Programmers and Debuggers

EDBG

USER GUIDE

Description

The Atmel® Embedded Debugger (EDBG) is an onboard debugger for integration into development kits with Atmel MCUs. In addition to programming and debugging support through Atmel Studio, the EDBG offers data streaming capabilities between the host computer and the target MCU.

Table of Contents

Description....1

1. Overview....3
   1.1. Features....3
2. Programming and Debugging....4
3. Virtual COM Port....5
4. Data Gateway Interface....6
   4.1. SPI Interface....6
   4.2. USART Interface....6
   4.3. I^2C Interface....6
      4.3.1. Information Interface....7
   4.4. GPIO Interface....8
   4.5. Timestamp Module....8
5. Technical Overview....9
   5.1. Pin Usage....9
   5.2. Power Consumption....9
   5.3. LED Control....9
6. Document Revision History....10
7. Firmware Release History....11

1. Overview

The Atmel Embedded Debugger (EDBG) is an onboard debugger for kits with Atmel devices.

EDBG enables the user to debug the target device without an external debugger. EDBG also brings additional features with a Data Gateway Interface and a Virtual COM Port for streaming of data to a host PC. The Atmel EDBG will enumerate as a composite USB device with separate interfaces for each function.

The Atmel EDBG is embedded on all Xplained Pro evaluation kits. All functionality of the Atmel EDBG is not necessarily available on all kits - the EDBG is factory configured depending on the specific kit capabilities. The configuration is read by Atmel Studio to present the correct capabilities and simplify the user interface. Supported extension boards connected to the kit will be detected by the EDBG and their features are reported to Atmel Studio.

1.1. Features

- On-board programming and debugging through SWD, JTAG, and PDI
- Virtual COM Port interface to target via UART
- Data Gateway Interface (DGI) for data streaming between target MCU and PC
  - SPI, USART, and I^2C interfaces available
  - GPIOs for accurate status indication
- Extension board identification
- Indication of power and status through LEDs
- Sleep mode to minimize power consumption

2. Programming and Debugging

The Atmel EDBG has the ability to program and debug Atmel AVR® and Atmel ARM® Cortex®-M core based microcontrollers. The following interfaces are supported:

- Atmel ARM Cortex-M programming and debug interfaces
  - Serial Wire Debug (SWD)
- Atmel megaAVR® programming and debug interfaces
  - JTAG
- Atmel AVR XMEGA® programming and debug interfaces
  - Program and Debug Interface (PDI)

Refer to the specific kit's user guide for details on connecting the interface.

3. Virtual COM Port

The EDBG features a CDC class USB interface that implements a Virtual COM Port. A UART connected to a target device is used to enable easy communication between a computer and the target.

The following configuration options are implemented

- Baud Rate: Flexible and accurate settings up to 2Mbps
- Parity: None, Even, Odd, Mark, Space
- Stop Bits: 1 bit, 1.5 bits, 2 bits

The configuration options must be specified in the terminal application, which will propagate the configuration to the EDBG Virtual COM Port on connection. The target MCU UART must be configured to match the Virtual COM Port.

Note that the UART pins of the EDBG are tri-stated when no terminal program is connected to the Virtual COM Port on the computer. This mechanism relies on the terminal program sending a DTR signal.

The Virtual COM Port is supported by the terminal extension in Atmel Studio. Most other terminal applications will work as well.

4. Data Gateway Interface

The Atmel EDBG features an interface for streaming data from the target device to a computer, called the Data Gateway Interface (DGI). This is meant as an aid in debugging and demonstration of features in the application running on the target device.

DGI consists of multiple channels for data streaming. The available channels are listed in the sections below. Note that not all interfaces need to be implemented on all kits, and that different kits can implement a different subset of these interfaces. Refer to the specific kit's user guide for details.

4.1. SPI Interface

The Serial Peripheral Interface (SPI) is connected through four digital signals; MOSI, MISO, SCK, and CS. The SPI is set to operate in slave mode, meaning that the target device must be set to master mode. The active low CS (Chip Select) line indicates to the SPI that it should expect data to be received and/or sent. If the master expects to receive data from the slave, it must poll for them by initiating a transfer. All pins are tri-stated until the interface is activated from the PC and the CS line is driven low.

It is possible to configure the mode (clock phase and data setup) of the SPI module. Valid settings are 0-3. The bit count for each transfer can also be set between 5 and 8 bits per transfer.

In normal operation, DMA will automatically buffer incoming data transfers. It is also possible to enable timestamping to get a more accurate timing of incoming data. Note that the timestamping will add an overhead to each data transfer, and a lower maximal throughput and a longer required inter-byte delay is expected. For sending data to the target device DMA is always used.

4.2. USART Interface

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) interface is connected through three digital signals; RX, TX, and XCK. All pins are tri-stated until the interface is activated from the PC.

Both synchronous and asynchronous modes are supported. If operated in asynchronous mode, the correct baud rate setting must be supplied. The baud rate is flexible and is accurate up to 2Mbps. Supported parity settings are none, even, odd, mark, and space. Stop bit can be set to 1, 1.5, and 2 stop bits. It is possible to use a transfer size of 5-8 bits. If used in synchronous mode, a clock signal must be supplied by the target device.

In normal operation, DMA will automatically buffer incoming data transfers. It is also possible to enable timestamping to get a more accurate timing of incoming data. Note that the timestamping will add an overhead to each data transfer, and a lower maximal throughput and a longer required inter-byte delay is expected. For sending data to the target device DMA is always used.

4.3. I^2C Interface

The Two-Wire Interface (I^2C) is connected to through two signals; SDA and SCL. The two-wire interface is set to slave mode, meaning that communication must be initiated by a target device in master mode. The interface must be enabled from the PC before communication can begin.

The slave address of the I^2C interface can be configured, but is default set to 0x28.

Communication from the target device to I^2C DGI is done by sending the slave address with the write bit, followed by the data bytes. The master must poll the DGI for data by sending the slave address with the read bit. Then the DGI will send a 1 byte length, directly followed by the data.

Important: An I²C write transaction (could be of length zero) must take place before an I²C read transaction is acknowledged by the EDBG.

4.3.1. Information Interface

The TWI-bus can be used to request information from the EDBG. This is done by sending a special sequence as described below.

- START Condition
- Address + W
- Request token
- START Condition (repeated start)
- Address + R
- Response
- ...
- STOP Condition

The sequence starts by sending a normal start condition followed by an address byte (default address is 0x28) with the RW bit cleared. Then a request token identifying the requested information is sent. Normally, a received byte would be put into the DGI buffer, but a repeated start will trigger the EDBG to parse the received token. Then an address with the RW bit set. The EDBG will then start to push the requested data onto the TWI-bus. All response bytes must be ACKed by the master, and the final byte must be NAKed.

4.3.1.1. Extension Boards

Token: **0xE1**

The EDBG has 10 pins designated for extension board identification. These pins are sampled on power-up and the extension board identification information is stored. This information can be retrieved by using the *Extension Boards* token.

The first part of the response from this request is a 2-byte BE (1st byte is MSB) extension map containing information of which extension slots are populated. A '1' in bit 0 of the extension map means that EXT1 is populated, a '0' means not populated. If there are extension modules present the 64-byte content of each of the ID chips is sent. It will start with EXT1. If the extension is present it will send the content, otherwise no data is sent.

Example: Extension boards are available in EXT2 and EXT5. The response will be:

- 0x00 (Extension map MSB)
- 0x12 (Extension map MSB)
- 0x41 'A' (First byte in EXT2 ID content)
- 0x74 't' (Second byte)
- 0x6D 'm' (Third byte)
- 0x65 'e' (Fourth byte)
- 0x6C 'l' (Fifth byte)
- 0x53 'S' (Sixth byte)
- ... Remaining 58 bytes
- 0x41 'A' (First byte in EXT5 ID content)
- 0x74 't' (Second byte)

- 0x6D 'm' (Third byte)
- ... Remaining 61 bytes
- 0x00 (End of data)

4.3.1.2. Kit Data

Token: **0xD2**

The EDBG has 256 bytes for storing kit-specific data such as MAC address, calibration values, etc. See the kit documentation for details on how the data is organized. The information can be retrieved by using the *Kit Data* token. Note that not all kits have kit-specific data, and the entire section will read as 0.

After sending the token, the stored kit-specific data will be sent byte-by-byte, starting with location 0, until a stop condition is detected.

4.4. GPIO Interface

Up to four General Purpose Input/Output (GPIO) signals are connected to DGI. In input mode, they can be used as a status indication from the target device. In output mode, they can be used as virtual control signals to simulate buttons.

The GPIO interface is always timestamped. It is only useful for signals with a relatively slow toggle rate (<250kHz).

4.5. Timestamp Module

Data received on DGI Interfaces can be routed through the timestamp module to embed timing references into the data stream. The timestamp is implemented as a counter, which increments in steps of (32/60) µs, giving a timestamp accuracy of about half a microsecond.

The timestamp module does not use DMA since it requires CPU intervention on each data unit received. This implies a lower maximum data throughput and demands a larger delay between each data unit.

5. Technical Overview

5.1. Pin Usage

All pins that are not in use on the EDBG, will be tri-stated. This is done to avoid signal contention. Note that the tri-stating of serial communication lines and external interference could lead to unintended data to be received by the target MCU.

5.2. Power Consumption

During enumeration, the EDBG reports a current consumption of 500mA, and must therefore be connected to a USB host that is capable of supplying this. It is impossible to predict the amount of current required by the setup, since the EDBG is a part of a flexible evaluation kit.

The EDBG alone consumes approximately 100mA during usage. When the EDBG does not detect a VBUS voltage, but is otherwise powered (for example through the Xplained Pro PWR header), the EDBG will go into sleep to minimize power consumption. The power consumption during sleep mode will vary with the design, but should be well below 1mA.

5.3. LED Control

The EDBG controls two LEDs; the power LED and the status LED. The power LED is on by default when the kit is powered, but can be disabled by the EDBG to lower the power consumption of the kit. The status LED is turned on when a host computer opens a connection to the USB interface. During communication activity the status LED will flash.

If the EDBG enters bootloader mode, both the status LED and the power LED will flash simultaneously. When a firmware upgrade is in progress the LEDs will blink alternately.

The behavior of the EDBG LEDs is summarized in the table below.

Table 5-1. EDBG LED Control

Operation mode	Power LED	Status LED
Normal operation	Power LED is lit when power is applied to the board.	Activity indicator, LED flashes every time something happens on the EDBG.
Bootloader mode (idle)	The power LED and the status LED blinks simultaneously.	
Bootloader mode (firmware upgrade)	The power LED and the status LED blinks in an alternating pattern.	

6. Document Revision History

Doc. Rev.	Date	Comments
42096C	10/2016	Updated *Firmware Release History*
42096B	02/2014	Added details about TWI info interface
42096A	02/2013	Initial document release

7. Firmware Release History

Table 7-1. Public Firmware Revisions

Firmware version (decimal)	Date	Relevant changes
3.31	29.09.2016	Added support for UPDI interface (tinyX devices) Made USB endpoint size configurable General bug fixes
3.12	24.02.2016	Added drag-and-drop programming (selected kits)
2.17	11.11.2015	CDC buffer fix
2.16	22.09.2015	Data polling improvements SWD speed improvement
2.09	11.05.2015	Improved power measurement Improved unlocking of SAM devices
2.01	27.01.2015	Improved DGI buffering
1.33	17.12.2015	Improved unlocking of SAM L21 Block illegal characters in USB serial number
1.27	06.04.2014	Variant without DGI
1.24	21.01.2014	Added SWO trace support
1.15	17.04.2013	Initial release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Microchip:
ATSAM4L8-XSTK