AVR315: Using the TWI module as I²C master

8-bit AVR® Microcontrollers

Application Note

Features

- C-code driver for TWI master
- Compatible with Philips' I²C protocol
- Uses the hardware TWI module
- Interrupt driven transmission
- Supports both Standard mode and Fast mode

Introduction

The Two Wire serial Interface (TWI) is compatible with Philips' I²C protocol. The bus was developed to allow simple, robust and cost effective communication between integrated circuits in electronics. The strengths of the TWI bus includes the capability of addressing up to 128 devices on the same bus, arbitration, and the possibility to have multiple masters on the bus.

A hardware TWI module is included in most of the AVR devices available.

This application note describes a TWI master implementation, in form of a full-featured driver and an example of usage for this driver. The driver handles transmission according to both Standard mode (<100kbps) and Fast mode (<400kbps).

Rev. 2564C-AVR-01/10

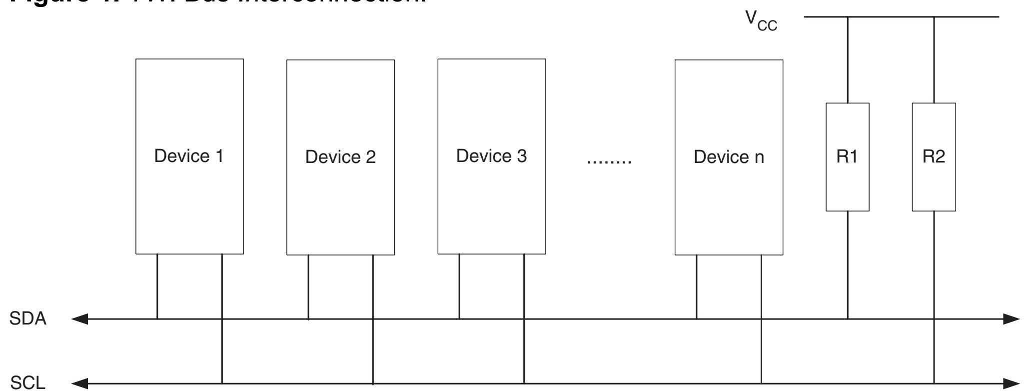
Theory

This section gives a short description of the TWI interface in general and the TWI module on the megaAVR's. For more detailed information refer to the datasheets.

Two-wire serial Interface

The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 individually addressable devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

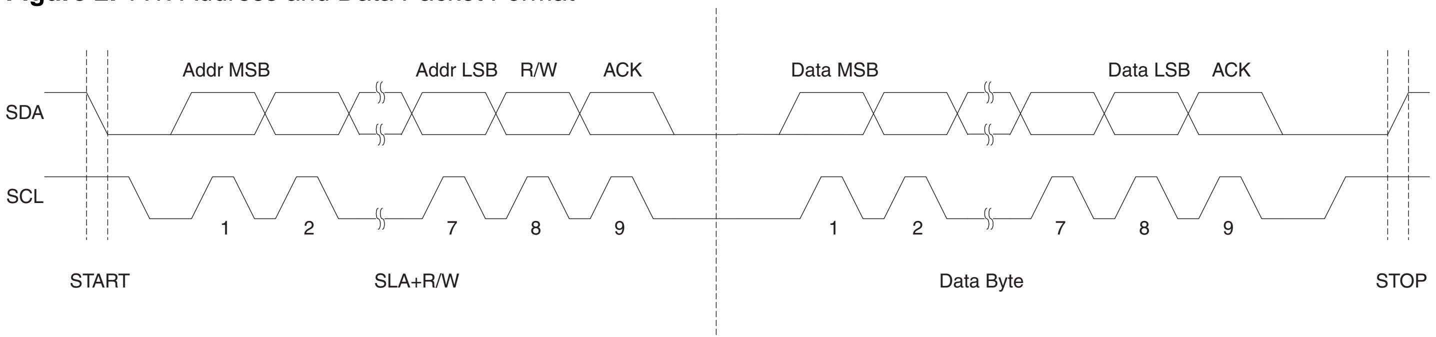
Figure 1. TWI Bus Interconnection.



The TWI bus is a multi-master bus where one or more devices, capable of taking control of the bus, can be connected. Only Master devices can drive both the SCL and SDA lines while a Slave device is only allowed to issue data on the SDA line.

Data transfer is always initiated by a Bus Master device. A high to low transition on the SDA line while SCL is high is defined to be a START condition or a repeated start condition.

Figure 2. TWI Address and Data Packet Format



A START condition is always followed by the (unique) 7-bit slave address and then by a Data Direction bit. The Slave device addressed now acknowledges to the Master by holding SDA low for one clock cycle. If the Master does not receive any acknowledge the transfer is terminated. Depending of the Data Direction bit, the Master or Slave now transmits 8-bit of data on the SDA line. The receiving device then acknowledges the data. Multiple bytes can be transferred in one direction before a repeated START or a STOP condition is issued by the Master. The transfer is terminated when the Master issues a STOP condition. A STOP condition is defined by a low to high transition on the SDA line while the SCL is high.

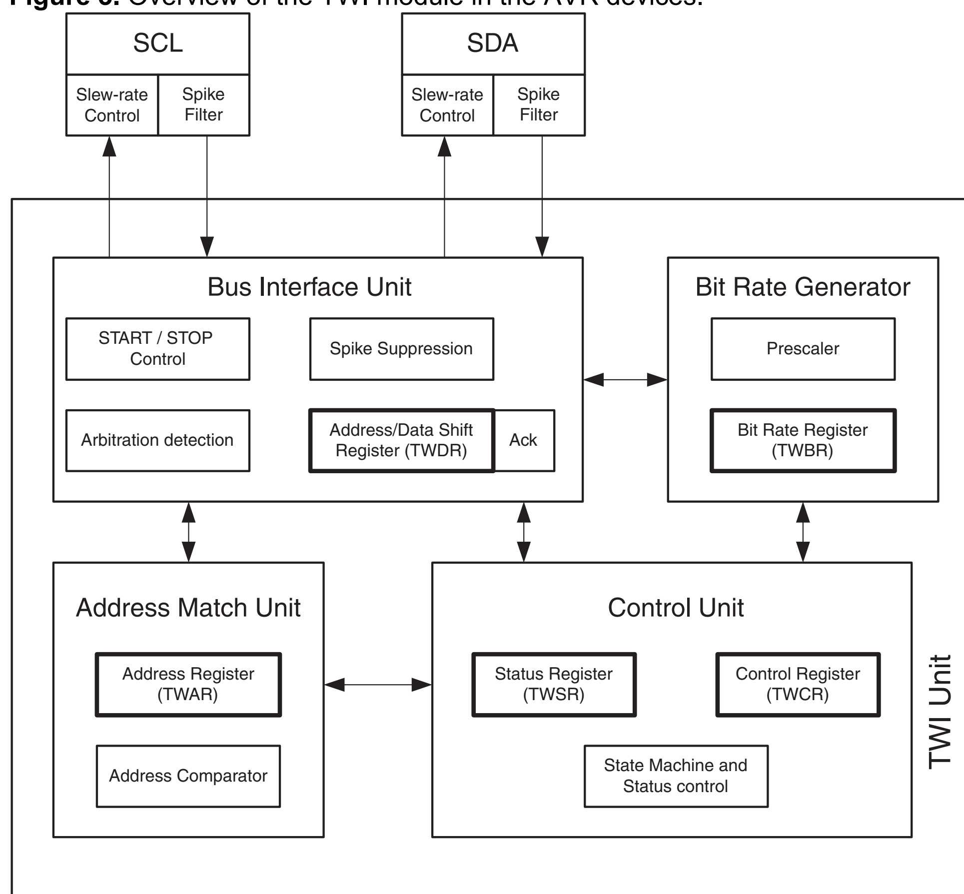
If a Slave device cannot handle incoming data until it has performed some other function, it can hold SCL low to force the Master into a wait-state.

All data packets transmitted on the TWI bus are 9 bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the master generates the clock and the START and STOP conditions, while the receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signaled by the receiver pulling the SDA line low during the ninth SCL cycle. If the receiver leaves the SDA line high, a NACK is signaled.

The AVR TWI Module

The TWI module is comprised of several sub modules, as shown in Figure 3. All registers drawn in a thick line are accessible through the AVR data bus.

Figure 3. Overview of the TWI module in the AVR devices.



Control Unit

The AVR TWI module can operate in both Master and Slave mode. The mode of operation is distinguished by the TWI status codes in the TWI Status Register (TWSR) and by the use of certain bits in the TWI Control Register (TWCR).

A set of predefined status codes covers the different states that the TWI can be in when a TWI event occurs. The status codes are divided in Master and Slave codes and further in receive and transmit related codes. Status codes for Bus Error and Idle also exist.

The TWI module operates as a state machine and is event driven: if a START CONDITION followed by a TWI address matches the address in the Slave's TWI Address Register (TWAR) the TWINT flag is set, resulting in the execution of the corresponding interrupt (if Global Interrupt and TWI interrupts are enabled). The firmware of the Slave responds by reading the status code in TWSR and responding accordingly. All TWI events will set the TWINT flag, and the firmware must respond based on the status in TWSR.

As long as the TWINT Flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT Flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition
- After the TWI has transmitted SLA+R/W
- After the TWI has transmitted an address byte
- After the TWI has lost arbitration
- After the TWI has been addressed by own Slave address or general call
- After the TWI has received a data byte
- After a STOP or REPEATED START has been received while still addressed as a Slave.
- When a bus error has occurred due to an illegal START or STOP condition

Bit Rate Generator

The Bite Rate Generator unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that Slaves may prolong the SCL low period, thereby reducing the average TWI bus clock frequency. The SCL frequency is generated according to the Equation 1 .

Equation 1.

$$SCL\ frequency = \frac{CPU\ Clock\ frequency}{16 + 2(TWBR) \cdot 4^{TWPS}}$$

- TWBR = Value of the TWI Bit Rate Register
- TWPS = Value of the prescaler bits in the TWI Status Register

TWPS is located in TWISR which is the same register as the TWI Status bits. TWPS should therefore always be set to 0 to simplify the handling of the TWI Status bits. Use TWBR to achieve the needed speed on the SCL. Table 1 shows a selection of pre calculated TWBR values based on CPU and SCL frequencies.

Table 1. CPU and SCL frequencies versus Bit Rate Generator register settings.

CPU Clock frequency [MHz]	TWBR	TWPS	SCL frequency [kHz]
16	12	0	400
16	72	0	100
14.4	10	0	400
14.4	64	0	100
12	7	0	400
12	52	0	100
8	2	0	400
8	32	0	100
4	12	0	100
3.6	10	0	100
2	2	0	100
2	12	0	50
1	2	0	50

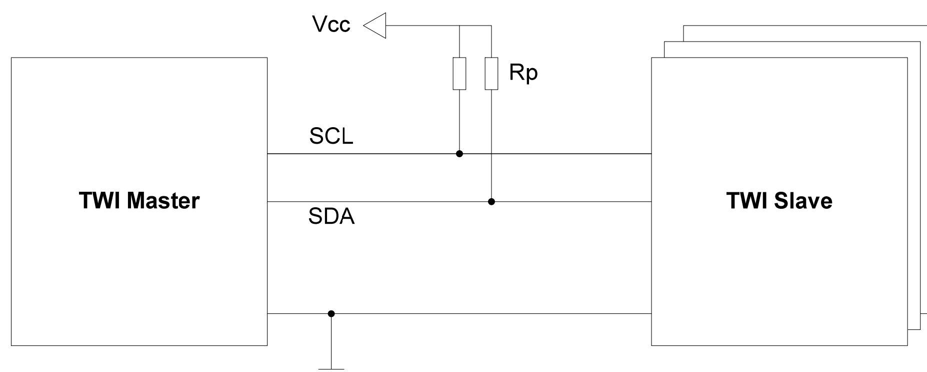
SCL and SDA Pins

Both TWI lines (SDA and SCL) are bi-directional, therefore outputs connected to the TWI bus must be of an open-drain or an open-collector type. Each line must be connected to the supply voltage via a pull-up resistor. A line is then logic high when none of the connected devices drives the line, and logic low if one or more is drives the line low.

The output drivers contain a slew-rate limiter. The input stages contain a spike suppression unit removing spikes shorter than 50 ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

Figure 4 shows how to connect the TWI units to the TWI bus. The value of Rp depends on V_{CC} and the bus capacitance (typically 4.7 k).

Figure 4. **TWI connection.**



Address Match Unit

The Address Match unit is only used in slave mode, and checks if the received address bytes match the 7-bit address in the TWI Address Register (TWAR). Upon an

address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR.

Although the clock system to the TWI is turned off in all sleep modes, the interface can still acknowledge its own Slave address or the general call address by using the TWI Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared.

Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition it also contains a register containing the (N)ACK bit to be transmitted or received.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a Master. If the TWI has initiated a transmission as Master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

Implementation

The implemented code in this application note is a pure master driver. The TWI modules also support slave operation. See “AVR311 TWI Slave Implementation” for a sample of a slave driver. The master and slave drivers could be merged to one combined master and slave driver, but this is not in the scope of this application note.

The master driver c-code consists of three files.

- TWI_Master.c
- TWI_Master.h
- Main.c

There is an example on how to use the driver in the main.c file. The TWI_Master.h file must be included in the main application and contains all function declarations, a define of the Bit Rate Register (TWBR) and defines for all TWI status codes. The TWBR must be set according to the description earlier in this application note. The TWI status code defines can be used to evaluate error messages and to take appropriate actions. The file TWI_Master.c contains all the driver functions.

Functions

The driver consists of the TWI Interrupt Service Routine and six functions. All functions are available for use outside the driver file scope. Some of them are however also used internally by the driver it self. All functions in the driver are listed in Table 2. The actual code sizes for the functions, compiled with the IAR compiler are listed in Table 4.

Table 2. Description of functions in the TWI Master driver.

Function name	Description
`void TWI_Master_Initialise( )`	Call this function to set up the TWI master to its initial standby state. Remember to enable interrupts from the main application after initializing the TWI.

`void TWI_Start_Transceiver_with_Data ( uchar *message, uchar messageSize)`	Call this function to send a prepared message. The first byte must contain the slave address and the read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read from the slave. Also include how many bytes that should be sent/read including the address byte. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.
`void TWI_Start_Transceiver( )`	Call this function to resend the last message. The driver will reuse the data previously put in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.
`uchar TWI_Transceiver_Busy( )`	Call this function to test if the TWI_ISR is busy transmitting.
`uchar TWI_Get_State_Info( )`	Call this function to fetch the state information of the previous operation. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation. If there was an error, then the function will return the TWI State code.
`uchar TWI_Get_Data_From_Transceiver ( uchar *message, uchar messageSize)`	Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call TWI_Start_Transceiver to send a request for data to the slave. Then Run this function to collect the data when they have arrived. Include a pointer to where to place the data and the number of bytes requested (including the address field) in the function call. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, before reading out the data and returning. If there was an error in the previous transmission the function will return the TWI error code.
`__interrupt void TWI_ISR( )`	This function is the Interrupt Service Routine (ISR), and automatically called when the TWI interrupt is triggered; that is whenever a TWI event has occurred. This function should not be called directly from the main application.

Table 3. Description of the driver register byte containing status information from the last transceiver operation. Available as bit fields within a byte.

TWI_statusReg	Description
`TWI_statusReg.lastTransOK`	Set to 1 when an operation has completed successfully.

Table 4. Code sizes with the IAR 3.10 compiler.

TWI Master functions	[bytes]
`TWI_Master_Initialise( )`	14
`TWI_Transceiver_Busy( )`	6
`TWI_Get_State_Info( )`	12
`TWI_Start_Transceiver_With_Data( )`	72
`TWI_Start_Transceiver( )`	8
`TWI_Get_Data_From_Transceiver( )`	50
`TWI_ISR( )`	194
	356

Figure 5 shows a flowchart of the process of sending and requesting data over the TWI interface through the drivers. Data is passed through parameters to the functions while the status of an operation is available trough a global status variable.

Figure 6 contains the flowchart for the TWI driver itself. A more detailed description of the actions for each event/state in the TWI Interrupt Service Routine can be found in a flowchart in Figure 6. TWI driver functions.

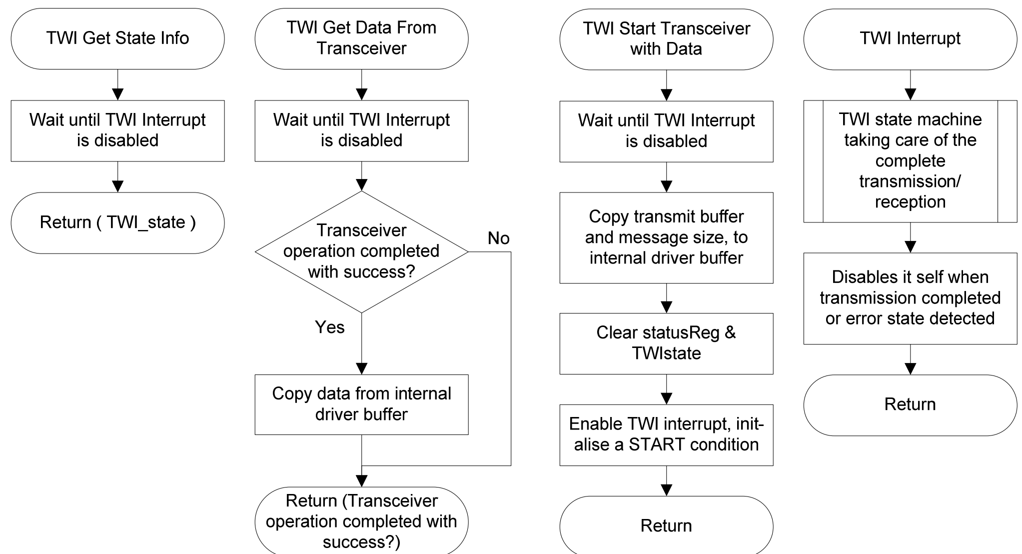


Figure 7. In Figure 6. TWI driver functions.

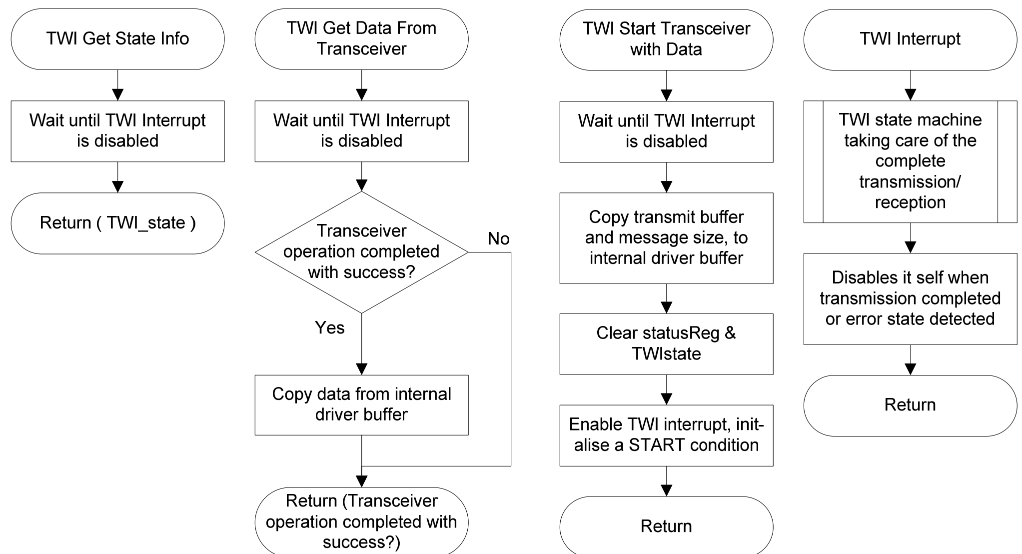


Figure 7 the left column contains the different states/events the TWI state machine can be in when entering the Interrupt. A case switch executes the different actions dependant on the cause of the interrupt call.

The Transceiver function copies the complete message into the transmission buffer. Then it enables the TWI interrupt to initiate the transmission. The Interrupt then takes care of the complete transmission and disables it self when the transmission is completed, or if an error state occurs. The driver can this way poll the interrupt enable bit to check if a transmission is on going. The main application is only allowed to access the global transceiver variables while the TWI transceiver is not busy. The interrupt stores eventual error states in a variable that is available for the main application through a function call.

Figure 5. Calling the TWI driver from the application.

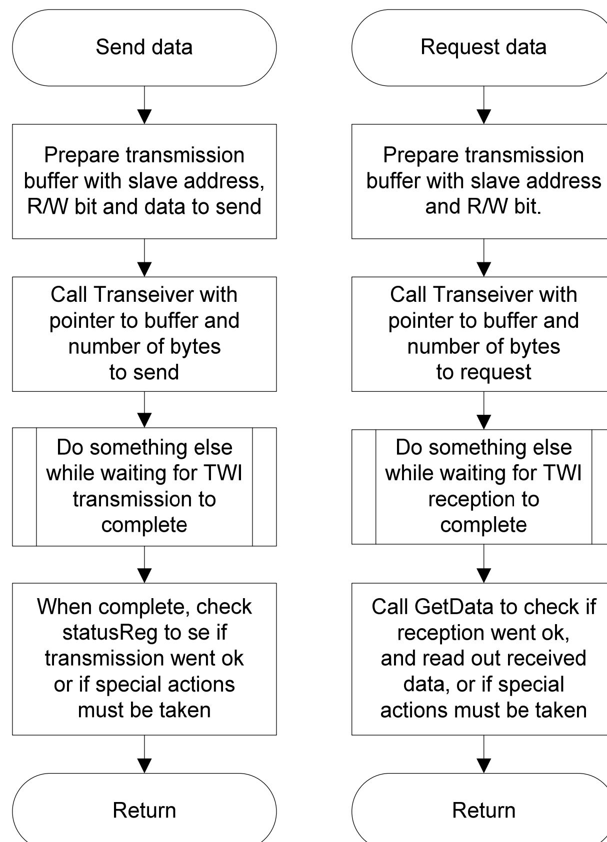


Figure 6. TWI driver functions.

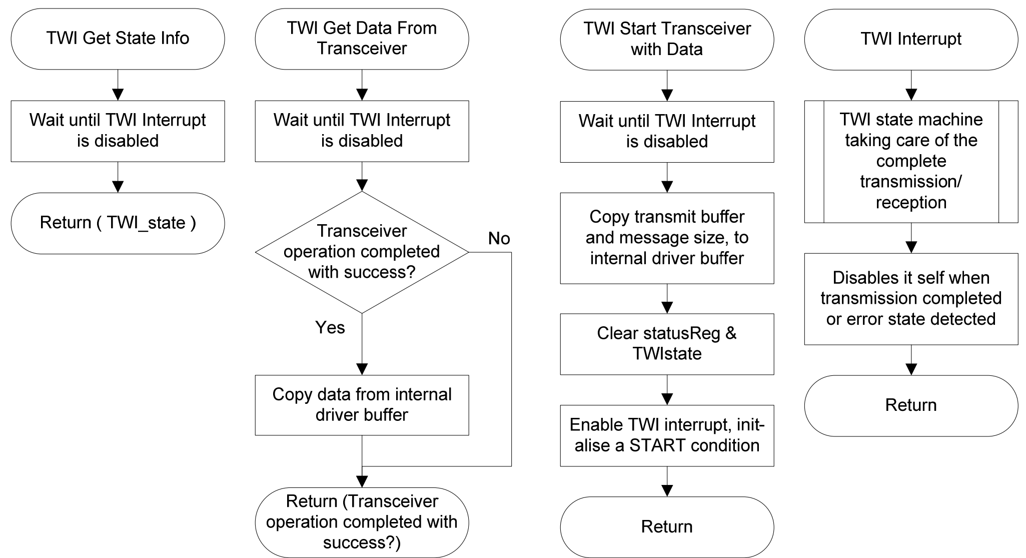


Figure 7. TWI interrupt service routine

TWI Interrupt

START has been transmitted

Repeated START has been transmitted

SLA+W has been tramsmitted and ACK received

Data byte has been tramsmitted and ACK received

Data byte has been received and ACK tramsmitted

SLA+R has been tramsmitted and ACK received

Data byte has been received and NACK tramsmitted

Arbitration lost

SLA+W/R has been tramsmitted and NACK received

Data byte has been tramsmitted and NACK received

Bus error due to an illegal START or STOP condition

Error States

Reset bufferPointer

buffer pointer at the end?

Yes

No

Set Success bit in StatusReg

Disable TWI interrupt, initalise a STOP condition and clear interrupt flag

Copy from data register to current buffer position. Post incr pointer.

Copy data from current buffer position to data register. Post incr pointer.

Clear TWI interrupt flag

Copy from data register to current buffer position.

Set Success bit in StatusReg

Disable TWI interrupt, initalise a STOP condition and clear interrupt flag

Initalise a RESTART condition and clear TWI interrupt flag

buffer pointer position at byte before the end?

Yes

No

Intialise a ACK after reception and clear TWI interrupt flag

Intialise a NACK after reception and clear TWI interrupt flag

Store TWI state information

Disable TWI interrupt

Return

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2010 Atmel Corporation. All rights reserved. Atmel®, Atmel logo and combinations thereof, AVR®, AVR® logo and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.