AVR918: Using the Atmel Tiny Programming Interface (TPI)

8-bit AVR® Microcontrollers

Application Note

Features

- **TPI driver for devices with TPI support**
- **Programming example for:**
  - **NVM read and write**
  - **Lock-byte read and write**
  - **Fuse-byte read and write**
  - **Calibration byte read**
  - **Device ID read**
  - **Chip erase**
- **Compatible with AVROSP (AVR911)**

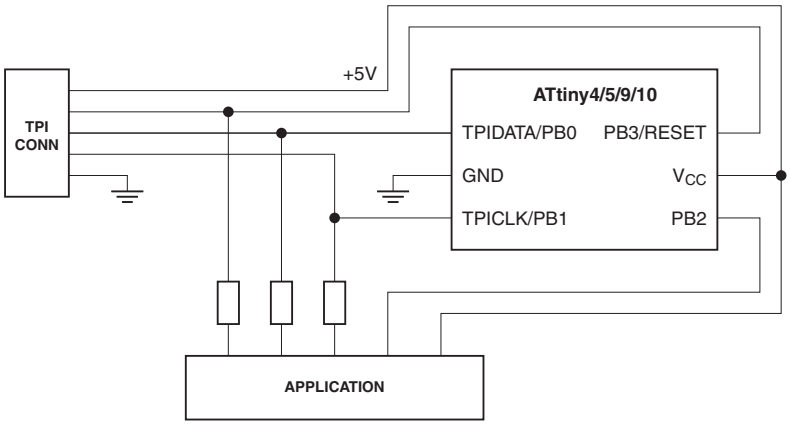
1 Introduction

The Atmel® Tiny Programming Interface (TPI) is featured on selected low-end Atmel AVR® microcontrollers, and allows external programmers to access the nonvolatile memory (NVM) of the device. The interface provides access to device lock bits, the program flash memory, and the signature, configuration, and calibration sections.

The TPI can be accessed via three pins:

- RESET: TPI enable input
- TPICLK: TPI clock input
- TPIDATA: TPI data input/output

Figure 1-1. The Atmel Tiny Programmer Interface for external programmers.



Refer to the corresponding device datasheet for more details on the TPI protocol.

This application note describes how to use an AVR microcontroller to access a device with TPI. The design uses the AVR open source programmer (AVROSP), as described in the Atmel AVR911 application note.

Rev. 8373A-AVR-02/11

2 Nonvolatile memories

This section describes the nonvolatile memory (NVM) sections that can be accessed via the Atmel Tiny Programming Interface (TPI). Access methods described in this section are specific to devices with TPI, and may not apply to other Atmel AVR microcontrollers.

The embedded NVM has:

- Nonvolatile memory lock bits
- Flash memory with four sections

2.1 Nonvolatile memory lock bits

Lock bits provide additional security for the device, when programmed. Lock bits can only be erased by a chip erase; hence, it is required to program them after the other NVM sections are programmed. By default, lock bits are un-programmed (set to 1).

2.2 Flash memory

The embedded flash memory has four sections:

- Code (program memory)
- Configuration
- Signature
- Calibration

2.2.1 Code (program memory) section

As the program memory cannot be accessed directly, it has been mapped to the data memory. The mapped program memory begins at byte address 0x4000 in data memory (see device datasheets for further information). This means that programs in program memory are executed starting from address 0x0000, but the same memory area is addressed starting from 0x4000 when accessed via the data memory.

Internal write operations to flash program memory have been disabled, and program memory, therefore, appears to firmware as read-only. Flash memory can only be written to by an external programmer.

2.2.2 Configuration section

The configuration byte resides in the configuration section. The following functions can be configured by writing appropriate values to the configuration byte:

- Brown-out detection level
- System clock output on port pin
- Watchdog timer on
- External reset disable

Changes to the configuration bit values will take effect after the device leaves programming mode.

Please consult individual device datasheets for further details on available functions.

2.2.3 Signature section

The signature section is used to store information such as the device signature. Typically, the device signature consists of three bytes.

Consult individual device datasheets for signature values.

2.2.4 Calibration section

The calibration section typically contains one calibration byte for the internal oscillator. This byte contains the calibration value, which is stored at device production. The calibration section is read-only.

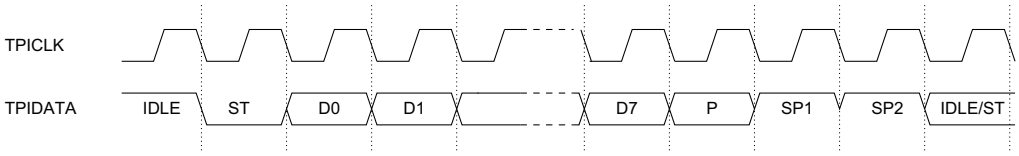
3 TPI target implementation

The Atmel Tiny Programming Interface (TPI) consists of two layers, the access layer and the physical layer. The TPI physical layer supports two modes of operation, transmit and receive. By default, the physical layer is in receiving mode, and waiting for a start bit. The TPI access layer controls the mode of operation.

3.1 TPI frame

The TPI physical layer supports a fixed frame format. A frame consists of one character, eight bits in length, one start bit, an even parity bit, and two stop bits. Data is transferred with the least-significant bit first. A break character of a 12-bit-long low level is supported; this can be extended beyond 12 bits, also.

Figure 3-1. Serial frame format.



Symbols used in Figure 3-1:

ST: Start bit (always low)
D0-D7: Data bits (least-significant bit sent first)
P: Parity bit (using even parity)
SP1: Stop bit 1 (always high)
SP2: Stop bit 2 (always high)

3.2 TPI physical layer

The TPI physical layer handles the basic low-level serial communication. The physical layer uses a bidirectional, half-duplex, serial receiver and transmitter. It includes serial-to-parallel and parallel-to-serial data conversion, start-of-frame detection, frame-error detection, parity-error detection, parity generation, and collision detection.

The TPI physical layer operates synchronously on the TPICLK signal provided by the external programmer. Data is changed at falling edges, and sampled at rising edges.

3.2.1 Serial data reception

In receive mode, data reception is started as soon as a start bit has been detected. When the complete frame is present in the shift register, the received data will be available for the TPI access layer.

There are three possible exceptions in the receive mode: frame error, parity error, and break detection. All these exceptions are signaled to the TPI access layer, which then enters the error state, and puts the TPI physical layer into receive mode, waiting for a break character.

3.2.2 Serial data transmission

Transmission is initiated by loading the shift register with the data to be transmitted. When the shift register has been loaded with new data, the transmitter shifts one complete frame out on the TPIDATA line at the transfer rate given by TPICLK.

If a collision is detected during transmission, the output driver is disabled. The TPI access layer enters the error state, and the TPI physical layer is put into receive mode, waiting for a break character.

3.2.3 Direction change

A guard-time mechanism is implemented in the physical layer to ensure correct timing during direction change. When the TPI physical layer changes from receive to transmit mode, a configurable number of additional idle bits are inserted before the start bit is transmitted. The default guard time is 128 bits. The minimum transition time between receive and transmit mode is two idle bits. The total idle time is the specified guard time plus two idle bits.

When the external programmer changes from receive mode to transmit, a minimum of one idle bit should be inserted before the start bit is transmitted.

3.3 TPI access layer

The TPI access layer controls the character transfer direction on the TPI physical layer. It also handles the recovery from the error state after an exception. TPI access layer handles the communication in message format. Each message consists of a byte-sized instruction followed by operands. The instructions are always sent from the programmer, while the operands can be from either the programmer or the device, depending on the type of the instruction.

The control and status space (CSS) of the Atmel Tiny Programming Interface is allocated for control and status registers in the TPI access layer. The access layer can also access the data space, either directly or indirectly, using the pointer register (PR) as the address pointer.

Consult the individual device datasheet for further information on exception handling.

4 External programmer implementation

The programmer described in this application note uses a USART to implement the TPI. The USART is used in synchronous mode, and XCK is used for TPICLK. The AVROSP, described in the Atmel AVR911 application note, is used as the user interface, and the programmer implements the AVROSP protocol via another USART.

4.1 Device selection

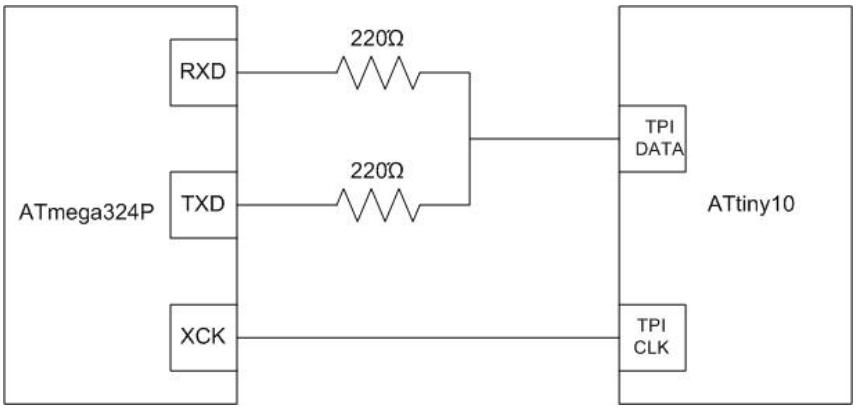
The implementation is tested with the Atmel ATmega324P, though any Atmel AVR device with two onboard USARTs can be used with minimal changes to the implementation. The device used in the external programming requires two USARTs, one for implementing TPI and another for AVROSP communication.

The device is driven by an external 11.059MHz crystal.

4.2 Target interface

The XCK pin is used as TPICLK, and RXD and TXD have been connected to TPIDATA via two 220Ω resistors, as shown in Figure 4-1.

Figure 4-1. TPI target interface.



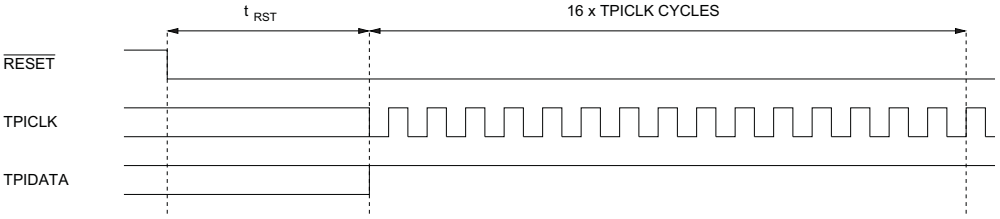
4.2.1 TPI enabling

The following sequence enables the TPI (see Figure 4-2 for guidance):

- Apply 5V between V_{CC} and GND
- Wait t_{TOUT} and then set the RESET pin low. This will reset the device and enable the TPI physical layer. The RESET pin must then be kept low for the entire programming session
- Wait t_{RST}
- Keep the TPIDATA pin high for 16 TPICLK cycles

NOTE Consult individual device datasheet for t_{TOUT} and t_{RST} values.

Figure 4-2. TPI enable sequence.



4.2.2 TPI instruction set

TPI has a compact instruction set, as shown in Table 4-1. These instructions can be used to access the NVM control status space and the data space. The external programmer can access the NVM controller and the NVM memories by using these instructions. All the instructions are one byte. Except for the SKEY instruction, all other instructions require one byte of operand to follow. SKEY requires eight bytes to follow.

Table 4-1. TPI instruction set.

Mnemonic	Operand	Description	Operation
SLD	data, PR	Serial LoaD from data space using indirect addressing	data ← DS[PR]
SLD	data, PR+	Serial LoaD from data space using indirect addressing and post-increment	data ← DS[PR]
SST	PR, data	Serial STore to data space using indirect addressing	DS[PR] ← data
SST	PR+, data	Serial STore to data space using indirect addressing and post-increment	DS[PR] ← data
SSTPR	PR, a	Serial STore to Pointer Register using direct addressing	PR[a] ← data
SIN	data, a	Serial IN from data space	data ← I/O[a] SIN data
SOUT	a, data	Serial OUT to data space	I/O[a] ← data
SLDCS	data, a	Serial LoaD from Control and Status space using direct addressing	data ← CSS[a]
SSTCS	a, data	Serial STore to Control and Status space using direct addressing	CSS[a] ← data
SKEY	Key, {8{data}}	Serial KEY	Key ← {8{data}}

4.2.3 Entering external programming mode

In order to enter the programming mode, it is required to enable the TPI communication by following the procedure described in Section 4.2.1, TPI enabling, on page 6. After enabling the TPI:

- Send an SKEY instruction via the TPI
- Send the NVM program enable data (0x1289AB45CDD888FF) byte by byte via the TPI
- Poll the status of the NVMEN bit in TPISR until it has been set

4.2.4 Exiting external programming mode

In order to exit the programming mode, the following procedure must be followed:

- Clear the NVM enable bit in TPISR
- Release RESET

4.2.5 Accessing the NVM

NVM is mapped to the data memory, and, hence, can be accessed via those mapped locations in the data memory.

The programming task is started after loading the appropriate NVM command (Table 4-2) to the nonvolatile memory command register (NVMCMD). The NVMBUSY flag bit in the nonvolatile memory control and status register (NVMCSR) will be set when the NVM controller is performing the instructed operation. The NVM command register is blocked for write access as long as the NVMBUSY flag is active. This is to ensure that the current command is fully executed before starting a new command.

4.2.5.1 NVM commands

NVM commands can be classified into general, section, and word operations, depending on the operation. In total, there are four commands for accessing NVM, as described in Table 4-2.

Table 4-2. NVM commands.

Operation type	NVMCMD	Mnemonic	Description
General	0x00	NO_OPERATION	No operation
General	0x10	CHIP_ERASE	Chip erase
Section	0x14	SECTION_ERASE	Section erase
Word (1)	0x1D	WORD_WRITE	Word write
Double word (2)	0x1D	DWORD_WRITE	Write double word
Flash words (3)	0x1D	CODE_WRITE	Write flash words

Notes: 1. ATtiny4/5/9/10
2. ATtiny20
3. ATtiny40

4.2.5.2 Addressing the flash

The data space uses byte accessing, but because the flash sections are accessed as words and organized in pages, the byte address of the data space must be converted to the word address of the flash section. The page address and the word address together form the absolute address of a word in flash. The most-significant bits of the data space address select the NVM lock bits or the flash section mapped to the data memory. The least-significant bit of the flash section address is used to select the low or high byte of the word.

During programming, the data space location is pointed to by the pointer register (PR), where the address must be stored before data is accessed. The sequence to store an address in the pointer register is as follows:

- Send a SSTPR instruction with the LSB set to 0
- Send the low byte of the address
- Send a SSTPR instruction with the LSB set to 1

- Send the high byte of the address

4.2.5.3 Reading the flash

The flash can be read from the data memory mapped locations one byte at a time. For read operations, the least-significant bit (bit 0) is used to select the low or high byte in the word address. If this bit is zero, the low byte is read, and if it is one, the high byte is read. The sequence to read a data byte from NVM is as follows:

- Send a SLD_POSTINC instruction
- Wait for an idle time of two clock cycles plus the guard time set in TPIPCR for reception
- Receive the data byte

4.2.5.4 Programming the flash

The flash can be written word-by-word or using multiple words (two or four) at a time, depending on the device used. The target location must be erased before a write operation. Otherwise, the flash word will corrupt its content. The erase operation can only be performed for the entire section; hence, it is required to erase the flash section before starting a write.

The low bytes must be written to the temporary buffer first, and then writing the high bytes will trigger the flash write operation.

During a multiple-word write, the low bytes and high bytes of all the words should be written in correct order to start the flash write operation.

The sequence to write flash words is as follows:

- Send the corresponding memory write command
- Send a SST_POSTINC instruction
- Send the low byte of the data to be written
- Send a SST_POSTINC instruction
- Send the high byte of the data to be written
- If the device supports programming more than one word at a time, send one idle character, and repeat the write sequence above for the required number of words (two or four) to be written
- Wait for the NVMBUSY bit to be cleared

While programming the configuration section on devices that support programming multiple words, dummy bytes have to be written to the configuration section after writing the actual data.

4.2.5.5 Chip erase

The chip erase command will erase the entire code section of the flash memory and the NVM lock bits. The NVM lock bits are not reset before the code section has been completely erased. Please note that the configuration, signature, and calibration sections are not affected by a chip erase. The sequence for performing a chip erase is as follows:

- Send a CHIP_ERASE command
- Set the address to a location inside code
- Write a dummy byte to the high byte of the location addressed
- Wait for the NVMBUSY bit to be cleared

4.2.5.6 Erase section

The section erase command will erase the selected NVM section. Code section, NVM lock bits, and configuration sections can be erased. The calibration section is read only, hence erase or write operations cannot be performed on it.

- Send a SECTION_ERASE command
- Set the address to a location inside selected section
- Write a dummy byte to the high byte of the location addressed
- Wait for the NVMBUSY bit to be cleared

5 User interface implementation

The Atmel TPI programmer described in this application note supports the Atmel AVR open source programmer (AVROSP). AVROSP is an Atmel AVR programmer application equivalent to the Atmel AVRProg tool included with Atmel AVR Studio®. It is a command-line tool, using the same syntax as the other command-line tools in AVR Studio. For more information on how the user interface works, please refer to the Atmel AVR911 application note; AVR911: AVR Open Source Programmer.

The only requirement is to have the avrosp.exe file available. The communication port settings (baud rate, parity, etc.) must be set manually before using the AVROSP. For example, to set COM1 for 115200bps, no parity, and eight data bits, run the following DOS command:

```
mode com1 baud=115200 parity=n data=8
```

The supported AVROSP commands are as described in Table 5-1.

Table 5-1. Supported AVROSP commands and actions.

AVROSP command	Action	Return
'T'	Do a dummy read for the AVROSP support	'\r'
'P'	Enable TPI	'\r'
'L'	Disable TPI	'\r'
'a'	Auto increment supported	'\r'
'A'	Get NVM word address	'\r'
'c'	Get low byte of data to be written	'\r'
'C'	Get high byte of data to be written, and start the NVM write	'\r'
'm'	Finished writing one page, for AVR911 support	'\r'
'R'	Read one word from the NVM, and return data to the AVROSP	Data high byte and data low byte
'e'	Do chip erase	'\r'
'.'	Read four bytes of universal command	'\r'
'N'	Read high fuse byte, do a dummy write support for the AVROSP	Dummy byte (0xFF)
'S'	Programmer identification	Send "AVR ISP"
's'	Read device ID	Return three bytes of device ID, high byte first

5.1 Universal command

The universal command sent by the AVROSP starts with a period (.), followed by four bytes of command. See the details in Table 5-2.

Table 5-2. Supported four-byte commands and actions.

Four byte command	Action
0x38 0x00 0xXX 0X00	Read OSCCAL
0xac 0xe0 0x00 data	Write lock byte
0x58 0x00 0x00 0x00	Read lock byte

Four byte command	Action
0xac 0xa0 0x00 data	Write fuse low
0xac 0xa8 0x00 data	Write fuse high
0x50 0x00 0x00 0x00	Read fuse low
0x58 0x08 0x00 0x00	Read fuse high

5.2 Example: Command line syntax for the Atmel ATtiny10

Table 5-3. Command line syntax example for ATtiny10.

Operation	Command line syntax
Read signature	avrosp.exe -dattiny10 -s
Read fuse byte	avrosp.exe -dattiny10 -q
Read lock byte	avrosp.exe -dattiny10 -y
Write fuse byte	avrosp.exe -dattiny10 –f<fuse bytes>
Write lock byte	avrosp.exe -dattiny10 –l<lockbyte>
Chip erase	avrosp.exe -dattiny10 -e
Read OSCCAL	avrosp.exe -dattiny10 –O
Read flash to <filename.hex>	avrosp.exe -dattiny10 -rf –of<filename.hex>
Program flash from <filename.hex>	avrosp.exe -dattiny10 -g -e -pf –if<filename.hex>

6 Quick start guide

The following is intended as a step-by-step guide on how to get started:

- Follow the hardware settings as described in Chapter 4, External programmer implementation, on page 6
- Download and unzip the source code for the Atmel AVR918
- Open \avr918_TPI_programming\trunk\Source\avr918\AVR918.APS
- ADD project configuration --> custom options --> custom compilation options -->[All Files]--> -DATTINY10 *
- Save all, clean, rebuild
- Program the Atmel ATmega324P
- Run command prompt, go to the avr918 directory
- Type any of the command line syntaxes mentioned in Section 5.2, Example: Command line syntax for the Atmel ATtiny10, on page 12

7 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

8373A-AVR-02/11