AVR312: Using the USI Module as a I2C Slave

APPLICATION NOTE

Introduction

The Two-wire serial Interface (TWI) is compatible with Philips' I^2C 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.

The Universal Serial Interface (USI) module on devices such as Atmel® ATmega169, ATtiny26, and ATtiny2313, has a dedicated two-wire mode. The USI provides the basic hardware resources needed for synchronous serial communication. Combined with a minimum of control software, the USI allows higher transfer rates and uses less code space than solutions based on software only. Interrupts are included to minimize the processor load.

This document describes how to use the USI for TWI slave communication. Source code for communication drivers for transmission and reception is provided. The code is complete with both data buffer handling and combined transmitter and receiver.

Features

- C-code driver for TWI slave, with transmit and receive buffers
- Compatible with Philips' I^2C protocol
- Interrupt driven, detection, and transmission/reception
- Wake up from all sleep modes, including Power Down

Table of Contents

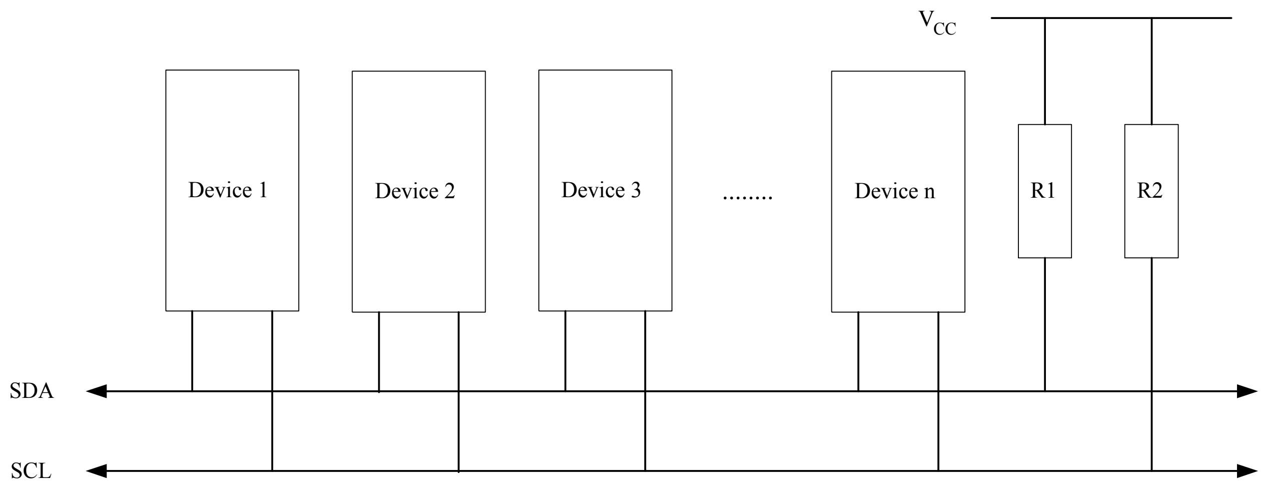
1. Theory

This chapter gives a short description of the TWI and USI interfaces. For more detailed information, refer to the datasheets.

1.1. 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.

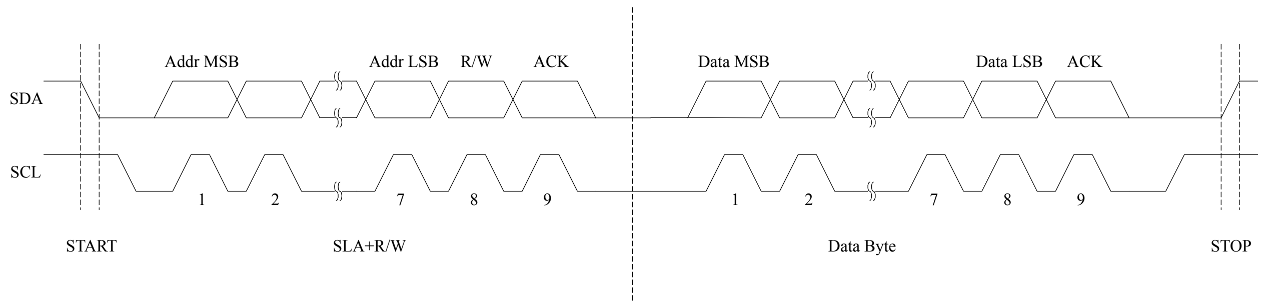
Figure 1-1. TWI Bus Interconnection



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 1-2. 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 eight bits 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 nine 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.

1.2. Universal Serial Interface – USI

The Universal Serial Interface (USI) provides the basic hardware resources needed for synchronous serial communication. Combined with a minimum of control software, the USI allows higher transfer rates and uses less code space than solutions based on software only. Interrupts are included to minimize the processor load. The main features of the USI are:

- Two-wire Synchronous Data Transfer
- Three-wire Synchronous Data Transfer
- Data Received Interrupt
- Wake-up from Idle Mode
- In Two-wire Mode: Wake-up from All Sleep Modes, including Power-down Mode
- Two-wire Start Condition Detector with Interrupt Capability

The USI Two-wire mode is compliant to the TWI bus protocol, but without slew rate limiting on outputs and input noise filtering.

Figure 1-3. Universal Serial Interface, Block Diagram

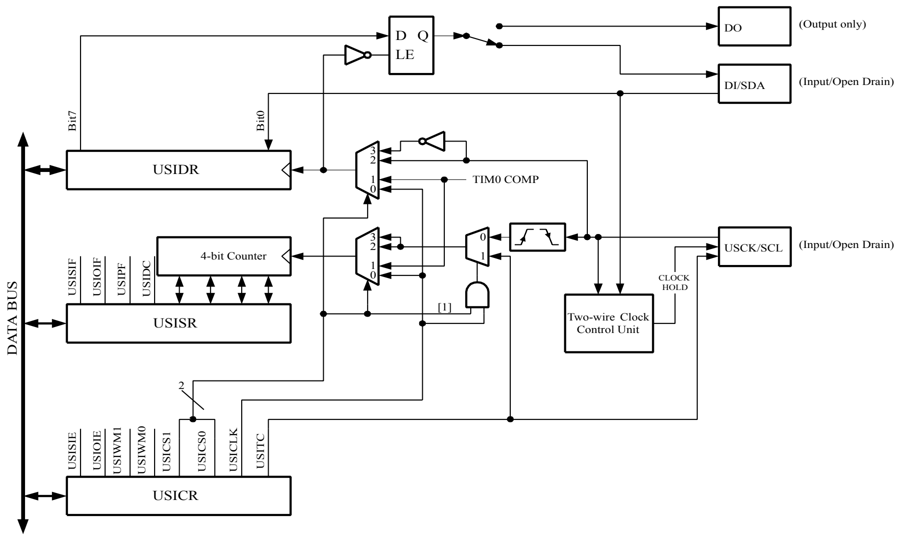
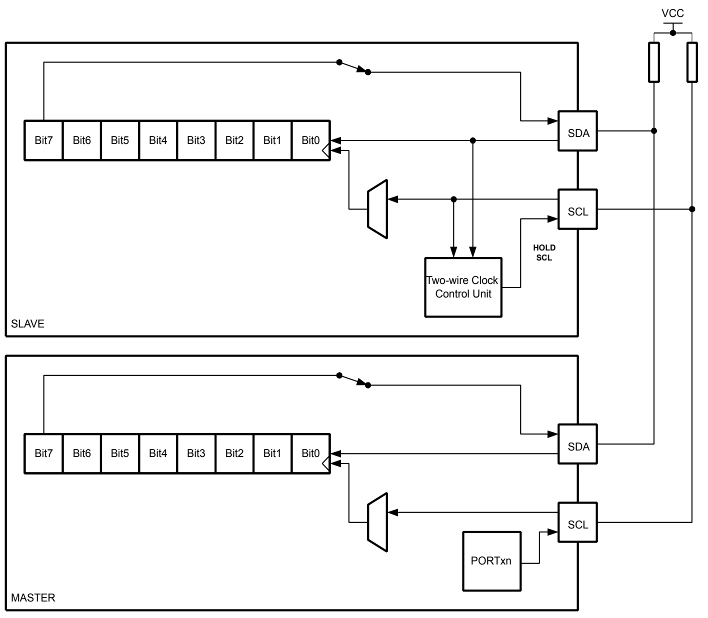


Figure 1-4. Two-wire Mode Operation, Simplified Diagram



The USI Data Register (USIDR) is an 8-bit Shift Register that contains the incoming and outgoing data. The register has no buffering so the data must be read as quickly as possible to ensure that no data is lost.

The USI Status Register (USISR) contains a 4-bit counter. Both the Serial Register and the counter are clocked simultaneously by the same clock source. This allows the counter to count the number of bits received or transmitted and generate an interrupt when the transfer is complete. The clock can be selected to use three different sources; The SCL pin, Timer/Counter0 Compare Match, or from software.

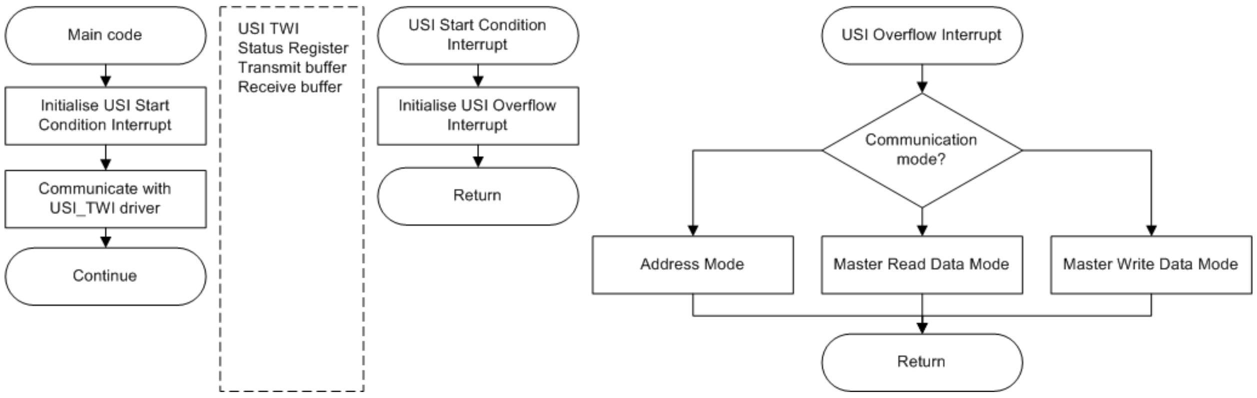
The two-wire clock control unit can generate an interrupt when a start condition is detected on the two-wire bus. It can also generate wait states by holding the clock pin low after a start condition is detected, or after the counter overflows.

2. Implementation

The application note describes the implementation of a TWI slave. The driver is written as a standalone driver that can easily be included into the main application. Use the code as an example, or customize it for own use. Defines and status registers are all set in the application note header file.

The core of the driver is interrupt driven and therefore “runs in parallel” to the other processes in the application. After initializing the driver, all communication with the driver is made through a global status register variable, and the transmit and receive buffers. The size of the communication buffers can be changed in the driver header file.

Figure 2-1. USI TWI Overview Flowchart



The main code communicates with the drivers through registers and buffers.

The driver takes care of the low level communication as transmission/reception of address, data, and ACK/NACK. High level operations like address setting, message interpreting, and data preparation, must be taken care of by the main application.

When a START condition is detected the USI Start Condition Interrupt sets up the USI Overflow Interrupt to start receiving the first package. The USI Overflow interrupt is always set to trigger after sampling eight bits. This way the CPU is free to operate on other application specific operations during the bit sampling/ transmission.

Since the USI Start Condition and USI Overflow event holds the SCL line when activated, it is not timing critical when the interrupts are to be executed. If other operations are preventing the interrupts to execute, then proper operation will still be ensured by the driver. However, to keep the communication speed on the TWI lines as high as possible the SCL time should be released a quickly as possible.

Detection of the START Condition event is always enabled. So during any state of the communication sequence, the driver will reset the reception if a USI Start Condition is detected.

Any TWI message has according to the standard, an undefined length. The data to be transmitted/ received needs to be held in buffers. The buffers contain all data bytes within one single message. The buffers must be read/rewritten before next the transmission.

TWI address administration must be controlled from the application itself. The USI_TWI_Slave_Initialize function takes the new address as a parameter and stores it for the driver to verify on each message reception. Return the initialize function to reset the address.

There are six different states gathered in three “main” states the communication can be in when entering the USI Overflow Interrupt. They are “Address Mode”, “Master Read Data Mode”, and “Master Write Data Mode”. The states are controlled internally with the USI_TWI_state variable.

2.1. Address Mode

This mode is only set the first time the USI Overflow Interrupt is executed after the START condition is detected. The data has already been sampled into the USI data register. If the address is not recognized the interface is reinitialized to wait for the next START condition, and therefore discards the rest of the message.

If it is a “general call” a bit in the status register will be set. Any actions that need to be taken based on this must be carried out by the main application.

The direction mode is read from the eight bits in the address transmission, and is stored in the masterReadDataMode status bit, before the byte is acknowledged by the slave.

Depending on the masterReadDataMode setting, a transmission/reception is prepared before leaving the interrupt and waiting for the sampling of the next eight bits.

2.2. Master Read Mode

Master Read Mode means that the slave has to transmit data to the master. Data is read from the transmit buffer and stored into the USI data register.

When the byte has been shifted out successfully, the slave listens for an ACK from the master. A NACK from the master is interpreted as the end of the message, and tags the transmission buffer as empty before resetting the interface.

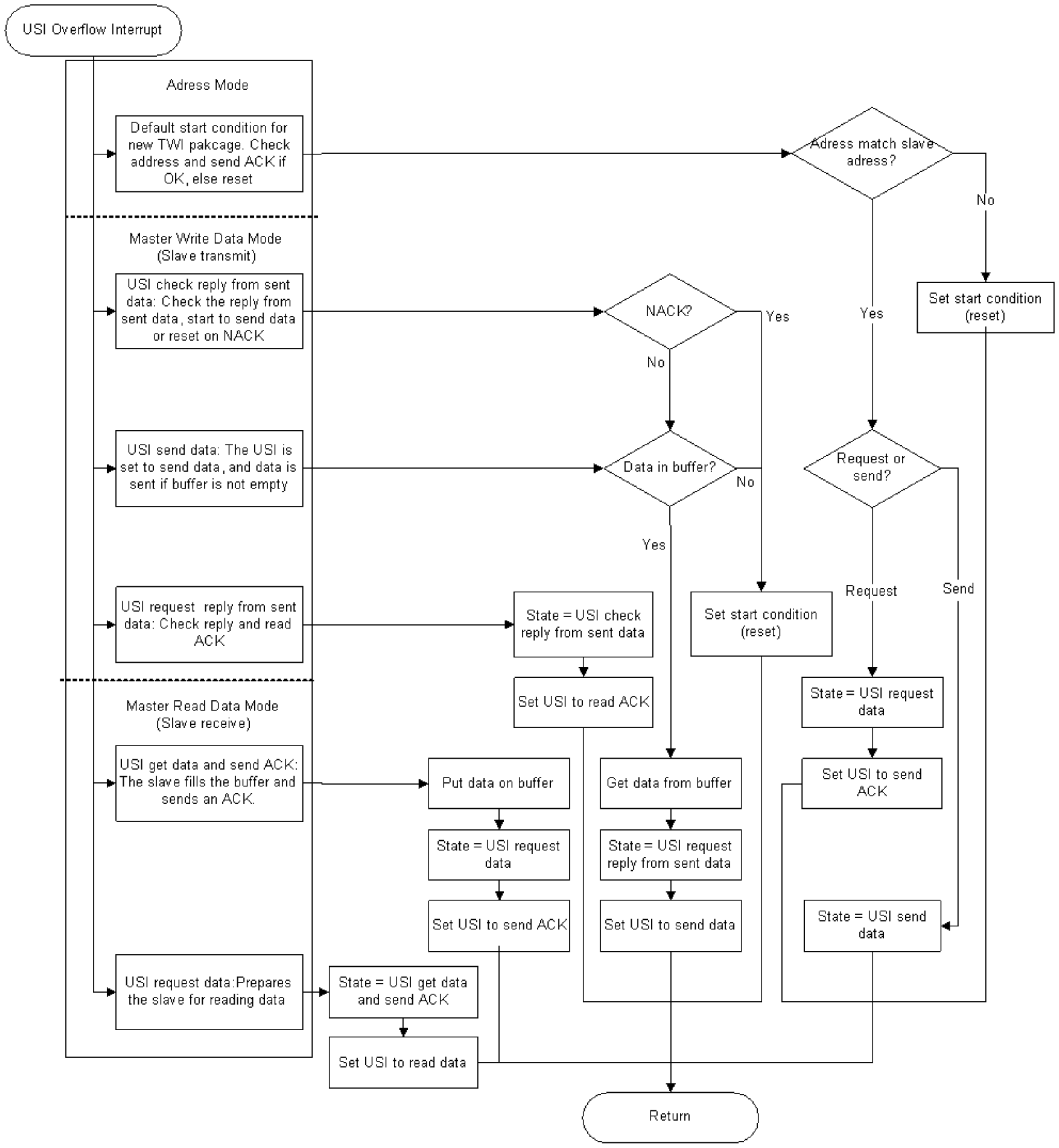
When getting an ACK the next data byte in the buffer is put in the USI data register for transmission. Etc.

2.3. Master Write Mode

Master Write Mode means that the master is going to transmit data to the slave. The USI is set up to sample a byte on the data line.

When the byte has been shifted in successfully, the slave sends an ACK to the master. After storing the data in the buffer, and setting up USI to sample the next byte, the application waits for one USI clock to test if there has been sent a STOP condition from the master. If there is a STOP condition, the receive buffer is tagged as full. And the interface is reset. If not then the transmission is continued.

Figure 2-2. Flowchart of the Processes in the USI Overflow Interrupt. Data Buffer Handling is not Included Here. The Interrupt is Initially Called after Eight Bits are Sampled.



USI Overflow Interrupt
Adress Mode
Default start condition for new TWI pakcage. Check address and send ACK if OK, else reset
Adress match slave adress?
No
Yes
Set start condition (reset)
Master Write Data Mode (Slave transmit)
USI check reply from sent data: Check the reply from sent data, start to send data or reset on NACK
NACK?
Yes
No
USI send data: The USI is set to send data, and data is sent if buffer is not empty
Data in buffer?
No
Yes
Request or send?
Request
Send
USI request reply from sent data: Check reply and read ACK
State = USI check reply from sent data
Set start condition (reset)
Set USI to read ACK
State = USI request data
Master Read Data Mode (Slave receive)
USI get data and send ACK: The slave fills the buffer and sends an ACK.
Put data on buffer
Get data from buffer
Set USI to send ACK
State = USI request data
State = USI request reply from sent data
Set USI to send ACK
Set USI to send data
State = USI send data
USI request data:Prepares the slave for reading data
State = USI get data and send ACK
Set USI to read data
Return

These actions will terminate and reset any transmission:

- The address in the message is not zero, nor the defined TWI address
- The master returns a NACK on a slave transmission
- The master sends a STOP Condition, after the slave sends a ACK
- The master sends a START Condition
- The master requests data, but the transmit buffer is empty
- The master sends data, but the receive buffer is full

Optionally one can also add more buffer control and the same response if the master requests/sends more information then the size of the transmit/receive buffers. This buffer control can be enabled by setting a define in the driver header file.

2.4. Sleep Mode

The USI Start Condition is able to wake up the AVR® from all sleep modes including Power Down. However, the device must stay in active mode until the complete message has been received/transmitted.

2.5. Source Code

The example code is written for Atmel START. It can be downloaded from "BROWSE EXAMPLES" entry of Atmel START for both Atmel Studio 7 and IAR™ IDE. Double click the downloaded .atzip file and project will be imported to Atmel Studio 7. To import the project in IAR, refer "Atmel START in IAR", select Atmel Start Output in External Tools → IAR.

3. Revision History

Doc Rev.	Date	Comments
2560D	08/2016	The example firmware is ported to Atmel Start
2560C	05/2009	Third Version
2560B	Unknown	Second Version
2560A	Unknown	Initial Document Released

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**