Atmel AVR1624: Using ATxmega128A1 Xplain Kit as USB-to-TWI Bridge

Features

- **Open source C++ code**
- **Modular design**
- **Device independent**
- **Supports the boot loader in Atmel® AVR1622**

1 Introduction

This application note describes how to use the Atmel ATxmega128A1 Xplain kit as a USB-to-TWI bridge. This application can be used to communicate with various applications which have TWI Slave functionality. In this application note we use a TWI slave running a TWI boot loader application on ATxmega128A1. As there is no hardware in a PC for sending data directly to a TWI interface, the Atmel AVR® Xplain kit is used as a USB-to-TWI bridge.

The application provides a command line utility, TWIGEN, for sending commands to the device from the PC. The open source code and its modular design application make it easy to port. This utility is not dependent on any parameters of the device other than its page size of flash memory.

This utility is capable of programming hex files directly to the device with TWI boot loader using the Xplain kit as a USB-to-TWI bridge. Apart from programming hex files, the utility can communicate with any TWI slave and is capable of writing and reading bytes from the device. Therefore it can also be used as a master for testing the functionality in TWI slaves being developed, and it can be used to generate a sequence of data to be sent to the device over TWI interface.

8-bit Atmel Microcontrollers

Application Note

Rev. 8438A-AVR-09/11

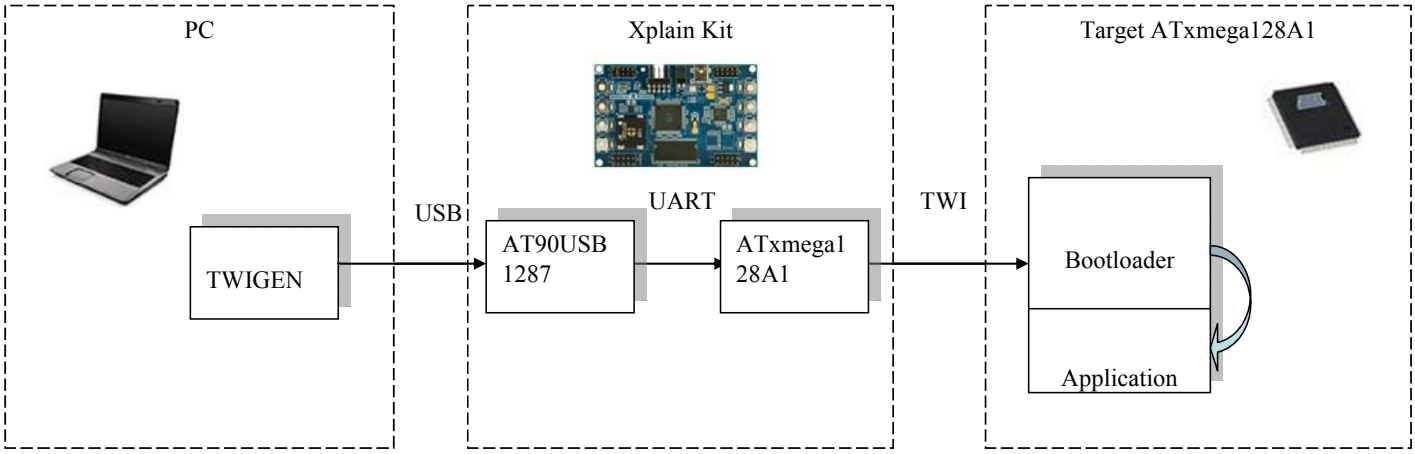
2 Getting up and running Xplain Bridge

This chapter walks you through the basic steps for getting the Xplain Bridge up and running. The necessary setup and requirements are described along with relevant information.

2.1 Theory of operation

This section explains how the Atmel AVR Xplain kit is capable of acting as a bridge between a PC and a device with TWI boot loader.

Figure 2-1. Block diagram of the operation.



The data from TWIGEN running on the PC is sent through the USB interface to the Xplain kit. The Atmel AT90USB1287 device on the Xplain kit receives this data over USB interface and sends it over UART interface. The Atmel ATxmega128A1 on the Xplain kit receives this data over UART and passes it over TWI interface to the device running a TWI boot loader.

2.2 USB to UART

The AT90USB1287 device on the Xplain kit is used for passing data from USB interface to data over UART interface. This is implemented using Communication Device Class (CDC). The CDC class is a standard USB class used for the communication devices (modem, Ethernet...) and is also used to emulate a virtual COM port.

The following explains the procedure to be followed for programming the AT90USB1287 device on the Xplain kit and enumerating it:

1. Connect the Atmel AVR JTAGICE mkII header to ‘JTAG USB’ header on the Xplain kit.
2. Start the Atmel AVR Studio® 5.
3. Select Tools -> AVR Programming.
4. Select the tool and interface used.
5. Set the device as AT90USB1287 and ‘Apply’ the settings.
6. Select ‘Memories’ tab and under ‘Flash’ section, browse to the folder containing ‘`Xplain_USB.a90`’.
7. Program the flash.

8. Unplug and plug the Xplain kit again.
9. A popup occurs showing “Xplain USB Gateway” and “Xplain Serial Port” and prompts to install software in ‘Found New Hardware Wizard’.
10. Windows® does not provide a native driver for the virtual COM port, so we need to provide an INF file to load the correct driver. Select ‘Install from a list or specific location (Advanced)’ and browse to the folder containing the ‘`at90usbxxx_cdc.inf`’ file.
11. Now it will be listed as ‘Xplain Serial Port’ under Ports (COM & LPT) in Device Manager with a specific COM number assigned to it.

NOTE The fuse settings used were EXTENDED: 0xF3, HIGH: 0x99 and LOW: 0x5E.

2.3 UART to TWI

The Atmel ATxmega128A1 device on the Atmel AVR Xplain kit is responsible for getting data through the UART interface and passing it over the TWI interface. In this case it uses a TWID module.

Explained below is the procedure to be followed for programming the ATxmega128A1 device on the Xplain kit:

1. Connect the Atmel AVR JTAGICE mkII header to ‘JTAG & PDI XMEGA’ header on the Xplain kit.
2. Follow the steps 2 to 4 given in the previous procedure.
3. Select the device as ATxmega128A1 and apply the settings.
4. Select ‘Memories’ tab and under ‘Flash’ section, browse to the folder containing the ‘`XplainSerialToI2CBootLoaderBridge.hex`’ file.
5. Program the flash.

NOTE The fuse settings used were FUSEBYTE0: 0xFF, FUSEBYTE1: 0x33, FUSEBYTE2: 0XF9, FUSEBYTE4: 0XFE and FUSEBYTE5: 0xFF.

3 Getting up and running the target ATxmega128A1 device

This chapter explains the setup procedure of the target Atmel ATxmega128A1 device, which has to update its application section by running a TWI boot loader. Please refer to Atmel AVR1622: TWI Boot Loader for XMEGA® which explains the procedure to do the same.

4 Getting up and running AVR TWIGEN

This chapter briefs the basic steps for getting up and running the AVR TWIGEN PC utility. Please refer to Atmel AVR1622 for demonstration of running the AVR TWIGEN.

4.1 Quick start information

This section describes the necessary steps to start using TWIGEN if you have no need to modify the application code.

The executable file `twigen.exe` is the only file required to use TWIGEN. It is located in the 'Windows side' folder in `avr1624.zip` file that comes with this application note. The zip-file also contains the complete source code and the hex files to be programmed to the Atmel AVR Xplain kit. Copy the executable file to a new directory and add the directory name to the PATH environment variable.

4.2 Command-line syntax

All parameters must start with a minus, one or more characters, and a number of optional values. Except for the input file name (-i), there should be one space between the parameter (-r, -w, etc.) and the characters of optional values (<numBytes>, <byte0>, etc.). The supported command-line parameters are listed in Table 4-1.

Table 4-1. Command-line parameters.

Parameter	Description
-e	Erases device
-i<infile>	Program flash with specified hex input file. The file format is Intel Extended HEX
-r <numBytes>	Read the number of bytes specified
-w <numBytes> <byte0> <byte1>...	Write the number of bytes specified. The data is defined by byte0, byte1, ...
-a <address size in bytes> <add byte0> <add byte1> ...	Use the sub address size specified. The address is defined by address byte0, address byte1, and so on
-x	Execute the queued commands
-v	Verify the specified operation after performing the operation
-c	Verify the specified operation without performing the operation
-s <Slave Address>	Specifies the TWI slave address
-p <COM port>	Specifies the COM port on the PC to use to communicate to the Xplain board
-q	Queues up the specified command. The queued commands can be executed by executing the –x command
-d <delay>	Specifies the delay in milliseconds for the queued commands. This delay takes place prior to executing the command
-n	Specifies to not retry the transaction if it is not acknowledged by the slave. This option is only valid for a queued command. Each command will retry by default if this option is not specified

4.3 Command-line examples

This section explains the working of AVR TWIGEN with some examples.

```
twigen -e -iLEDCHASER.hex -a 3 0x00 0x00 0x00 -s 0x55 -p 14
```

The above example first erases the device and then programs the specified hex file at triple byte sub address to slave 0x55 and COM port 14.

```
twigen -e -a 3 0x00 0x00 0x00 -s 0x55 -p 14
```

The above example will erase the flash memory of slave 0x55 and COM port 14. The address parameter should be passed essentially on all commands.

```
twigen -r 0x20 -a 1 0x00 -s 0x5E -p 26
```

The above example reads 0x20 bytes from slave 0x5E using a single byte sub address of 0x00 and COM port 26.

```
twigen -r 0x30 -a 2 0x00 0x00 -s 0x52 -p 26
```

The above example reads 0x30 bytes from slave 0x52 using a double byte sub address of 0x0000 and COM port 26.

```
twigen -w 1 0x11 -a 1 0x07 -s 0x5E -p 26
```

The above example writes one byte of value 0x11 to slave 0x5E using a single byte sub address of 0x07 and COM port 26.

```
twigen -w 2 0x55 0xAA -a 2 0x00 0x10 -s 0x52 -p 26
```

The above example writes two bytes of value 0x55AA to slave 0x52 using a double byte sub address of 0x0010 and COM port 26.

4.4 Rules

This section explains the rules to be followed when using AVR TWIGEN.

- Only one operation can be specified at once
- -v and –c cannot be specified for the same operation
- -v and –c cannot be used with a zero byte write or any size read
- -n can only be specified for a queued command

5 Implementation

This chapter assumes that the reader has some knowledge of object-oriented programming concepts, and the C++ programming language in particular.

The top-level work is encapsulated in the `twigen` class. It uses objects of `HEXFile` to read and write hex files. The two helper classes `Utility` and `ErrorMsg` are used throughout the application.

The `twigen` decodes the command line and processes it accordingly. It takes care of the main job and the COM port handling is taken care by the comport file.

5.1 Class description

This section describes each class with a brief introduction to the class' purpose and then each of its public methods with return type, parameters and purpose.

5.1.1 HEXFile

This is a class providing basic functionality of read and write of Intel® extended hex files. It also has methods for defining the memory range to be used. This is useful for reading or writing only parts of the AVR memories.

5.1.1.1 HEXFile

This is the constructor for the class. It takes two parameters, a `long` indicating the required maximum data buffer size and a `long` containing the default byte value to be used when initializing the buffer. An exception is thrown if not enough memory is available.

5.1.1.2 ~HEXFile

This is the destructor for the class. It de-allocates all previously allocated memory.

5.1.1.3 readFile

This method reads data from a hex file. This method takes one parameter, the hex file name, and returns no value. An exception is thrown if any file access errors occur, or the file format is invalid.

5.1.1.4 writeFile

This method writes data to a hex file. The method takes one parameter, the hex file name and returns no value. An exception is thrown if any file access errors occur.

5.1.1.5 setUsedRange

This method overrides the memory range indicators. This can be used to limit the range for read and write operations. The method takes two parameters, two `long` variables containing the new start and end limits respectively. The method returns no values. An exception is thrown if the provided range is invalid.

5.1.1.6 clearAll

This method sets the entire data buffer to the desired byte value. The method takes one parameter, a `long` containing the desired byte value, and returns no value.

5.1.1.7 getRangeStart

This is an access method for the start address of the current range. The method takes no parameters, and returns the start address.

5.1.1.8 getRangeEnd

This is an access method for the end address of the current range. The method takes no parameters, and returns the end address.

5.1.1.9 getData

This is an access method for the data in the buffer. It takes one parameter, a `long` containing the byte address, and returns a `long` containing the byte value. An exception is thrown if the address is outside legal ranges.

5.1.1.10 setData

This is an access method for setting the data in the buffer. It takes two parameters, a `long` containing the byte address and a `long` containing the byte value. An exception is thrown if the address is outside legal ranges.

5.1.1.11 getSize

This is an access method for retrieving the buffer size. The method takes no parameters and returns a `long` containing the buffer size in bytes.

5.1.2 twigen

This is a class holding all information extracted for the command line parameters. The class also contains the functionality for performing the necessary operations.

5.1.2.1 twigen

This is the constructor for the class. It erases the string holding the input file name to be programmed to flash.

5.1.2.2 parseArguments

This method parses the command line parameters. It takes two parameters, the familiar `int argc` and `char *argv[]` from the `main()` function. The method returns an `int` containing the result of the method. It returns 0 when successful and -1 otherwise.

5.1.2.3 checkArguments

This method checks whether the parameters comply with the rules stated above in the ‘Rules’ section. It takes no parameters and returns an `int` containing the result of the method. It returns 0 if successful and -1 otherwise.

5.1.2.4 displayBytes

This method displays the requested number of bytes. It takes two parameters, a `short` containing the byte count and a `char *` containing the buffer address to which the data is to be stored. It returns no parameters.

5.1.2.5 getAck

This method gets the acknowledge byte. It takes no parameters and returns the acknowledgement status. It returns 0 if there was acknowledgement and -1 otherwise.

5.1.2.6 Ack

This method sends acknowledgement byte. It takes no parameters. This method returns 0 if acknowledgement was sent successfully and -1 otherwise.

5.1.2.7 executeQueue

This method executes the queued commands. It takes no parameters and returns an `int` containing the status of the method. It returns 0 if the queued operations were executed successfully and -1 otherwise.

5.1.2.8 Write

This method writes specified data to the specified slave at the specified address. It takes four parameters, a `char` containing the slave address, a `char *` containing the address to which data is to be written, a `short` containing the byte count and `char *` containing the buffer address from which data is to be written. It returns 0 if it was written successfully and -1 otherwise.

5.1.2.9 Read

This method reads specific number of bytes from the specified slave at the specified address. It takes four parameters, a `char` containing the slave address, a `char *` containing the address from which the data is to be read, a `short` containing the byte count and a `char *` containing the buffer address to which data is to be stored. It returns 0 if it was read successfully and -1 otherwise.

5.1.2.10 eraseFlash

This method issues commands to erase the flash memory of the device. It takes no parameters. It returns 0 if the commands were issued successfully and -1 otherwise.

5.1.3 Utility

This class serves as a container and namespace for often used functions. It is instantiated in the source file and an external reference to an Util object is provided in the header file. It is especially used for log and progress messages and for enabling silent operation.

5.1.3.1 Utility

This is the constructor for the class. It takes no parameters. The constructor initializes the internal log and progress status to enable both log and progress messages.

5.1.3.2 ~Utility

This is the destructor for the class. It currently has no function, just a placeholder for future extensions.

5.1.3.3 muteLog

This method prevents all further log messages from being displayed on the screen. It takes no parameters and returns no value.

5.1.3.4 muteProgress

This method prevents all further progress messages from being displayed on the screen. It takes no parameters and returns no value.

5.1.3.5 log

This method prints log-type messages to the screen, if not muted. The method takes one parameter, the message string. The method returns no value.

5.1.3.6 progress

This method prints progress-type messages to the screen, if not muted. The method takes one parameter, the message string. The method returns no value.

5.1.3.7 convertHex

This method converts a hexadecimal string to a number. The method takes one parameter, the string, and returns a `long` containing the converted number. An exception is thrown if any conversion errors occur.

5.1.3.8 convertLong

This method converts a number to a string, using a specified radix. It takes two parameters, a `long` containing the number to be converted and a `long` containing the desired radix to be used. This method converts a number to a string, using a specified radix.

5.1.3.9 getRegistryValue

This method retrieves a value from the Windows registry database. The method takes two parameters, a string containing the registry key path and a string containing the key name. The method returns a string containing the retrieved value. An exception is thrown if any errors occur during the database operations.

5.1.4 ErrorMsg

This class serves as a container for error messages to be thrown as exceptions.

5.1.4.1 ErrorMsg

This is the constructor for the class. It takes one parameter, the error message string.

5.1.4.2 ~ErrorMsg

This is the destructor for the class. It currently has no function, just a placeholder for future extensions.

5.1.4.3 What

This is an access method for the error message string. It takes no parameters and returns a copy of the error message.

5.1.5 Commport

This file does not have a class but has modular functions for communicating through the COM port.

5.1.5.1 CommPortOpen

This function opens the COM port with the specified name and gets the handle. It takes two parameters, a handle to the COM port and a string containing the COM port name. It returns Boolean value true if the function was successful and false otherwise.

5.1.5.2 CommPortInit

This function initializes the COM port with the specified handle at the specified baud rate. It takes three parameters, a handle to the COM port, a `long` containing the baud rate and a `char` containing the time to wait in seconds. It returns true if the function was successful and false otherwise.

5.1.5.3 CommPortClose

This function closes the COM port with the specified handle. It takes a parameter, a handle to the COM port. This function returns true if it was successful and false otherwise.

5.1.5.4 CommPortTx

This function transmits the specified data out the specified COM port. It takes three parameters, a handle to the COM port, buffer containing the data to be transmitted and a `long` containing the number of bytes. This function returns true if it was successful and false otherwise.

5.1.5.5 CommPortRx

This function receives the specified amount of data from the specified COM port into the specified buffer. It takes three parameters, a handle to the COM port, a buffer to which the received data will be stored, and a `long` containing the number of bytes. This function returns true if it was successful and false otherwise.

5.1.5.6 CommPortGetLastError

This function takes no parameters and returns a double word containing the last error.

5.1.5.7 CommPortFlushRxBuffer

This function flushes the buffer that stores received data. It takes a parameter, a handle to the COM port. This function returns true if it was successful and false otherwise.

5.1.5.8 CommPortRxReady

This function checks whether the COM port is ready to receive data. It takes two parameters, a handle to the COM port and a `long` containing the number of bytes. It returns true if it is ready to receive data and false otherwise.

6 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

8438A-AVR-09/11