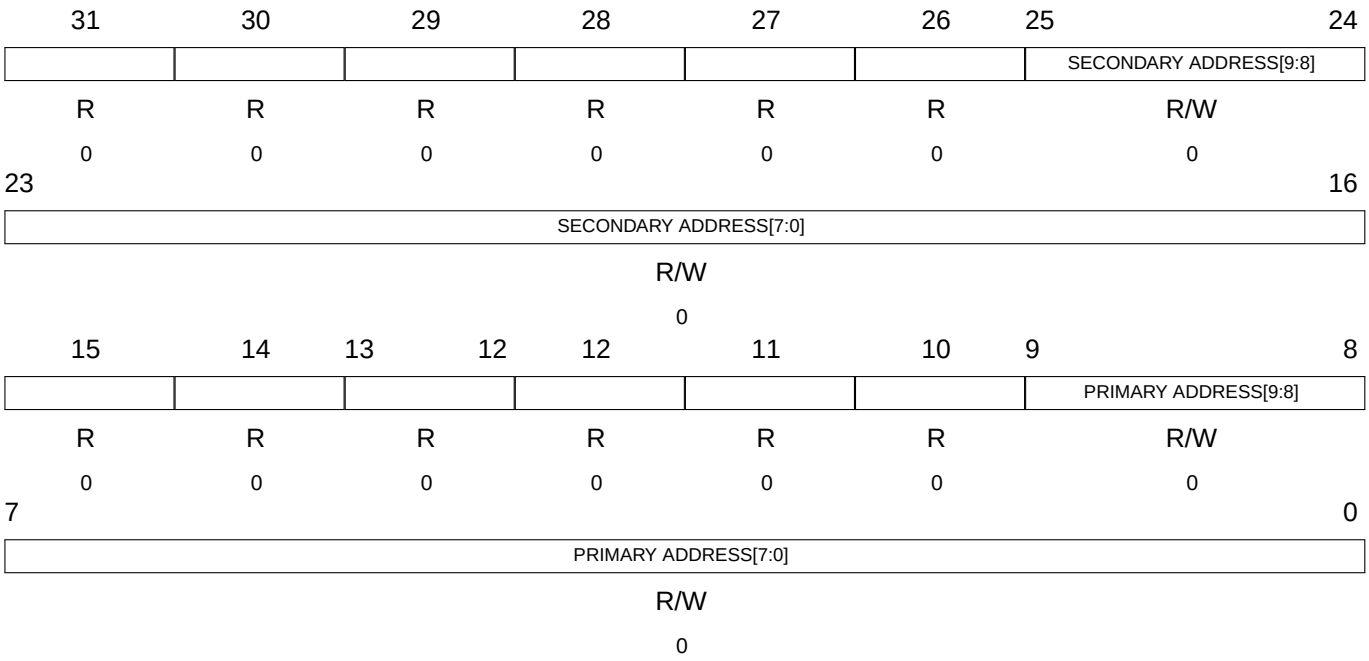


COUNT[15:0] *The counter of sent bytes via I2C interface*

In the automatic byte counting mode (the AUTO_CNT bit in the Control (1.7.3) 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 received and then incremented after each data frame that has been sent / received.

1.7.8 Address Register

Address: 0x18



PRIMARY ADDRESS[9:0] *Primary Slave Address*

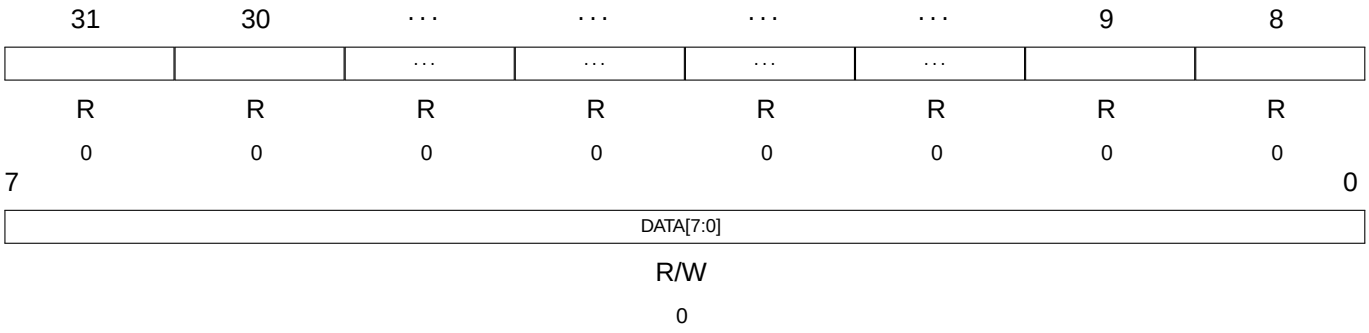
Configured address, module will respond to (for both modes - read and write).

SECONDARY ADDRESS[9:0] *Secondary Slave Address*

Configured address, module will respond to (for both modes - read and write).

1.7.9 Transmission Data Register

Address: 0x1C

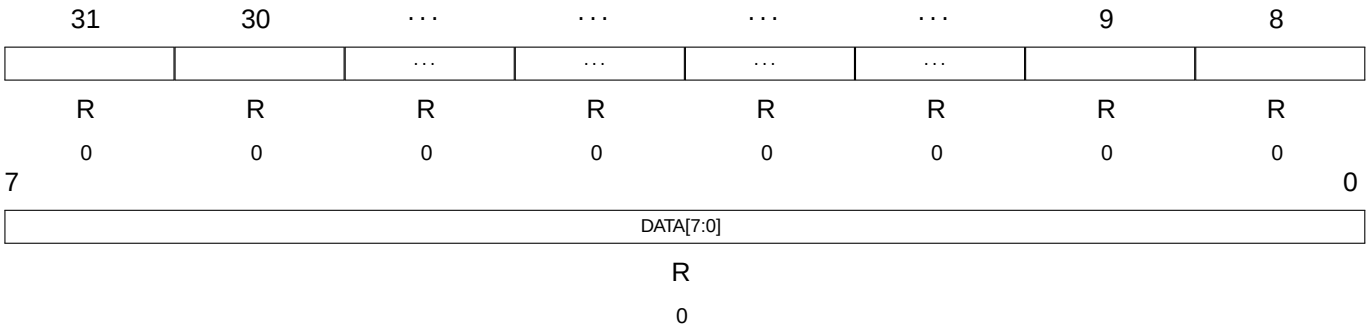


DATA[7:0] *Transmission Buffer*

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 Slave module has been succesfully addressed in Slave transmission mode, writing data into the TDR register will start the transmission of the data frame.

1.7.10 Reception Data Register

Address: 0x20

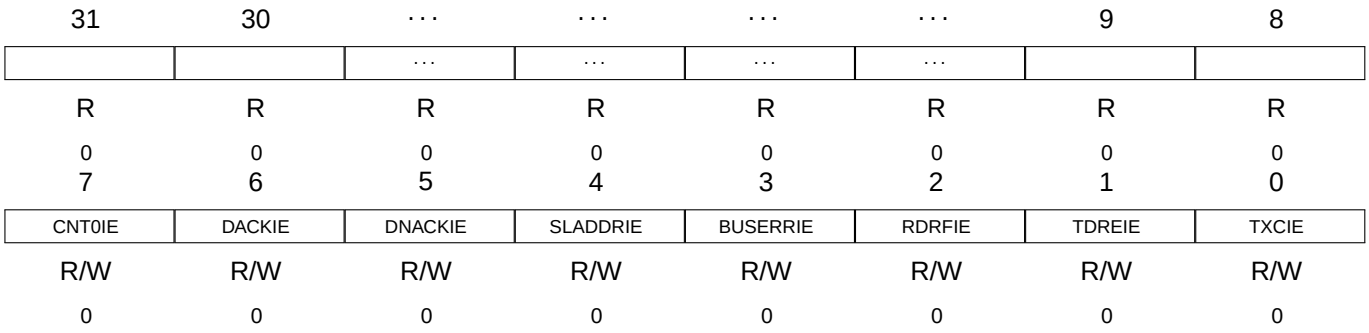


DATA[7:0] *Reception Buffer*

The data received from the Master 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



TXCIE *Transmission Completed Interrupt Enable*

0 Interrupt after reception of the STOP bit is disabled.

1 Interrupt after reception 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.

BUSERRIE *Bus Error Interrupt Enable*

0 Interrupt signaling collision or other I2C protocol error is disabled.

1 Interrupt signaling collision or other I2C protocol error is enabled.

SLADDRIE *Slave Addressed Interrupt Enable*

0 Interrupt signaling reception of a matching address packet is disabled.

1 Interrupt signaling reception of a matching 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 Slave transmission mode only).

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

1 Interrupt signaling reception of the ACK bit in response to the data packet is enabled (in the Slave transmission mode only).

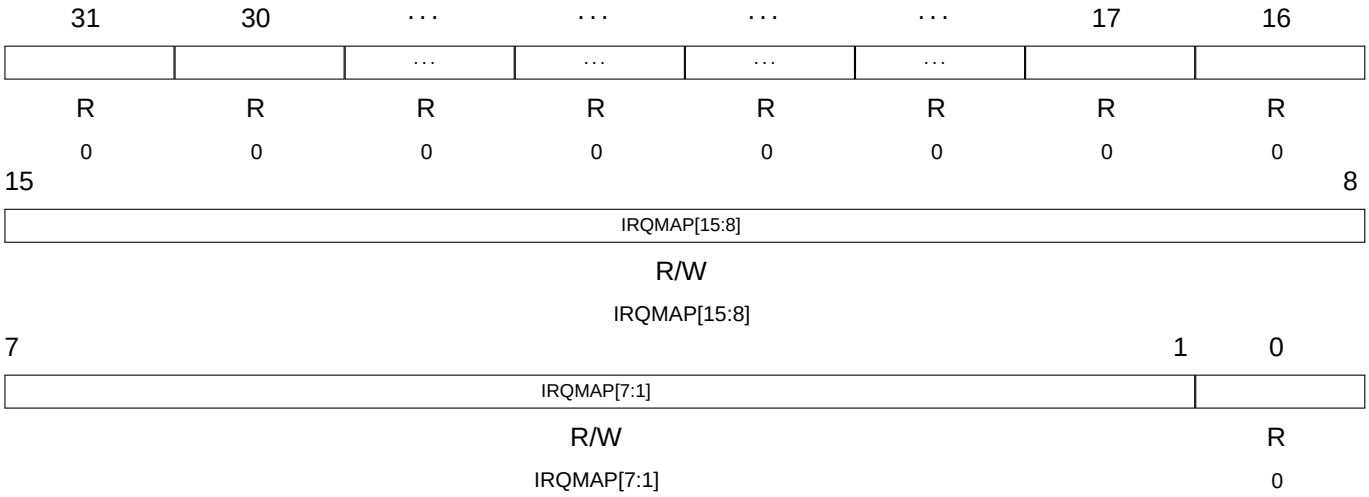
CNT0IE *Count Register Equals 0 Interrupt Enable*

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.

1.8 Implementation

1.8.1 Design Structure

The synthesible RTL IP core part (*AXI/rtl*, *COMMON/rtl* and *I2C_SLV/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*AXI/tb* and *I2C_SLV/tb* folder) uses SystemVerilog language.

```
AXI
├── rtl
│   ├── AXI_PERIPH
│   │   └── amba_axilite_apb_bridge.v
└── tb
    ├── AXI_PERIPH
    │   ├── APB
    │   │   └── virtual_APB_slave.sv
    │   ├── AXI
    │   │   └── tb_amba_axilite_tasks.sv
    │   ├── common
    │   │   └── timescale.v
    │   ├── run
    │   │   └── ncvlog_amba_axilite_apb_bridge.sh
    │   ├── tests
    │   │   └── tb_read_write_test.sv
    │   └── tb_amba_axilite_apb_bridge.sv
COMMON
├── rtl
    ├── DFF_en.v
    ├── edge_detector.v
    ├── input_filter.v
    └── synchronizer.v
I2C_SLV
├── beh
├── rtl
│   ├── APB_I2C_SLV.v
│   ├── AXILITE_I2C_SLV.v
│   ├── I2C_SLV.v
│   ├── I2C_SLV_defines.v
│   └── I2C_SLV_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_master.sv
    │   └── virtual_I2C_master_2.sv
    ├── run
    │   └── irun_apb_i2c_slv.sh
    ├── tests
    │   ├── tb_interrupt_MAPPING_test.v
    │   └── tb_RegisterAccess_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_DATA_NACK_INTERRUPT_test.v
    ├── tb_RX_32bytes_DATA_NACK_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_address_match_handling_INTERRUPT_test.v
    ├── tb_RX_address_match_handling_test.v
    ├── tb_RX_BusError_handling_INTERRUPT_test.v
    ├── tb_RX_BusError_handling_test.v
    ├── tb_RX_PDMA_1byte_INTERRUPT_test.v
    ├── tb_RX_PDMA_1byte_test.v
    ├── tb_RX_PDMA_32bytes_INTERRUPT_test.v
    ├── tb_RX_PDMA_32bytes_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_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_address_match_handling_INTERRUPT_test.v
    ├── tb_TX_address_match_handling_test.v
    ├── tb_TX_BusError_handling_INTERRUPT_test.v
    ├── tb_TX_BusError_handling_test.v
    ├── tb_TX_PDMA_1byte_INTERRUPT_test.v
    ├── tb_TX_PDMA_1byte_test.v
    ├── tb_TX_PDMA_32bytes_INTERRUPT_test.v
    └── tb_TX_PDMA_32bytes_test.v
  └── tb_APB_I2C_SLV.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. The testbench is divided into two environments. The first one tests the APB_I2C_SLV module. To run the simulation using Cadence® Incisive® Enterprise Simulator run *irun_apb_i2c_slv.sh* script located in the *I2C_SLV/tb/run* folder. The simulation should end with reporting no errors. The second environment tests the AXI4-Lite to APB3 converter. To run the simulation using Cadence® Incisive® Enterprise Simulator run *ncvlog_amba_axilite_apb_bridge.sh* script located in the *AXI/tb/AXI_PERIPH/run* folder. The simulation should end with reporting no errors. The AXILITE_-I2C_SLV top module is composed of the APB_I2C_SLV core and the amba_axilite_apb_bridge AXI4-Lite to APB3 converter.

1.8.3 Clock and Reset

The CC-I2C_SLV-AXI 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
i2cHoldPeriodWidth	Configure hold period counter width (sum of hold and setup widths can't exceed 32)	1:32	8
i2cSetupPeriodWidth	Configure setup period counter width (sum of hold and setup widths can't exceed 32)	1:32	8
i2cPeriodCounterWidth	Configure clock prescaler width (max of i2cHoldPeriodWidth and i2cSetupPeriodWidth)	1:32	8
i2cCountWidth	Configure data counter width	1:32	16
i2cSignalSampleCountWidth	Configure input sample counter width	1:32	4
dual_address_support_enable	Configure support for secondary address recognition	0,1	4
interface_inputs _synchronization_enable	Configure SDA and SCL synchronizing flip flops for I2C slave	0,1	1
interface_inputs_filter_enable	Configure SDA and SCL input filter	0,1	1
default_interrupt_MAPPING	Reset value of interrupt_MAPPING register	1:32767	0

1.8.6 Signals Description

Signal name	Description	I/O	Active	Type
ACLK	Synchronous clock	I	rising	clock
ARESETn	Asynchronous reset	I	low	reset
AWADDR[5:2]	AXI4-Lite write address	I	data	comb.
AWPROT[2:0][1]	AXI4-Lite write address protection type	I	data	comb.
AWVALID	AXI4-Lite write address valid	I	high	comb.
AWREADY	AXI4-Lite write address ready	O	high	reg.
WDATA[31:0]	AXI4-Lite write data	I	data	comb.
WSTRB[3:0]	AXI4-Lite write strobe	I	high	comb.
WVALID	AXI4-Lite write valid	I	high	comb.
WREADY	AXI4-Lite write ready	O	high	reg.
BRESP[1:0]	AXI4-Lite write response	O	data	reg.
BVALID	AXI4-Lite write response valid	O	high	reg.
BREADY	AXI4-Lite write response ready	I	high	comb.
ARADDR[5:2]	AXI4-Lite read address	I	data	comb.
ARPROT[2:0][1]	AXI4-Lite read address protection type	I	data	comb.
ARVALID	AXI4-Lite read address valid	I	high	comb.

ARREADY	AXI4-Lite read address ready	O	high	reg.
RDATA[31:0]	AXI4-Lite read data	O	data	reg.
RRESP[1:0]	AXI4-Lite read response	O	data	reg.
RVALID	AXI4-Lite read valid	O	high	reg.
RREADY	AXI4-Lite read ready	I	high	comb.
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_DNACK	Data NACK received interrupt	O	high	reg.
interrupt_DACK	Data ACK received interrupt	O	high	reg.
interrupt_CountEqu0	Count equals zero interrupt	O	high	reg.
interrupt_SlaveAddressed	Module addressed interrupt	O	high	reg.
interrupt_BusError	Protocol error 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[2]
SCL_pad_input	I2C clock pin input	I	As I2C spec.	reg./comb[2]
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] Signal is not used in the design.
[2] Depending on interface_inputs_synchronization_enable setting, input synchronization flip-flops may be inserted.

1.8.7 Instantiation

```
icg
    icg_i2c_x  (     .E(ARVALID|AWVALID|i2c_clock_request),
                     .clk(ACLK),
                     .gclk(i2c_clk),
                     .scan_enable(scan_enable));

AXILITE_I2C_SLV   #( .PDMA_support(CFG_DMA_EN),
                     .default_interrupt_MAPPING(CFG_DEF_INT_MAPPING))

AXILITE_I2C_SLV_u  ( .ACLK(i2c_clk),
                     .ARESETn(ARESETn),
                     .AWADDR(AWADDR[5:2]),
                     .AWPROT(AWPROT),
                     .AWVALID(AWVALID),
                     .AWREADY(AWREADY),
                     .WDATA(WDATA),
                     .WSTRB(WSTRB),
                     .WVALID(WVALID),
                     .WREADY(WREADY),
                     .BRESP(BRESP),
                     .BVALID(BVALID),
                     .BREADY(BREADY),
                     .ARADDR(ARADDR[5:2]),
                     .ARPROT(ARPROT),
                     .ARVALID(ARVALID),
                     .ARREADY(ARREADY),
                     .RDATA(RDATA),
                     .RRESP(RRESP),
                     .RVALID(RVALID),
                     .RREADY(RREADY),
                     .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_BusError(i2c_interrupt_BusError),
                     .interrupt_CountEqu0(i2c_interrupt_CountEqu0),
```


```
                    .interrupt_SlaveAddressed(i2c_interrupt_SlaveAddressed),
                    .interrupt_DACK(i2c_interrupt_DataACK),
                    .interrupt_DNACK(i2c_interrupt_DataNACK),
                    .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_BusError |
                        i2c_interrupt_CountEqu0 | i2c_interrupt_SlaveAddressed |
                        i2c_interrupt_DataNACK | i2c_interrupt_DataACK;

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
1.0	11-2017	First Issue.

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

 CC-I2C_SLV-AXI-Doc_112017.