AVR 8-bit Microcontrollers

AVR310: Using the USI module as a TWI Master

APPLICATION NOTE

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.

The Universal Serial Interface (USI) module on devices like 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.

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). The example can be found from the Atmel START web-page, http://start.atmel.com/.

Features

- C-code driver for TWI master
- Compatible with Philips' I^{2}C protocol
- Uses the USI module
- Uses no interrupts or timers
- Supports both Standard mode and Fast mode

Table of Contents

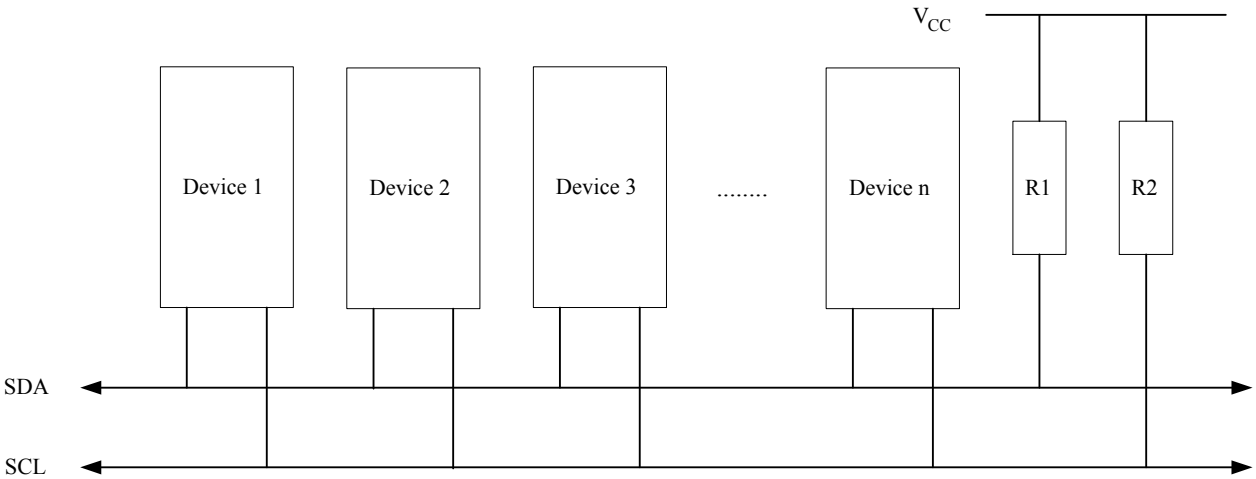
1. Theory

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

1.1. Two-wire Serial Interface

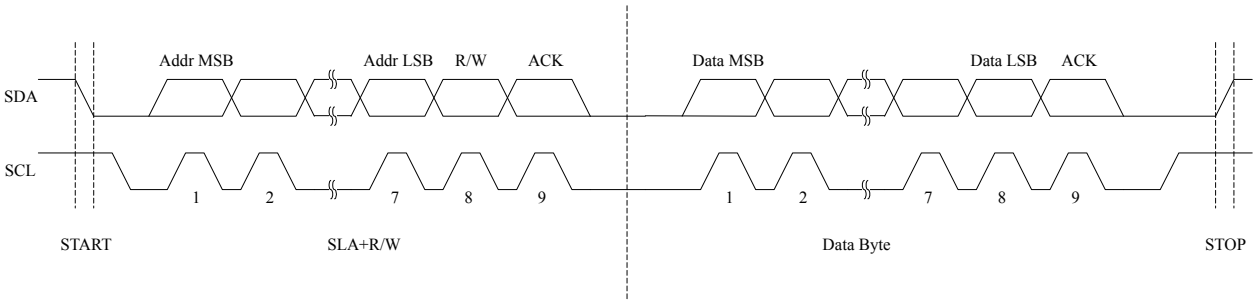
The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 individually addressable devices using only two bi-directional bus lines; one for clock (SCL) and one for data (SDA). The only external 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 Connection



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 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 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
- Wakeup 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

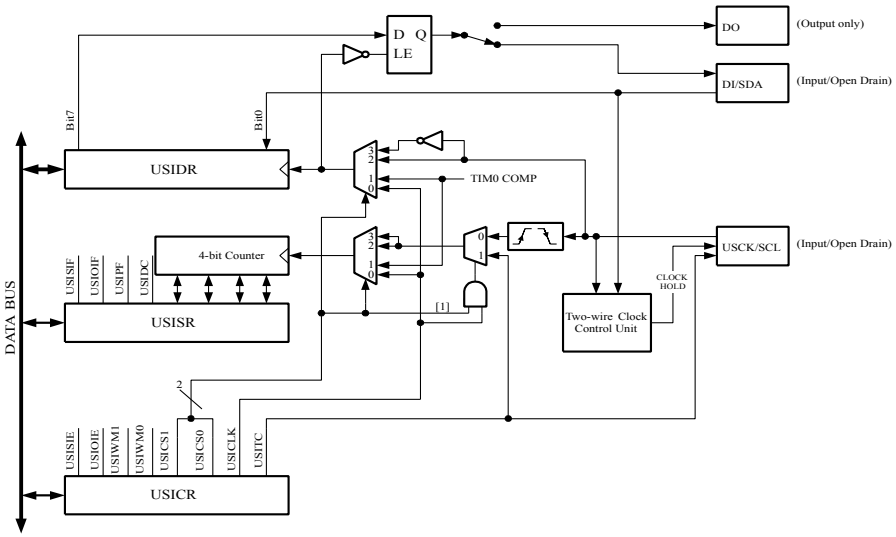
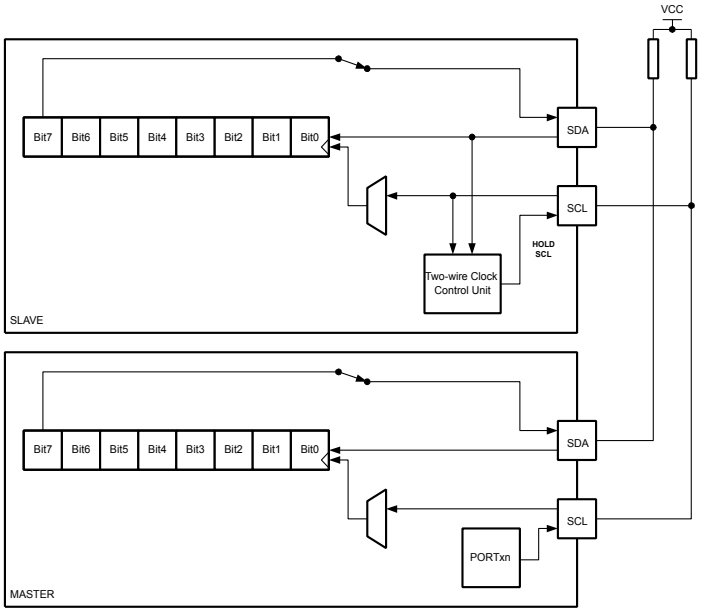


Figure 1-4. Two-wire Mode Operation, Simplified Diagram



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.

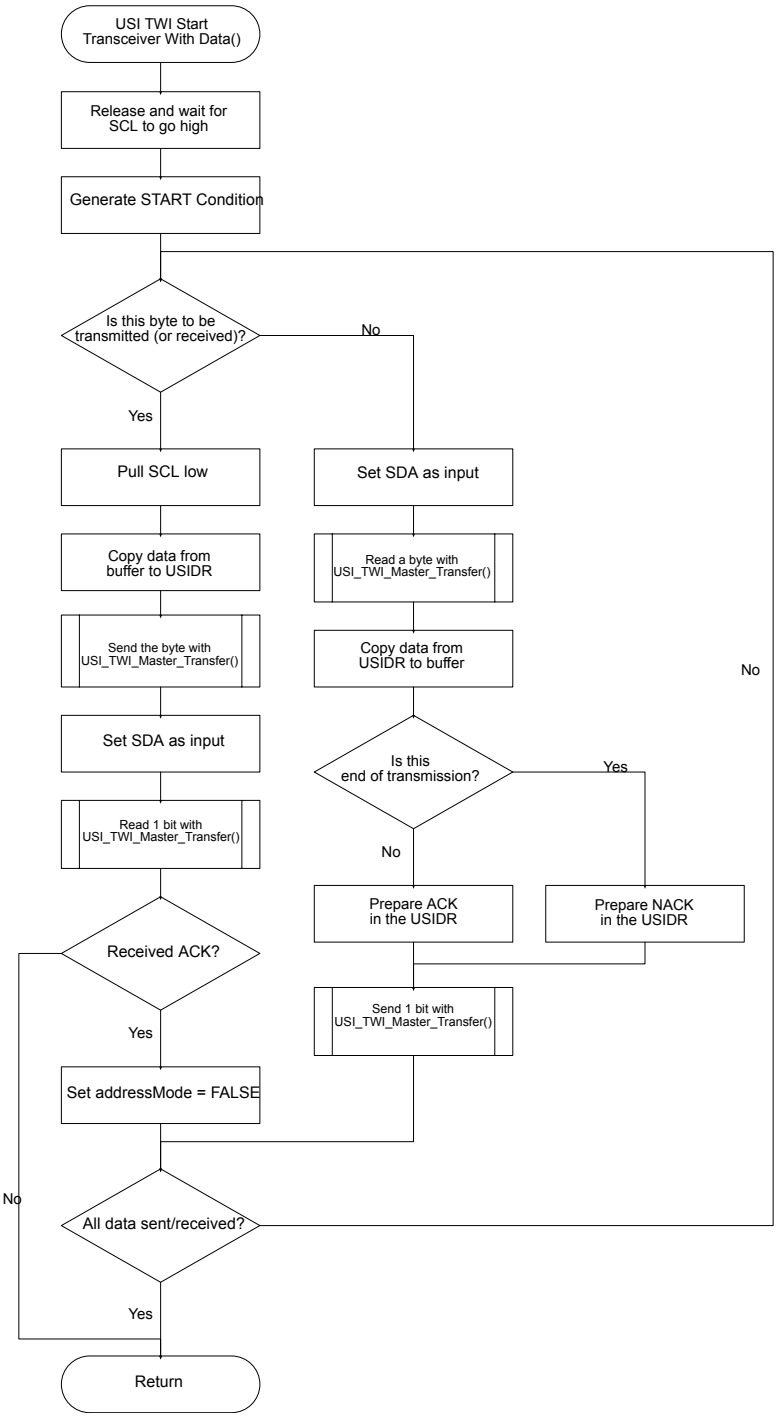
The USI Status Register (USISR) contains a 4-bit counter. Both the Serial Register and the counter are clocked simultaneously by the same clock source. This allows the counter to count the number of bits received or transmitted and sets a flag, alternatively generates an interrupt when the transfer is 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 generate flags when a start condition, data collision, or stop condition is detected on the Two-wire bus.

2. Implementation

The application note describes the implementation of a TWI master. The driver is written as a standalone driver that easily can 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 driver uses the USI module and standard I/O pin control. No additional resources as timers or any other interrupt sources are needed. The driver ensures correct timing even if it gets any interrupt signals during execution. The execution is however sequential, i.e. all CPU resources are therefore used during transmission.

Figure 2-1. Flowchart of the Transceiver Function. A Flowchart of the Sub Function USI_TWI_Master_Transfer is Found in the Next Figure



The driver consists of these functions:

- USI_TWI_Master_Initialize
- USI_TWI_Start_Transceiver_With_Data
- USI_TWI_Master_Transfer
- USI_TWI_Master_Stop
- USI_TWI_Get_State_Info

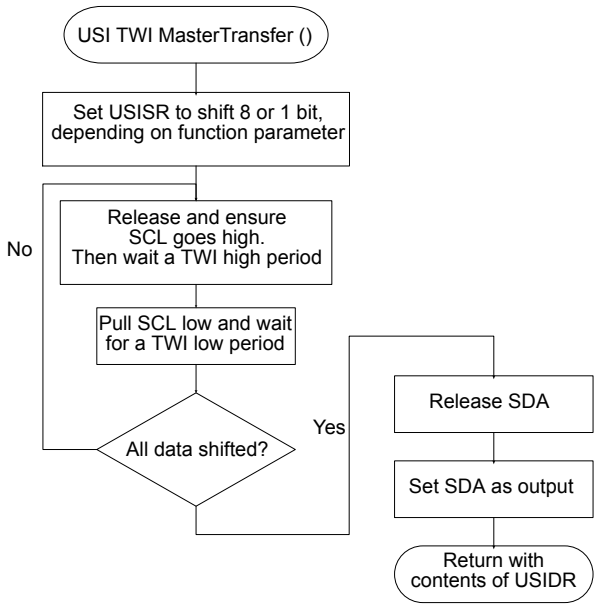
The *USI_TWI_Master_Initialize* function is used to set the USI module in TWI mode, and setting the TWI bus in idle/released mode.

The START and RESTART conditions are included into the transceiver function; *USI_TWI_Start_Transceiver_With_Data*. A flowchart of the function can be found in the figure above. The same function is used for both transmit and receive operations. The transceiver takes a pointer to a transmission buffer as parameter, together with the number of bytes in the buffer. The first location in the buffer must always contain both the address of the slave and the read/write bit determining the transmission type. If the master is requesting data from the slave, the transmit buffer only contains the slave address (with the read bit set), and a data size parameter indicating the number of bytes requested. The transceiver function will put the received data into the transmission buffer.

USI_TWI_Master_Transfer (see the figure below) is called from within *USI_TWI_Start_Transceiver_With_Data*.

USI_TWI_Master_Stop is called from within *USI_TWI_Start_Transceiver_With_Data*.

Figure 2-2. Flowchart of the General Transfer Function. The Function is Used by the USI_TWI_Start_Transceiver_With_Data Function in the Figure Above



On completion the transceiver function holds the TWI bus by pulling the SCL line low. A new transmission can be initiated immediately by rerunning the transceiver function. The transceiver function generates error codes if the transmission fails. The codes are listed in the header file and in the table below. Use the function USI_TWI_Get_State_Info to get hold of the error state if the transceiver returns a fail.

Table 2-1. Error Codes Returned from the Transceiver Function

Define name of error code	#	Description
USI_TWI_NO_DATA	0x00	Transmission buffer is empty
USI_TWI_DATA_OUT_OF_BOUND	0x01	Transmission buffer is outside SRAM space
USI_TWI_UE_START_CON	0x02	Unexpected Start Condition
USI_TWI_UE_STOP_CON	0x03	Unexpected Stop Condition
USI_TWI_UE_DATA_COL	0x04	Unexpected Data Collision (arbitration)

USI_TWI_NO_ACK_ON_DATA	0x05	The slave did not acknowledge all data
USI_TWI_NO_ACK_ON_ADDRESS	0x06	The slave did not acknowledge the address
USI_TWI_MISSING_START_CON	0x07	Generated Start Condition not detected on bus
USI_TWI_MISSING_STOP_CON	0x08	Generated Stop Condition not detected on bus

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. A small sample code of how to use the driver is included.

This implementation does not support TWI bus arbitration. The device using this driver must therefore be the only master on the bus. As according to the TWI standard, all 127 slaves can be addressed individually on the bus. The lack of bus arbitration is not a limit of the USI module, and can be implemented into the driver, but is not in the scope for this application note.

The driver does not use interrupts and uses loops to control the TWI bus activity. To add additional execution control and security one can use a Watchdog Timer. This can prevent unintentional behavior on the TWI bus from blocking the application. All AVR®s have an on-chip Watchdog Timer. For more information on the watchdog timer, check out the application note “AVR132: Using the Enhanced Watchdog Timer” and the datasheets.

The driver has code for both standard and fast mode TWI timing. Set selected mode in the header file of the driver. The default setting is fast mode.

2.1. Code Size

Table 2-2. Code Sizes with IAR EWAVR 3.10 with all Code Optimization On

Function	Size[bytes]
USI_TWI_Master_Initialize( )	28
USI_TWI_Start_Transceiver_With_Data( )	142
USI_TWI_Master_Transfer( )	56
USI_TWI_Master_Stop( )	24
USI_TWI_Get_State_Info( )	6
	256

3. Revision History

Doc.Rev.	Date	Comments
2561D	09/2016	New template
2561C	12/2013	Updated
2561B	09/2004	Updated
2561A	Unknown	Initial document release

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**