

Report No:

AN127

Title:

In-System Programming (ISP) of the Atmel XMEGA AVR FLASH Microcontroller Family

Author:

John Marriott

Date:

06th January 2013

Version Number:

1.14



All rights are reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without prior notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights

Contents

1.0 Introduction	4
1.1 Overview of programmers supporting XMEGA PDI devices	6
1.1.1 Portable ISP Programmers	6
1.1.2 Single channel Production ISP Programmers	6
1.1.3 Multiplexed sequential Production ISP Programmers	7
1.1.4 Multi-channel GANG Production ISP Programming Systems	8
1.2 XMEGA PDI – programmer cross reference guide	9
1.3 XMEGA Device Support	11
2.0 XMEGA Programming Interfaces	12
2.1 Overview of XMEGA Programming Interfaces	12
2.2 Comparison of XMEGA PDI and JTAG algorithms	13
2.3 Overview of PDI Interface	16
2.4 PDI – Physical Interface	16
2.5 Overview of JTAG Interface	17
3.0 PDI Algorithm	19
3.1 Overview of PDI Interface	19
3.2 PDI – Physical Interface	19
3.3 PDI – Clock Signal (XMEGA RESET pin)	20
3.4 PDI – Data Signal	21
3.4 PDI – Data Signal	21
3.5 ISPnano Series 3 - Target ISP Port – PDI pin-out	22
3.6 Single XMEGA device – PDI programming connections	23
3.7 Single XMEGA PDI + AVR SPI – programming connections	25
3.8 XMEGA PDI – ISP cable length recommendations	27
3.9 Signal / Power GROUND (0V) connections	27
3.10 Enabling PDI programming mode	28
4.0 Creating an EDS Development Project	29
4.1 Overview	29
4.2 Information required to create an XMEGA PDI Project	29
4.3 Launching the EDS Wizard	30
4.4 Selecting the attached programmer	31
4.5 Selecting the target XMEGA device	31
4.6 Selecting the XMEGA oscillator settings	32
4.7 Setting up the Target Power Supply	33
4.8 Setting up the XMEGA ‘Erase Options’	33
4.9 Setting up the XMEGA ‘FLASH Programming’	34
4.10 Setting up the XMEGA ‘EEPROM Programming’	35
4.11 Saving the EDS setup file	36
4.12 Testing an EDS programming Project	36
4.13 Checking the Device Signature (ID) of a target device	37
4.14 Cannot Enter Programming Mode error	37
Appendix 1 – CONMOD Module + XMEGA PDI	39
1.0 Overview	39
Appendix 2 – ISPnano-QC1 Quick Connect Module	40
1.0 Overview	40
1.1 Quick-Connect connector pin-out	41
Appendix 3 – Using ConsoleEDS to program XMEGA PDI devices	43
1.0 Overview	43
1.1 Explanation of ConsoleEDS ‘Base Projects’	43
1.2 Setting up a ConsoleEDS ‘Base Project’	43
1.3 Reading the ‘Device Signature / ID’	44
1.4 Erasing the FLASH and EEPROM	44

1.5 Programming the FLASH using the /FLASHWRITE command	45
1.6 Programming the 'User Signature Row'	45
1.7 Programming the Fuses using the /FUSEWRITE command	46
1.8 Reading the Fuses using the /FUSEREAD command	46
1.9 Programming the 'Security Fuses' using the /SECURITYWRITE command	47
1.10 Reading the 'Security Fuses' using the /SECURITYREAD command	47
1.11 Executing a 'Standalone Programming Project'	47
1.12 Mixing a 'Standalone project' with individual programming commands	48
1.13 Typical XMEGA programming sequence	49
Appendix 4 – XMEGA Fuse Verify problems.....	50
1.0 Problem description	50
1.1 Error numbers / message for XMEGA Fuse Verify problem	50
1.2 Fix for this problem	51
Appendix 5 – PDI Clock Buffer module	52
1.0 Overview	52
1.2 XMEGA PDI Buffered ISP Cable	52
1.3 XMEGA PDI Buffered ISP Cable - Features	53
1.4 Recommended Clock Buffer module location	53
• The <i>PDI_CLOCK</i> and <i>PDI_DATA</i> wiring between the 'Clock Buffer module' and fixture probe- pins should be a maximum of e.g. 3 – 4 cm.....	53
1.5 Clock Buffer functional explanation.....	54
1.6 Clock Buffer Module – 10-way IDC pin-out	55
1.7 Clock Buffer Module – Outputs to the DUT	56
1.8 Controlling the PDI Clock Buffered output.....	57
Appendix 6 – XMEGA PDI Design guidelines	58
1.0 Overview	58
1.1 XMEGA RESET circuit	58
1.2 XMEGA PDI - CLOCK signal line (RESET)	58
1.3 XMEGA PDI - DATA signal line	58
1.4 PDI programming cable length	59

1.0 Introduction

This application note describes how to develop and implement **In-System Programming (ISP)** support for the **Atmel ATxmega AVR microcontroller** family the '**2-wire PDI**' programming interface. The document details how to create a '**Programming Project**' which will operate on any Equinox ISP programmer. The document describes the physical connections required from the programmer to the target XMEGA microcontroller and also details the different ISP Header Connector pin-outs which are currently available.

The Equinox programming range includes solutions for development, low / mid / high volume production and field programming of Atmel XMEGA AVR microcontrollers.

General features.....

- High-speed In-System Programming (ISP) support of **Atmel XMEGA AVR** microcontrollers
- Programming solutions for development, low / mid / high volume production and field programming applications
- Supports high-speed programming of the XMEGA on-chip FLASH / EEPROM memory areas and Configuration Fuses / Security Fuses
- Uses Atmel proprietary '**2-wire PDI**' serial programming interface
- Very high-speed programming due to local data storage, optimised programming algorithms and high-speed PDI driver circuitry
- Dedicated high-speed PDI driver circuitry with full ESD and over-voltage protection
- Programmers can be used in "**Standalone Mode**" (no PC required)
- Supports high-speed program / verify of the on-chip FLASH, EEPROM, Configuration Fuses and Security Fuses in a single operation.
- Support for writing / reading the XMEGA '**User Signature Row**' memory area
- Support for programming the XMEGA '**User Signature Row**' memory area in Standalone Mode (new)
- Support for reading the XMEGA '**Production Signature Row**' memory area
- Optimised Erase operations – supports individual control over erasing of the Application Sector, Application Table, User Signature Row and EEPROM areas.
- Supports programming of non volatile '**Fuse Bits**'
- Supports programming of the '**Security Fuses**' to protect code from being read out
- Supports importing of '**Atmel ELF File**' which contains XMEGA FLASH / EEPROM File + Fuse / Security hex values
- ISPnano programmers support programming of both an Atmel XMEGA and a standard AVR microcontroller from a single programmer

In 'Development Mode'.....

- Powerful yet simple-to-use Development Suite called '**EDS**'
- All aspects of programming the XMEGA AVR device can be controlled from **EDS**
- Supports reading back/ display of the '**User Signature Row**' from a target XMEGA device

In 'Production Mode'.....

- Programmers can be used in "**Standalone Mode**" (no PC required)
- Programmers can be controlled via TTL wire control, ASCII protocol, ConsoleEDS console application and ISP-PRO production programmer monitoring application
- Scalable production programming solution – up to 32 x ISPnano programmers can be controlled from a single PC COM port
- A single '**Standalone Programming Project**' can Erase the device, program /verify the FLASH / EEPROM, program the Configuration Fuses and finally secure the device in a single operation.
- Up to 64 x independent XMEGA AVR '**Standalone Programming Projects**' can be stored inside an ISPnano or ISPjuno programmer.
- Support for programming the XMEGA '**User Signature Row**' memory area in Standalone Mode (new)
- Supports importing of production settings from an '**Atmel ELF file**'
- Programmer can store multiple versions of firmware for different '**customer product versions**' or '**functional test versions**' of firmware.
- It is possible to program unique '**Serial Numbers**', '**MAC addresses**' and '**Calibration Data**' into either the FLASH or EEPROM area of the target XMEGA device using either our **ConsoleEDS** or **ISP-PRO** control applications.
- Batch statistics - It is possible to log batch programming statistics including a product barcode label ID number to a database.

1.1 Overview of programmers supporting XMEGA PDI devices

Programming support for the **Atmel XMEGA AVR** microcontroller family is currently available on the '**ISPnano Series 3**' and '**ISPnano Series 4**' range of production ISP programmers.

1.1.1 Portable ISP Programmers

The ISPjuno programmer is portable ISP programmer designed for development, field and small volume production programming applications.



ISPjuno

Portable ISP Programmer supporting standalone operation and capable of storing up to 64 'standalone programming projects'

1.1.2 Single channel Production ISP Programmers

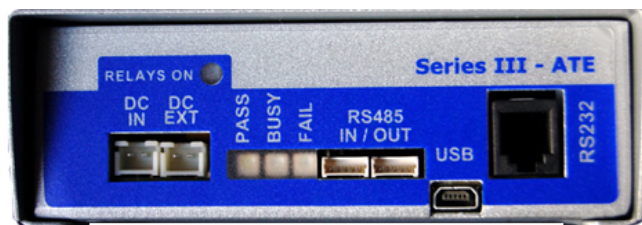
The ISPnano range of programmers are designed for direct integration into programming / test fixtures. Their compact design allows them to be placed directly under the bed-of-nails in a fixture. The range includes versions with galvanic relay isolation which are ideal for use in fixtures where both programming and testing take place on the same fixture.



ISPnano

Series III

Single channel ISP programmer



ISPnano

Series III - ATE

Single channel ISP programmer with relay isolation









ISPnano

Series IV

Single channel ISP programmer with plug-in connector module, relay signal isolation and opto-isolated TTL control port

1.1.3 Multiplexed sequential Production ISP Programmers

The **ISPnano-MUX** range of programmers are specially designed to program PCBs on a multi-PCB panel. A single programmer is multiplexed to 2 or 4 or 8 PCBs (DUT's) allowing up to 8 PCBs to be programmed in a sequential manner.

	 <p>ISP nano mux 2 2 channel multiplexed Production ISP Programmer 2 channel Multiplexed programming system</p>
	 <p>ISP nano mux 4 4 channel multiplexed Production ISP Programmer 4 channel Multiplexed programming system</p>
	 <p>ISP nano mux 8 8 channel multiplexed Production ISP Programmer 8 channel Multiplexed programming system</p>

1.1.4 Multi-channel GANG Production ISP Programming Systems

The **ISPnanoS3-GANG** range of programmers are comprised of 4, 6 or 8 independent '**ISPnano Series 3**' programmers mounted to a bracket. These systems are capable of programming 4, 6 or 8 PCBs (DUTs) in parallel (at the same time) and so are ideally suited to very high volume production environments where all the PCBs on a multi-PCB panel must be programmed concurrently.









ISPnano
Gang Production ISP Programming Systems



**ISPnano
Series IV
Gang Systems
4 / 6 / 8 channel systems**

Multi-channel
parallel programming systems for
programming multiple DUTs
concurrently on a '**PCB Panel**'

1.2 XMEGA PDI – programmer cross reference guide

The table below details the range of programmers available which support programming of the Atmel XMEGA PDI programming.

Programmer picture	Programmer name	PDI algorithm	JTAG algorithm	Programming channels	Comment
 Series III	ISPnano Series 3	Yes – available now	Please contact Equinox	1 - 32	Requires CONMOD or Quick-Connect module
 Series III ATE	ISPnano Series 3 ATE	Yes – available now	Please contact Equinox	1 - 32	Requires CONMOD or Quick-Connect module
 Series IV	ISPnano Series 4	Yes – available now	Please contact Equinox	1 - 32	All XMEGA PDI circuitry is included on plug-in IO-CON module.
 ISP nano MUX2	ISPnano-MUX2	Yes – available now	Please contact Equinox	2 sequential	64 (32 x 2)
 ISP nano MUX4	ISPnano-MUX4	Yes – available now	Please contact Equinox	4 sequential	128 (32 x 4)
 ISP nano MUX8	ISPnano-MUX8	Yes – available now	Please contact Equinox	8 sequential	256 (32 x 8)

 	ISPjuno	Please contact Equinox	Please contact Equinox	1	1
--	---------	------------------------	------------------------	---	---

Please note:

- The **'ISPnano Series 3'** programmer range supports uploading of up to 64 independent **'Standalone Programming Projects'**.
- The **XMEGA 2-wire PDI** programming interface **cannot** be supported on the Equinox FS2003, FS2009, Epsilon5, PPM3-MK2 or PPM4-MK1 programmers as these programmers do not feature the correct hardware to allow PDI to be implemented.

1.3 XMEGA Device Support

The Equinox *'ISPnano Series 3'* programmer is capable of supporting *'2-wire PDI'* programming of the entire **Atmel XMEGA AVR** microcontroller family.

Important note:

- We only support programming of XMEGA AVR devices via the *'2-wire PDI'* programming interface.
- The JTAG programming interface is currently not supported.

The table below lists all the devices in the **XMEGA 'A1'** family.

Ordering Code	Flash (B)	E ²	SRAM	Speed (MHz)	Power Supply
ATxmega384A1-AU	384K + 8K	4 KB	32 KB	32	1.6 - 3.6V
ATxmega256A1-AU	256K + 8K	4 KB	16 KB	32	1.6 - 3.6V
ATxmega192A1-AU	192K + 8K	2 KB	16 KB	32	1.6 - 3.6V
ATxmega128A1-AU	128K + 8K	2 KB	8 KB	32	1.6 - 3.6V
ATxmega64A1-AU	64K + 4K	2 KB	4 KB	32	1.6 - 3.6V
ATxmega384A1-CU	384K + 8K	4 KB	32 KB	32	1.6 - 3.6V
ATxmega256A1-CU	256K + 8K	4 KB	16 KB	32	1.6 - 3.6V
ATxmega192A1-CU	192K + 8K	2 KB	16 KB	32	1.6 - 3.6V
ATxmega128A1-CU	128K + 8K	2 KB	8 KB	32	1.6 - 3.6V
ATxmega64A1-CU	64K + 4K	2 KB	4 KB	32	1.6 - 3.6V
ATxmega128A1-C7U	128K + 8K	2 KB	8 KB	32	1.6 - 3.6V
ATxmega64A1-C7U	64K + 4K	2 KB	4 KB	32	1.6 - 3.6V

The table below lists all the devices in the **XMEGA 'D4'** family.

Ordering Code	Flash	E ²	SRAM	Speed (MHz)	Power Supply
ATxmega128D4- AU	128 KB + 8 KB	2 KB	8 KB	32	1.6 - 3.6V
ATxmega64D4-AU	64 KB + 4 KB	2 KB	4 KB	32	1.6 - 3.6V
ATxmega32D4-AU	32 KB + 4 KB	1 KB	4 KB	32	1.6 - 3.6V
ATxmega16D4-AU	16 KB + 4 KB	1 KB	2 KB	32	1.6 - 3.6V
ATxmega128D4- MH	128 KB + 8 KB	2 KB	8 KB	32	1.6 - 3.6V
ATxmega64D4-MH	64 KB + 4 KB	2 KB	4 KB	32	1.6 - 3.6V
ATxmega32D4-MH	32 KB + 4 KB	1 KB	4 KB	32	1.6 - 3.6V
ATxmega16D4-MH	16 KB + 4 KB	1 KB	2 KB	32	1.6 - 3.6V
ATxmega32D4-CU	32 KB + 4 KB	1 KB	4 KB	32	1.6 - 3.6V
ATxmega16D4-CU	16 KB + 4 KB	1 KB	2 KB	32	1.6 - 3.6V

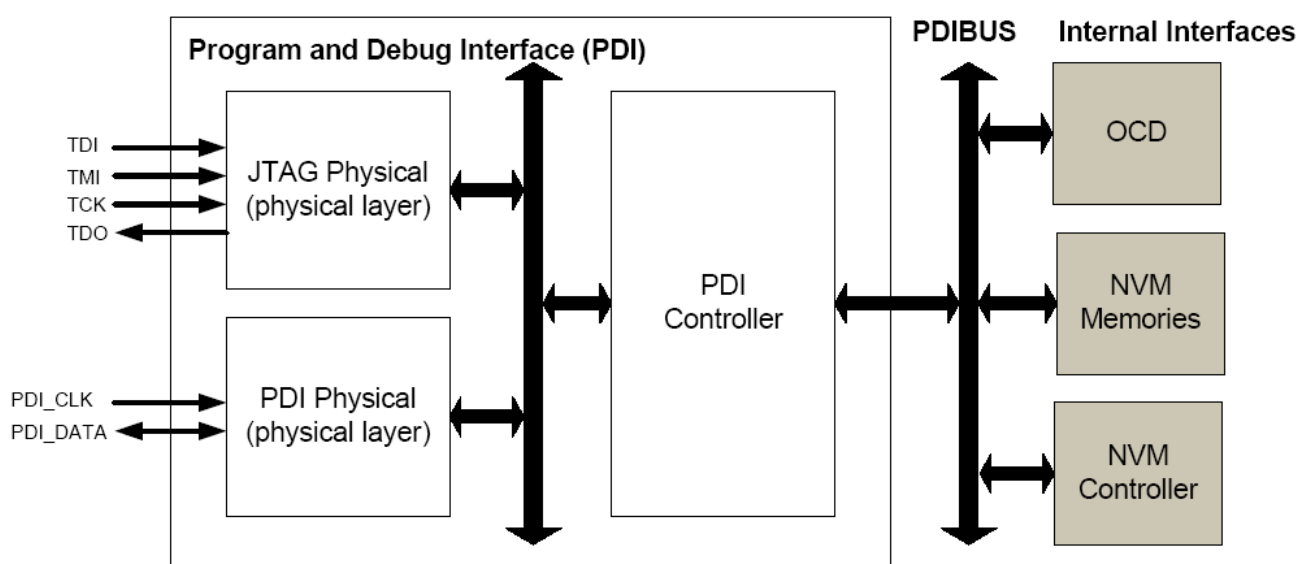
2.0 XMEGA Programming Interfaces

2.1 Overview of XMEGA Programming Interfaces

The Atmel XMEGA AVR microcontroller family feature either one or both of the following physical interfaces:

Interface Name	Interface description	Number of interface signals	XMEGA pins required for programming
PDI	Program and Debug Interface	2-wire	<ul style="list-style-type: none"> • PDI_CLK (RESET) • PDI_DATA
JTAG	JTAG Program / Debug Interface	4-wire + RESET	<ul style="list-style-type: none"> • TDI, TMI, TCK, TDO • RESET

A graphical overview of the '**PDI**' and '**JTAG**' interfaces is shown in the diagram below.



It is possible to perform both programming and debugging using either of the two physical interfaces. The primary interface is the '**PDI Physical Interface**'. This is a 2-pin interface which uses the **RESET pin** for the clock input (**PDI_CLK**), and the dedicated test pin **PDI_DATA** for data input and output

A 4-pin JTAG interface is also available on most of the higher pin-count XMEGA devices (but not all), and this interface can be used for programming and debugging. The JTAG interface is IEEE std. 1149.1 compliant, and supports JTAG boundary scan. Control of the RESET pin is also required by the programmer in order to halt user code execution.

Important note:

- The 2-wire PDI interface uses the XMEGA RESET pin as the PDI_CLOCK pin. This means that **the RESET pin must be dedicated for programming** and cannot be used as a standard RESET pin.

2.2 Comparison of XMEGA PDI and JTAG algorithms

The table below compares the JTAG and SPI programming algorithms for the Atmel XMEGA AVR family of microcontrollers.

Parameter	PDI algorithm	JTAG algorithm	Comments
Equinox programming support availability	Yes All XMEGA devices supported.	No We have no plans to support the JTAG algorithm	Equinox only currently support the 2-wire PDI algorithm.
PDI / JTAG Port availability on XMEGA devices	PDI port available on all XMEGA devices	JTAG port only available on higher pin-count XMEGA devices.	Use PDI if you need a common interface to all XMEGA devices. See note 1.
Programming speed	Marginally faster than JTAG	Marginally slower than PDI	Depends on PDI / JTAG clock frequencies See note 2.
Programming reliability	Very good Not reliable with long ISP cables.	Very good	Possible problems with PDI data integrity See note 3.
In-System Debugging	Yes – use Atmel JTAG-ICE debugger	Yes – use Atmel JTAG-ICE debugger	JTAG port normally used during development phase for both PDI and JTAG. See note 4.
ISP programming cable length	Max PDI cable length is 10 cm	Max JTAG cable length is 25 cm	Problem with clock integrity in PDI mode when using long cables.
Boundary Scan Testing	Not possible	Yes – requires external JTAG tester	Very useful for production testing. See note 5.
Multiple XMEGA AVR programming on same Target Board	Very difficult in PDI mode	Possible to daisy-chain multiple XMEGA AVR devices in a JTAG chain.	Only one device can be programmed at a time.
Programming pins required	2 x PDI pins PDI_CLK (RESET), PDI_DATA	4 x JTAG pins + RESET TDI, TDO, TCK, TMS	PDI uses RESET pin for the PDI_CLK. See note 6.
Programming pins can be used for user I/O?	No – PDI pins are dedicated for programming.	Not recommended	No other components are allowed on the pins.
RESET pin control required?	Yes RESET pin is the PDI Clock. It be used as normal RESET pin.	Yes RESET pin is used to stop user code from disabling JTAG access.	The RESET pin is essential for PDI and JTAG operation. See note 6.
RESET pin isolation during programming	The PDI_CLK pin is the XMEGA RESET pin. Customer RESET circuit must be isolated during PDI programming.	No	Careful design of the target RESET circuit is required to allow production programming via the PDI interface. See note 6.

XMEGA static current consumption	PDI interface does not affect static current	XMEGA device will take more current if JTAG port is enabled	Always disable JTAG port if low current consumption is required.
On-chip accurate oscillator calibration	Probably not possible	Yes The JTAG TDI pin is used to inject a calibration signal	Please contact Equinox for further information

PDI / JTAG Port availability (note 1)

- The 2-wire PDI port is available on all XMEGA devices in the family.
- The 4-wire JTAG port is only available on higher pin-count XMEGA devices.
- If you would like to use the same programming interface for all XMEGA devices, then PDI would be the best choice.

Programming speed (note 2)

It is likely that the PDI algorithm will be marginally faster than the JTAG algorithm as the JTAG algorithm is effectively the PDI protocol sent via JTAG with lots of passing bytes. So even though PDI features bi-directional data transfer, it will still probably be faster than JTAG.

Programming reliability (note 3)

The PDI algorithm uses a synchronous clock on the RESET pin of the XMEGA device. If there is any capacitance on the RESET pin eg. a CR reset circuit, then this could affect the reliability of PDI programming. It is therefore essential that any reset circuit is physically isolated from the XMEGA RESET pin during the PDI programming process.

There are known reliability problems when programming XMEGA devices using long (>10cm) ISP cables. The reliability problems are due to noise / skew on the PDI_CLOCK signal line. The solution is to use a '**Clock Buffer**' circuit at the target end of the long ISP cable to clean up the clock.

In-System Debugging (note 4)

The Atmel JTAG-ICE MK2 debugger supports debugging of XMEGA AVR microcontrollers via both the PDI and JTAG interface.

Boundary Scan Testing (note 5)

All XMEGA AVR devices which feature a JTAG port are capable of being tested in-circuit using the so called JTAG '**Boundary Scan Testing**' technique. This technique is not supported by Atmel or Equinox. It should be available on request from any good **Boundary Scan Tester** company.

Multiple XMEGA AVR programming on same Target Board (note 6)

It is possible to program multiple XMEGA AVR devices on the same Target Board using a single JTAG programming interface by connecting the XMEGA devices in a so called '**JTAG Chain**'. This mode will be supported by Equinox in the future.

It is not possible to connect the PDI ports of multiple XMEGA AVR devices together. Each device would require its own dedicated PDI programmer.

Programming pins required (note 6)

The PDI interface only uses two pins on XMEGA device. The **PDI_CLK** pin is actually the XMEGA RESET pin, so this pin would not be used for user I/O anyway. The **PDI_DATA** pin is a dedicated pin only used for PDI so there are no user I/O pins wasted by using the PDI interface. However, one possible disadvantage of the PDI interface is that the **PDI_CLK** pin which is in fact the **RESET** pin cannot have any RESET circuit on it as this would skew the clock signal and stop PDI working. The RESET circuit on a PDI Target Board must therefore be carefully designed to allow PDI programming during production programming but also to act as a normal reset circuit during normal operation.

2.3 Overview of PDI Interface

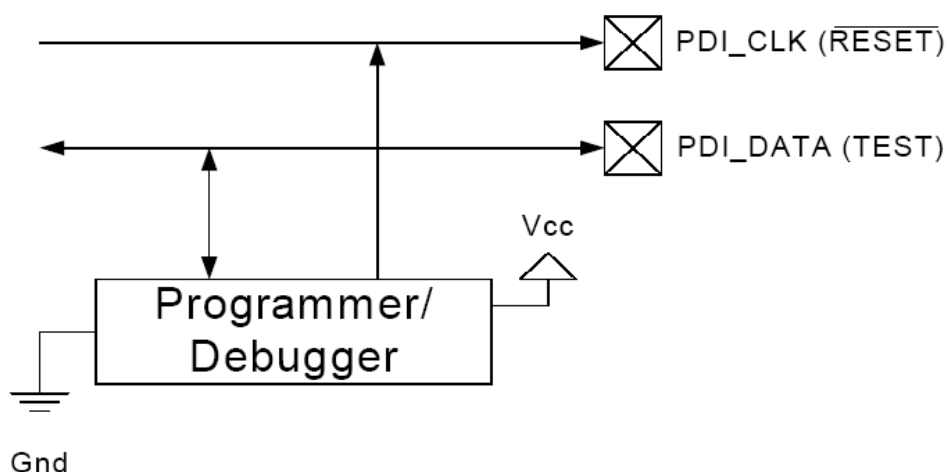
The **Program and Debug Interface (PDI)** is an Atmel proprietary interface for **external programming (ISP)** and **on-chip debugging (OCD)** of Atmel XMEGA AVR microcontrollers. The PDI interface supports high-speed programming of all on-chip Non-Volatile Memory (NVM) spaces including the Flash, EEPROM, Configuration Fuses, Security Fuses plus the User Signature Row.

2.4 PDI – Physical Interface

The **Program and Debug Interface (PDI)** is a 2-wire interface which allows the XMEGA device to be programmed using an external programmer. The pins required for programming via the PDI interface are detailed in the table below.

PDI Signal Name	Signal description	Direction from programmer	Pin name on XMEGA device
PDI_CLK (/RESET)	PDI Clock Signal (This pin is also the XMEGA RESET pin !!!)	Output	RESET
PDI_DATA	PDI Data Signal (bi-directional)	Bi-directional	TEST
GROUND (0V)	Target / Programmer Signal GROUND	Passive	GND
Vcc	Target / Programmer Vcc Supply	Passive	VCC

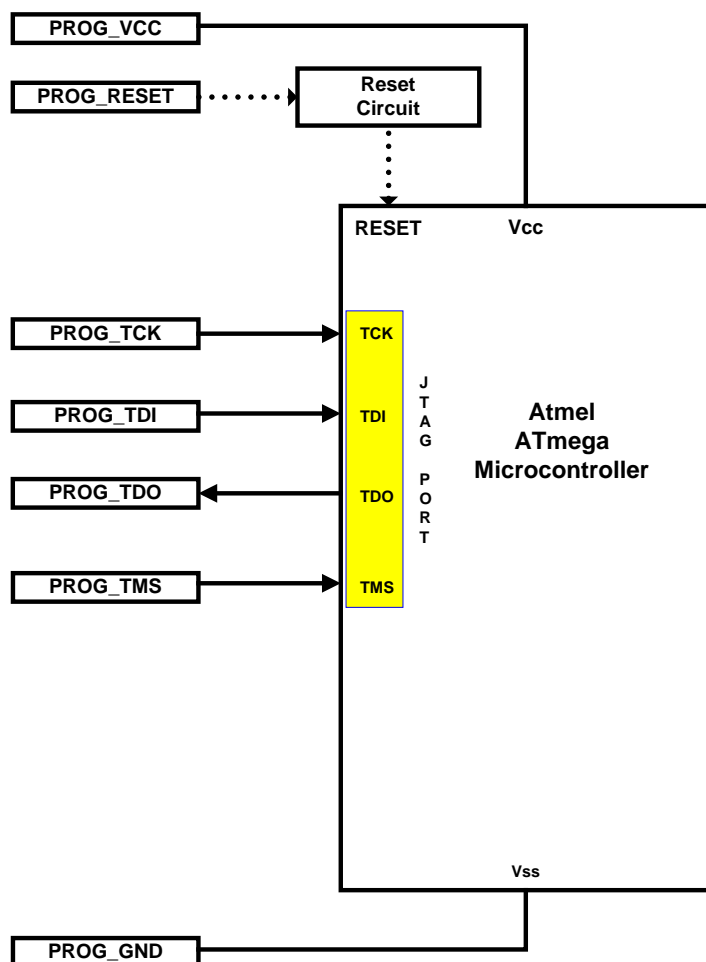
The connections between an '**External programmer**' and the **PDI Interface** of an XMEGA device are shown in the diagram below.



2.5 Overview of JTAG Interface

The JTAG algorithm provides a method of performing high-speed programming of Atmel XMEGA AVR microcontroller devices. The same JTAG port can also be used for on-chip debugging of code using the Atmel JTAG-ICE Debugger.

Fig 2.5 – XMEGA AVR – JTAG Programming Interface connections



Programmer Signal Name	Signal description	Signal direction (from Programmer)	Connect to AVR Microcontroller Pin	Signal direction (from Microcontroller)
PROG_TCK	Test Clock Pin	Output	TCK	Input
PROG_TDI	Test Data Input	Output	TDI	Input
PROG_TDO	Test Data Output	Input	TDO	Output
PROG_TMS	Test Mode Select	Output	TMS	Input
PROG_RESET	RESET	Output	RESET	Input

The advantages and disadvantages of the JTAG algorithm are detailed below.

Advantages

- The JTAG algorithm uses the same '**JTAG Port**' as the Atmel JTAG-ICE Debugger. This means that the same port can be used for both debugging during the development phase and also programming during the production phase of the product.
- It is possible to use the JTAG port of the Target Microcontroller to perform in-circuit testing of the microcontroller and surrounding circuitry. This testing is performed by shifting Test Data through the JTAG port of the Target Microcontroller. A JTAG Test System is required to perform this testing. It is not supported by any Equinox Programmer or the Atmel JTAG ICE.
- It is possible to daisy-chain multiple JTAG devices on the JTAG bus in a so-called '**JTAG Chain**' and then select to program a particular device in the chain.

Disadvantages

- Equinox do NOT currently offer support for the XMEGA JTAG algorithm.
- The JTAG algorithm is possibly marginally slower than the PDI algorithm.
- The JTAG Programming Interface uses 5 pins: TCK, TDI, TDO, TMS and RESET.
- The JTAG interface pins are all user I/O pins so the application will lose 4 I/O pins if JTAG is used.
- Not all XMEGA AVR devices have a JTAG port – only the larger pin-count devices can be programmed via JTAG!
- If the JTAG port is enabled on the XMEGA device, then the device will take more current than if PDI programming only was used.
- The JTAG pins of the microcontroller are not designed for off-board use and should not be shared with any other circuitry on Target Board. This means that the JTAG port pins must be dedicated for programming / debugging.

3.0 PDI Algorithm

3.1 Overview of PDI Interface

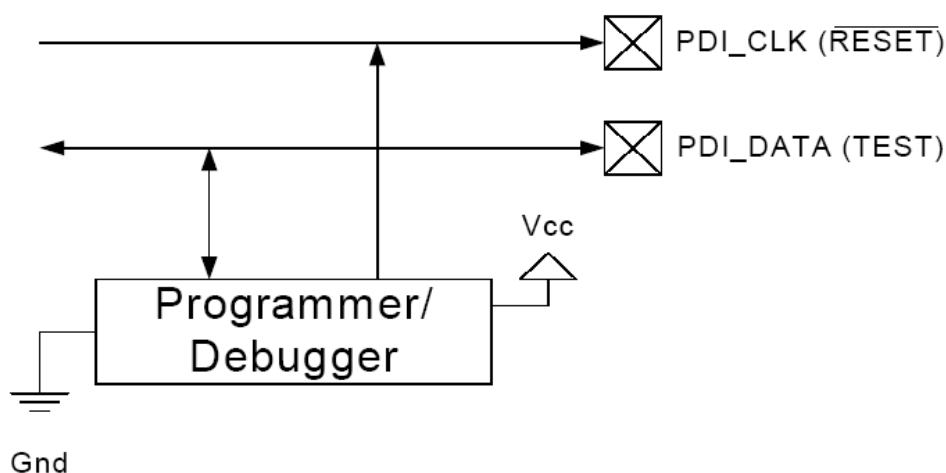
The **Program and Debug Interface (PDI)** is an Atmel proprietary interface for external programming and on-chip debugging of Atmel XMEGA AVR microcontrollers. The PDI interface supports high-speed programming of all on-chip Non-Volatile Memory (NVM) spaces including the Flash, EEPROM, Fuses, Lockbits plus the User Signature Row.

3.2 PDI – Physical Interface

The **Program and Debug Interface (PDI)** is a 2-wire interface which allows the XMEGA device to be programmed using an external programmer. The pins required for programming via the PDI interface are detailed in the table below.

PDI Signal Name	Signal description	Direction from programmer	Pin name on XMEGA device
PDI_CLK (RESET)	PDI Clock Signal	Output	RESET
PDI_DATA	PDI Data Signal (bi-directional)	Bi-directional	TEST
GROUND (0V)	Target / Programmer Signal GROUND	Passive	GND
Vcc	Target / Programmer Vcc Supply	Passive	VCC

The connections between an '**External programmer**' and the **PDI Interface** of an XMEGA device are shown in the diagram below.



3.3 PDI – Clock Signal (XMEGA RESET pin)

The '**PDI – Clock**' is a clock signal which is continuously generated by the programmer during PDI programming. This clock is fed from the programmer into the **PDI_CLK (RESET)** pin of the target XMEGA device. The clock signal must be a continuous waveform with a frequency ≥ 10 kHz, otherwise the target XMEGA device will exit PDI programming mode.

Important notes:

- As the XMEGA **RESET** pin is being used as a high-speed clock pin during PDI programming / debugging, it is therefore very important that this pin is free to oscillate without any external capacitive or resistive loading.
- The use of any form of reset circuit which prevents the external programmer from driving a clock into the RESET pin will probably cause the PDI programming to either be very unreliable or to not work at all.
- As the **PDI_CLK (RESET)** pin of the target XMEGA device is used as the **PDI CLOCK** pin, the **RESET** pin of the programmer must **NOT** be connected to this pin.

Recommendations:

- It is recommended that no other components are connected to the **PDI_CLK (RESET)** pin of the target XMEGA device during PDI programming.
- The programmer must connect directly to the actual **PDI_CLK (RESET)** pin of the XMEGA device.
- This can be achieved by using an '**Option link**' or '**0 ohm resistor**' on the **PDI_CLK (RESET)** pin which allows any other circuitry to be disconnected from this pin during PDI programming.
- If a capacitor must be placed on the **PDI_CLK (RESET)** pin for EMC purposes, then use the lowest value possible and isolate it from the actual **PDI_CLK (RESET)** pin by using e.g. a 10k ohm resistor.
- The length of cable between the programmer and the XMEGA DUT should be 10cm or less. If your application requires longer cables than 10cm, then it is recommended that the **PDI_CLOCK** signal is buffered at the target end of the cable using the '**Serial Clock Buffer Module**'.

3.4 PDI – Data Signal

The **PDI – Data** is a bi-directional signal line which is used to transfer data between the programmer and the target XMEGA device and vice-versa. This is a dedicated pin for PDI programmer and should not be used for any other purposes except PDI. When PDI programming mode is first entered, the programmer automatically becomes the '**master**' on the PDI bus and drives the **PDI_DATA (TEST)** signal line. As part of the PDI protocol, the programmer can then instruct the target XMEGA device to transmit data back to the programmer. In order to achieve this, the programmer must reverse the direction of the **PDI_DATA** signal so that the XMEGA device can then drive this line back to the programmer. The automatic reversal of the data direction is handled by special high-speed driver hardware on the external programmer.

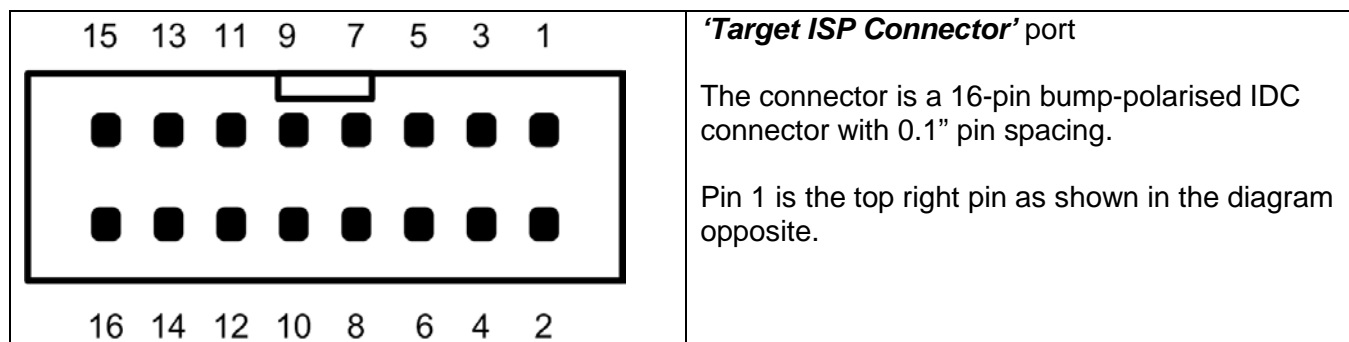
Recommendations:

- It is recommended that no other components are connected to the **PDI_DATA (TEST)** pin.
- This pin must be dedicated for PDI data transfer.
- The length of cable between the programmer and the XMEGA DUT should be 10cm or less. If your application requires longer cables than 10cm, then it is recommended that the **PDI_CLOCK** signal is buffered at the target end of the cable using the '**Serial Clock Buffer Module**'. The PDI_DATA signal cannot be buffered as it is a bi-directional signal.

3.5 ISPnano Series 3 - Target ISP Port – PDI pin-out

The '**Target ISP Connector**' port of the '**ISPnano Series 3**' programmer features all the signals required to implement In-System Programming (ISP) of a target XMEGA device using the '**2-wire PDI**' interface.

The illustration below shows the location of the '**Target ISP Connector**' port on the rear panel of the programmer.



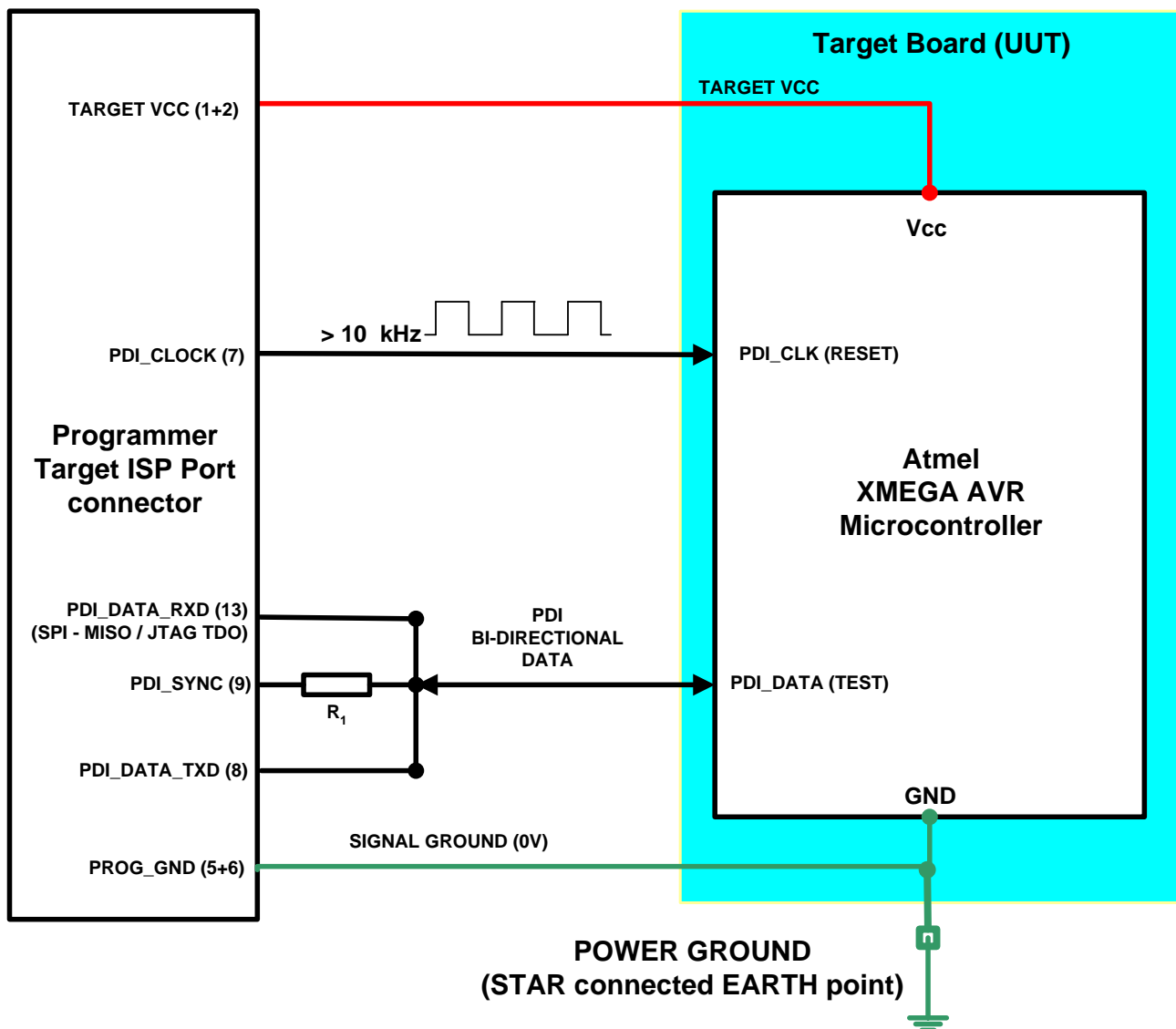
This connector also features the programmable "**Target Vcc**" and "**Target Vpp**" voltages plus a switched "**EXTERNAL Vcc**" supply.

The pins on this connector which are used for the '**PDI Interface**' are detailed in the table below.

Programmer Signal name (16-way IDC)	IDC pin	Signal description	Direction from programmer	Pin name on XMEGA device
TARGET_VCC	1+2	Target Vcc Supply	Passive	VCC
GROUND (0V)	5+6	Target / Programmer Signal GROUND	Passive	GND
PDI_CLOCK	7	PDI Clock Signal	Output	RESET
PDI_DATA_TXD	8	PDI Data Signal - TRANSMIT	Output	TEST
PDI_SYNC	9	PDI Synchronisation Signal	Output	TEST
PDI_DATA_RXD	13	PDI Data Signal - RECEIVE	Input	TEST

3.6 Single XMEGA device – PDI programming connections

The diagram below shows the connections required between the programmer and a Target Board for programming a single XMEGA microcontroller using the PDI interface.



Programmer Signal name (16-way IDC)	IDC pin	Signal description	Direction from programmer	Pin name on XMEGA device
TARGET_VCC	1+2	Target Vcc Supply	Passive	VCC
GROUND (0V)	5+6	Target / Programmer Signal GROUND	Passive	GND
PDI_CLOCK	7	PDI Clock Signal	Output	RESET
PDI_DATA_TXD	8	PDI Data Signal - TRANSMIT	Output	TEST
PDI_SYNC	9	PDI Synchronisation Signal	Output	TEST
PDI_DATA_RXD	13	PDI Data Signal - RECEIVE	Input	TEST

Please note:

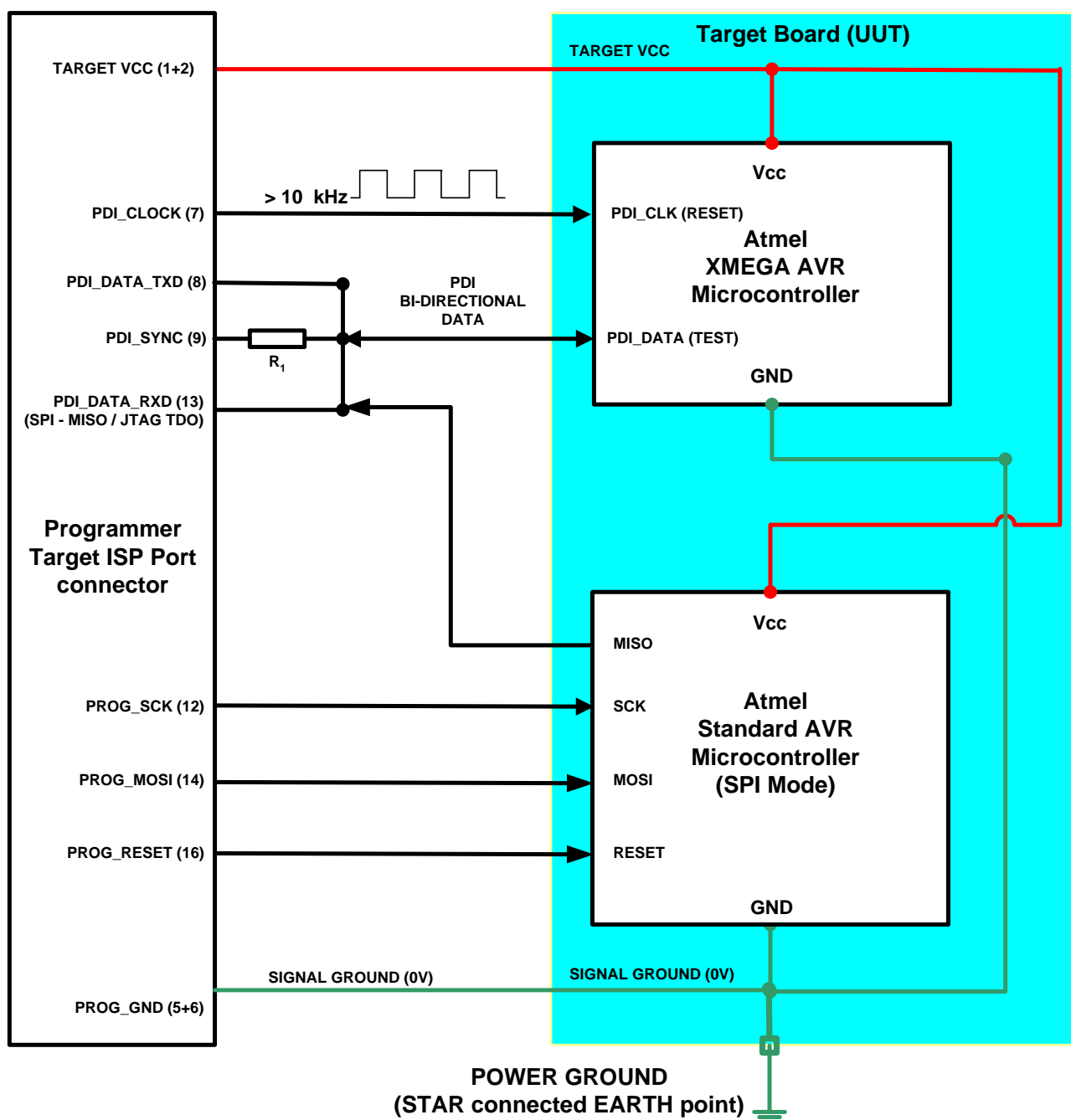
- **Very important** - The ISP cable length between the programmer and XMEGA DUT should be **no more than 10 cm long**. If longer ISP cables are necessary because of the programming fixture design then a '**Serial Clock Buffer Module**' is required to clean up the clock at the DUT end of the ISP cable.
- The **PDI_SYNC** signal (pin 9 on the Target ISP connector) should be connected to the **XMEGA TEST (PDI_DATA)** pin via a resistor – R1. The value of R1 should be 470 ohms. This pin is used to force the XMEGA device into PDI programming mode.
- The **RESET** line from the programmer must **NOT** be connected to the XMEGA device RESET pin (unless you are using some sort of external reset circuit).
- A separate "**SIGNAL GROUND**" and "**POWER GROUND**" should be implemented so that large fluctuations in the target 0V do not affect the PDI or SPI signals.
- The "**SIGNAL GROUND**" is connected between the programmer and the UUT 0V.
- The "**POWER GROUND**" is connected between the UUT 0V and the '**STAR Connected EARTH**' of the fixture. This should be the common EARTH point for the power supplies which are powering the programmer(s) and the UUT(s).

3.7 Single XMEGA PDI + AVR SPI – programming connections

The diagram below shows the connections required between the programmer and the Target Board for programming both an XMEGA microcontroller and a standard AVR microcontroller on the same Target Board.

This schematic shows how to program the following devices using a single programmer:

- Atmel XMEGA AVR microcontroller – PDI programming interface
- Atmel standard AVR microcontroller – SPI programming interface



Programmer Signal name (16-way IDC)	IDC pin	Signal description	Direction from programmer	Pin name on target device
TARGET_VCC	1+2	Target Vcc Supply	Passive	VCC
GROUND (0V)	5+6	Target / Programmer Signal GROUND	Passive	GND
PDI_CLOCK	7	PDI Clock	Output	XMEGA - RESET
PDI_DATA_TXD	8	PDI Data Signal - TRANSMIT	Output	XMEGA - TEST
PDI_SYNC	9	PDI Synchronisation Signal	Output	TEST
PROG_SCK	12	SPI – SCK Clock	Output	AVR - SCK
PDI_DATA_RXD PROG_MISO	13	PDI Data Signal – RECEIVE SPI – MISO	Input	XMEGA- TEST
PROG_MOSI	14	SPI – MOSI signal	Output	AVR - MOSI
PROG_RESET	16	Programmer RESET – AVR SPI device	Output	AVR - RESET

Please note:

- **Very important** - The ISP cable length between the programmer and XMEGA DUT should be **no more than 10 cm long**. If longer ISP cables are necessary because of the programming fixture design then a '**Clock Buffer**' circuit is required to clean up the clock at the DUT end of the ISP cable.
- The **PDI_SYNC** signal (pin 9 on the Target ISP connector) should be connected to the **XMEGA TEST (PDI_DATA)** pin via a resistor – R1. The value of R1 should be 470 ohms. This pin is used to force the XMEGA device into PDI programming mode.
- It is also recommended that a resistor R2 (value 470 ohms) is inserted in the **MISO** line to protect the programmer against a clash of both the target device and programmer drive this signal by mistake at the same time.
- There is one shared programmer signal line between the "**SPI**" and "**PDI**" port. The **MISO** line is shared with the '**PDI_DATA_RXD**' pin so this pin must be routed to both devices.
- When programming in PDI mode, it is important to keep the "**AVR SPI**" micro in reset (RESET pin held LOW) so that the AVR tri-states all its SPI lines allowing the **MISO** pin to be used as the '**PDI_DATA_RXD**' pin. This can be done by setting the RESET pin to '**LAL**' in the pre-programming state machine.
- The **RESET** line from the programmer must **NOT** be connected to the XMEGA device RESET pin (unless you are using some sort of external reset circuit).
- A separate "**SIGNAL GROUND**" and "**POWER GROUND**" should be implemented so that large fluctuations in the target 0V do not affect the PDI or SPI signals.
- The "**SIGNAL GROUND**" is connected between the programmer and the UUT 0V.

- The **"POWER GROUND"** is connected between the UUT 0V and the **'STAR Connected EARTH'** of the fixture. This should be the common EARTH point for the power supplies which are powering the programmer(s) and the UUT(s).

3.8 XMEGA PDI – ISP cable length recommendations

The ISP cable length between the programmer and XMEGA DUT should be **no more than 10 cm long**. There is a known issue with all newer versions of XMEGA devices with 2010 or later date code which makes these devices susceptible to noise / skew on the PDI_CLOCK pin. This problem can render PDI programming non-functional if long ISP cables are used.

Recommendation:

If longer ISP cables (>10 cm) are necessary because of the programming fixture design then a remote **'Serial Clock Buffer'** module is required to clean up the **PDI_CLOCK** signal at the DUT end of the ISP cable. See appendix 5 for further information about the **'Serial Clock Buffer Module'**.

3.9 Signal / Power GROUND (0V) connections

It is very important that both the programmer and Target System (UUT) are earthed correctly. Incorrect grounding can lead to current flowing in the 0V signal back to the PC which could cause ESD damage to either the programmer or the UUT. The ISPnano programmer features a **'Signal GROUND'** which is a specially filtered (cleaned) 0V signal which is used only for the programming signals. The UUT should then use its own dedicated **'Power GROUND'** as this will be much noisier than the programmer 0V.

Signal GROUND (0V)

The **'Signal GROUND'** is the 0V to which the programming signals (PDI, SPI, JTAG etc) are referenced to. This is a specially filtered 0V signal line which is used only for the programming signals. The **'Signal GROUND'** should be connected from the GROUND pins on the 'Target ISP Port' connector directly to the main GROUND on the UUT. The minimum cable length should be used for this connection.

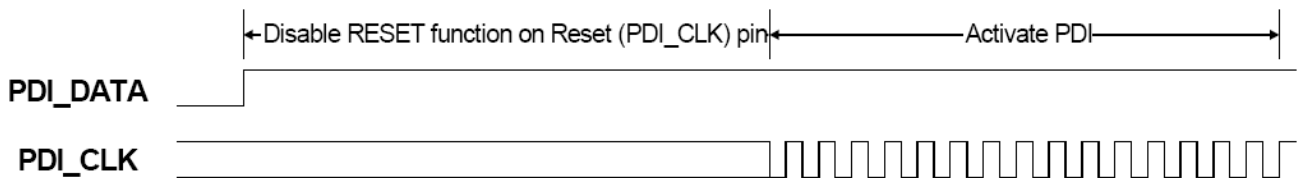
Power GROUND (0V)

The **'Power GROUND'** is the 0V to which the Target Board (UUT) uses as its 0V reference. The **'Power GROUND'** should be connected from the main GROUND (0V) point on the UUT to the 'Star connected GROUND' of the overall programming fixture. This is usually the point where all 0V GROUND connections are made for the power supplies in the fixture.

3.10 Enabling PDI programming mode

The PDI port must be enabled before it can be used. The external programmer must first force the **PDI_DATA (TEST)** line high for a period longer than the equivalent external reset minimum pulse width (refer to device data sheet for external reset pulse width data). This will disable the RESET functionality of the RESET pin, if not already disabled by the fuse settings. This can be achieved by making the correct settings in the 'Pre-programming statemachine' in the EQTools programming project.

The **PDI_CLK (RESET)** line must then be kept high for 16 PDI_CLK cycles (16 positive edges detected). The first PDI_CLK cycle must start no later than 100uS after the RESET functionality of the Reset pin was disabled. If this does not occur in time the RESET functionality of the Reset pin is automatically enabled again and the enabling procedure must start over again.



4.0 Creating an EDS Development Project

4.1 Overview

This section describes how to create a '**Programming Project**' for an Atmel XMEGA AVR microcontroller using the '**PDI – Serial Programming Algorithm**'. If you have used the Atmel '**AVR Studio**' software to develop the firmware for your application, it may then be necessary to convert the '**Fuses**' and '**Lock Bits**' to the correct format for inclusion in your EQTools project – please see section 5 for further details.

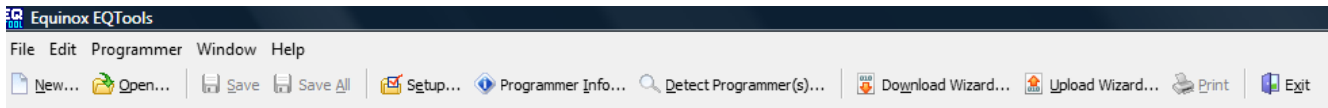
4.2 Information required to create an XMEGA PDI Project

The following information about the Target System is required in order to create an AVR SPI Programming Project:

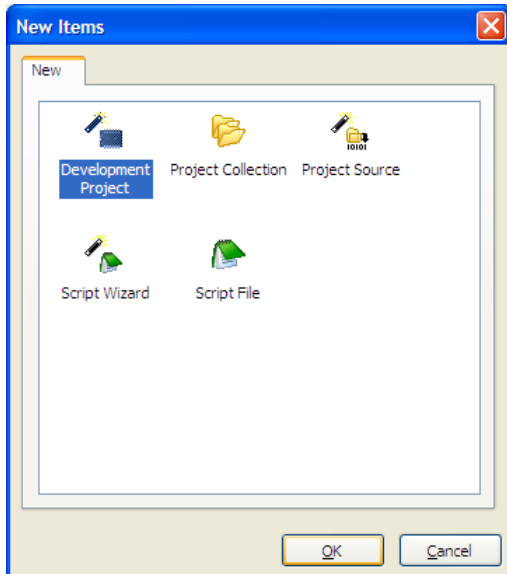
#	Information / data required	Example
1	XMEGA AVR Device part number	ATxmega128A1
2	PDI connections / connector on Target board	Atmel 6-way PDI connector
3	PDI Programming configuration	i. Single XMEGA device or ii. XMEGA device + a secondary SPI or JTAG microcontroller
4	Target System Vcc voltage	e.g. 3.3V
5	Target System maximum current consumption	e.g. 100mA
6	FLASH area 'Program File'	Binary (*.bin) or Intel Hex (*.hex)
7	EEPROM area 'Data File'	Binary (*.bin) or Intel Hex (*.hex)
8	Configuration Fuse values These fuse values describe how the 'Configuration Fuses' in the XMEGA AVR device are to be programmed.	i. Boolean fuse values: e.g. RSTDISBL=0, CKSEL=1, CKSEL2=0 etc ii. Fuse Hex values from 'AVR Studio' e.g. 0x22 0x45 0x34
9	Reset circuit parameters	Make sure any reset circuit is completely isolated from the XMEGA RESET pin during programming.

4.3 Launching the EDS Wizard

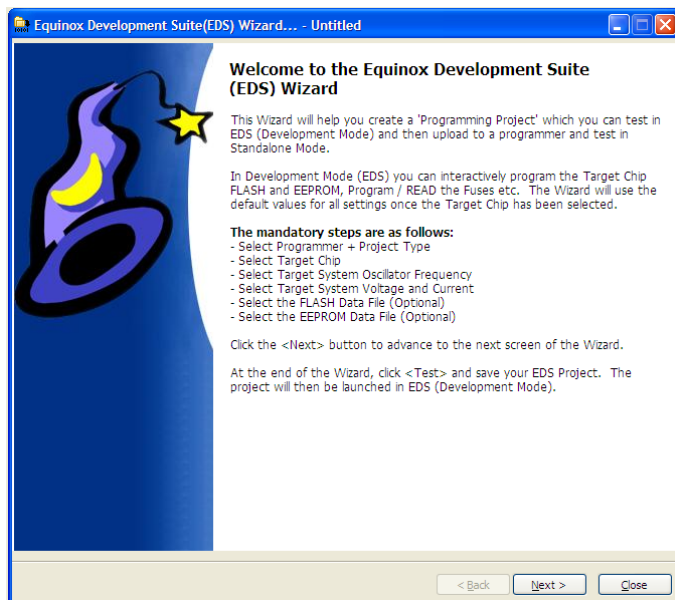
- Click the **<New>** icon on the top icon bar



- Select **<Development Project>** icon



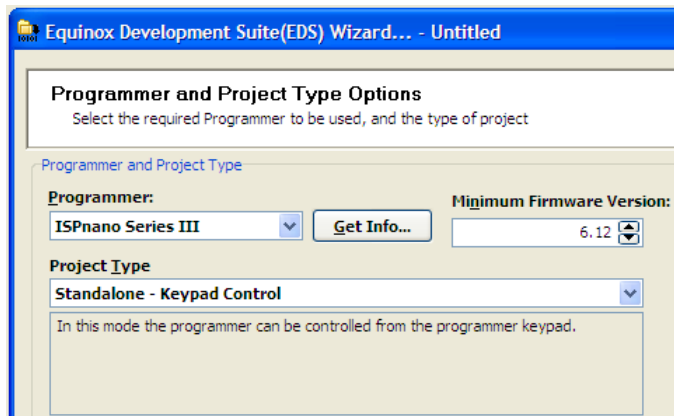
→ The EDS (Development) Wizard will launch



- Click **<Next>** → the **<Programmer and Project Type>** screen is displayed.

4.4 Selecting the attached programmer

This screen allows you to select the attached programmer and also to set up the correct 'Project Type' for your project.

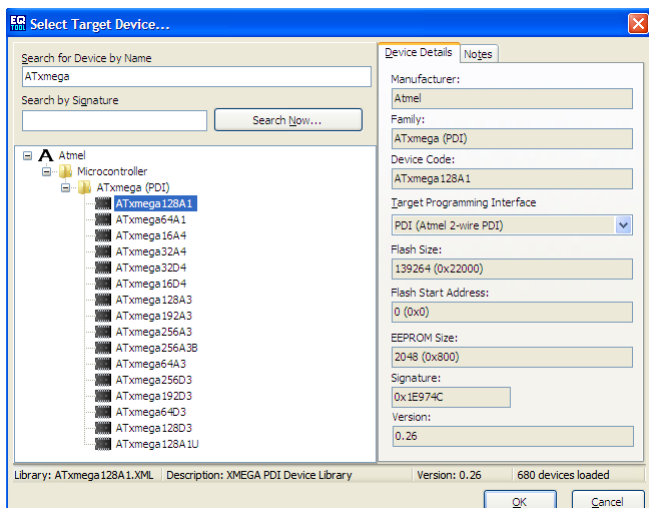


- Select your programmer from the drop-down list
- OR
- Click the **<Get Info>** button if your programmer is attached and powered on
→ This will automatically select the attached programmer.
- Set the '**Project Type**' to '**Standalone – keypad control**'.
This is correct for most applications. You can change this setting at a later stage if required.
- Click **<Next>** → the **<Select Target Device>** screen will be displayed.

4.5 Selecting the target XMEGA device

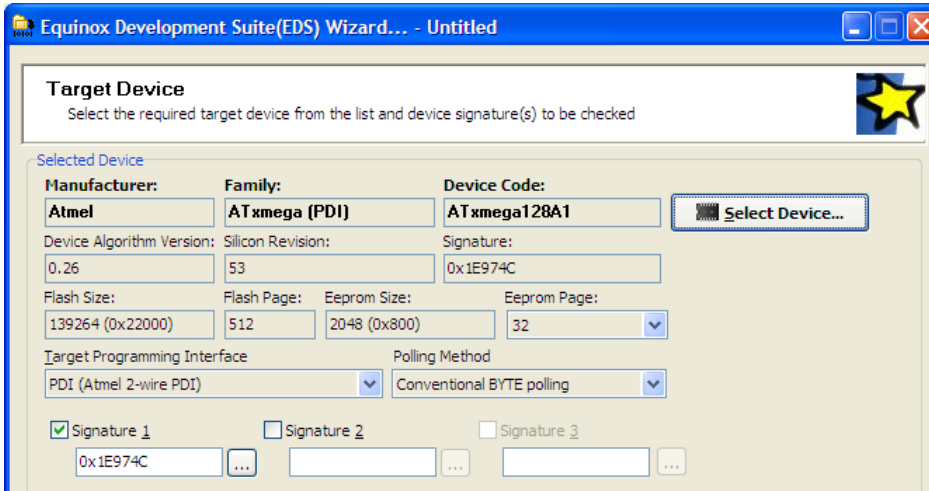
This screen allows you to select the target XMEGA device to program.

- Expand the '**Atmel**' tree, followed by the '**Microcontroller**' tree, followed by the '**ATxmega (PDI)**' tree.
→ A list of all the supported Atmel XMEGA PDI devices is displayed.....



- Select the required device e.g. ATxmega128A1 and then click the **<OK>** button.

→ The **'Target Device'** screen is now displayed...



Target Device
Select the required target device from the list and device signature(s) to be checked

Selected Device

Manufacturer: Atmel **Family:** ATmega (PDI) **Device Code:** ATmega128A1 **Select Device...**

Device Algorithm Version: 0.26 **Silicon Revision:** 53 **Signature:** 0x1E974C

Flash Size: 139264 (0x22000) **Flash Page:** 512 **Eeprom Size:** 2048 (0x800) **Eeprom Page:** 32

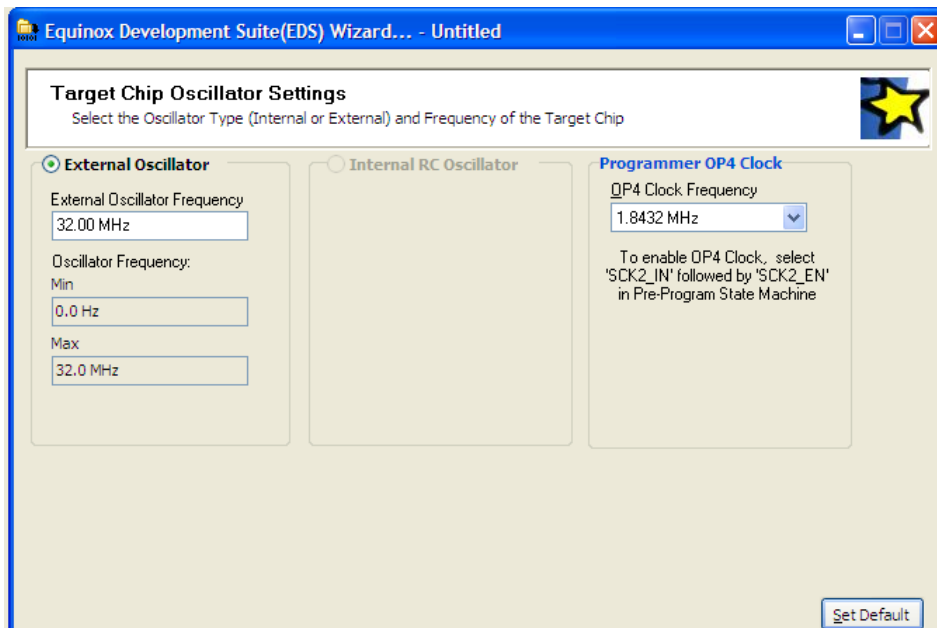
Target Programming Interface: PDI (Atmel 2-wire PDI) **Polling Method:** Conventional BYTE polling

☒ **Signature 1** 0x1E974C ☐ **Signature 2** ☐ **Signature 3**

- Click **<Next>** → the **<Target Oscillator Settings>** screen is displayed.

4.6 Selecting the XMEGA oscillator settings

This screen allows you to select the target XMEGA oscillator settings.



Target Chip Oscillator Settings
Select the Oscillator Type (Internal or External) and Frequency of the Target Chip

☒ **External Oscillator** ☐ **Internal RC Oscillator**

External Oscillator Frequency
32.00 MHz

Oscillator Frequency:
Min: 0.0 Hz
Max: 32.0 MHz

Programmer OP4 Clock
OP4 Clock Frequency: 1.8432 MHz

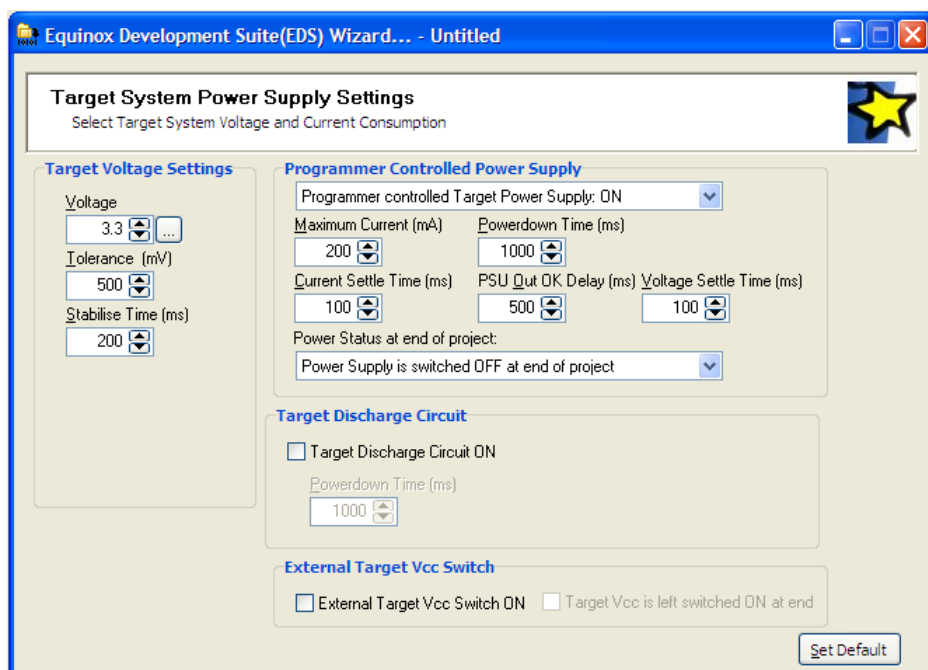
To enable OP4 Clock, select 'SCK2_IN' followed by 'SCK2_EN' in Pre-Program State Machine

Set Default

- Enter the frequency of the oscillator connected to the XMEGA device in the **'External Oscillator Frequency'** field.
- Do not change the **'Programmer OP4 Clock'** as this is not required for XMEGA programming.
- Click **<Next>** → the **<Target Power Supply>** screen is displayed.

4.7 Setting up the Target Power Supply

This screen allows you to set up how the programmer powers the Target System.



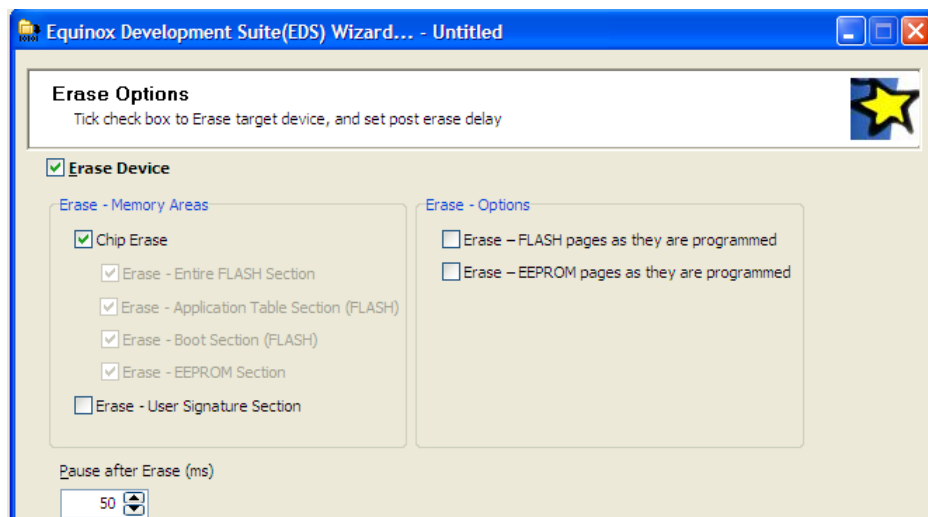
For most XMEGA applications, the programmer should power the **'Target System'** at e.g. 2.7 – 3.3V

To power the Target System from the programmer:

- Set the **'Programmer controlled Target Power Supply'** to 'ON'
- Set the **'Target Voltage'** to voltage at which the XMEGA device should be powered on your Target System.
- Click **<Next>** → the **<Erase options>** screen is displayed.

4.8 Setting up the XMEGA 'Erase Options'

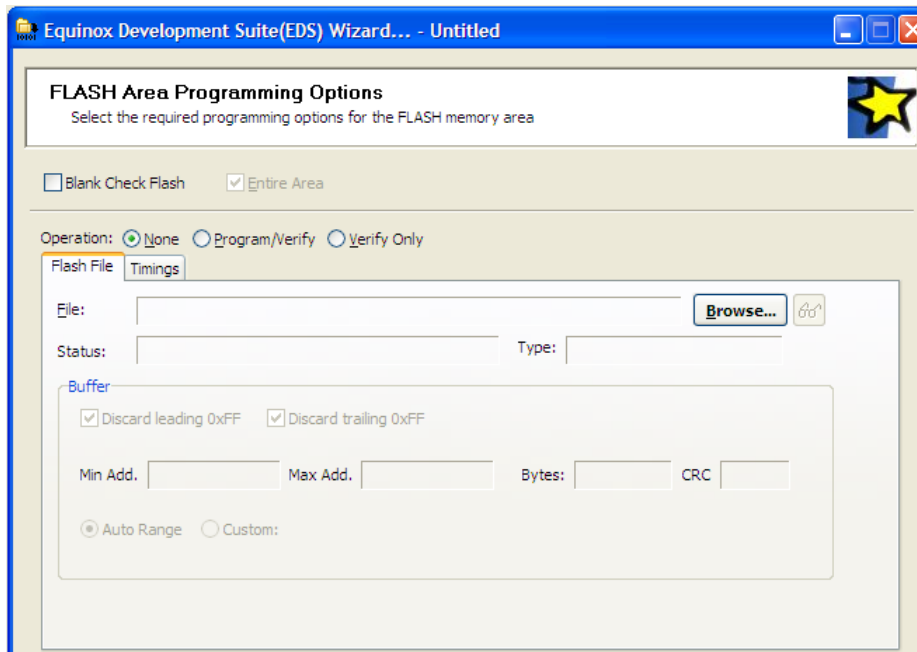
This screen allows you to set up how programmer handles the erasing of an XMEGA device.



- By default, the programmer will perform a '**Chip Erase**' operation which will erase all the individual FLASH sectors (Application table, Boot Section) and also the EEPROM.
- For now, leave the settings as default. **The 'Erase options'** are covered in more detail later on this application note.
- Click **<Next>** → the **<FLASH Area Programming options>** screen is displayed.

4.9 Setting up the XMEGA 'FLASH Programming'

This screen allows you to set up how to program the FLASH area(s) of the XMEGA device.

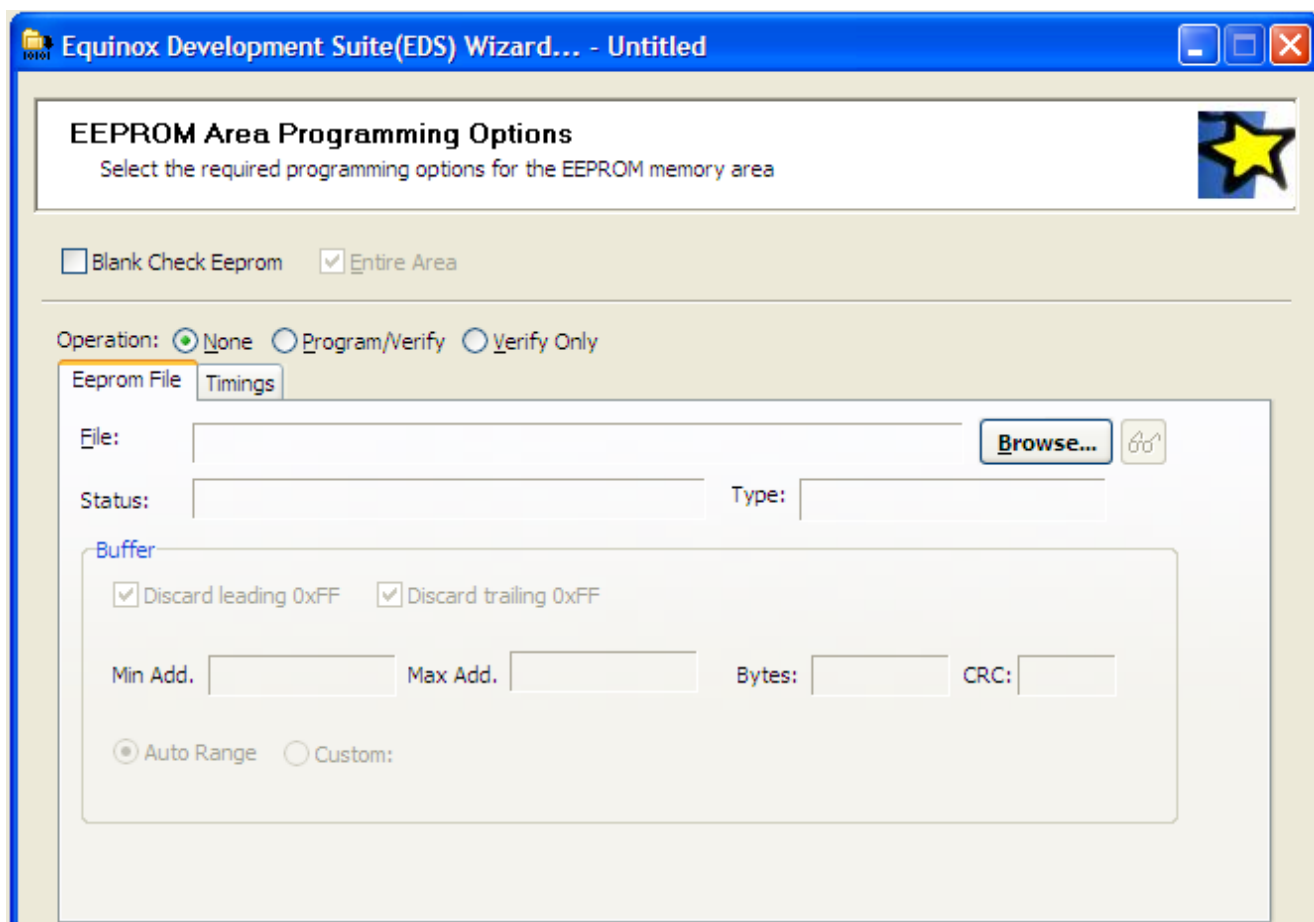


To load a file for programming into the FLASH area...

- Click the **<Browse>** button
 - Select the required input file e.g. *.bin, *.hex, *.SREC
- The start and end address of the input file will be displayed.
- The '**Operation**' is automatically set to '**Program/Verify**' which means the programmer will program each page and then verify each page of FLASH.

4.10 Setting up the XMEGA 'EEPROM Programming'

This screen allows you to set up how to program the EEPROM area(s) of the XMEGA device.

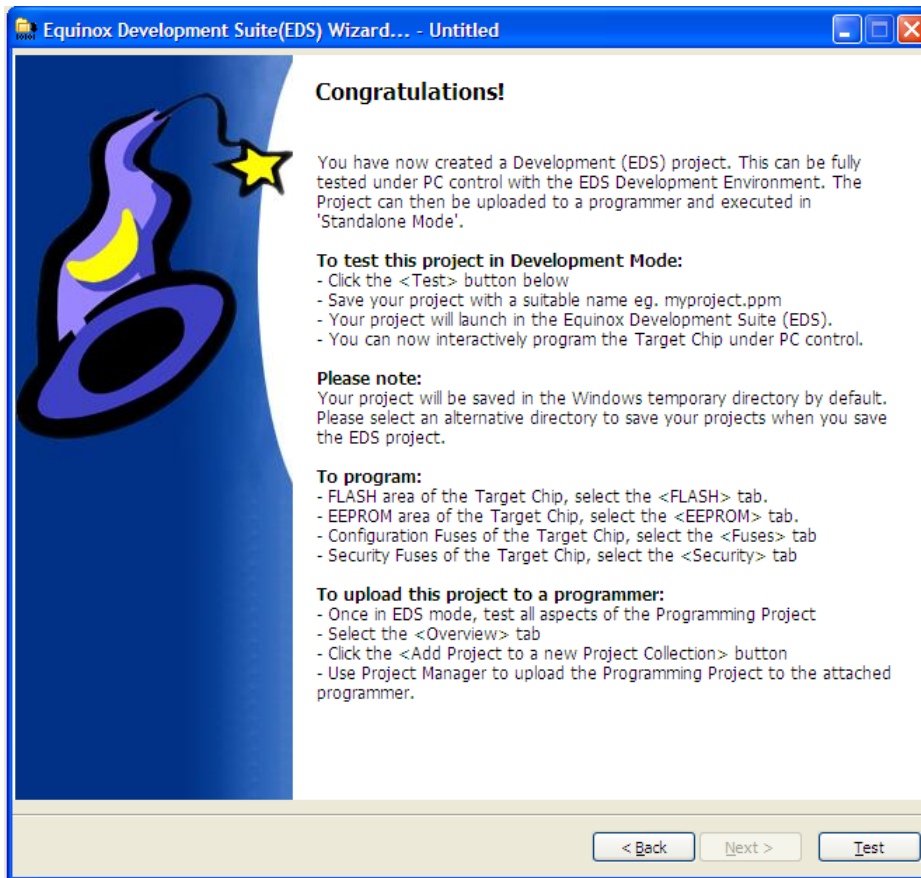


To load a file for programming into the EEPROM area...

- Click the **<Browse>** button
- Select the required input file e.g. *.bin, *.hex, *.SREC
- The start and end address of the input file will be displayed.
- The '**Operation**' is automatically set to '**Program/Verify**' which means the programmer will program each page and then verify each page of EEPROM.
- Click **<Next>** → the **<Congratulations >** screen is displayed.

4.11 Saving the EDS setup file

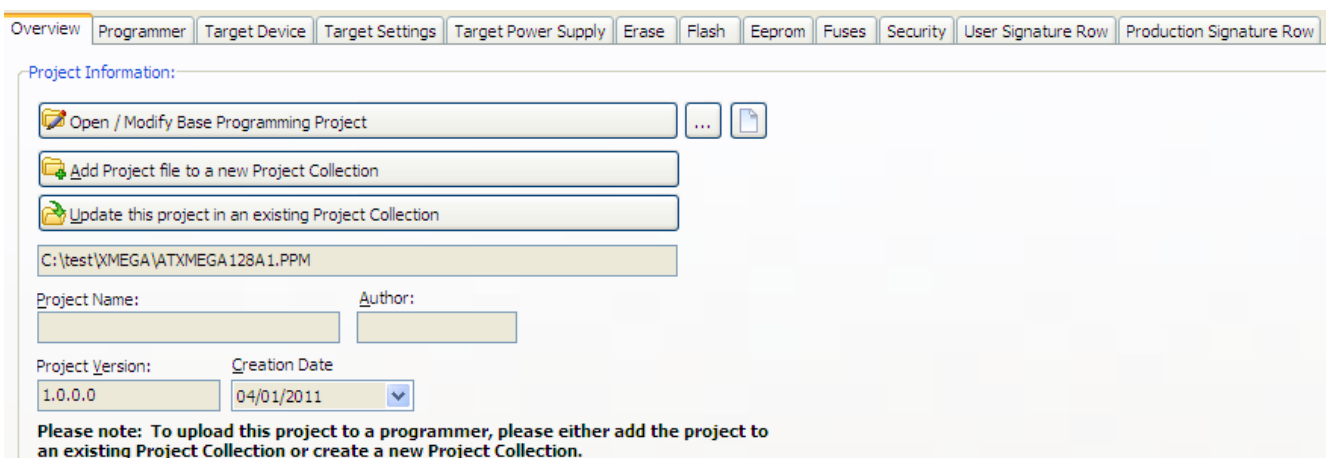
When you reach the '**Congratulations**' screen, you can now save this EDS project file to disk.



- Click the **<Test>** button
 - Save the project with a sensible name e.g. **ATxmega128A1.PPM**
- Your newly created EDS project will automatically start in a new EDS session.

4.12 Testing an EDS programming Project

Once an EDS programming project has been created, it can be tested by loading it into EDS.

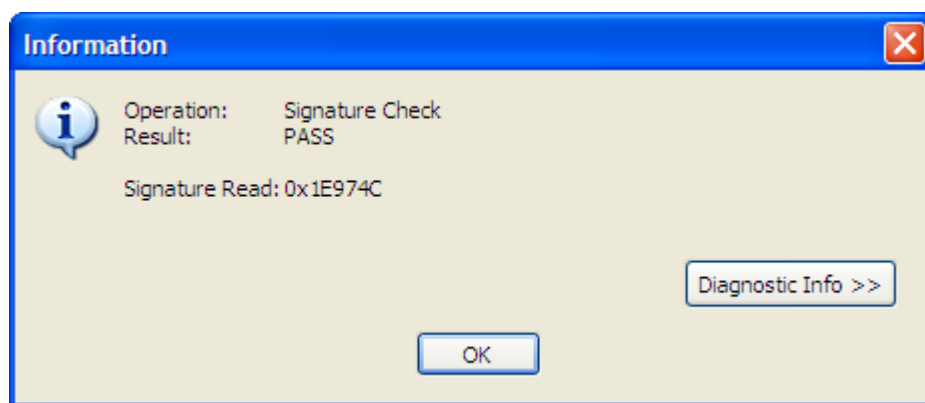


4.13 Checking the Device Signature (ID) of a target device

The best way to check that the programmer can communicate with an attached XMEGA device is to try to read back / check the 'Device Signature'.

Instructions:

- Select the '**Target Device**' tab
 - Click the **<Check ID>** button
- If the programmer manages to enter programming mode then it should report back the correct '**Device Signature**' for the selected XMEGA device.



If the programmer does not manage to enter programming mode, then EDS will report one of the following errors:

- Error 40 – Cannot enter programming mode
- Incorrect Device Signature – Read 0x??????, expected 0xyyyyyy

Please refer to section 4.14 to debug these error conditions.

4.14 Cannot Enter Programming Mode error

If the programmer cannot communicate with the target XMEGA device, then EDS will report '**Cannot enter programming mode**'.

If you receive this error, please check the following:

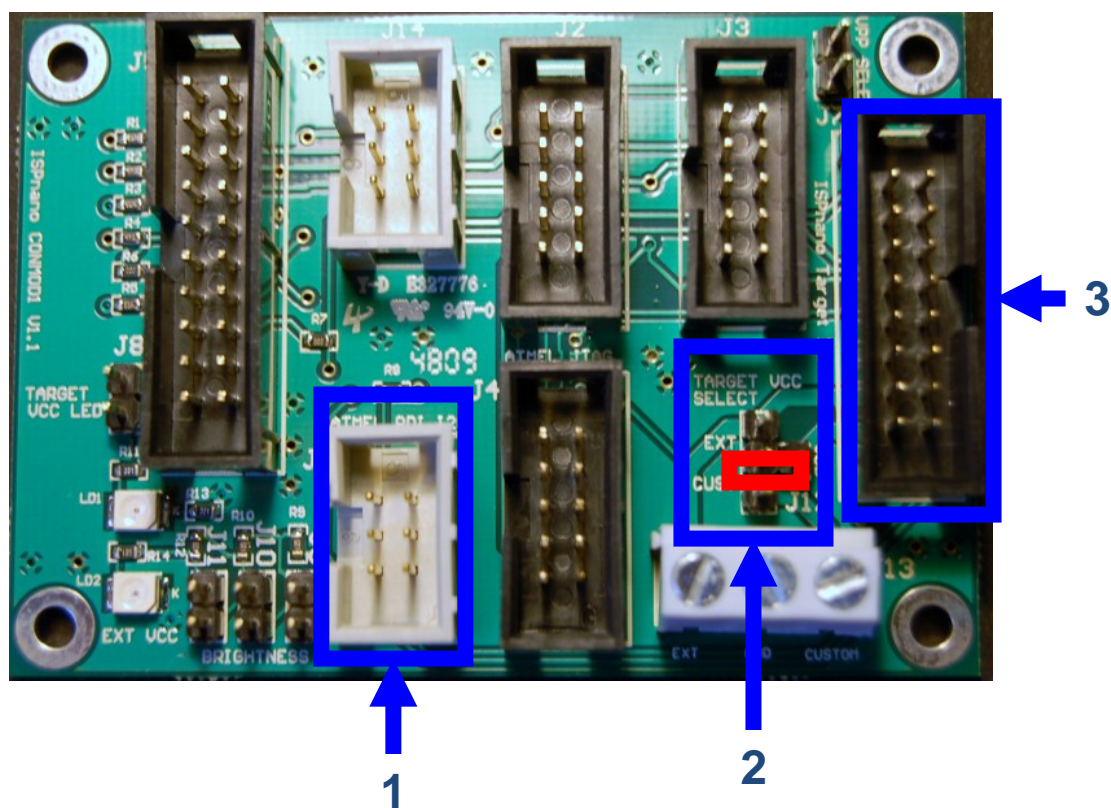
1. The PDI programming signal connections between the programmer and the Target System are correct.
2. Does the XMEGA RESET pin have any other components connected to it?
 - If there is a capacitor on this pin, then the clock signal from the programmer may be skewed causing the device to fail to enter PDI programming mode.
 - Try removing this capacitor and then attempt to '**Check ID**' again.
3. How long are the PDI_CLOCK and PDI_DATA signal wires?
 - It is recommended that these signal lines be kept as short as possible e.g. 10 cm or less.

- Try reducing the length of the connecting cables.
4. Is your project set up to power the Target System?
 - Check the power supply settings on the **'Target Power Supply'** settings of your project.
 5. Is the correct voltage applied to the XMEGA Vcc pin?
 - Measure the actual voltage on the Vcc pin of the target XMEGA device using a DVM (volt meter). Is it correct?

Appendix 1 – CONMOD Module + XMEGA PDI

1.0 Overview

This appendix describes how to use the **'ISPnano CONMOD Module'** to connect an **ISPnano Series III** programmer to an **"Atmel XMEGA AVR"** device using the 2-wire PDI interface. The CONMOD module features all the required circuitry to support programming of an XMEGA microcontroller via the PDI interface. The programmer connects to the 16-way IDC port labelled (3) and the XEMGA PDI device connects to the 6-way IDC connector labelled (1) in the picture below.



Please note:

- The **"Atmel PDI / I2C"** 6-way IDC connector – marked (1) in the above picture has the same pin-out as the standard **'PDI connector'** found on the Atmel STK600 kit.
- All relevant connections for PDI are already made on the CONMOD board so there is no need to add any other connections to get PDI to work.

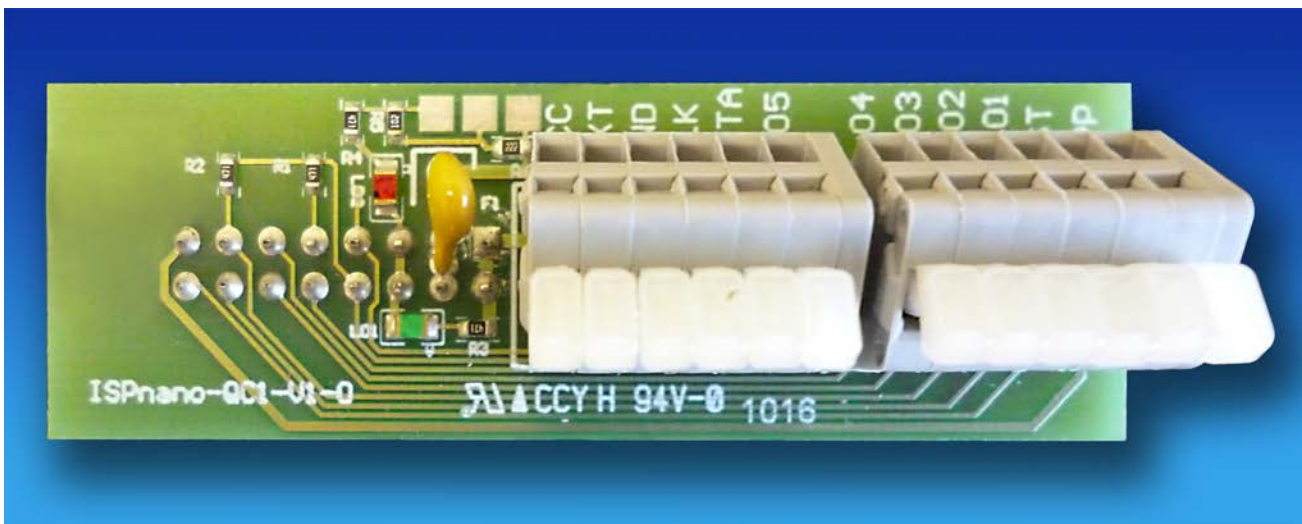
Instructions

- Referring to the annotated picture above
- Plug the 16-way IDC cable supplied with the programmer between the programmer **'Target ISP Port'** (16-way IDC connector) and the CONMOD Module 16-way header (J7) – see arrow (3).
- The PDI port is the 6-way IDC connector labelled **'Atmel PDI I2C'** – see arrow (1)
- Set up the **'Target Vcc Select'** jumper so that the programmer powers the Target Board – see red box marked (2) in the picture.

Appendix 2 – ISPnano-QC1 Quick Connect Module

1.0 Overview

This appendix describes the '**ISPnano-QC1**' Quick Connect Module. This module features all the required circuitry required to implement XMEGA AVR programming via the 2-wire PDI programming interface. The module plugs into the 16-way '**Target ISP Port**' on the '**ISPnano Series 3**' or '**ISPnano Series 3 ATE**' programmer. It provides so-called '**Quick connect**' connections allowing both an XMEGA PDI Target System and either a JTAG or SPI Target System to be connected to the programmer at the same time.



Features

- Plugs into the 16-way '**Target ISP Port**' on the '**ISPnano Series 3**', '**ISPnano Series 3 ATE**' or '**ISPnano Series 4**' programmers.
- Features all circuitry required for programming an Atmel XMEGA AVR microcontroller via the XMEGA PDI interface
- Allows an second device to be connected to the same programmer via either the JTAG or SPI interface
- All Target I/O signals are available via Quick-connect connectors
- External-Vcc in-line fuse
- '**Target Vcc**' LED
- '**External Vcc**' LED

1.1 Quick-Connect connector pin-out

The pin-out of the Quick Connect connectors is detailed in the table below.

QC pin	QC Signal Name	Signal description	Direction from programmer	Pin name on target device
1	VCC	Target Vcc Supply	Passive	VCC
2	EXT	EXTERNAL switched Target Vcc Supply	Passive	See note 1
3	GND	Target / Programmer Signal GROUND	Passive	GND
4	CLK	XMEGA PDI Clock	Output	XMEGA - RESET
5	DATA	XMEGA PDI Data	Output	XMEGA - TEST
6	I/O5	Spare I/O	Input / Output	See note 2
7	I/O4	JTAG - TMS	Output	JTAG - TMS
8	I/O3	<ul style="list-style-type: none"> SPI – SCK JTAG - TCK 	Output	<ul style="list-style-type: none"> SPI – SCK JTAG - TCK
9	I/O2	<ul style="list-style-type: none"> SPI – MISO JTAG - TDO 	Input	<ul style="list-style-type: none"> SPI – MISO JTAG - TDO
10	I/O1	<ul style="list-style-type: none"> SPI – MOSI JTAG - TDI 	Output	<ul style="list-style-type: none"> SPI – MOSI JTAG - TDI
11	RST	RESET	Output	See note 3
12	VPP	VPP Voltage	PASSIVE	See note 4

Please note:

- The signal names printed on the '**ISPnano-QC1**' Quick Connect Module are shown in the '**QC Signal Name**' column in the table.
- The XMEGA '**PDI Clock**' and '**PDI Data**' signals are marked as '**CLK**' and '**DATA**' respectively and should be connected to the target XMEGA RESET pin and TEST pin

Note 1

The '**EXTERNAL switched Target Vcc Supply**' is an external voltage applied to the '**DC-EXT**' pin of the programmer which can be switched to the Target System. It is usually used to switch a voltage to the input of a voltage regulator circuit on the Target System. If you are not using this functionality, then do not connect this pin.

Note 2

This is a spare I/O line which is not used for either the SPI or JTAG algorithms.

Note 3

The RESET pin of the programmer should **NOT** be connected to the target XMEGA AVR RESET pin as XMEGAs use the RESET pin as the CLOCK pin.

If you are planning to connect an SPI or JTAG device to the programmers as well as an XMEGA PDI device, then the RESET pin of the programmer should connect to the RESET pin of the SPI or JTAG device.

Note 4

The **VPP** pin outputs a programmable voltage from 5.0 to 13.5V which is used to place certain devices eg. ATtiny microcontrollers into '**High-voltage programming mode**'. This pin should not be connected if you are programming any other devices.

Appendix 3 – Using ConsoleEDS to program XMEGA PDI devices

1.0 Overview

This appendix describes how to use the ConsoleEDS utility to program Atmel XMEGA microcontrollers via the PDI programming interface. It is possible to use ConsoleEDS to program the FLASH, EEPROM, User Signature Row, Configuration Fuses and Security Fuses of an XMEGA device using simple command strings executed via the command line.

Please note:

This section provides specific instructions of how to use ConsoleEDS to program an XMEGA device. For further information about how to use the ConsoleEDS application in general, please refer to Application Note AN111.

1.1 Explanation of ConsoleEDS ‘Base Projects’

Most ConsoleEDS commands require that a **‘Base Project’** is created for each device to be programmed. The **‘Base Project’** is used by ConsoleEDS to define the following parameters about the target device / target system:

- Target Device e.g. ATxmega128A1
- Target Programming Interface e.g. PDI
- Target Vcc Voltage e.g. +3.3V
- Target Power Supply characteristics e.g. current
- Target Programming speed e.g. 415 kBaud
- Device Signature / Device ID

The **‘Base Project’** is declared on the ConsoleEDS command line as follows:

ConsoleEDS BaseProject.prj /FLASHWRITE=FlashData.hex

1.2 Setting up a ConsoleEDS ‘Base Project’

The simplest way to set up a ConsoleEDS **‘Base Project’** is to use the EDS **‘Development Wizard’**.

Here is an overview of how to set up a **‘Base Project’**:

- Launch the EDS Development Wizard
- Select the required device eg. **ATxmega128A1**
- Make sure the **‘Erase task’** is enabled in the project and all **‘Erase options’** are enabled.
- Make sure the **‘Fuse task’** is enabled in the project. It doesn’t matter what fuse values are selected, only that the **‘Fuse task’** is enabled.
- Make sure the **‘Security task’** is enabled in the project. It doesn’t matter what fuse values are selected, only that the **‘Security task’** is enabled.
- You should not select any FLASH file in the project.
- Compile the project to make a *.prj project eg. ATxmega128A1.PRJ

- You have now created a '**Base Project**'.

1.3 Reading the 'Device Signature / ID'

It is possible to read the '**Device Signature**' from the target device using the **/READSIG** command. This command allows ConsoleEDS to check that the correct device is connected to the programmer and that the device will enter programming mode OK.

Typical command useage:

ConsoleEDS ATxmega128A1.PRJ /READSIG

Typical response:

Console EDS - Signature read back: 0x1E974C

1.4 Erasing the FLASH and EEPROM

The XMEGA devices feature FLASH technology where each byte of FLASH can be re-programmed but only if the entire FLASH has been erased first. The programmer can program any bit in the FLASH from a '1' to a '0' but cannot program a '0's back to a 1. This means that the programmer must send the '**Chip Erase**' command to erase all FLASH locations back to 0xFF **BEFORE** any new data can be programmed into the FLASH area.

Typical command useage:

ConsoleEDS ATxmega128A1.PRJ /ERASE

Typical response:

Console EDS - Erasing target

Console EDS - TARGET_ERASE

Please note:

- i. The **/ERASE** command does not receive any feedback from the target device to tell it whether the erase operation worked or not. It is therefore up to the next commands in the sequence to handle any errors if the erase did not work.
- ii. The '**Erase task**' must be enabled in the '**Base Project**' otherwise the erase operation will fail.

1.5 Programming the FLASH using the /FLASHWRITE command

The **/FLASHWRITE** command is used to program data from a file on the PC hard disk to the FLASH area of the target device. It is possible to program any address range from a single byte to the entire FLASH area of the device.

Typical example:

ConsoleEDS ATxmega128A1.PRJ /ERASE /FLASHWRITE=FLASH_DATA.HEX

This example will erase the entire FLASH area first and then program the data contained in the file FLASH_DATA.HEX into the FLASH area of the device.

i. Supported file types

EQTools supports loading of the following file types:

- *.BIN – Binary file
- *.HEX – Intel Hex file
- *.SREC – Motorola S-Record file

ii. ERASE the FLASH before programming

The FLASH technology used on Atmel XMEGA devices only supports a '**Chip Erase**' command which erases the entire FLASH area. The address range of the FLASH to be programmed must be erased to 0xFF **BEFORE** programming the file otherwise the programming operation will fail. This can be achieved by using the **/ERASE** command.

Typical example:

ConsoleEDS ATxmega128A1.PRJ /ERASE

→ This will erase the entire FLASH area.

1.6 Programming the 'User Signature Row'

The **/FLASHWRITE** command is used to program data from a file on the PC hard disk into the '**User Signature Row**' area of the target device.

Typical example:

***ConsoleEDS ATxmega128A1.PRJ /ERASE /FLASHWRITE=Signature_Row_Data.HEX
/OFFSET=0x0E0400***

This example will erase the entire FLASH area first and then program the data contained in the **Signature_Row_Data.HEX** file into the FLASH area of the device.

i. FLASH offset start address

The '**User Signature Row**' of an XMEGA device does not start at address 0x00000.

It actually starts at **0x0E0400** (see XMEGA memory map) so it is necessary to use the **/OFFSET=0x0E0400** command to program this special area of the FLASH using ConsoleEDS.

If the **'User Signature Row'** area to be programmed is already erased to 0xFF:

```
ConsoleEDS ATxmega128A1.PRJ /FLASHWRITE=Signature_Row_Data.HEX  
/OFFSET=0x0E0400
```

1.7 Programming the Fuses using the /FUSEWRITE command

The simplest method of programming the fuses of an XMEGA devices is to use the **/FUSEWRITE** command. This command allows the **'hex value'** of the fuses eg. 0xFF to be programmed directly into the **'Fuse array'** of the target device.

Typical example:

```
ConsoleEDS ATxmega128A1.PRJ PRJ /FUSEWRITE=0x00,0x00,0xFF,0xFF,0xDF  
/RESETAFTERFUSEWRITE
```

The **'hex value'** of the fuses can be found from any one of the following sources:

- 'AVR Studio' project or screenshot
- Atmel ELF File
- Equinox – EQTools Project file

Please note:

The **/RESETAFTERFUSEWRITE** command is required to force the device in and out of PDI programming mode between the **'Fuse Program'** and **'Fuse verify'** operations. This step is required to get around the **'Fuse verify'** problem with all XMEGA devices – see Appendix 4 for further details.

1.8 Reading the Fuses using the /FUSEREAD command

The hex value of the fuses of an XMEGA device can be read back using the **/FUSEREAD** command. This command will return the **'hex value'** of the fuses eg. 0xFF.

Typical example:

```
ConsoleEDS ATxmega128A1.PRJ /FUSEREAD  
Console EDS - MISP_FUSE_READ  
Console EDS - Fuses Read: 0xFF  
Console EDS - FINISH
```

1.9 Programming the ‘Security Fuses’ using the /SECURITYWRITE command

The simplest method of programming the ‘**Security fuses**’ (Lock bits) of an XMEGA devices is to use the **/SECURITYWRITE** command. This command allows the ‘**hex value**’ of the ‘**Security fuses**’ eg. 0xFE to be programmed directly into the ‘**Security Fuse array**’ of the target device.

Typical example:

ConsoleEDS ATxmega128A1.PRJ /SECURITYWRITE=0xFE

Console EDS - MISP_LOCK_WRITE

Console EDS - Security Fuses Written: 0xFE

Console EDS - Reading security fuses

Console EDS - MISP_LOCK_READ

Console EDS - Security Fuses Read: 0xFE

The ‘**hex value**’ of the ‘**Security fuses**’ can be found from any one of the following sources:

- ‘AVR Studio’ project or screenshot
- Atmel ELF File
- Equinox – EQTools Project file

1.10 Reading the ‘Security Fuses’ using the /SECURITYREAD command

The ‘**hex value**’ of the ‘**Security fuses**’ of an XMEGA device can be read back using the **/SECURITYREAD** command. This command will return the ‘**hex value**’ of the ‘**Security fuses**’ eg. 0xFF.

Typical example:

ConsoleEDS ATxmega128A1.PRJ /SECURITYREAD

Console EDS - Reading security fuses

Console EDS - MISP_LOCK_READ

Console EDS - Security Fuses Read: 0xFF

1.11 Executing a ‘Standalone Programming Project’

The fastest method of programming an ATtiny device will always be to use a ‘**Standalone programming project**’ stored inside the programmer memory. This is also a very simple and neat way of programming in a production environment as a ‘**Standalone programming project**’ performs all the required programming actions in a single pre-compiled project.

Typical example:

ConsoleEDS /AUTOPROGRAM=PROJECTNAME

This command will execute the '**Standalone programming project**' called '**PROJECTNAME**' which is stored inside the programmer memory. This project must have already been pre-configured and uploaded to the programmer **BEFORE** executing the **/AUTOPROGRAM** command.

A '**Standalone programming project**' can perform all or some of the actions below:

- Power up the UUT
- Enter programming mode
- Validate the '**Device Signature / ID**'
- Erase the 'Lock Bits' and then the entire FLASH area
- Program the FLASH area
- Program the '**Configuration fuses**'
- Program the '**Security fuses**' (Lock Bits)
- Power down the UUT

Why use a '**Standalone programming project**' ?

The '**Standalone programming project**' will be much faster at programming large areas of FLASH compared to programming the same data using the **/FLASHWRITE** command. This is because in standalone mode, the data is stored locally on the programmer so the data retrieval time is much faster than transferring it from the PC.

1.12 Mixing a '**Standalone project**' with individual programming commands

It is possible to combine multiple ConsoleEDS commands in a single session to create quite complex programming sequences. In the example below, a '**Standalone programming project**' stored inside the programmer memory is used to program the '**main firmware**' into the FLASH at high speed. A unique '**serial number**' is then programmed from a file called **SerialNumber.hex**. Finally the fuses are programmed and the device is then locked.

Typical example:

```
ConsoleEDS ATxmega128A1.PRJ /AUTOPROGRAM=MAINPROG
/FLASHWRITE=SerialNumber.HEX FUSEWRITE=0x00,0x00,0xFF,0xFF,0xDF
/SECURITYWRITE=0xFE
```

This single ConsoleEDS command line session will perform the following programming actions:

- Execute the '**Standalone programming project**' - **MAINPROG**
 - Power up the UUT (if programmer controlling target power)
 - Enter '**low-voltage**' TPI programming mode
 - Erase the 'Lock Bits' and then the entire FLASH area
- Programs the data contained in the file '**SerialNumber**' into the FLASH area
- Programs the '**Configuration fuses**'
- Exits PDI programming mode
- Enter PDI programming mode
- Verifies the '**Configuration fuses**'

- Programs the '**Security fuses**' (Lock Bits)
- Exits '**low-voltage**' TPI programming mode
- Power down the UUT

1.13 Typical XMEGA programming sequence

It is possible to combine multiple ConsoleEDS commands in a single session to create quite complex programming sequences. The example below enters programming mode, erases the FLASH area and then programs some data contained in the file '**FLASH_DATA.HEX**'. Finally the '**Configuration fuses**' and '**Security fuses**' are programmed.

Typical example:

```
ConsoleEDS ATxmega128A1.PRJ /ERASE /FLASHWRITE=FLASH_DATA.HEX  
/FUSEWRITE=0x00,0x00,0xFF,0xFF,0xDF /SECURITYWRITE=0xFE
```

This single ConsoleEDS command line session will perform the following programming actions:

- Power up the UUT (if programmer controlling target power)
- Enter PDI programming mode
- Erase the 'Lock Bits' and then the entire FLASH area (depends on settings of the 'Erase options')
- Programs the data contained in the file '**FLASH_DATA.HEX**' into the FLASH area
- Programs the '**Configuration fuses**'
- Exits PDI programming mode
- Enter PDI programming mode
- Verifies the '**Configuration fuses**'
- Programs the '**Security fuses**' (Lock Bits)
- Exits PDI programming mode
- Power down the UUT

The programmer will keep the target device in programming mode throughout the ConsoleEDS session which can save a considerable amount of time as it can take 300 – 500ms exiting and re-entering programming mode.

Appendix 4 – XMEGA Fuse Verify problems

1.0 Problem description

When the value of certain '**Configuration fuses**' in an XMEGA microcontroller are changed from a '**1**' to a '**0**' or a '**0**' to '**1**', the device will not read back the correct value of these fuses until the programmer has exited '**PDI programming mode**' and then re-entered programming mode. This is an unfortunate feature of the XMEGA PDI programming mode which is beyond the control of Equinox Technologies.

As the read back fuse values are incorrect, this will lead to a '**Fuse Verify Error**' when the programmer attempts to program the XMEGA fuses as the programmer writes the fuses and then immediately reads them back and verifies that they are correct. The read back values will be the previous values (i.e. the values before programming) not the newly programmed values so the fuse verify operation will fail.

The fuses which will trigger this problem are as follows:

- **RSTDISBL** – RESET pin disable fuse
- **JTAGEN** – JTAG ENABLE fuse
- **SUT0** – Start up time fuse
- **SUT1** – Start up time fuse
- **WDLOCK** – Watchdog time lock fuse

Important notes:

- The '**Fuse Verify Error**' problem is only triggered when the value of one of these fuses is changed from a '**1**' to a '**0**' or a '**0**' to '**1**'.
- If one of these fuses is programmed with the same value that it already contains then this programming action will not trigger the problem.

1.1 Error numbers / message for XMEGA Fuse Verify problem

The '**Fuse Verify Error**' can generate the following error codes / messages:

- **Error 44** – Pre-Erase Fuse Verify Error (with firmware 6.17 or below)
- **Error 45** – Post-Erase Fuse Verify Error (with firmware 6.18 or above)

The '**Fuse Verify Error**' will affect all methods of programmer control including EDS, ConsoleEDS, ASCII Control Protocol, keypad mode and using the hardware START signal or TTL control. The '**Error number**' will be the same no matter which control method is used.

1.2 Fix for this problem

This problem has been fixed in programmer firmware 6.29.

The programmer now always programs the fuses, exits PDI mode, re-runs the pre-programming Statemachine, re-enters PDI mode and then verifies the fuses. This process is completely automatic so no user intervention or changes to project settings should be required.

Appendix 5 – PDI Clock Buffer module

1.0 Overview

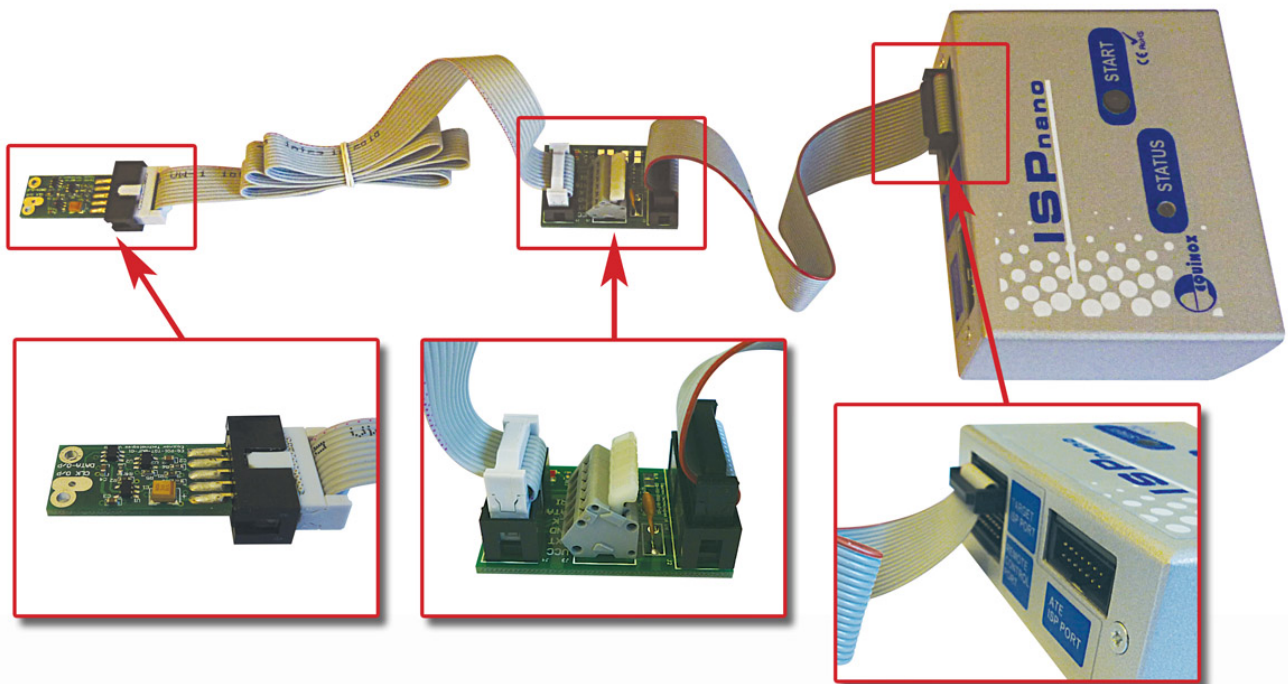
There is a known issue with XMEGA device with date code 2010 or later where they appear to be more susceptible to noise / clock skew on the **PDI_CLOCK** pin than earlier silicon versions. If long cables (>10cm) are used to connect the programmer to the XMEGA DUT, then in certain circumstances the programmer cannot physically enter PDI programming mode.

The only known solutions to this problem are as follows:

- Reduce the ISP cable length to <10cm or until the programmer can reliably enter PDI programming mode.
- or
- Use a '**Clock Buffer**' circuit to buffer the **PDI_CLOCK** signal at the target (DUT) end of the ISP cable.

1.2 XMEGA PDI Buffered ISP Cable

The solution to programming an XMEGA device over long cables is to buffer the **PDI_CLOCK** signal at the target (DUT) end of the ISP cable. The illustration below shows the typical cabling arrangements for the '**Clock Buffer Module**' when used with an ISPnano programmer.



Please note:

It is possible to use cable lengths of up to 2.0 meters between the programmer and the '**Clock Buffer Module**' using the cabling arrangement detailed in the diagram above.

1.3 XMEGA PDI Buffered ISP Cable - Features

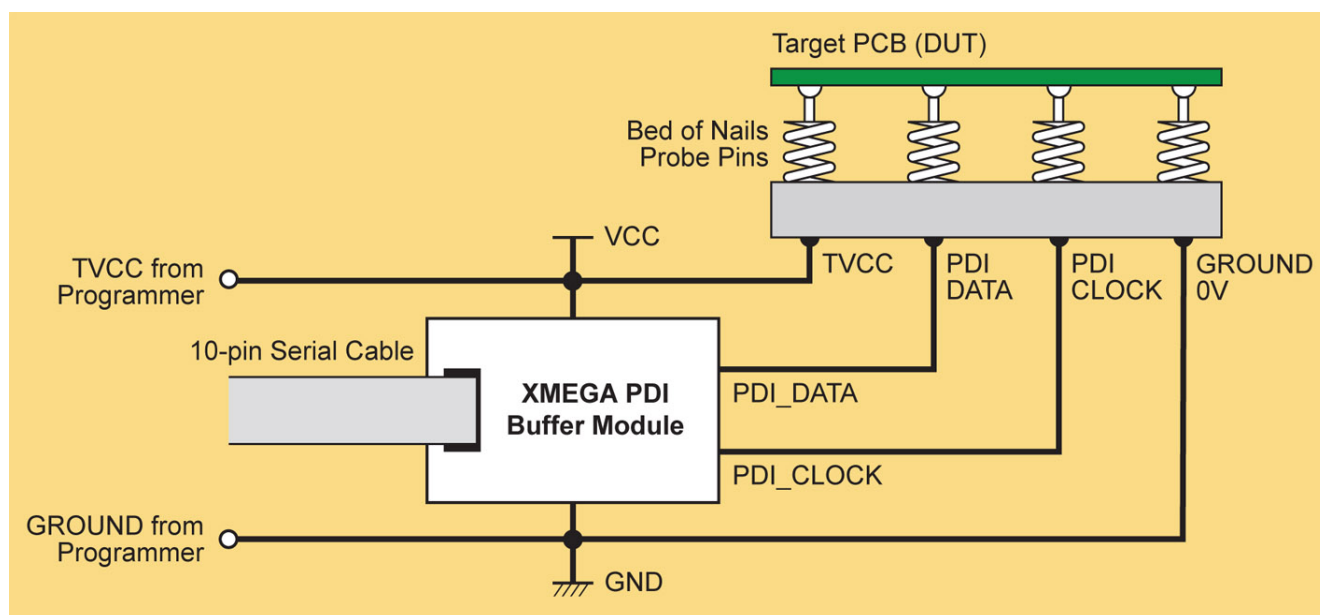
The '**Buffered ISP Cable**' features are as follows...

- Supports programming of Atmel XMEGA microcontrollers using ISP cable lengths of up to 2m.
- Simple to retrofit to any existing programming fixture.
- Buffers the **PDI_CLOCK** signal at target end of the ISP cable.
- Cross-talk between the **PDI_CLOCK** and **PDI_DATA** signals is minimised by running additional GROUND wires between the clock and data signals.
- An analogue switch on the output of the **PDI_CLOCK** buffer is used to isolate the programmer / buffer circuit from the DUT when not in programming mode.
- The buffer circuitry can be powered from the programmer or the Target System (DUT).
- Comprehensive ESD protection of the **PDI_CLOCK** and **PDI_DATA** signals.
- Simple integration with existing programming projects – only one change is required to the statemachine settings in order to control the analogue switch.

1.4 Recommended Clock Buffer module location

The '**Clock Buffer module**' must be inserted as close as possible to the XMEGA DUT. The module is designed so that it can be soldered directly to the bed-of-nails **PDI_CLOCK** probe pin in the fixture thereby minimising the distance from the buffered clock output to the target XMEGA clock input.

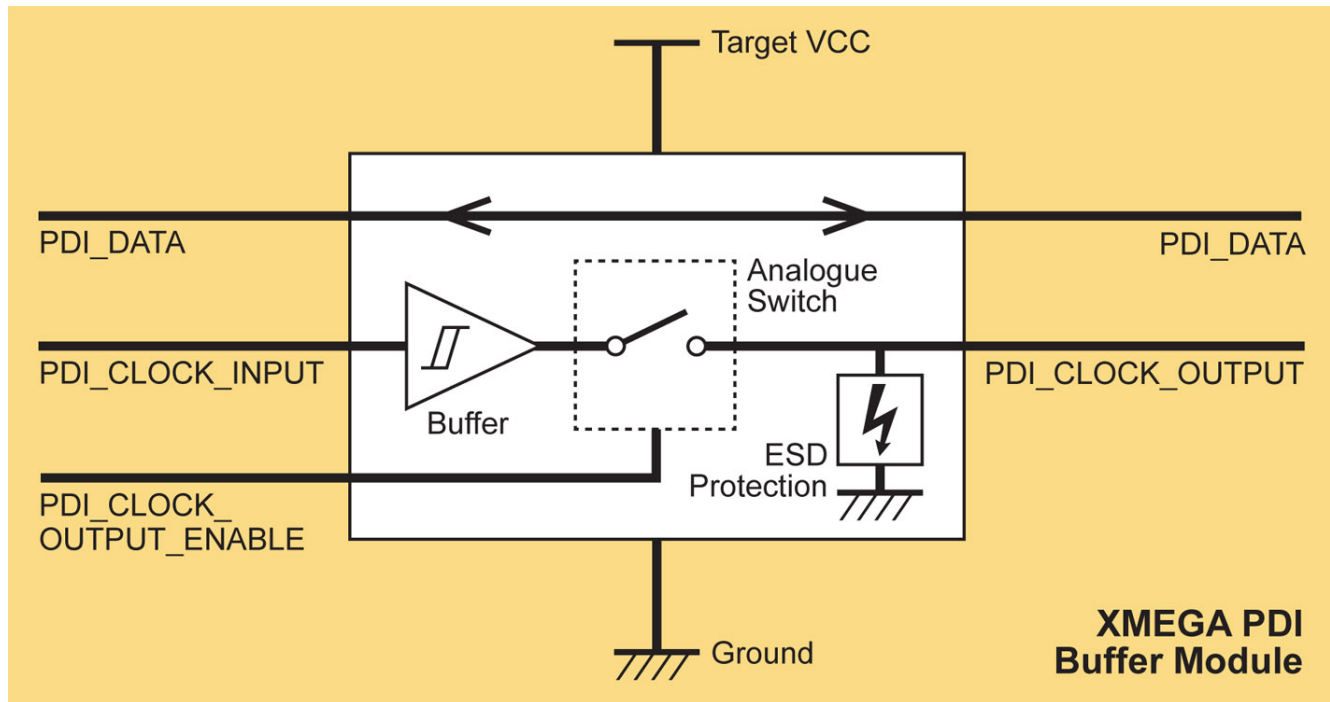
The diagram below shows the recommended location of the '**Clock Buffer module**' circuit inside a programming fixture...



- The **PDI_CLOCK** and **PDI_DATA** wiring between the '**Clock Buffer module**' and fixture probe-pins should be a maximum of e.g. 3 – 4 cm.
- The 10-way ISP cable between the '**Clock Buffer module**' and the programmer can be up to 3 meters in length.

1.5 Clock Buffer functional explanation

The diagram below shows the general operation of the '**Clock Buffer module**' circuit...



Important features:

- The **PDI_CLOCK** signal from the programmer is buffered so that the clock edges are square again at the output of the buffer.
- The output of the buffer is then fed through a low-on resistance analogue switch which allows the **PDI_CLOCK** signal to be tri-stated under programmer control when not being used for programming. This prevents the programmer from permanently driving the RESET pin of the target XMEGA controller. i.e. the pin is only driven during programming.
- The **PDI_CLOCK** signal is enabled by the programmer using the spare programmer output – OP5. The programmer must drive OP5 HIGH to enable the analogue switch output.
- The output of the analogue switch is protected by an ESD protection device to prevent damage due to ESD or over-voltage.

1.6 Clock Buffer Module – 10-way IDC pin-out

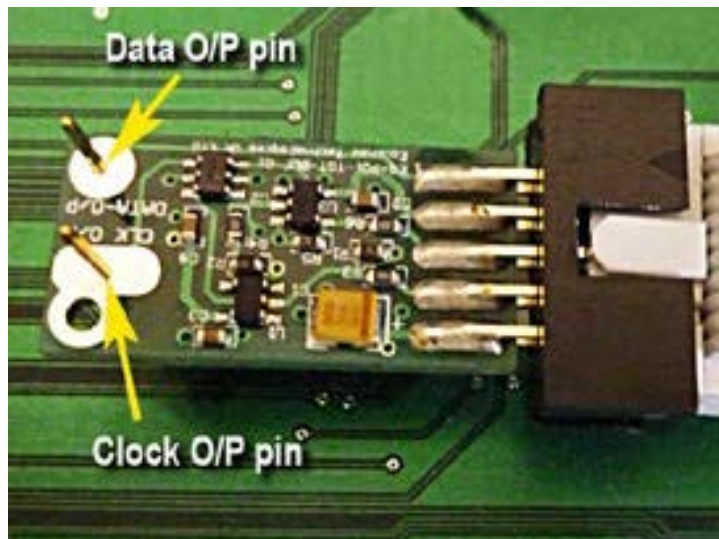
The 10-way IDC cable on the '**Clock Buffer Module**' brings the **PDI_CLOCK**, **PDI_DATA** and **CLOCK_BUFFER_ENABLE** signals from the programmer to the module. The simplest method to bring these signals up from the programmer is to use a 10-way ID cable made to the required length. However, it is also entirely possible to use separate wires to connect just the required signals from the programmer to the '**Clock Buffer Module**'.

The pin-out of the 10-way male IDC connector on the '**Clock Buffer Module**' is shown in the table below.

10-way IDC pin #	Signal name	Signal description	Direction from programmer	Pin name on target device
1	PROG_PDI_DATA	PDI Bi-directional DATA signal	Passive	PDI_DATA
2, 4, 6,8,10	GND	Target / Programmer Signal GROUND	Passive	GND
3,9	VCC	Target / Programmer VCC supply	Passive	VCC / VDD
5	CLOCK_BUFF_ENABLE	Clock Buffer ENABLE signal	Output	No connect
7	PROG_PDI_CLK	XMEGA PDI Clock signal from the programmer (not buffered)	Output	XMEGA - RESET

1.7 Clock Buffer Module – Outputs to the DUT

The outputs **PDI_CLOCK_OUTPUT** (buffered) and **PDI_DATA** (not buffered) of the '**Clock Buffer Module**' are brought out to two large pads on the module PCB.

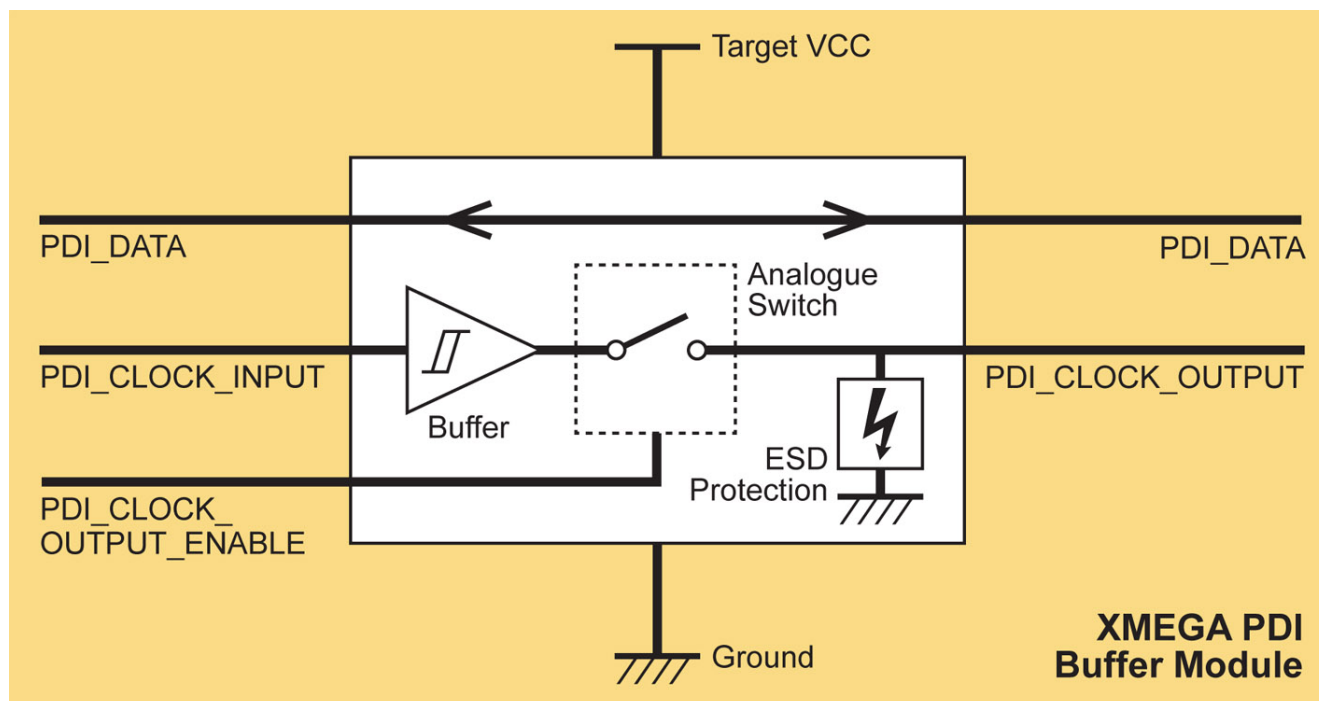


As the purpose of the module is to buffer the **PDI_CLOCK** signal, it is very important that the module is positioned as close as possible to the **PDI_CLOCK** test-pin on the DUT. The module has been designed so that a typical fixture pogo-pin will fit through the '**CLK-OP**' hole in PCB allowing the module to be mounted directly to the target **PDI_CLOCK** pin. If it is not possible to fix the module directly to the pogo-pin, then try to locate it within 2 – 3 cms of the pogo-pin and use a piece of wire to connect between the module and the pogo-pin.

10-way IDC pin #	Signal name	Signal description	Direction from module	Pin name on target device
DATA / OP	PROG_PDI_DATA	PDI Bi-directional DATA signal	Passive	PDI_DATA
CLK O/P	PDI_CLK_OUTPUT	XMEGA PDI Clock signal (buffered)	Output	XMEGA - RESET

1.8 Controlling the PDI Clock Buffered output

The output of the '**Clock Buffer**' is gated via a low-impedance analogue switch to the DUT. This allows the programmer to control when the PDI_CLOCK_OUTPUT signal is driving and or not.



The programmer controls the analogue switch using the control line **CLOCK_BUFF_ENABLE** on the module which is on pin 5 of the 10-way IDC connector. The **CLOCK_BUFF_ENABLE** signal is pulled low by default which disables the output drive.

The programmer uses the spare **output O/P5** to control the **CLOCK_BUFF_ENABLE** signal. It needs to drive **output O/P5** HIGH during any programming operation in order to enable the analogue switch which will then gate the programmer clock signal to the DUT.

The spare **output O/P5** can be found on pin 6 of the 16-way IDC '**Target ISP Port**' of any ISPnano programmer.

To enable the **output O/P5** during a programming operation, the pre-programming statemachine in the programming project must be altered to say '**LAH**' in the O/P5 column.

Appendix 6 – XMEGA PDI Design guidelines

1.0 Overview

This section provides an overview of suggested '*design guidelines*' which should be followed when designing a target board which is to be programmed using the Atmel 2-wire PDI interface.

1.1 XMEGA RESET circuit

- The XMEGA **RESET** pin is used as the **PDI_CLOCK** pin during the production programming process.
- The XMEGA RESET pin must therefore be **completely isolated** from any on-board '**reset circuit**' during programming. This includes isolating even simple C/R reset circuits as even a small amount of capacitance can upset the PDI clock signal.
- A simple jumper link or solder blob link can be used to isolate the XMEGA RESET pin during the programming process.

!!! Warning !!!

Failure to isolate any reset circuit during the programming process will prevent PDI programming from working properly.

1.2 XMEGA PDI - CLOCK signal line (RESET)

- The XMEGA **PDI CLOCK** signal (**PDI_CLOCK**) is a high-frequency signal line during the programming process and should therefore be routed with care.
- High-frequency PCB design guidelines should be followed when routing this signal.
- The signal line length should be as short as possible.
- The **PDI_CLOCK** should not be routed near other high-frequency signals especially the **PDI_DATA** signal. This is to minimise any chance of cross-talk between the PDI clock and data lines.

1.3 XMEGA PDI - DATA signal line

- The XMEGA **PDI DATA** signal (**PDI_DATA**) is a high-frequency signal line during the programming process and should therefore be routed with care.
- This signal carries high-speed bi-directional data between the programmer and the XMEGA DUT.
- This is a dedicated signal line only for PDI data. There must be no other components on this signal line.
- The **PDI_DATA** signal line should also be isolated from the **PDI_CLOCK** line to avoid any cross-talk from the PDI clock signal. This could possibly be achieved by applying good high-frequency PCB design rules, proper use of a ground plane and perhaps inserting a 0V track between the **PDI_DATA** and **PDI_CLOCK** signals on the PCB.

1.4 PDI programming cable length

- There are known PDI programming issues with many XMEGA devices when attempting to use long (>10 cm) ISP cables between the programmer and the XMEGA DUT.
- If longer ISP cables are necessary because of the programming fixture design then an external '**Clock Buffer**' circuit is required on the programming fixture to clean up the clock at the DUT end of the ISP cable.