AVR1320: True 400 kHz operation for TWI slave

8-bit AVR® Microcontrollers

Application Note

Features

- **AVR® XMEGA™ family devices**
- **C-code driver for TWI slave**
- **Compatible with Philips® I^{2}C™ protocol**
- **Uses the hardware TWI module**
- **Interrupt driven transmission**
- **No clock-stretching – true 400 kHz operation**

1 Introduction

The Two Wire serial Interface (TWI) is compatible with Philips' I^{2}C 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 AVR XMEGA devices and implements both master and slave functionalities. This module is I^{2}C and SMB bus compliant.

If needed, to slow down the clock frequency or to insert wait states, AVR XMEGA devices allow in compliance with the I^{2}C protocol, to stretch the low period of the clock. In this case, the master is forced into a wait state until the slave is ready. In some systems no clock stretching is allowed despite that the standard allows for it.

This application note describes a TWI slave driver implementation for AVR XMEGA that ensures that the clock is not stretched.

Rev. 8281A-AVR-04/10

2 TWI theory

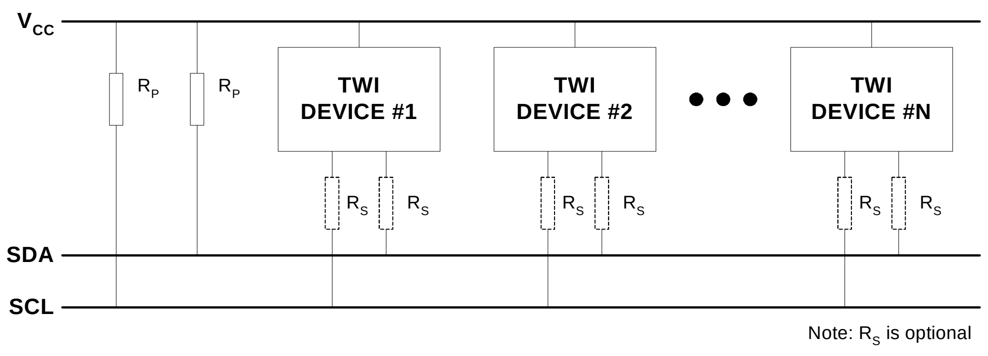
This section gives a short description of the TWI interface in general and the TWI module on AVR XMEGA. For more detailed information refer to the datasheet.

2.1 Two wire serial interface

The TWI is ideally suited for typical microcontroller applications. The TWI protocol allows the system 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 2-1. TWI Bus Topology



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 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.

A START condition is always followed by the (unique) 7-bit slave address and then by a Data Direction bit. The addressed Slave device sends an acknowledge to the Master by holding SDA low for one clock cycle. If the Master does not receive any acknowledge the transfer is terminated.

Depending on the Data Direction bit, the Master or Slave 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.

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, 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 sent.

2.2 TWI Clock stretching

According to the I^{2}C standard, all devices connected to the bus are allowed to stretch the low period of the clock to slow down the overall clock frequency or to insert wait states while processing data. A device that needs to stretch the clock can do this by holding/forcing the SCL line low after it detects a low level on the line (possible since the lines are driven low by open-drain and have pull-up resistors).

If the device is in a sleep mode and a START condition is detected the clock is stretched during the device wake-up period. To avoid clock stretching in relation to use of sleep, only sleep mode with sufficiently fast wake-up time should be used. This could be e.g. IDLE, Standby and Extended Standby mode.

3 AVR XMEGA Slave mode - stretchless operation

If the customer application doesn’t support clock stretching, the AVR1320 proposes a true 400 kHz TWI operation for an AVR XMEGA TWI slave.

The TWI hardware module is common to AVR XMEGA. The AVR1320 software tests are performed on ATxmega128A1, but the driver is compatible with any AVR XMEGA.

3.1 TWI slave operation

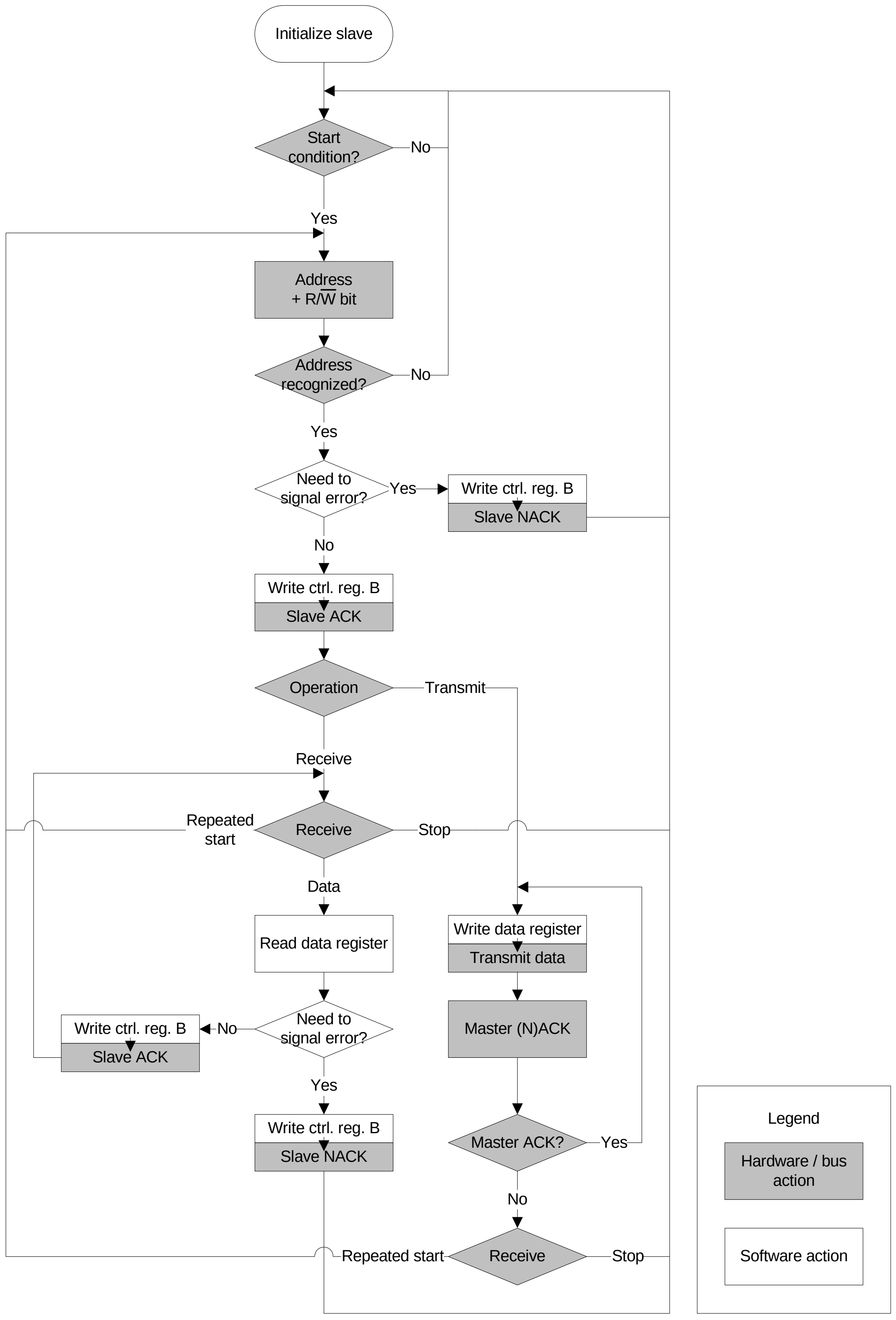
The TWI slave is byte-oriented with optional interrupts after each byte. There are separate slave Data Interrupt and Address/Stop Interrupt. Interrupt flags can also be used for polled operation. There are dedicated status flags for indicating ACK/NACK received, clock hold, collision, bus error and read/write direction.

When an interrupt flag is set, the SCL line is forced low. This will give the slave time to respond or handle any data, and will in most cases require software interaction.

The interrupt service routine (ISR) process needs to manage three major states:

- Slave address interrupt (see section 3.2)
- Data byte interrupt in write mode (see section 3.4)
- Data byte interrupt in read mode (see section 3.3).

Figure 3-1. Slave operation.



3.2 Reception of first byte (address)

In slave mode, the AVR XMEGA AVR receives a first byte with address and R/W bit (Data Direction). Then, the 9th bit is dedicated to the ACK or NACK bit.

An interrupt occurs after the 8th bit. During the execution of the ISR, the AVR XMEGA holds SCL line low. SCL is released when Address/Stop Interrupt Flag (APIF) is cleared in STATUS register by clearing APIF or writing to CMD bits.

If the TWI master does not support the clock stretching, the SCL line frequency during the 8 first bits and the 9th must be same, and the SCL line must thus be released by the slave in due time to not stall the TWI master. In this case, the interrupt timing determines the maximum frequency allowed by the TWI interface.

3.3 Write operation – slave received data

If the R/W bit of the address byte is equal to 0, a write operation is performed by the master. After the 8th bit of the address byte, an interrupt occurs and the program jumps to the ISR. The first data byte from the master will be available in TWI DATA register when the next interrupt occurs. After a TWI write command, the data from the master must be recovered before the next byte is received.

3.4 Read operation – slave transmits data

If the R/W bit of the address byte is equal to 1, a read operation is performed by the master. After the 8th bit of the address byte, an interrupt occurs and the program jumps to the ISR. The first data byte to read needs to be loaded in the TWI DATA register during the ISR, without stretching the clock.

3.5 SCL description

The TWI frequency limit depends on time to release SCL in the ISR function. When APIF or Data Interrupt Flag (DIF) are set (see STATUS register), the slave forces the SCL line low. Clearing the interrupt flags will release the SCL line.

DIF or APIF are automatically cleared when:

- Writing a 1 in the corresponding bit location
- Writing to the slave DATA register
- Reading the slave DATA register
- Writing a valid command to the CMD bits in the CTRLB register

4 Hardware description

4.1 CPU clock

If the CPU clock of the TWI slave is increased more CPU cycles are available before the SCL must be release, and it is thus an advantage to run the TWI slave from a fast clock source to avoid clock stretching. AVR XMEGA offers a 32 MHz internal oscillator. The system clock control register is used to select the clock source.

4.2 Pull up values on SDA and SCL pins

SCL and SDA pads are open drain and need pull up resistors.

2kOhms resistor values are recommended to have correct edges for a 400 kHz frequency on TWI bus.

5 Software description

5.1 List of files

The source code package consists in the following files (and corresponding header files):

- twi_example.c
- twi_slave_driver.c
- twic_it.asm
- clksys_driver.c

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation (see readme.html).

5.2 Software algorithms

The TWI process is divided in two parts:

- A TWI ISR function
- An INT0 ISR function.

When a byte is sent/received by the TWI hardware block and is ready to process, the TWI ISR enables INT0 ISR. The TWI ISR uses HIGH level interrupt and the INT0 ISR uses MEDIUM level interrupt.

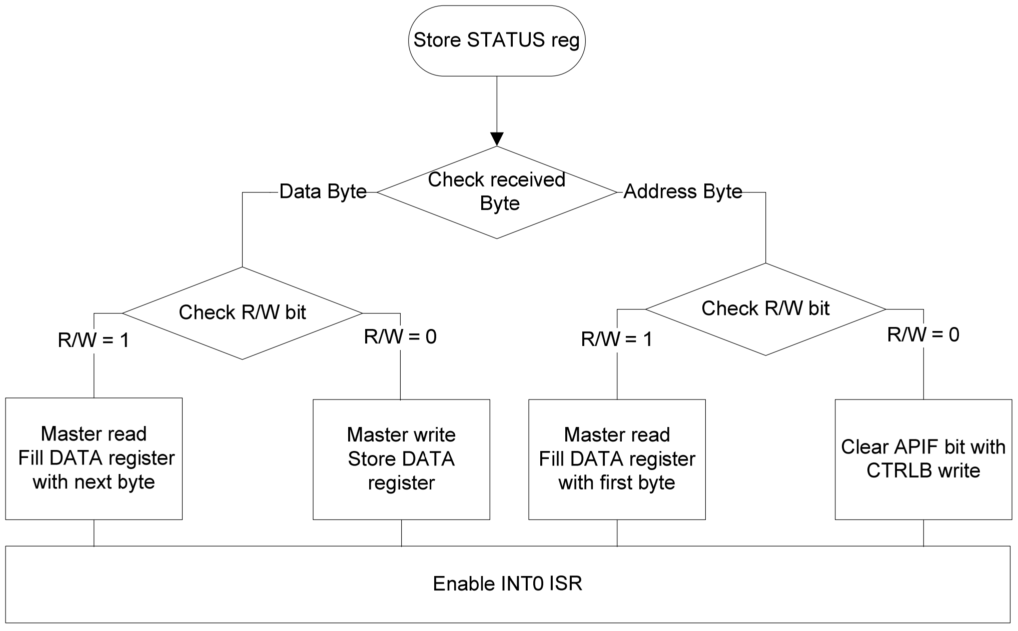
The SCL pin is connected to INT0 ISR. The next rising edge on SCL will start INT0 process function.

The TWI ISR is short and implemented in assembly to ensure a quick SCL release. The INT0 ISR hence processes the TWI data. The splitting of ISR is used to ensure that long interrupt execution time does not violate the timing of the TWI driver and cause clock stretching.

The TWI ISR function stores STATUS register and DATA register value for the TWI_process() function.

SCL is released when DATA register is read or write. When an address byte is received with R/W = 0, SCL is released after CTRLB write operation.

Figure 5-1. ISR function.


Store STATUS reg
Check received Byte
Data Byte
Address Byte
Check R/W bit
R/W = 1
R/W = 0
Check R/W bit
R/W = 1
R/W = 0
Master read Fill DATA register with next byte
Master write Store DATA register
Master read Fill DATA register with first byte
Clear APIF bit with CTRLB write
Enable INT0 ISR

5.3 TWI test measurements

A TWI master sends TWI write/read commands in loop. Test description is:

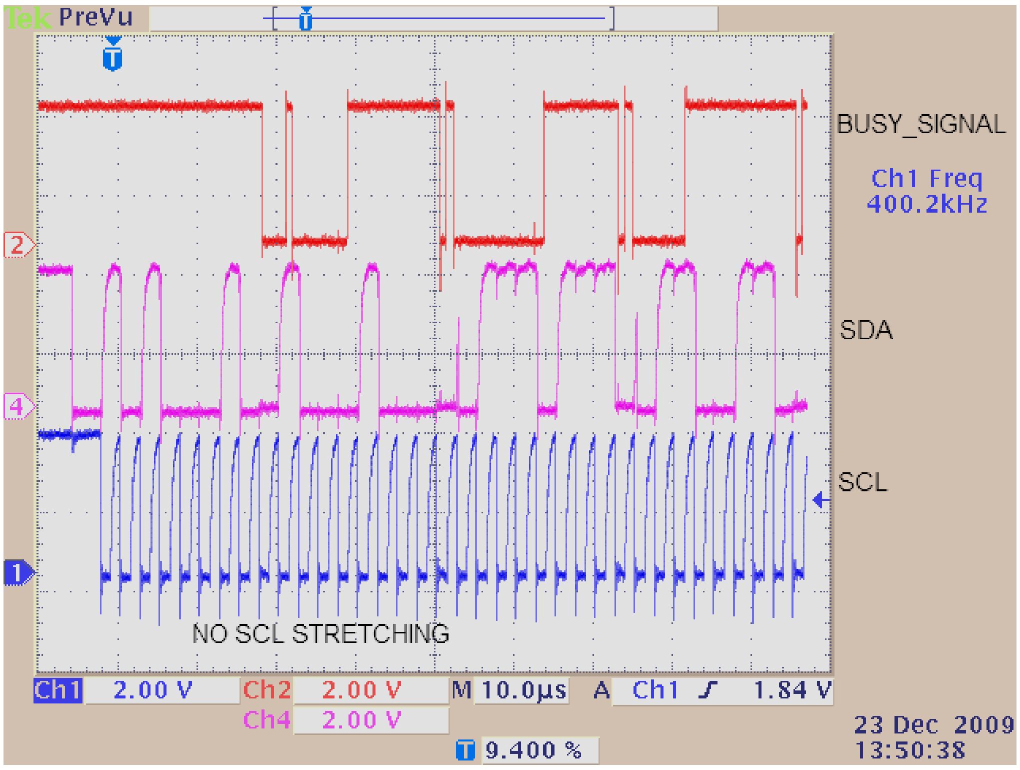
- TWI clock = 400 kHz
- Write command of 8 bytes
- Read command of 8 bytes

A scope monitors stretching. No clock stretching is apparent for the example code.

A signal (BUSY_SIGNAL) is added on scope to control the time spent in ISR function and in TWI_process. During ISR (in first) and TWI_process (just after), BUSY_SIGNAL is 0.

The TWI software uses about half of the CPU bandwidth during a continues 400 kHz transfer.

Figure 5-2. True 400kHz TWI signals



5.4 Limitations

- Because of the speed of the communication and the nature of the handling, if data is to be calculated and provided to the TWI module in real-time, the user processing routine must not exceed 16µs between packets before data underrun occurs. This equates to roughly 500 CPU cycles at 32 MHz.
- The HIGH and MEDIUM interrupt level should be reserved exclusively for the TWI functionality to ensure glitch-free stretchless operation. The MEDIUM level is used by a secondary (pin change) interrupt generated to run the TWI_process() function, since there is no time to do so in the TWI ISR. All interrupts added by the user should have LOW priority to guarantee stretchless operation of the TWI driver.
- The driver automatically sends the action defined in the ACKACT bit of the CTRLA register to the master when responding to a MASTER READ. If the user wishes to send NACK, this must be prepared by setting the ACKACT bit in advance.

5.5 Conclusion

The AVR XMEGA supports true 400 kHz TWI communication. The software example provided in this application note explains how to process this TWI interface in slave mode. The IAR™ program is available in AVR1320 application note. The driver does not compile with the GCC compiler.

The memory mapping of this code is the following:

Table 5-1. Memory mapping

Code memory	**DATA memory**
1552 bytes	109 bytes

5.6 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.

6 Getting started

The user application can be added in the main function.

After a TWI master write-command, the twiSlave.receivedData[] buffer contains data.

Before a TWI master read command, the twiSlave.sendData[] buffer must be ready.

The TWIC_SlaveProcessData() is called after each received byte and allows to process sent/received buffers.

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
http://www.atmel.com/

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature