24 Inter-integrated circuit (I^2C) interface

24.1 I^2C introduction

I^2C (inter-integrated circuit) bus Interface serves as an interface between the microcontroller and the serial I^2C bus. It provides multimaster capability, and controls all I^2C bus-specific sequencing, protocol, arbitration and timing. It supports the standard mode (Sm, up to 100 kHz) and Fm mode (Fm, up to 400 kHz).

It may be used for a variety of purposes, including CRC generation and verification, SMBus (system management bus) and PMBus (power management bus).

Depending on specific device implementation DMA capability can be available for reduced CPU overload.

24.2 I^{2}C main features

- Parallel-bus/I^{2}C protocol converter
- Multimaster capability: the same interface can act as Master or Slave
- I^{2}C Master features:
  - Clock generation
  - Start and Stop generation
- I^{2}C Slave features:
  - Programmable I^{2}C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
  - Standard Speed (up to 100 kHz)
  - Fast Speed (up to 400 kHz)
- Analog noise filter
- Programmable digital noise filter
- Status flags:
  - Transmitter/Receiver mode flag
  - End-of-Byte transmission flag
  - I^{2}C busy flag
- Error flags:
  - Arbitration lost condition for master mode
  - Acknowledgment failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/Underrun if clock stretching is disabled
- 2 Interrupt vectors:
  - 1 Interrupt for successful address/ data communication
  - 1 Interrupt for error condition
- Optional clock stretching
- 1-byte buffer with DMA capability
- Configurable PEC (packet error checking) generation or verification:
  - PEC value can be transmitted as last byte in Tx mode
  - PEC error checking for last received byte
- SMBus 2.0 Compatibility:
  - 25 ms clock low timeout delay
  - 10 ms master cumulative clock low extend time
  - 25 ms slave cumulative clock low extend time
  - Hardware PEC generation/verification with ACK control
  - Address Resolution Protocol (ARP) supported
- PMBus Compatibility

Note: Some of the above features may not be available in certain products. The user should refer to the product data sheet, to identify the specific features supported by the I^2C interface implementation.

24.3 I^2C functional description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I^2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz) or fast (up to 400 kHz) I^2C bus.

24.3.1 Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

Communication flow

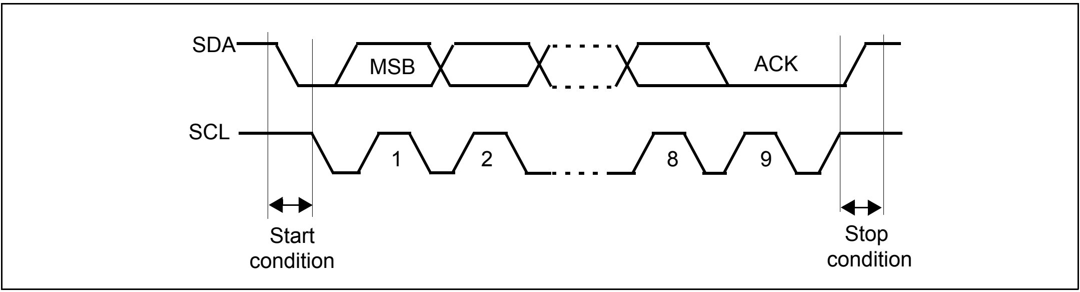
In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to *Figure 271*.

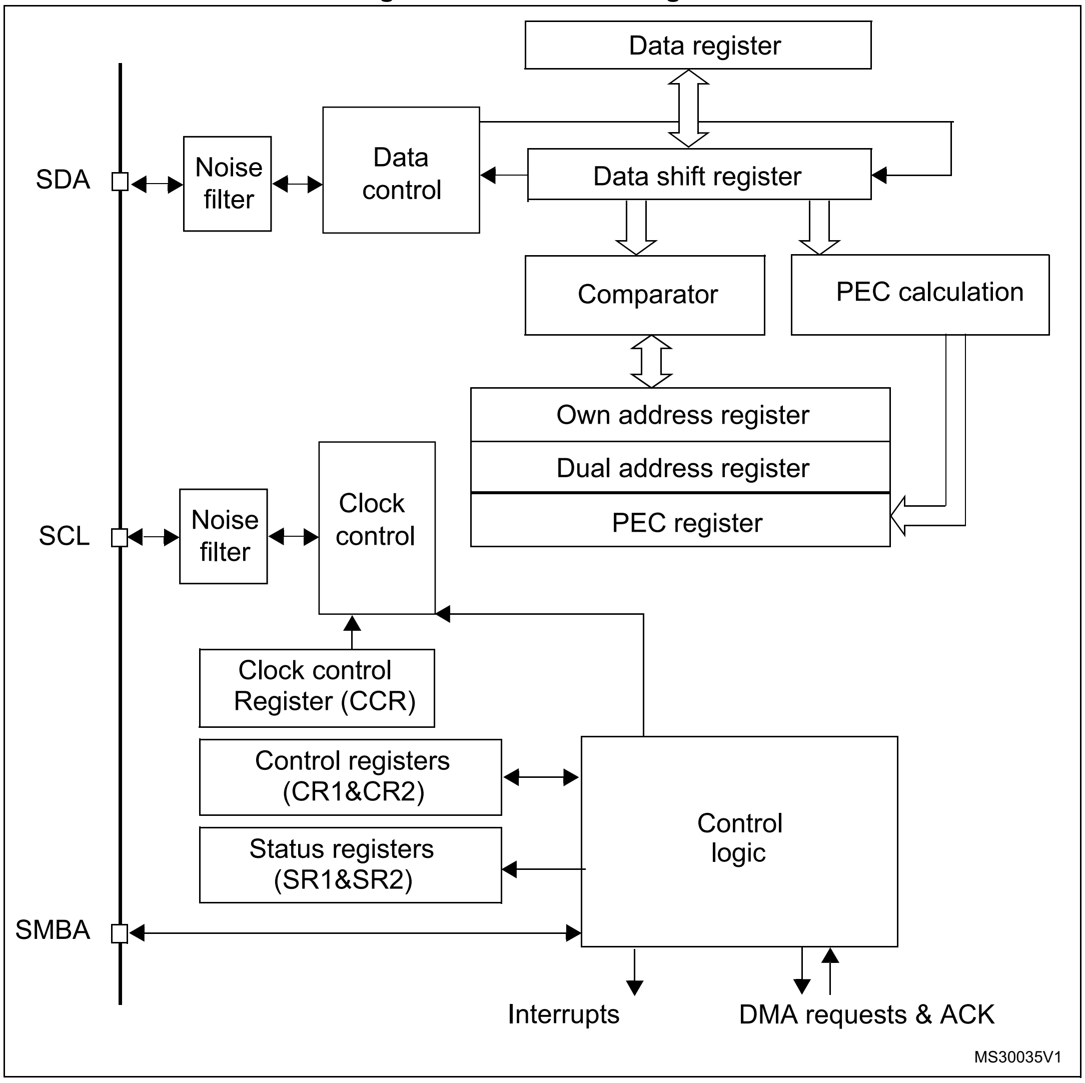
Figure 271. I^2C bus protocol



Acknowledge may be enabled or disabled by software. The I^2C interface addresses (dual addressing 7-bit/ 10-bit and/or general call address) can be selected by software.

The block diagram of the I^2C interface is shown in *Figure 272*.

Figure 272. I^2C block diagram



1. SMBA is an optional signal in SMBus mode. This signal is not applicable if SMBus is disabled.

24.3.2 I^2C slave mode

By default the I^2C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

The peripheral input clock must be programmed in the I2C_CR2 register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 2 MHz in Sm mode
- 4 MHz in Fm mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OAR1) and with OAR2 (if ENDUAL=1) or the General Call address (if ENGC = 1).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

Header or address not matched: the interface ignores it and waits for another Start condition.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

Address matched: the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.
- If ENDUAL=1, the software has to read the DUALF bit to check which slave address has been acknowledged.

In 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It enters Transmitter mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

Slave transmitter

Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

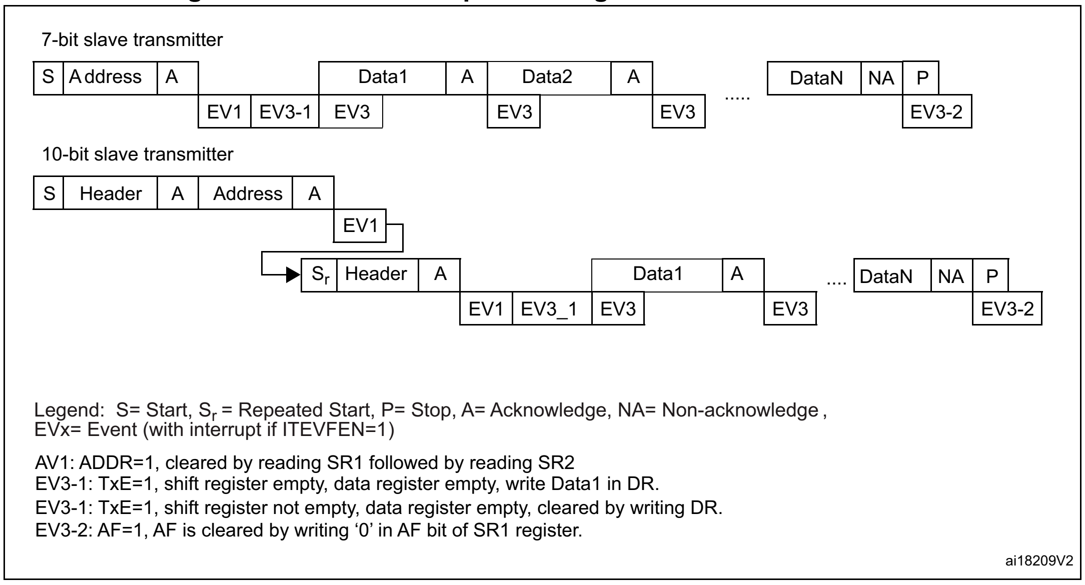
The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see *Figure 273* Transfer sequencing EV1 EV3).

When the acknowledge pulse is received:

- The TxE bit is set by hardware with an interrupt if the ITEVFEN and the ITBUFEN bits are set.

If TxE is set and some data were not written in the I2C_DR register before the end of the next data transmission, the BTF bit is set and the interface waits until BTF is cleared by a read to I2C_SR1 followed by a write to the I2C_DR register, stretching SCL low.

Figure 273. Transfer sequence diagram for slave transmitter



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

AV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV3-1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV3-1: TxE=1, shift register not empty, data register empty, cleared by writing DR.
EV3-2: AF=1, AF is cleared by writing ‘0’ in AF bit of SR1 register.

1. The EV1 and EV3_1 events stretch SCL low until the end of the corresponding software sequence.
2. The EV3 event stretches SCL low if the software sequence is not completed before the end of the next byte transmission

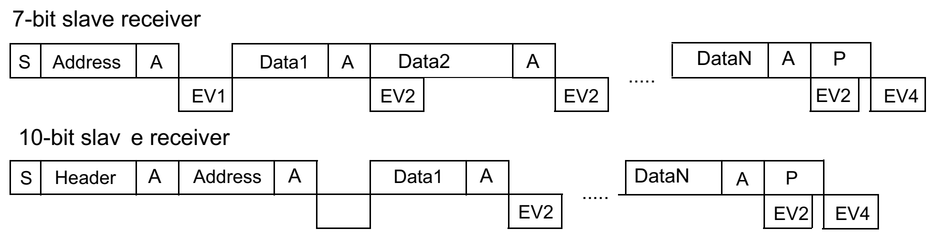
Slave receiver

Following the address reception and after clearing ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bit is set.

If RxNE is set and the data in the DR register is not read before the end of the next data reception, the BTF bit is set and the interface waits until BTF is cleared by a read from the I2C_DR register, stretching SCL low (see *Figure 274*).

Figure 274. Transfer sequence diagram for slave receiver



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge,
EVx= Event (with interrupt if ITEVFEN=1)

EV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV2: RxNE=1 cleared by reading DR register.
EV4: STOPF=1, cleared by reading SR1 register followed by writing to the CR1 register

ai18208V2

1. The EV1 event stretches SCL low until the end of the corresponding software sequence.
2. The EV2 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
3. After checking the SR1 register content, the user should perform the complete clearing sequence for each flag found set.
   Thus, for ADDR and STOPF flags, the following sequence is required inside the I2C interrupt routine:
   READ SR1
   if (ADDR == 1) {READ SR1; READ SR2}
   if (STOPF == 1) {READ SR1; WRITE CR1}
   The purpose is to make sure that both ADDR and STOPF flags are cleared if both are found set.

Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets:

- The STOPF bit and generates an interrupt if the ITEVFEN bit is set.

The STOPF bit is cleared by a read of the SR1 register followed by a write to the CR1 register (see *Figure 274: Transfer sequence diagram for slave receiver* EV4).

24.3.3 I^{2}C master mode

In Master mode, the I^{2}C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a Start condition and ends with a Stop condition. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

- Program the peripheral input clock in I2C_CR2 Register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR1 register to generate a Start condition

The peripheral input clock frequency must be at least:

- 2 MHz in Sm mode
- 4 MHz in Fm mode

SCL master clock generation

The CCR bits are used to generate the high and low level of the SCL clock, starting from the generation of the rising and falling edge (respectively). As a slave may stretch the SCL line, the peripheral checks the SCL input from the bus at the end of the time programmed in TRISE bits after rising edge generation.

- If the SCL line is low, it means that a slave is stretching the bus, and the high level counter stops until the SCL line is detected high. This allows to guarantee the minimum HIGH period of the SCL clock parameter.
- If the SCL line is high, the high level counter keeps on counting.

Indeed, the feedback loop from the SCL rising edge generation by the peripheral to the SCL rising edge detection by the peripheral takes time even if no slave stretches the clock. This loopback duration is linked to the SCL rising time (impacting SCL VIH input detection), plus delay due to the noise filter present on the SCL input path, plus delay due to internal SCL input synchronization with APB clock. The maximum time used by the feedback loop is programmed in the TRISE bits, so that the SCL frequency remains stable whatever the SCL rising time.

Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (MSL bit set) when the BUSY bit is cleared.

Note: In master mode, setting the START bit causes the interface to generate a ReStart condition at the end of the current byte transfer.

Once the Start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the Slave address (see *Figure 275* and *Figure 276* Transfer sequencing EV5).

Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

- In 10-bit addressing mode, sending the header sequence causes the following event:
  - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (see *Figure 275* and *Figure 276* Transfer sequencing).
  - The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see *Figure 275* and *Figure 276* Transfer sequencing).
- In 7-bit addressing mode, one address byte is sent.

  As soon as the address byte is sent,
  - The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see *Figure 275* and *Figure 276* Transfer sequencing).

The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.

- In 7-bit addressing mode,
  - To enter Transmitter mode, a master sends the slave address with LSB reset.
  - To enter Receiver mode, a master sends the slave address with LSB set.
- In 10-bit addressing mode,
  - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address, (where xx denotes the two most significant bits of the address).
  - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

  The TRA bit indicates whether the master is in Receiver or Transmitter mode.

Master transmitter

Following the address transmission and after clearing ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

The master waits until the first data byte is written into I2C_DR (see *Figure 275* Transfer sequencing EV8_1).

When the acknowledge pulse is received, the TxE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set.

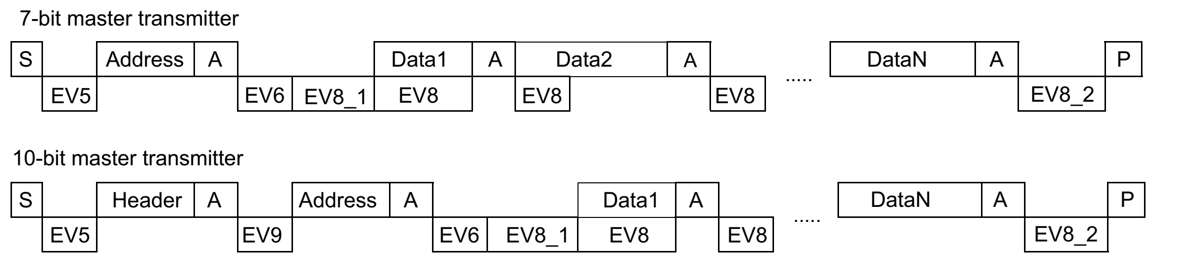
If TxE is set and a data byte was not written in the DR register before the end of the last data transmission, BTF is set and the interface waits until BTF is cleared by a write to I2C_DR, stretching SCL low.

Closing the communication

After the last byte is written to the DR register, the STOP bit is set by software to generate a Stop condition (see *Figure 275* Transfer sequencing EV8_2). The interface automatically goes back to slave mode (MSL bit cleared).

Note: Stop condition should be programmed during EV8_2 event, when either TxE or BTF is set.

Figure 275. Transfer sequence diagram for master transmitter



Legend: S = Start, SR = Repeated start, P = stop, A = Acknowledge
EVx = Event (with interrupt if ITEVFEN = 1)
EV5: SB=1, cleared by reading SR1 register followed by writing DR register with address.
EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2.
EV8_1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV8: TxE=1, shift register not empty, data register empty, cleared by writing DR register.
EV_2: TxE=1, BTF=1, Program stop request, TxE and BTF are cleared by hardware by the stop condition.
EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

ai18210V2

1. The EV5, EV6, EV9, EV8_1 and EV8_2 events stretch SCL low until the end of the corresponding software sequence.
2. The EV8 event stretches SCL low if the software sequence is not complete before the end of the next byte transmission.

Master receiver

Following the address transmission and after clearing ADDR, the I^2C interface enters Master Receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

1. An acknowledge pulse if the ACK bit is set
2. The RxNE bit is set and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set (see *Figure 276* Transfer sequencing EV7).

If the RxNE bit is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set by hardware and the interface waits until BTF is cleared by a read in the DR register, stretching SCL low.

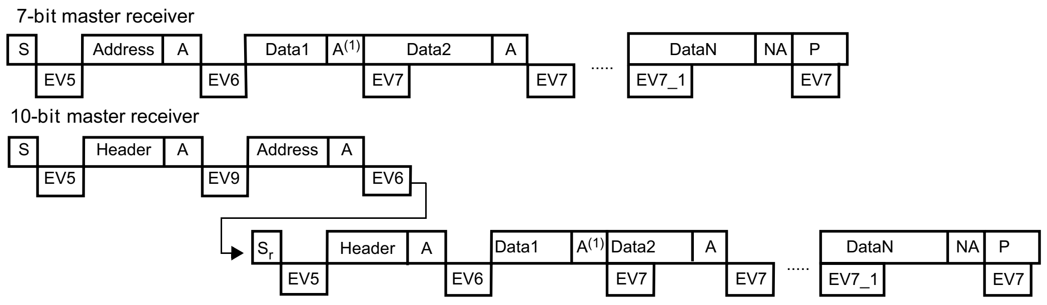
Closing the communication

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Restart condition.

1. To generate the nonacknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RxNE event).
2. In order to generate the Stop/Restart condition, software must set the STOP/START bit after reading the second last data byte (after the second last RxNE event).
3. In case a single byte has to be received, the Acknowledge disable is made during EV6 (before ADDR flag is cleared) and the STOP condition generation is made after EV6.

After the Stop condition generation, the interface goes automatically back to slave mode (MSL bit cleared).

Figure 276. Transfer sequence diagram for master receiver



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge,
EVx= Event (with interrupt if ITEVFEN=1)

EV5: SB=1, cleared by reading SR1 register followed by writing DR register.

EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
In case of the reception of 1 byte, the Acknowledge disable must be performed during EV6 event, i.e. before clearing ADDR flag.

EV7: RxNE=1 cleared by reading DR register.

EV7_1: RxNE=1 cleared by reading DR register, program ACK=0 and STOP request

EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

ai17540d

1. If a single byte is received, it is NA.
2. The EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. The EV7 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
4. The EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

The procedures described below are recommended if the EV7-1 software sequence is not completed before the ACK pulse of the current byte transfer.

These procedures must be followed to make sure:

- The ACK bit is set low on time before the end of the last data reception
- The STOP bit is set high after the last data reception without reception of supplementary data.

For 2-byte reception:

- Wait until ADDR = 1 (SCL stretched low until the ADDR flag is cleared)
- Set ACK low, set POS high
- Clear ADDR flag
- Wait until BTF = 1 (Data 1 in DR, Data2 in shift register, SCL stretched low until a data 1 is read)
- Set STOP high
- Read data 1 and 2

For N >2 -byte reception, from N-2 data reception

- Wait until BTF = 1 (data N-2 in DR, data N-1 in shift register, SCL stretched low until data N-2 is read)
- Set ACK low
- Read data N-2
- Wait until BTF = 1 (data N-1 in DR, data N in shift register, SCL stretched low until a data N-1 is read)
- Set STOP high
- Read data N-1 and N

24.3.4 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

This error occurs when the I^{2}C interface detects an external Stop or Start condition during an address or a data transfer. In this case:

- the BERR bit is set and an interrupt is generated if the ITERREN bit is set
- in Slave mode: data are discarded and the lines are released by hardware:
  - in case of a misplaced Start, the slave considers it is a restart and waits for an address, or a Stop condition
  - in case of a misplaced Stop, the slave behaves like for a Stop condition and the lines are released by hardware
- In Master mode: the lines are not released and the state of the current transmission is not affected. It is up to the software to abort or not the current transmission

Acknowledge failure (AF)

This error occurs when the interface detects a nonacknowledge bit. In this case:

- the AF bit is set and an interrupt is generated if the ITERREN bit is set
- a transmitter which receives a NACK must reset the communication:
  - If Slave: lines are released by hardware
  - If Master: a Stop or repeated Start condition must be generated by software

Arbitration lost (ARLO)

This error occurs when the I^{2}C interface detects an arbitration lost condition. In this case,

- the ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set)
- the I^{2}C Interface goes automatically back to slave mode (the MSL bit is cleared). When the I^{2}C loses the arbitration, it is not able to acknowledge its slave address in the same transfer, but it can acknowledge it after a repeated Start from the winning master.
- lines are released by hardware

Overrun/underrun error (OVR)

An overrun can occur in slave mode when clock stretching is disabled and the I^{2}C interface is receiving data. The interface has received a byte (RxNE=1) and the data in DR has not been read, before the next byte is received by the interface. In this case,

- The last received byte is lost.
- In case of Overrun error, software should clear the RxNE bit and the transmitter should re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I^{2}C interface is transmitting data. The interface has not updated the DR with the next byte (TxE=1), before the clock comes for the next byte. In this case,

- The same byte in the DR register is sent again
- The user should make sure that data received on the receiver side during an underrun error are discarded and that the next bytes are written within the clock low time specified in the I^{2}C bus standard.

For the first byte to be transmitted, the DR must be written after ADDR is cleared and before the first SCL rising edge. If not possible, the receiver must discard the first data.

24.3.5 Programmable noise filter

In Fm mode, the I^{2}C standard requires that spikes are suppressed to a length of 50 ns on SDA and SCL lines.

An analog noise filter is implemented in the SDA and SCL I/Os. This filter is enabled by default and can be disabled by setting the ANOFF bit in the I2C_FLTR register.

A digital noise filter can be enabled by configuring the DNF[3:0] bits to a non-zero value. This suppresses the spikes on SDA and SCL inputs with a length of up to DNF[3:0] * T_{PCLK1}.

Enabling the digital noise filter increases the SDA hold time by (DNF[3:0] +1)* T_{PCLK}.

To be compliant with the maximum hold time of the I^{2}C-bus specification version 2.1 (Thd:dat), the DNF bits must be programmed using the constraints shown in *Table 144*, and assuming that the analog filter is disabled.

Note: DNF[3:0] must only be configured when the I^{2}C is disabled (PE = 0). If the analog filter is also enabled, the digital filter is added to the analog filter.

Table 144. Maximum DNF[3:0] value to be compliant with Thd:dat(max)

PCLK1 frequency	Maximum DNF value	
	Sm mode	Fm mode
2 <= F_{PCLK1} <= 5	2	0
5 < F_{PCLK1} <= 10	12	0
10 < F_{PCLK1} <= 20	15	1
20 < F_{PCLK1} <= 30	15	7
30 < F_{PCLK1} <= 40	15	13
40 < F_{PCLK1} <= 50	15	15

Note: For each frequency range, the constraint is given based on the worst case which is the minimum frequency of the range. Greater DNF values can be used if the system can support maximum hold time violation.

24.3.6 SDA/SCL line control

- If clock stretching is enabled:
  - Transmitter mode: If TxE=1 and BTF=1: the interface holds the clock line low before transmission to wait for the microcontroller to write the byte in the Data Register (both buffer and shift register are empty).
  - Receiver mode: If RxNE=1 and BTF=1: the interface holds the clock line low after reception to wait for the microcontroller to read the byte in the Data Register (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
  - Overrun Error in case of RxNE=1 and no read of DR has been done before the next byte is received. The last received byte is lost.
  - Underrun Error in case TxE=1 and no write into DR has been done before the next byte must be transmitted. The same byte is sent again.
  - Write Collision not managed.

24.3.7 SMBus

Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I^2C principles of operation. SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of toggling individual control lines.

The System Management Bus Specification refers to three types of devices. A *slave* is a device that is receiving or responding to a command. A *master* is a device that issues commands, generates the clocks, and terminates the transfer. A *host* is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

Similarities between SMBus and I^2C

- 2 wire bus protocol (1 Clk, 1 Data) + SMBus Alert line optional
- Master-slave communication, Master provides clock
- Multi master capability
- SMBus data format similar to I^2C 7-bit addressing format (*Figure 271*).

Differences between SMBus and I^2C

The following table describes the differences between SMBus and I^2C.

Table 145. SMBus vs. I^2C

SMBus	I^2C
Max. speed 100 kHz	Max. speed 400 kHz
Min. clock speed 10 kHz	No minimum clock speed
35 ms clock low timeout	No timeout
Logic levels are fixed	Logic levels are V_{DD} dependent
Different address types (reserved, dynamic etc.)	7-bit, 10-bit and general call slave address types
Different bus protocols (quick command, process call etc.)	No bus protocols

SMBus application usage

With System Management Bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status. SMBus provides a control bus for system and power management related tasks.

Device identification

Any device that exists on the System Management Bus as a slave has a unique address called the Slave Address. For the list of reserved slave addresses, refer to the SMBus specification version. 2.0 (http://smbus.org/).

Bus protocols

The SMBus specification supports up to 9 bus protocols. For more details of these protocols and SMBus address types, refer to SMBus specification version. 2.0. These protocols should be implemented by the user software.

Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. The Address Resolution Protocol (ARP) has the following attributes:

- Address assignment uses the standard SMBus physical layer arbitration mechanism
- Assigned addresses remain constant while device power is applied; address retention through device power loss is also allowed
- No additional SMBus packet overhead is incurred after address assignment. (i.e. subsequent accesses to assigned slave addresses have the same overhead as accesses to fixed address devices.)
- Any SMBus master can enumerate the bus

Unique device identifier (UDID)

In order to provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique device identifier (UDID).

For the details on 128 bit UDID and more information on ARP, refer to SMBus specification version 2.0.

SMBus alert mode

SMBus Alert is an optional signal with an interrupt line for devices that want to trade their ability to master for a pin. SMBA is a wired-AND signal just as the SCL and SDA signals are. SMBA is used in conjunction with the SMBus General Call Address. Messages invoked with the SMBus are 2 bytes long.

A slave-only device can signal the host through SMBA that it wants to talk by setting ALERT bit in I2C_CR1 register. The host processes the interrupt and simultaneously accesses all SMBA devices through the *Alert Response Address* (known as ARA having a value 0001 100X). Only the device(s) which pulled SMBA low acknowledges the alert Response address. This status is identified using SMBALERT Status flag in I2C_SR1 register. The host performs a modified Receive Byte operation. The 7 bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.

If more than one device pulls SMBA low, the highest priority (lowest address) device wins communication rights via standard arbitration during the slave address transfer. After acknowledging the slave address the device must disengage its SMBA pull-down. If the host still sees SMBA low when the message transfer is complete, it knows to read the ARA again.
A host which does not implement the SMBA signal may periodically access the ARA.

For more details on SMBus Alert mode, refer to SMBus specification version 2.0 (http://smbus.org/).

Timeout error

There are differences in the timing specifications between I^2C and SMBus.
SMBus defines a clock low timeout, TIMEOUT of 35 ms. Also SMBus specifies TLOW: SEXT as the cumulative clock low extend time for a slave device. SMBus specifies TLOW: MEXT as the cumulative clock low extend time for a master device. For more details on these timeouts, refer to SMBus specification version 2.0.

The status flag Timeout or Tlow Error in I2C_SR1 shows the status of this feature.

How to use the interface in SMBus mode

To switch from I^2C mode to SMBus mode, the following sequence should be performed.

- Set the SMBus bit in the I2C_CR1 register
- Configure the SMBTYPE and ENARP bits in the I2C_CR1 register as required for the application

If you want to configure the device as a master, follow the Start condition generation procedure in *Section 24.3.3: I2C master mode*. Otherwise, follow the sequence in *Section 24.3.2: I2C slave mode*.

The application has to control the various SMBus protocols by software.

- SMB Device Default Address acknowledged if ENARP=1 and SMBTYPE=0
- SMB Host Header acknowledged if ENARP=1 and SMBTYPE=1
- SMB Alert Response Address acknowledged if SMBALERT=1

24.3.8 DMA requests

DMA requests (when enabled) are generated only for data transfer. DMA requests are generated by Data Register becoming empty in transmission and Data Register becoming full in reception. The DMA must be initialized and enabled before the I2C data transfer. The DMAEN bit must be set in the I2C_CR2 register before the ADDR event. In master mode or in slave mode when clock stretching is enabled, the DMAEN bit can also be set during the ADDR event, before clearing the ADDR flag. The DMA request must be served before the end of the current byte transfer. When the number of data transfers which has been programmed for the corresponding DMA stream is reached, the DMA controller sends an End of Transfer EOT signal to the I^2C interface and generates a Transfer Complete interrupt if enabled:

- Master transmitter: In the interrupt routine after the EOT interrupt, disable DMA requests then wait for a BTF event before programming the Stop condition.
- Master receiver
  - When the number of bytes to be received is equal to or greater than two, the DMA controller sends a hardware signal, EOT_1, corresponding to the last but one data byte (number_of_bytes – 1). If, in the I2C_CR2 register, the LAST bit is set, I^2C automatically sends a NACK after the next byte following EOT_1. The user can generate a Stop condition in the DMA Transfer Complete interrupt routine if enabled.
  - When a single byte must be received: the NACK must be programmed during EV6 event, i.e. program ACK=0 when ADDR=1, before clearing ADDR flag. Then the user can program the STOP condition either after clearing ADDR flag, or in the DMA Transfer Complete interrupt routine.

Transmission using DMA

DMA mode can be enabled for transmission by setting the DMAEN bit in the I2C_CR2 register. Data are loaded from a Memory area configured using the DMA peripheral (refer to the DMA specification) to the I2C_DR register whenever the TxE bit is set. To map a DMA stream x for I^2C transmission (where x is the stream number), perform the following sequence:

1. Set the I2C_DR register address in the DMA_SxPAR register. The data are moved to this address from the memory after each TxE event.
2. Set the memory address in the DMA_SxMA0R register (and in DMA_SxMA1R register in the case of a bouble buffer mode). The data are loaded into I2C_DR from this memory after each TxE event.
3. Configure the total number of bytes to be transferred in the DMA_SxNDTR register. After each TxE event, this value is decremented.
4. Configure the DMA stream priority using the PL[0:1] bits in the DMA_SxCR register
5. Set the DIR bit in the DMA_SxCR register and configure interrupts after half transfer or full transfer depending on application requirements.
6. Activate the stream by setting the EN bit in the DMA_SxCR register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I^2C interface and the DMA generates an interrupt, if enabled, on the DMA stream interrupt vector.

Note: Do not enable the ITBUFEN bit in the I2C_CR2 register if DMA is used for transmission.

Reception using DMA

DMA mode can be enabled for reception by setting the DMAEN bit in the I2C_CR2 register. Data are loaded from the I2C_DR register to a Memory area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA stream x for I^2C reception (where x is the stream number), perform the following sequence:

1. Set the I2C_DR register address in DMA_SxPAR register. The data are moved from this address to the memory after each RxNE event.
2. Set the memory address in the DMA_SxMA0R register (and in DMA_SxMA1R register in the case of a bouble buffer mode). The data are loaded from the I2C_DR register to this memory area after each RxNE event.
3. Configure the total number of bytes to be transferred in the DMA_SxNDTR register. After each RxNE event, this value is decremented.
4. Configure the stream priority using the PL[0:1] bits in the DMA_SxCR register
5. Reset the DIR bit and configure interrupts in the DMA_SxCR register after half transfer or full transfer depending on application requirements.
6. Activate the stream by setting the EN bit in the DMA_SxCR register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I^2C interface and DMA generates an interrupt, if enabled, on the DMA stream interrupt vector.

Note: Do not enable the ITBUFEN bit in the I2C_CR2 register if DMA is used for reception.

24.3.9 Packet error checking

A PEC calculator has been implemented to improve the reliability of communication. The PEC is calculated by using the $C(x) = x^8 + x^2 + x + 1$ CRC-8 polynomial serially on each bit.

- PEC calculation is enabled by setting the ENPEC bit in the I2C_CR1 register. PEC is a CRC-8 calculated on all message bytes including addresses and R/W bits.
  - In transmission: set the PEC transfer bit in the I2C_CR1 register after the TxE event corresponding to the last byte. The PEC is transferred after the last transmitted byte.
  - In reception: set the PEC bit in the I2C_CR1 register after the RxNE event corresponding to the last byte so that the receiver sends a NACK if the next received byte is not equal to the internally calculated PEC. In case of Master-Receiver, a NACK must follow the PEC whatever the check result. The PEC must

be set before the ACK of the CRC reception in slave mode. It must be set when the ACK is set low in master mode.

- A PECERR error flag/interrupt is also available in the I2C_SR1 register.
- If DMA and PEC calculation are both enabled:-
  - In transmission: when the I^2C interface receives an EOT signal from the DMA controller, it automatically sends a PEC after the last byte.
  - In reception: when the I^2C interface receives an EOT_1 signal from the DMA controller, it automatically considers the next byte as a PEC and checks it. A DMA request is generated after PEC reception.
- To allow intermediate PEC transfers, a control bit is available in the I2C_CR2 register (LAST bit) to determine if it is really the last DMA transfer or not. If it is the last DMA request for a master receiver, a NACK is automatically sent after the last received byte.
- PEC calculation is corrupted by an arbitration loss.

24.4 I^2C interrupts

The table below gives the list of I^2C interrupt requests.

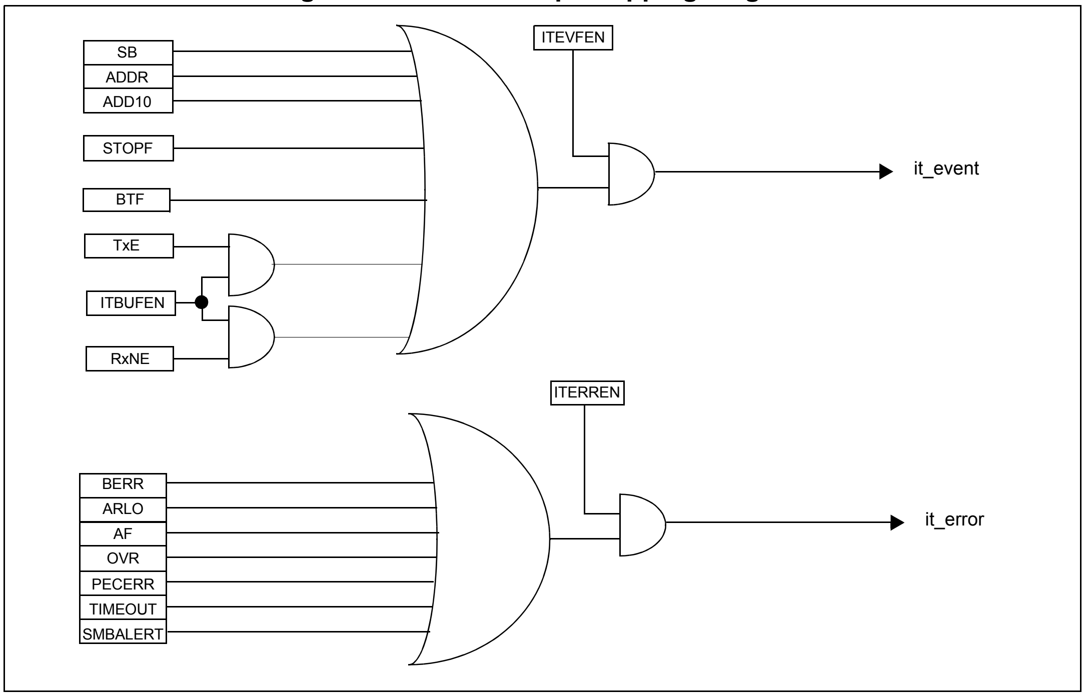
Table 146. I^2C Interrupt requests

Interrupt event	Event flag	Enable control bit
Start bit sent (Master)	SB	ITEVFEN
Address sent (Master) or Address matched (Slave)	ADDR	
10-bit header sent (Master)	ADD10	
Stop received (Slave)	STOPF	
Data byte transfer finished	BTF	
Receive buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Bus error	BERR	ITERREN
Arbitration loss (Master)	ARLO	
Acknowledge failure	AF	
Overrun/Underrun	OVR	
PEC error	PECERR	
Timeout/Tlow error	TIMEOUT	
SMBus Alert	SMBALERT	

Note: *SB, ADDR, ADD10, STOPF, BTF, RxNE and TxE are logically ORed on the same interrupt channel.*

BERR, ARLO, AF, OVR, PECERR, TIMEOUT and SMBALERT are logically ORed on the same interrupt channel.

Figure 277. I^2C interrupt mapping diagram


SB
ADDR
ADD10
STOPF
BTF
TxE
ITBUFEN
RxNE
ITEVFEN
it_event
ITERREN
BERR
ARLO
AF
OVR
PECERR
TIMEOUT
SMBALERT
it_error