Atmel AVR290: Avoid Clock Stretch with Atmel tinyAVR

Features

- **Atmel® tinyAVR® family devices**
- **C-code driver for the Atmel TWI slave**
- **Compatible with Philips I²C protocol**
- **Uses the Atmel TWI hardware module**
- **Interrupt driven transmission**
- **No clock stretching – true 100kHz operation**

1 Introduction

The serial Atmel two-wire interface (TWI) bus is compatible with the Philips I²C protocol. The bus was developed to allow simple, robust, and cost-effective communication between integrated circuits in electronic devices.

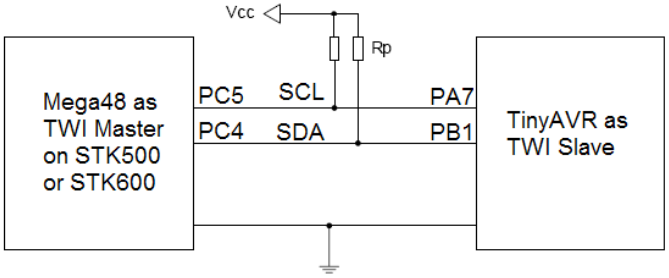
The strength of the TWI bus includes the capability to address up to 128 devices on the same TWI bus arbitration, and the ability to have multiple masters on the same bus.

The Atmel tinyAVR family, and, in particular, the Atmel ATtiny20 and ATtiny40 devices, contain an improved TWI module that generates a faster ACK response, thereby allowing data rates as fast as 100Kbps. This module is also I²C and SMBus compliant.

This application note describes a TWI slave driver for the ATtiny20 and ATtiny40. A slave software driver is included. The application note explains how the TWI SCL clock is not stretched, provided that the AVR® clock speed is fast enough to release the serial clock (SCL) and the acknowledge (ACK) on the serial data (SDA) signals.

For demonstration purposes, a TWI master is included in the software folder. It is a modified Atmel AVR315 program. Figure 1-1 shows a block diagram of the configuration.

Figure 1-1. Configuration for this application note.



8-bit Atmel Microcontrollers

Application Note

Rev. 8380A-AVR-03/11

2 Prerequisites

The Atmel TWI slave and master demo discussed in this document requires basic familiarity with the following skills and technologies. Refer to Chapter 10, References, to learn more.

- C programming language for embedded systems
- Compiling C projects with IAR Embedded Workbench® for Atmel AVR
- IAR™ kick start version of the IAR Embedded Workbench for Atmel AVR C compiler, version 5.51, or later, from ftp://ftp.iar.se/WWWfiles/datasheet/New/DS-EWAVR-551.pdf
- WinAVR GCC C compiler is not supported at this time; however, the code could be converted to compile with WinAVR
- Atmel AVR Studio® 4.18 or newer
- Atmel STK®500, or Atmel STK600, with adapter for the Atmel ATtiny20
- The Atmel AVR JTAGICE mkII debugger, optional for debugging if code modification is planned
- General familiarity with the TWI interface, detailed in the ATtiny20 datasheet

3 Limitations

- The Atmel TWI slave driver is targeted to run on the Atmel ATtiny20 or the Atmel ATtiny40, with its TWI peripheral only. Support for other devices may require modifications to the TWI slave driver code
- The software solution with this application note is tested on the IAR Embedded Workbench for Atmel AVR version 5.51, or later. Earlier versions of the compiler may require some modifications
- If the ATtiny20 CPU clock is set by application software to be 8MHz, there is a limited TWI rate of 100Kbps

4 Resource requirements

Table 4-1. Typical requirements.

Peripheral	Pins	Configurable?
Atmel TWI slave	SCL, SDA	No
PORTA	PA0-PA6	For demo only
Internal EEPROM	Not available in this device	

Table 4-2. Memory requirements [1].

Memory	Typical size	Maximum size
Program memory	514bytes	2048bytes
Data memory	73bytes	128bytes
Internal EEPROM memory	None in this device	

Note: 1. Exact memory requirements depend on a variety of factors, such as compiler version, optimization levels, addition or removal of configurable functionality.

5 TWI slave driver: How it avoids clock stretch

5.1 Optimized code for minimum execution time

The slave driver described here has been optimized to achieve zero clock stretch. This is accomplished by reducing the number of AVR instructions in the slave driver, thus requiring a minimum amount of execution time. This allows the slave driver to release the SCL clock and the ACK fast enough for 100Kbps operation and eliminate SCL clock stretch. The following instruction in the slave driver releases the SCL clock so that it is not stretched.

```
TWI_SLAVE_CTRLB=(uint8_t)((1<<TWI_SLAVE_CMD1)|(1<<TWI_SLAVE_CMD0));
```

IMPORTANT

To prevent SCL clock stretch, the above instruction must be executed within one bit time of ISR entry. Example: if the master is running at 100Kbps, equivalent to 100kHz, one bit time is 10 microseconds (µs).

Figure 5-1. ACK and SCL no-stretch oscilloscope image.

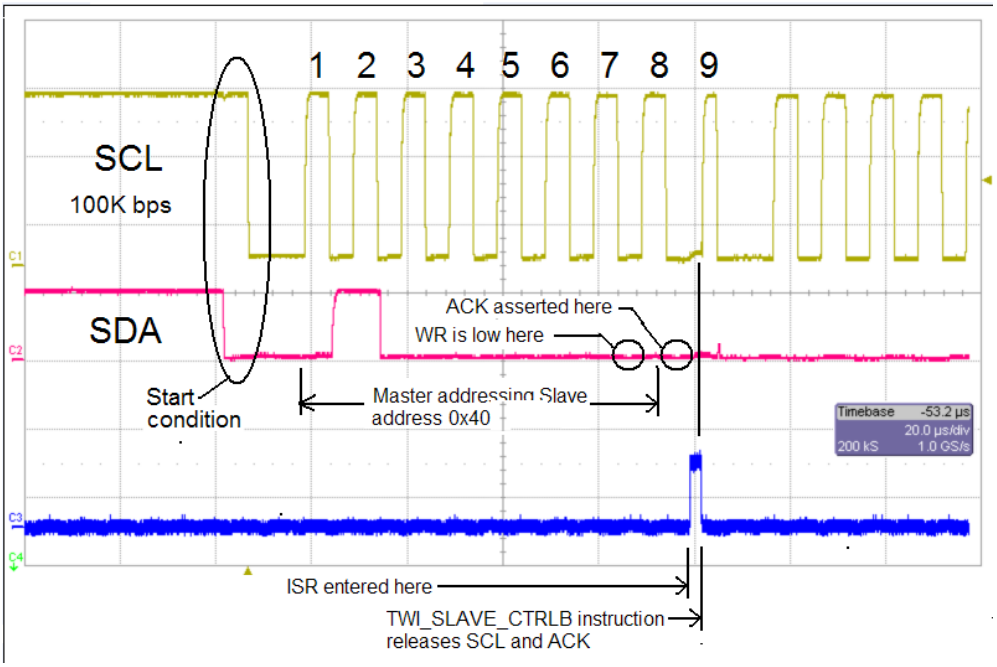


Figure 5-1 shows the sequence of the master addressing the slave.

IMPORTANT

Observe that the slave address is 0x40 = 0b01000000. However, within the slave and master code, this address is referred to as 0x20. The TWI software shifts this 0x20 one bit left to allow room for the ACK.

The slave responds to the master as follows:

1. The ISR is entered, indicated by the rising edge shown on scope channel 3 (C3). (This is done by using PORTB bit 0 for explanation purposes only.)
2. The slave holds the SDA data signal low, as well as holding the SCL clock signal low.

3. Within one bit time, shown here as 10µs, the slave driver software executes an instruction to the TWI_SLAVE_CTRLB register. This releases SCL, thereby not stretching the SCL clock.

5.2 CPU clock speed is critically important

This application note achieves zero clock stretch at 100Kbps only because the slave driver executes relatively few instructions and the CPU clock speed is fast enough. In this example, 8MHz is selected. The Atmel ATtiny20 and ATtiny40 can operate at 12MHz, but only via an external oscillator source.

Table 5-1. Maximum Kbps for various CPU clock speeds.

Clock speed (MHz)	Maximum Kbps	Maximum time in application code (1)
12	150	666µs
8	100	1ms
4	50	2ms
2	25	4ms
1	12.5	8ms

Note: 1. At a data rate of 100Kbps, a new byte from the applications' code could be sent to or from the master at a rate of one byte per millisecond (ms). Care must be directed not to spend too much time in the application code when the master requests data to be sent or received faster than this rate.

5.3 How to connect the TWI master and slave

For this demo, the master consists of an Atmel STK500 or STK600 with an Atmel ATmega48 running its RC oscillator at 8MHz. Un-check the ATmega48 divide-by-8 fuse to enable it to run at 8MHz.

The software details of the master and slave are explained below. Refer to Figure 1-1 when connecting master to slave.

The slave consists of an ATtiny20 running on a STK600.

1. Connect jumper wires for SCL and SDA as shown in Figure 1-1.
2. Two 2kΩ resistors are recommended for Rp.
3. A third jumper is necessary to connect grounds on each board.
4. On the master board, connect a 10-pin jumper between the eight switches, SW0-SW7, and a second 10-pin cable between PORTB and the LED port.
5. On the slave board, connect a 10-pin cable between PORTB and the LED port. The LEDs will display the value of the count1 variable, which increments once every three seconds.

6 How to build and run the software

6.1 Atmel TWI ATtiny20 slave software

1. From the www.atmel.com website, locate the zip-compressed software file associated with this Atmel AVR290 application note. In the zip file directory `twi_slave_tiny`, examine the following files:
   `twi-no_stretch-tiny20-demo.c`
   `twi-no_stretch-tiny20-drvr.c`
   `twi-no_stretch-tiny20-drvr.h`
2. Unzip the file. Locate the `iar` folder.
3. After installation of the IAR Embedded Workbench for Atmel AVR C compiler, open the IAR .eww workspace and compile the project.
4. A .hex file will be generated. Using the Atmel STK600 or another AVR programmer, download this .hex file into the Atmel ATtiny20.
5. Power cycle the ATtiny20. Seven LEDs should slowly sequence upward in a binary pattern, one increment per three seconds.

6.2 How to call and use the functions

Callback functions exchange data between the Atmel TWI driver and the application code. These functions, defined in the application code, are:

`twi_data_from_master()` and `twi_data_to_master()`

unsigned char twi_data_to_master (void)
The Atmel TWI slave driver callback function `twi_data_to_master` is an example of command-specific response logic. It demonstrates how the TWI driver gets data from the main application code.

Input: None.

Output: This function returns a value in count1 or ~count1 based on the value of the command previously sent from the master to the slave and this code.

unsigned char twi_data_from_master (unsigned char)
This driver callback function, an example of command-specific response logic, allows the slave driver to send data to the main application code.

Input: Data from the slave driver. Specifically, the master sends data to the slave, and this data is the input to the function. In the application code, this data is referred to as the command.

Output: Data available to the application code.

void twi_slave_initialise (void)
This function initializes the Atmel AVR TWI hardware module to allow the TWI master to write to or read from the slave. No variables are involved.

void twi_bus_error_check (void)
This function tests for errors due to multiple TWI devices attempting to use the TWI bus at the same time; that is, collisions. The function is included here for reference only, and is not used by the code at this time.

6.3 How to build and run the modified TWI master software

A modified version of the TWI master implementation described in the Atmel AVR315 application note serves as an TWI master for this demo, although another master of the user's choice could be used instead. The modifications include changing the slave address to 0x20 and redefining the SW0 and SW1 switches to send specific commands to the Atmel ATtiny20 slave.

1. Locate the Atmel AVR290 software folder, previously unzipped as described in Section 6.1.
2. Locate the `twi_master_demo` sub-folder.
3. Using the installed IAR Embedded Workbench for Atmel AVR C compiler, open the IAR .eww workspace and compile the project.
4. A .hex file will be generated. Using the Atmel STK500 or another AVR programmer, download the compiled and linked .hex file into the Atmel ATmega48, which has been wired to the slave, as previously described, using SDA, SCL, GND, and the two 2kΩ pull-up resistors.
5. Run the downloaded code. Once the slave program is running, a press and release of switch SW0 on the STK500 will initiate the TWI master to access the ATtiny20 slave and request the value of count1, which will be displayed on the master's LEDs.
6. Similarly, a press and release of switch SW1 on the STK500 will initiate the TWI master to access the ATtiny20 slave and request the inverted (binary compliment) value of count1, which will be displayed on the master's LEDs.
7. The SCL clock frequency can be changed on the master as follows:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot (PrescalerValue)}$$

This frequency is set as follows

1. Locate the `TWI_master.h` file.
2. As an example, to set the application software to 100Kbps, set TWI_TWBR = 0x22.

7 How to observe the bps data rate and the TWI no-stretch

1. Connect an oscilloscope to the demo, as follows: channel 1 on SCL, channel 2 on SDA.
2. Measure the period between SCL clock edges.
    a. 100Kbps is equivalent to 100kHz, or 10µs between clock edges.
3. See Section 6.3 for details on to how to change the bit-per-second (bps) rate.

Figure 7-1. Complete TWI master-slave access.

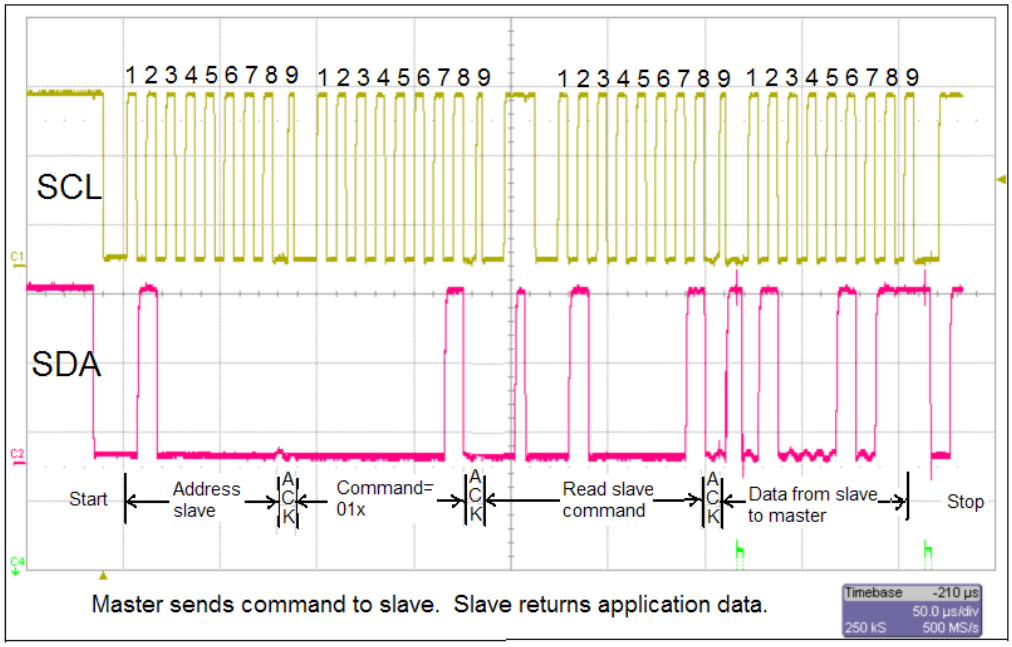


Figure 7-1 is the actual oscilloscope waveform of a complete master to slave request for data. The data rate is 100Kbps. Observe the following:

1. After the start command, the TWI master asserts the slave address of 0x40 (2 × 0x20).
2. This is followed by a no-stretch ACK and a release of the 9th SCL clock rising edge.
3. Next, the master sends a command of 0x01 (alternately, 0x02 for this demo) followed by a no-stretch ACK and a release of the 9th SCL clock rising edge.
4. The slave software responds to the master command via the slave callback function `twi_data_from_master()`.
5. The master issues a “read slave” command, indicated by the SDA signal being high at the 8th SCL clock rising edge. This is followed by a no-stretch ACK and a release of the 9th SCL clock rising edge.
6. The slave responds to the master’s read request as follows:
    The slave driver software accesses the application via the callback function `twi_data_to_master()`. This function is located in the application code, and supplies an application variable to the slave driver.
7. The slave TWI module places the data on the SDA bus. The TWI master software receives this data and displays it on the LEDs on the Atmel STK500 master’s PORTB, if used.

8 TWI theory

This section gives a short description of the TWI interface and the TWI module on Atmel AVR devices, including Atmel tinyAVR. For more detailed information, please refer to the AVR datasheet.

8.1 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 TWI devices connected to the bus have individual addresses. They also have mechanisms for resolving bus contention.

The TWI bus is a multi-master bus, where one or more devices capable of taking control of the bus can be connected. Only a master device 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 a (unique) 7-bit slave address, and then by an 8^{th} bit, the data direction bit. The addressed slave device sends an acknowledge (ACK) to the master by holding SDA low for the 9^{th} clock cycle. The transfer is terminated if the master does not receive any acknowledgment.

Depending on the data direction bit, the master or the slave 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 SCL is high.

All data transmissions on the TWI bus are nine bits long, consisting of eight bits of data (one data byte) and an acknowledge bit (ACK).

During a data transfer, the master generates the clock and the START and STOP conditions. The TWI slave is responsible for acknowledging the reception. An acknowledge signal (ACK) is generated by the slave, which pulls the SDA line low during the 9^{th} SCL clock cycle. If the receiver leaves the SDA line high, a no-acknowledge (NACK) is sent.

8.2 TWI clock stretching

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

8.3 Atmel tinyAVR slave module – no-stretch operation

If the TWI master's application doesn't support clock stretching, this Atmel AVR290 application note demonstrates 100Kbps slave operation with an Atmel ATtiny20 or an Atmel ATtiny40.

The TWI slave is byte-oriented, and demonstrates interrupts after each byte. 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.

9 Conclusion

The Atmel ATtiny20 and ATtiny40 support 100Kbps TWI slave operation. The 100Kbps rate is a limitation based on the tinyAVR device running at 8MHz, as described in Section 5.2. The software example provided in this application note explains how to process this TWI interface in slave mode. The code in this application note compiles with the IAR C Compiler, but does not compile with the GCC compiler, at this time.

9.1 Doxygen documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code automatically 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.

10 References

- Atmel AVR Studio
- Atmel AVR1320: True 400kHz operation for TWI slave
- Atmel AVR315: Using the TWI module as I^2C master USB Specification – appropriate URL
- Atmel AVR JTAGICE mkII
- Embedded Workbench for Atmel AVR C compiler, version 5.51, or later, from ftp://ftp.iar.se/WWWfiles/datasheet/New/DS-EWAVR-551.pdf

11 Table of contents

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

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

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

8380A-AVR-03/11