AVR32107: Using TWI as a Master on the AVR32

32-bit AVR® Microcontrollers

Application Note

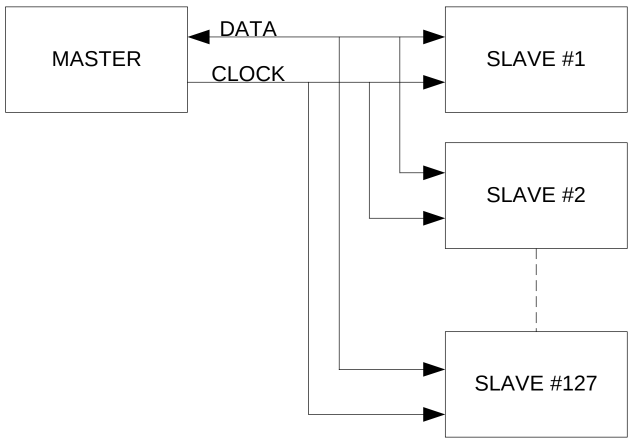
Features

- Compatible with Philips' I^{2}C protocol
- Master transmitter mode
- Master receiver mode
- 7-bit slave address – up to 127 devices on the same bus
- Normal (100kbps) and Fast (400kbps) operation
- Interrupt driven communication
- Sequential read and write operation
- Compatible with Standard Two-Wire Serial Memory
- Currently only master mode is supported

1 Introduction

The AVR®32 microcontroller communicate as the master on a TWI bus (Philips' I2C compatible Two-Wire Interface). Up to 127 TWI devices can be connected to each bus. The bus is half-duplex, and the current master device initiates transfers both ways.

Figure 1.1: Conceptual schematics



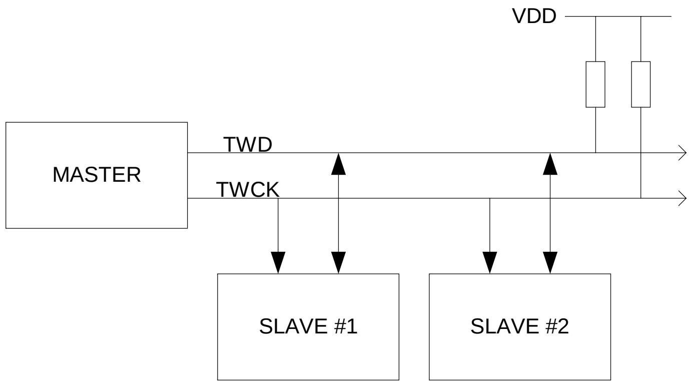
Rev. 32011A-AVR-04/06

2 Functional description

2.1 Electrical interconnection

The TWI bus consists of two lines, one for data and one for the clock signal (and possibly one for ground). Both bus lines are connected to the positive supply voltage through pull-up resistors. When a device wants to output a logical 0, it drives the bus line low and for logical 1, the output is tri-stated. In this way the device relies on the pull-up resistor to pull the line high. This technique is often referred to as wired AND. This is described in further detail in chapter 2.2.

Figure 2.1: Electrical schematics



2.2 Wired AND

Both lines in the TWI bus are connected to the power source with pull-up resistors. As a result, both lines on the bus will go high, if no device is driving the bus. The wired-AND is a result of that the devices currently transmitting on the bus releases it when logical 1 is to be transmitted, and only drives the bus when they are to transmit logical 0. Thus, the signal on the bus will be the result of an AND operation on all bits currently transmitted, and a device transmitting 0 will dominate over one transmitting 1.

2.3 Bus events: START and STOP conditions

In order to either start or stop a transmission, START and STOP conditions are issued. These conditions are defined as:

- START: The data line is driven low while by the master the clock line is high. Then the master drives the clock line low.
- STOP: The clock line is released (driven high), and then the master releases the data line.

The start condition for a TWI device may in some cases be defined by the master driving the data line low. The master will then drive the clock line low, independent the definition of the start condition. Whatever is the case, these transitions will generate a START condition.

2.4 Acknowledgement

Whenever the master is transferring data to a slave, every byte sent has to be acknowledged by the slave. This is done by pulling the data line low after each byte transmission. If the slave acknowledges the data, this is read as ACK (acknowledge) by the master. But if the slave fails to acknowledge, for some reason or another, the line is not pulled down and is read as NACK (not acknowledged) from the master's point of view.

2.5 Transfer format

Data is transferred MSB first, and each character is 8 bit long. There is no parity check, but every character must be followed by an acknowledgement. The data line (TWD) must remain stable while the clock line (TWCK) is high to ensure data consistency. The exceptions from this are the START condition and STOP condition. Every transmission is initiated by a START condition. When the master wants to end the transmission, a STOP condition is transmitted. Repeated START conditions can be used if the master wishes to address other devices without risking losing the bus to another master.

2.6 Transmitting data

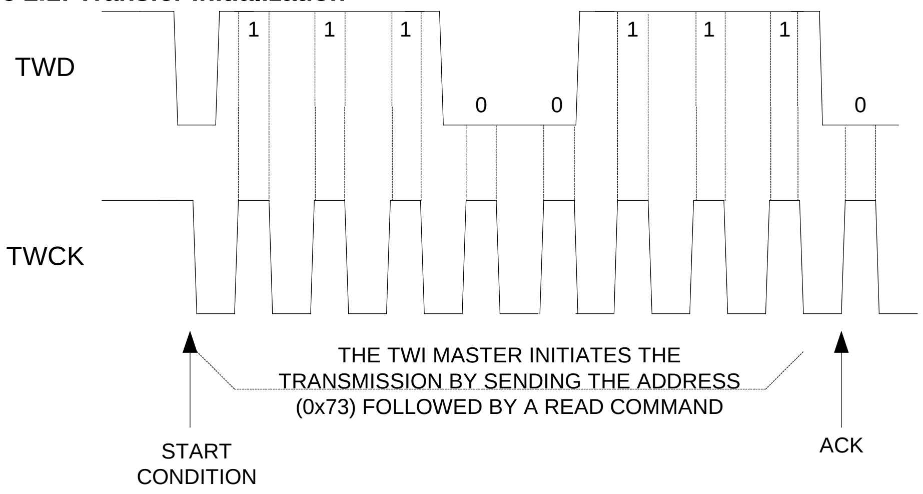
As TWI is half duplex, data can be written from the master to the slave or vice versa. These two operations have a lot in common, but there are also some differences. These two operations are described in this chapter.

2.6.1 Transmission initialization

To initiate a TWI transfer, a master must initiate a START condition. After the master initiates a START condition, it sends a 7-bit slave address. The 8th bit indicates the transfer direction (write or read). Read is logical 1, while write is 0. After the first byte, the master will release the data line, allowing a slave to drive it low. If a device on the bus recognizes its own address, it will pull down the bus on the following cycle, giving the master an ACK. In Figure 2.1 a master tries to access a device with address (id) 0x73 with a read operation and the slave acknowledges the request.

The Figure 2.1 shows how data is read on the TWI bus. When the clock is high, the data line must be stable and no transitions may be made. If such transitions do happen, unspecified behavior may occur.

Figure 2.1: Transfer initialization


TWD
1
1
1
0
0
1
1
1
0
TWCK
THE TWI MASTER INITIATES THE TRANSMISSION BY SENDING THE ADDRESS (0x73) FOLLOWED BY A READ COMMAND
START CONDITION
ACK

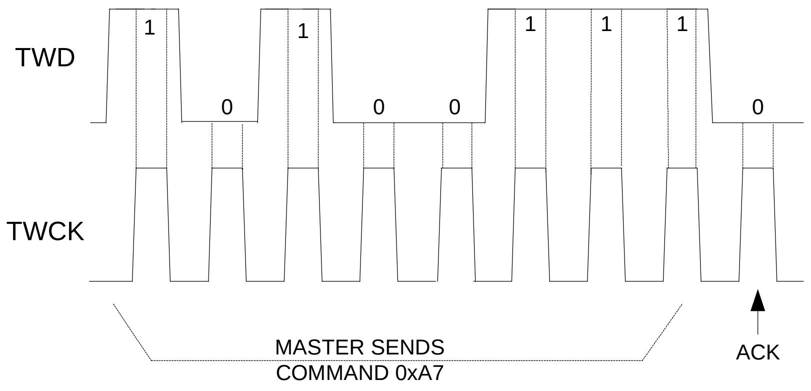
When a slave has been accessed, data is transmitted according to the read/write instruction in the initialization phase.

2.6.2 Read request

Before reading data from the slave device, a command may be given. This command can either be 0, 1, 2 or 3 bytes long. Each sequential command is followed by an acknowledgement by the slave in the same manner as in the initialization of the transfer. This command is device specific and may sometimes be referred to as an address for some devices.

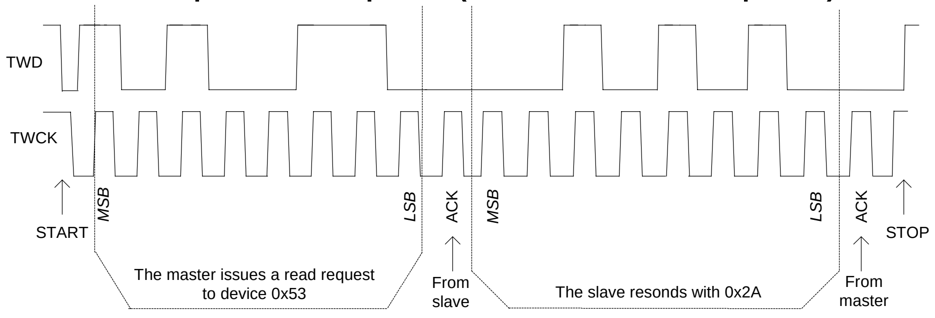
After the command sequence of a transmission, the slave device is responsible for the data line and drives it. The master is still responsible for driving the clock line. As the master is responsible for driving the clock line, no acknowledge bit is passed after a byte transfer. The most significant bit is always transmitted first. The slave will not change the data line while the clock is high. This is extremely important, as this may generate a START or STOP condition.

Figure 2.2: Command sequence



A complete read sequence is found in Figure 2.2. The master tries to contact a node with address 0x53 with a read instruction. The slave responds by acknowledging this. Then the slave puts out 0x2A as an answer to that request. In turn, the master acknowledges this and sends out a STOP condition, finalizing the transfer.

Figure 2.2: A complete read sequence (without command sequence)

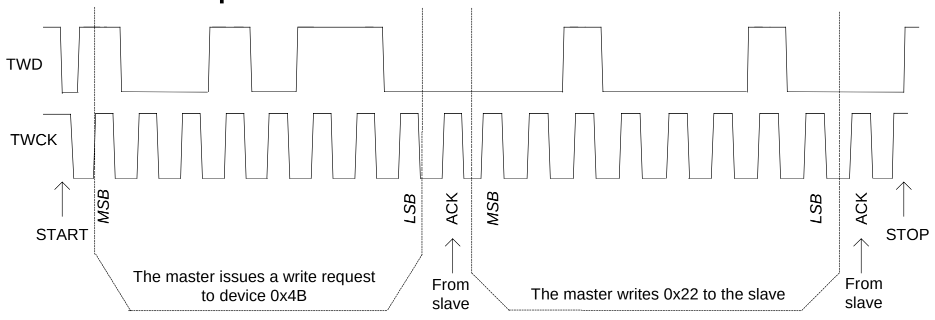

TWD
TWCK
START
MSB
LSB
ACK
MSB
LSB
ACK
STOP
The master issues a read request to device 0x53
From slave
The slave resonds with 0x2A
From master

2.6.3 Write request

If a write request is made to a slave during initialization, the master is responsible for driving the data line and feeding the slave with bytes. After each transmission of a byte, the slave must acknowledge the byte sent. This is done in the same matter as acknowledging a transfer request, described in chapter 2.4.1.

The transmission ends when the transmitter stops sending, or if the transmitter does not receive an ACK for the transmitted character. The master will then set a STOP condition on the bus, releasing it, or initiate a new session by sending another START condition.

Figure 2.3: A write operation


TWD
TWCK
START
MSB
LSB
ACK
MSB
LSB
ACK
STOP
The master issues a write request to device 0x4B
From slave
The master writes 0x22 to the slave
From slave

2.7 More than one master - arbitration

Only one master can be in control of the communication on the bus at any time. No other device will interfere with an ongoing transmission. If the bus is in use and another master starts driving the bus lines, hazardous behavior may occur. The master currently in control is providing the clock signal on the bus.

However, if two masters initiate a START condition simultaneously, they will both start transmitting clock and data signal. To avoid corruption of data, one of them has to switch to idle or slave mode. This problem is solved through arbitration; both (or all) masters start transmitting, but the moment one of them discovers that the TWD line is driven by some other master, it will stop transmitting, and go to idle or slave mode. The mode is dependent of the features supported by the TWI master. A consequence of the wired-AND logic is that if 1 and 0 is transmitted on a bus wire simultaneously, the bus will read 0. Every master transmitting data will read the value on the bus to see if they really are driving the bus. As soon as a device tries to transmit 1, but the bus reads 0, it will know that some other device is transmitting, and switch to slave or idle mode.

The result is that the master sending the lowest address character will win the bus (if two or more masters are trying to address the same device, the arbitration will continue in the direction bit, and possibly the data bits). The AVR32 microcontroller is unable to enter slave mode. Thus, when it loses the bus, it will enter idle mode until a STOP condition is transmitted on the bus, signaling that the master currently in control of the bus is done. When a STOP condition is detected, and the AVR32 wants to communicate on the bus, a START condition is immediately transmitted.

2.8 Usage

Before any data can be sent on the TWI bus, it must first be initialized. Then a specific device can be probed to check whether it is responding or not. It is then also possible to receive or send data. The functions for these operations are described in chapter 2.6.1.

Baudrate(s) for your TWI slave is described in the appropriate datasheet for your device. It is important to select the correct baudrate, as it is the master's responsibility to drive the clock line.

3 Package information

Included with the application note is a driver package. This package contains drivers, example code and documentation.

3.1 Drivers

Drivers are available in the package. These drivers are written to be independent of a specific compiler and are successfully tested on gcc and IAR Embedded Workbench.

3.2 Examples

Examples are available from the corresponding driver package. All functionality is divided into libraries and an example that utilizes the library.

3.3 Documentation

Function specific documentation is available in the package. Refer to readme.html in the source code directory.

4 Further reading

The AVR32 TWI also has support for Direct Memory Access (DMA) and interrupt-driven communication. These two concepts are described in two other Application notes, namely AVR32108 title and AVR32109 title respectively.