

Report No:

AN132

Title:

## In-System Programming (ISP) of the Atmel ATtiny AVR FLASH Microcontroller Family using the TPI interface

Author:

John Marriott

Date:

24<sup>th</sup> October 2014

Version Number:

1.09



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

|  |           |
|--|-----------|
| <b>1.0 Introduction</b> .....  | <b>3</b>  |
| 1.1 Programmers supporting ATtiny TPI programming.....                       | 3         |
| 1.2 Programmer firmware for ATtiny TPI support .....                         | 4         |
| 1.3 ISPnano Series 3 - Equipment required for ATtiny TPI support .....       | 5         |
| 1.4 ATtiny – TPI - Device Support.....                                       | 6         |
| <b>2.0 ATtiny TPI Programming Interfaces</b> .....                           | <b>7</b>  |
| 2.1 Overview of the ATtiny TPI Programming Interface.....                    | 7         |
| 2.2 Low voltage / high voltage TPI programming.....                          | 7         |
| 2.3 Connecting an ATtiny TPI device to an ISPnano programmer.....            | 9         |
| 2.4 TPICLK – Clock Signal.....   | 10        |
| 2.5 TPIDATA – Data Signal .....  | 11        |
| 2.6 RESET/VPP pin .....  | 11        |
| 2.7 ISPnano Series 3 / 4 - Target ISP Port – TPI pin-out.....                | 12        |
| 2.8 Signal / Power GROUND (0V) connections.....                              | 13        |
| <b>3.0 Creating an EDS Development Project</b> .....                         | <b>14</b> |
| 3.1 Overview.....  | 14        |
| 3.2 Selecting a Development Project .....                                    | 14        |
| 3.3 Selecting ‘Programmer and Project Type’ .....                            | 15        |
| 3.4 Selecting ‘Target Device’.....   | 16        |
| 3.5 Selecting the ATtiny oscillator settings .....                           | 17        |
| 4.7 Setting up the Target Power Supply .....                                 | 18        |
| 4.8 Setting up the ATtiny ‘Erase Options’ .....                              | 19        |
| 4.9 Setting up the ATtiny ‘FLASH Programming’.....                           | 20        |
| <b>Appendix 1 – ISPnano-QC1 Quick Connect Module</b> .....                   | <b>22</b> |
| 1.0 Overview.....  | 22        |
| 1.1 Quick-Connect connector pin-out .....                                    | 23        |
| <b>Appendix 2 – Using ConsoleEDS to program ATtiny TPI devices</b> .....     | <b>25</b> |
| 1.0 Overview.....  | 25        |
| 1.1 Explanation of ConsoleEDS ‘Base Projects’.....                           | 25        |
| 1.2 Creating ATtiny low-voltage and high-voltage base projects .....         | 25        |
| 1.3 Setting up a ConsoleEDS ‘Base Project’ .....                             | 26        |
| 1.4 Reading the ‘Device Signature / ID’.....                                 | 26        |
| 1.5 Erasing the FLASH and Fuses.....   | 27        |
| 1.6 Programming the FLASH using the /FLASHWRITE command.....                 | 28        |
| 1.7 Programming the Fuses using the /FUSEWRITE command .....                 | 29        |
| 1.8 Reading the Fuses using the /FUSEREAD command.....                       | 29        |
| 1.9 Programming the ‘Security Fuses’ using the /SECURITYWRITE command.....   | 30        |
| 1.10 Reading the ‘Security Fuses’ using the /SECURITYREAD command .....      | 30        |
| 1.11 Executing a ‘Standalone Programming Project’.....                       | 31        |
| 1.12 Mixing a ‘Standalone project’ with individual programming commands..... | 31        |
| 1.13 Typical ATtiny programming sequence .....                               | 33        |
| 1.14 Disabling the ATtiny RESET pin.....                                     | 34        |
| 1.15 Resurrecting a device where the RESET pin is disabled .....             | 34        |

## 1.0 Introduction

This application note describes how to develop and implement In-System Programming (ISP) support for the **Atmel ATtiny AVR TPI microcontroller** family using the '**3-wire TPI**' 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 Microcontroller and also details the different ISP Header Connector pin-outs which are currently available.

### 1.1 Programmers supporting ATtiny TPI programming

Programming support for the **Atmel ATtiny AVR TPI microcontroller** family is currently only available on the '**ISPnano Series 3**' range of programmers.

The table below details the Equinox ISP programmers which are capable of supporting TPI programming of Atmel ATtiny AVR microcontrollers.

|   | Programmer  | Programming channels                | Low-voltage TPI algorithm  | High-voltage TPI algorithm |
|---|---|-------------------------------------|----------------------------|----------------------------|
|  | <b>ISPnano Series 3</b>   | 1<br>(network up to 32 programmers) | Yes<br>– available now     | Yes<br>– available now     |
|  | <b>ISPnano Series 3 ATE</b>                                     | 1<br>(network up to 32 programmers) | Yes<br>– available now     | Yes<br>– available now     |
|  | <b>ISPnano-MUX4</b><br>4-channel multiplexed programming system | 4 sequential programming channels   | Yes<br>– available now     | Yes<br>– available now     |
|  | <b>ISPnano-MUX8</b><br>8-channel multiplexed programming system | 8 sequential programming channels   | Yes<br>– available now     | Yes<br>– available now     |
|  | <b>ISPjuno</b><br>Portable ISP Programmer                       | 1                                   | Yes<br>– available Q2 2013 | Yes<br>– available Q2 2013 |

**Please note:**

- The '**ISPnano Series 3**' / '**ISPnano Series 4 ATE**' programmer range supports uploading of up to 64 independent '**Standalone Programming Projects**'.
- The **ATtiny TPI** 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 TPI to be implemented.

## 1.2 Programmer firmware for ATtiny TPI support

The minimum firmware version for programming Atmel ATtiny devices using the TPI interface is detailed in the table below.

| Programmer           | Minimum Firmware version | Comment                   |
|----------------------|--------------------------|---------------------------|
| ISPnano Series 3     | 6.29 and above           | Fully tested and released |
| ISPnano Series 3 ATE | 6.29 and above           | Fully tested and released |
| ISPnano Series 4 ATE | 6.67                     | Fully tested and released |
| ISPjuno              | TBA                      | No released yet           |

## 1.3 ISPnano Series 3 - Equipment required for ATtiny TPI support

The table below lists the equipment required to implement 'Atmel ATtiny TPI' programming on an 'ISPnano Series 3' or 'ISPnano Series 4' programmer.

| Equipment  | Picture / illustration  | Comment   |
|--|---|---|
| ISPnano Series 3 / 3 ATE / Series 4 ATE programmer |    | <ul style="list-style-type: none"> <li>This is the actual programming module.</li> </ul>  |
| ISPnano-QC1 Quick Connect Module                   |   | <ul style="list-style-type: none"> <li>This is a small connector module which plugs into the 16-way IDC connector on the ISPnano programmer.</li> <li>It features all the circuitry required for ATtiny TPI programming.</li> <li>All TPI connections are brought out to 'Quick Connect' connectors.</li> </ul> |
| ISPnano-UPG23                                      |  | <ul style="list-style-type: none"> <li><b>ATtiny TPI – License Upgrade</b></li> <li>This is an '<b>electronic programming license</b>' which enables the programmer to support programming of Atmel ATtiny devices via the TPI interface.</li> </ul>  |

### Please note:

The ISPnano Series 3 programmers feature an on-board 'TPI' port for programming Atmel ATtiny devices via the TPI interface. However, in order to use this port, a simple external circuit is required. This circuit is included on the 'ISPnano-QC1 - Quick Connect Module' and also the **CONMOD** module.

## 1.4 ATtiny – TPI - Device Support

The Equinox *'ISPnano Series 3'* programmer is capable of supporting any Atmel ATtiny AVR device which supports the *'3-wire Tiny Programming Interface (TPI)'* programming interface. The table below details the devices which are currently supported.

| Device   | On-chip FLASH Size (bytes) | FLASH Page Size (bytes) | Minimum firmware version | LV TPI algorithm | HV TPI algorithm | Tested on actual device |
|----------|----------------------------|-------------------------|--------------------------|------------------|------------------|-------------------------|
| ATtiny4  | 512                        | 128                     | 6.67                     | YES              | -                | -                       |
| ATtiny5  | 512                        | 128                     | 6.67                     | YES              | -                | -                       |
| ATtiny9  | 1024                       | 128                     | 6.67                     | YES              | -                | -                       |
| ATtiny10 | 1024                       | 128                     | 6.67                     | YES              | YES              | YES                     |
| ATtiny20 | 2048                       | 128                     | 6.67                     | YES              | -                | -                       |
| ATtiny40 | 4096                       | 128                     | 6.67                     | YES              | YES              | -                       |

### Please note:

Equinox will usually add any new ATtiny TPI device on request. The above list may therefore not be up-to-date. Please refer to the latest *'Device Support List'* for the devices which are currently supported by the Equinox range of programmers.

This can be found as follows:

- as a **Download** available on the website:  
Click on the *'Downloads'* tab. Under *'Download Type'* choose *Device Support Lists / Release notes* then click *Search*.
- Browsing on the **Device Support** tab under each product.
- In the latest version of **EQ-Tools**:  
Launch EQ-Tools and then select **<Programmer> <Create a Device Support list>**  
All programmers and devices supported are listed in this document.  
You will need the most recent EQ-Tools build version – please refer to the website for further details.

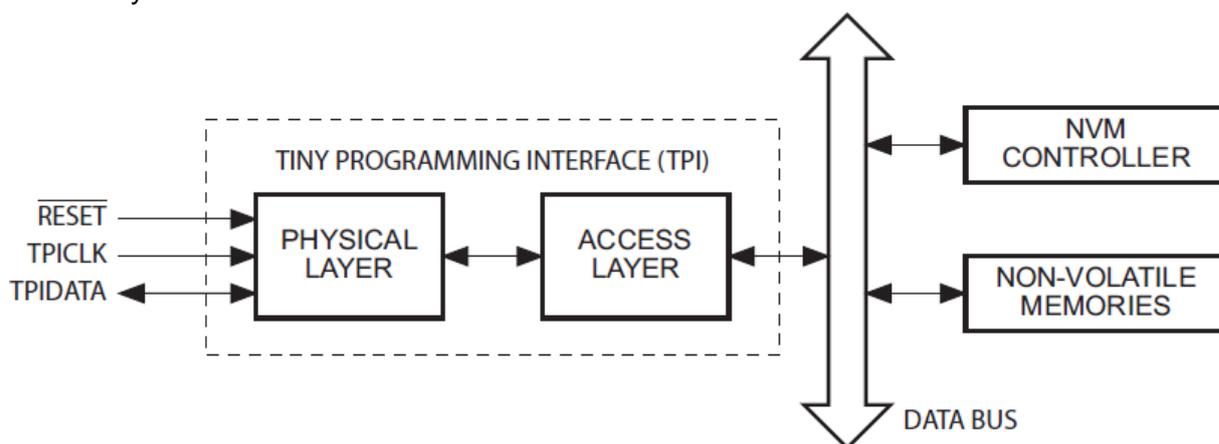
### Please note:

- Please see **Application Note – AN112** for instructions on updating your programmer firmware.

## 2.0 ATtiny TPI Programming Interfaces

### 2.1 Overview of the ATtiny TPI Programming Interface

The devices in the *Atmel ATtiny (TPI) AVR microcontroller family* can only be programmed using the 3-wire '*Tiny Programming Interface (TPI)*' interface. The *Tiny Programming Interface (TPI)* is an Atmel proprietary interface for external programming and on-chip debugging of selected Atmel ATtiny AVR microcontrollers. The illustration below shows how the TPI interface is implemented within an ATtiny AVR device.



The TPI interface is an ideal choice for these low pin-count microcontrollers as only two programming pins and RESET are required to implement In-System Programming. The TPI interface supports high-speed programming of all on-chip Non-Volatile Memory (NVM) spaces including the Flash, Fuses and Lock bits.

### 2.2 Low voltage / high voltage TPI programming

An ATtiny device can normally be programmed at +5.0V using the so called '*low voltage TPI*' programming algorithm. The device must be powered at +5.0V during programming otherwise the programming operation will fail.

However, if the '*RESET Disable Fuse (RSTDISBL)*' has been previously enabled (set to '0'), then the target device will no longer respond to the programmer using the '*low-voltage TPI*' programming algorithm. To re-program an ATtiny device in this state, it is necessary to use the '*high-voltage TPI*' programming algorithm which applies +12V to the *RESET/VPP* pin of the device.

The table below compares the '*low-voltage TPI*' and '*high-voltage TPI*' programming algorithms:

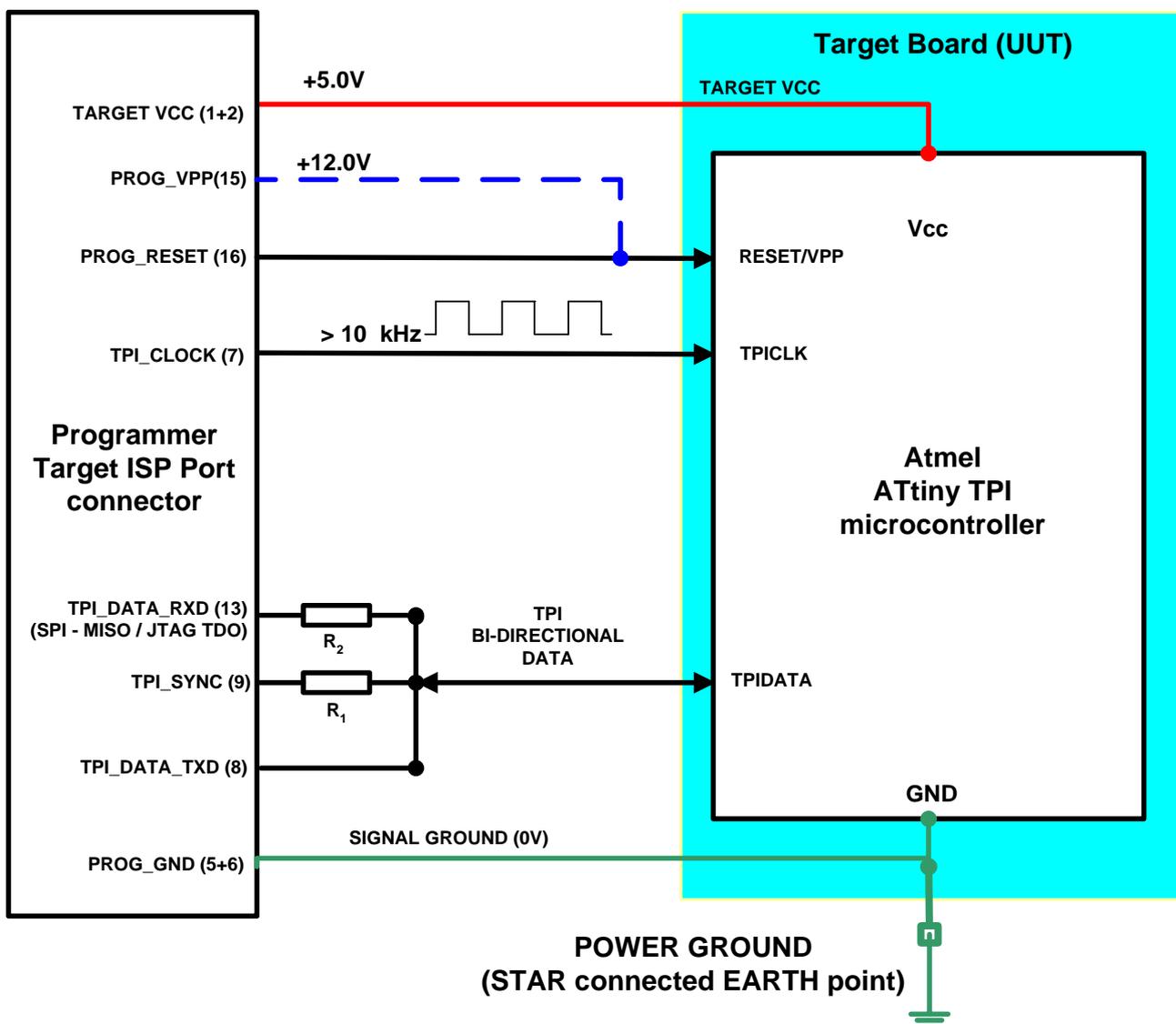
| Interface Name | Interface description                  | Number of interface signals | Vcc Voltage | Vpp Voltage  | ATtiny pins required for programming   |
|----------------|--|-----------------------------|-------------|--------------|--|
| LV TPI         | Low-voltage TPI programming interface  | 3-wire                      | +5.0V       | Not required | <ul style="list-style-type: none"> <li>• TPI_CLK</li> <li>• TPI_DATA</li> <li>• RESET</li> </ul>     |
| HV TPI         | High-voltage TPI programming interface | 3-wire                      | +5.0V       | +12.0V       | <ul style="list-style-type: none"> <li>• TPI_CLK</li> <li>• TPI_DATA</li> <li>• RESET/VPP</li> </ul> |

### Important notes:

- The '**Target Vcc Voltage**' must be set to +5.0V during programming otherwise programming operations will fail. This voltage can be generated by the programmer or it can be supplied by the Target System.
- The '**Target Vpp Voltage**' is generated by the programmer and applied to the **RESET/VPP** pin of the target device. The Target System (UUT) must be designed so that it can withstand +12V on the **RESET/VPP** pin.

## 2.3 Connecting an ATtiny TPI device to an ISPnano programmer

The diagram below shows the connections required between an '*ISPnano Series 3*' or '*ISPnano Series 4*' programmer and a Target Board (UUT) for programming a single ATtiny microcontroller using the TPI interface.



### Please note:

- The above circuit is already implemented on both the '*ISPnano CONMOD module*' and the '*ISPnano Quick Connect module*' so it is recommended that one of these modules is used for any production fixture.
- The **TPI\_SYNC** signal (pin 9 on the Target ISP connector) should be connected to the **ATtiny TPIDATA** pin via a resistor –  $R_1$ . The value of  $R_1$  should be 470 ohms. This pin is used to force the ATtiny device into TPI programming mode.
- The **TPI\_DATA\_RXD** pin should be connected to the ATtiny **TPIDATA** pin via a resistor –  $R_2$ . The value of  $R_2$  should be 470 ohms.
- The **RESET** line from the programmer should be connected to the ATtiny **RESET/VPP** pin (unless you are using some sort of external reset circuit).

- If the '**high-voltage TPI**' algorithm is to be used, then the '**PROG\_VPP**' pin must also be connected to the ATtiny **RESET/VPP** pin. The Target System must be able to withstand +12.0V applied to this pin!!!
- 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).

The pins required for programming via the TPI interface are detailed in the table below.

| Programmer Signal name (16-way IDC) | IDC pin | Signal description                | Direction from programmer | Pin name on ATtiny device |
|-------------------------------------|---------|-----------------------------------|---------------------------|---------------------------|
| TARGET_VCC                          | 1+2     | Target Vcc Supply                 | Passive                   | VCC                       |
| GROUND (0V)                         | 5+6     | Target / Programmer Signal GROUND | Passive                   | GND                       |
| TPI_CLOCK                           | 7       | TPI Clock Signal                  | Output                    | RESET                     |
| TPI_DATA_TXD                        | 8       | TPI Data Signal - TRANSMIT        | Output                    | TEST                      |
| TPI_SYNC                            | 9       | TPI Synchronisation Signal        | Output                    | TPIDATA                   |
| TPI_DATA_RXD                        | 13      | TPI Data Signal - RECEIVE         | Input                     | TPIDATA                   |
| PROG_VPP                            | 15      | Programmer Vpp Voltage Output     | Output                    | RESET/VPP                 |
| PROG_RESET                          | 16      | Programmer RESET                  | Output                    | RESET/VPP                 |

## 2.4 TPICLK – Clock Signal

The '**TPI – Clock**' is a clock signal which is continuously generated by the programmer during TPI programming. This clock is fed from the programmer into the **TPICLK** pin of the target ATtiny device. The clock signal must be a continuous waveform with a frequency  $\geq 10$  kHz, otherwise the target ATtiny device will exit TPI programming mode.

### Important notes:

- As the ATtiny **TPICLK** pin is being used as a high-speed clock pin during TPI programming / debugging, it is therefore very important that this pin is free to oscillate without any external capacitive or resistive loading.

## Recommendations:

- It is recommended that no other components are connected to the **TPICLK** pin of the target ATtiny device during TPI programming.

## 2.5 TPIDATA – Data Signal

The **TPIDATA** signal is a bi-directional signal line which is used to transfer data between the programmer and the target ATtiny device and vice-versa. This is a dedicated pin for TPI programmer and should not be used for any other purposes except TPI programming. When TPI programming mode is first entered, the programmer automatically becomes the '**master**' on the TPI bus and drives the **TPIDATA** signal line. As part of the TPI protocol, the programmer can then instruct the target ATtiny device to transmit data back to the programmer. In order to achieve this, the programmer must reverse the direction of the **TPIDATA** signal so that the ATtiny 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 **TPIDATA** pin.
- This pin must be dedicated for TPI data transfer.

## 2.6 RESET/VPP pin

The ATtiny **RESET/VPP** pin is used to force the target device into programming mode.

### *i. Low-voltage TPI*

When using the '**low-voltage TPI**' algorithm, the programmer applies 0V to the ATtiny **RESET/VPP** pin to force the device into programming mode. When not in programming mode, the programmer applies +5.0V to the **RESET/VPP** pin.

### *ii. High-voltage TPI*

When using the '**high-voltage TPI**' algorithm, the programmer must apply 12.0V to the ATtiny **RESET/VPP** pin to force the device into programming mode. When not in programming mode, the programmer applies +5.0V to the **RESET/VPP** pin.

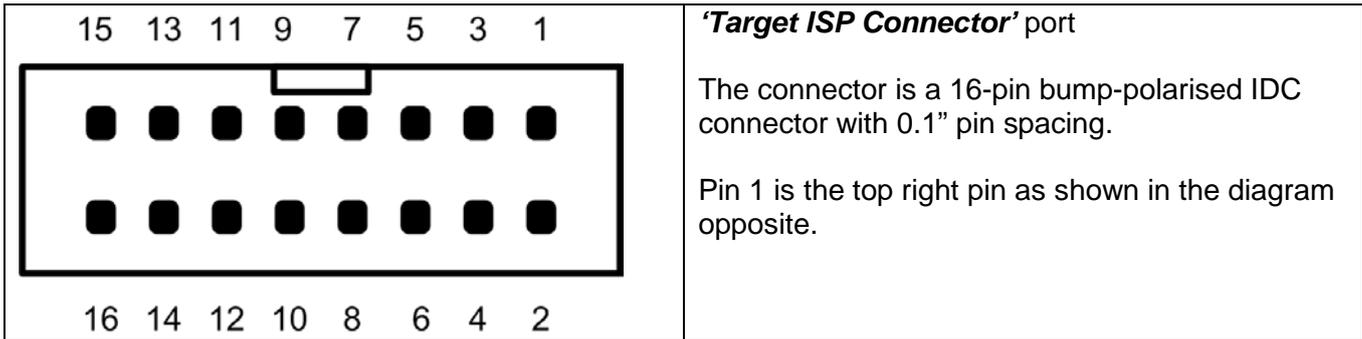
## Warning!

The Target System must be specifically designed to allow +12.0V to be applied to the ATtiny **RESET/VPP** pin otherwise the target device may be damaged.

## 2.7 ISPnano Series 3 / 4 - Target ISP Port – TPI 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 ATtiny device using the '**3-wire TPI**' 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 '**TPI Interface**' are detailed in the table below.

| Programmer Signal name (16-way IDC) | IDC pin | Signal description                | Direction from programmer | Pin name on ATtiny device |
|-------------------------------------|---------|-----------------------------------|---------------------------|---------------------------|
| TARGET_VCC                          | 1+2     | Target Vcc Supply                 | Passive                   | VCC                       |
| GROUND (0V)                         | 5+6     | Target / Programmer Signal GROUND | Passive                   | GND                       |
| TPI_CLOCK                           | 7       | TPI Clock Signal                  | Output                    | TPICLK                    |
| TPI_DATA_TXD                        | 8       | TPI Data Signal - TRANSMIT        | Output                    | TPIDATA                   |
| TPI_SYNC                            | 9       | TPI Synchronisation Signal        | Output                    | TPIDATA                   |
| TPI_DATA_RXD                        | 13      | TPI Data Signal - RECEIVE         | Input                     | TPIDATA                   |
| PROG_VPP                            | 15      | Programmer Vpp Voltage Output     | Output                    | RESET/VPP                 |
| PROG_RESET                          | 16      | Programmer RESET                  | Output                    | RESET/VPP                 |

## 2.8 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.

## 3.0 Creating an EDS Development Project

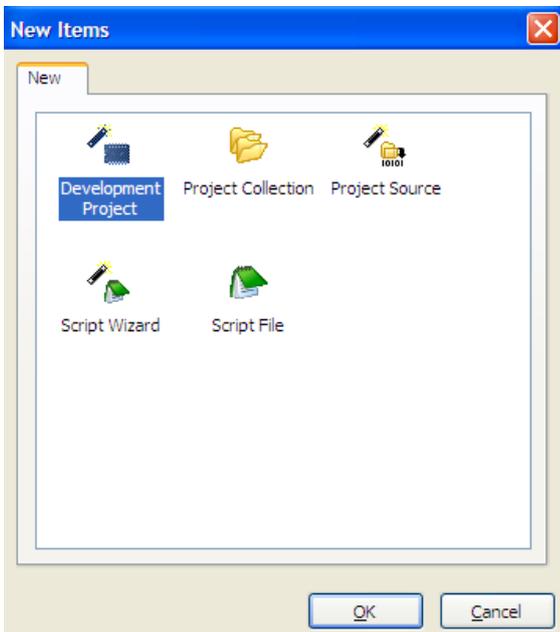
### 3.1 Overview

This section describes the steps required to create an **'EDS – Development Project'** which can be used to program an ATtiny device under PC control.

### 3.2 Selecting a Development Project

To create a **'Development Project'**...

- Launch EQTools
- Click the **'New'** icon
- The **'New items'** screen is displayed.



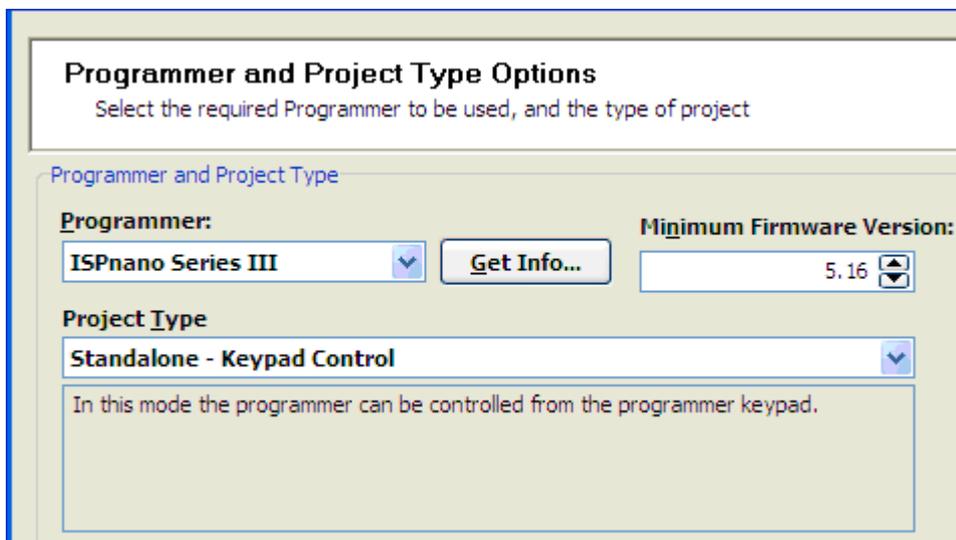
- Select **the 'Development Project'** icon and then click the <OK> button  
→ The **'Equinox Development Suite (EDS)'** wizard will now start.



## 3.3 Selecting 'Programmer and Project Type'

This screen allows you to select:

- The '**Programmer**'
- The '**Project Type**'



### i. Programmer

- Select the '**Programmer**' for which the project is to be compiled for.
- If the programmer is attached to the PC, you can simply click the **<Get Info>** button to automatically select the correct programmer.

### ii. Project Type

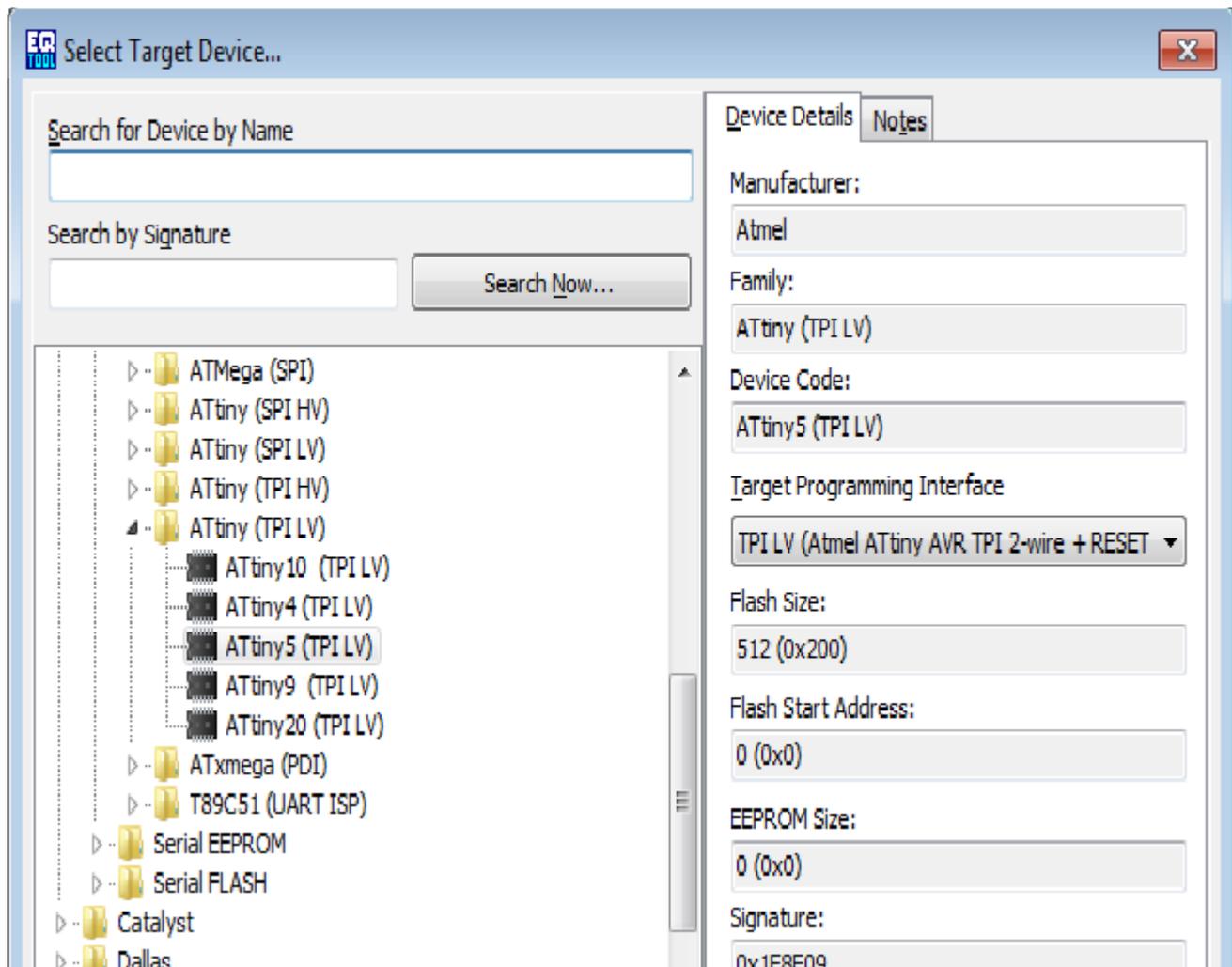
- The '**Project Type**' should be set to '**Standalone – keypad control**' for all standalone projects and ConsoleEDS projects.

## 3.4 Selecting 'Target Device'

This screen allows you to select the '**Target Device**' to be programmed.

The simplest way to find a particular device is as follows:

- Type the '**Device Name / code**' into the '**Search for device**' field e.g. **ATtiny5**
- Click the **<Search now>** button
- All instances of the **ATtiny5** device are now displayed



- Select the required device from the drop-down list and then click the **<OK>** button.

### Important note:

- The ATtiny5 device can be programmed in either '**Low-voltage TPI LV**' mode at +5V or using the '**High voltage TPI HV**' mode which applies +12V to the RESET pin of the target device.
- Please take care to select the correct device / programming mode which matches your target system. See section 2.2 for further details.

Once the device has been selected, the following screen will be displayed...



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

**Selected Device**

**Manufacturer:** Atmel  
**Family:** ATtiny (TPI LV)  
**Device Code:** ATtiny5 (TPI LV) Select Device...

Device Algorithm Version: 1.02R  
Silicon Revision: 0  
Signature: 0x1E8F09

Flash Size: 512 (0x200)  
Flash Page: 128

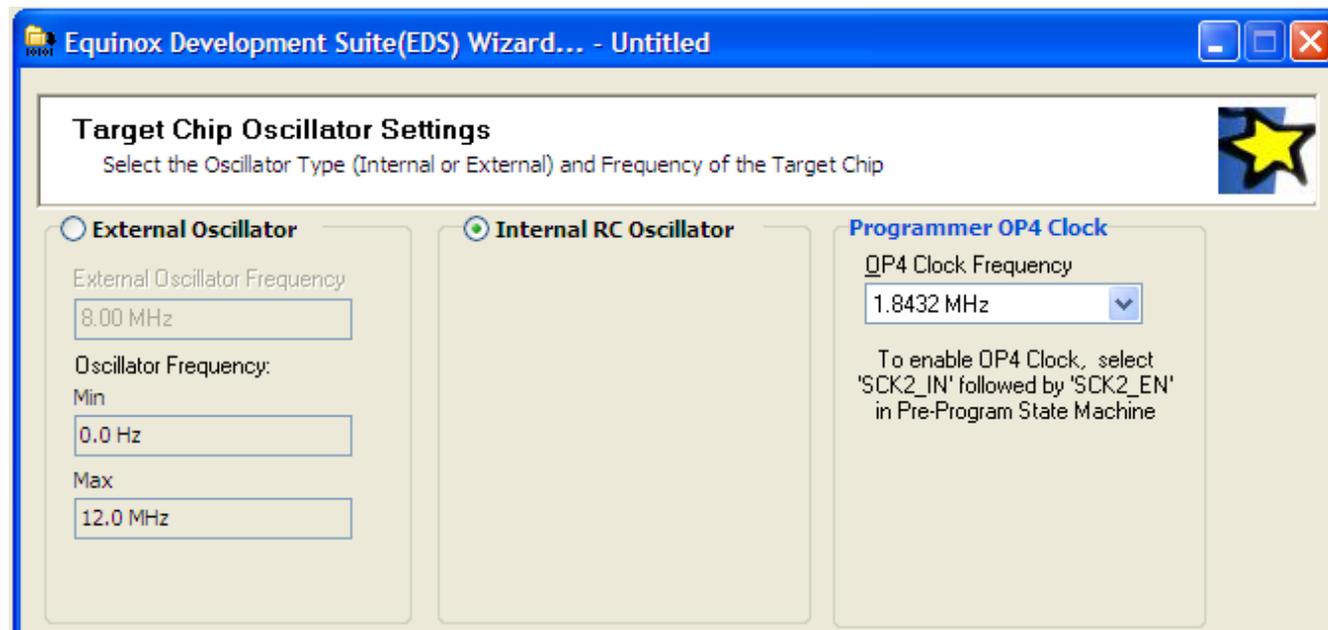
Target Programming Interface: TPI LV (Atmel ATtiny AVR TPI 2-wire + RESET)  
Polling Method: Conventional BYTE polling

Signature 1: 0x1E8F09  
 Signature 2  
 Signature 3

Click **<Next>**

## 3.5 Selecting the ATtiny oscillator settings

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



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

External Oscillator  
 Internal RC Oscillator

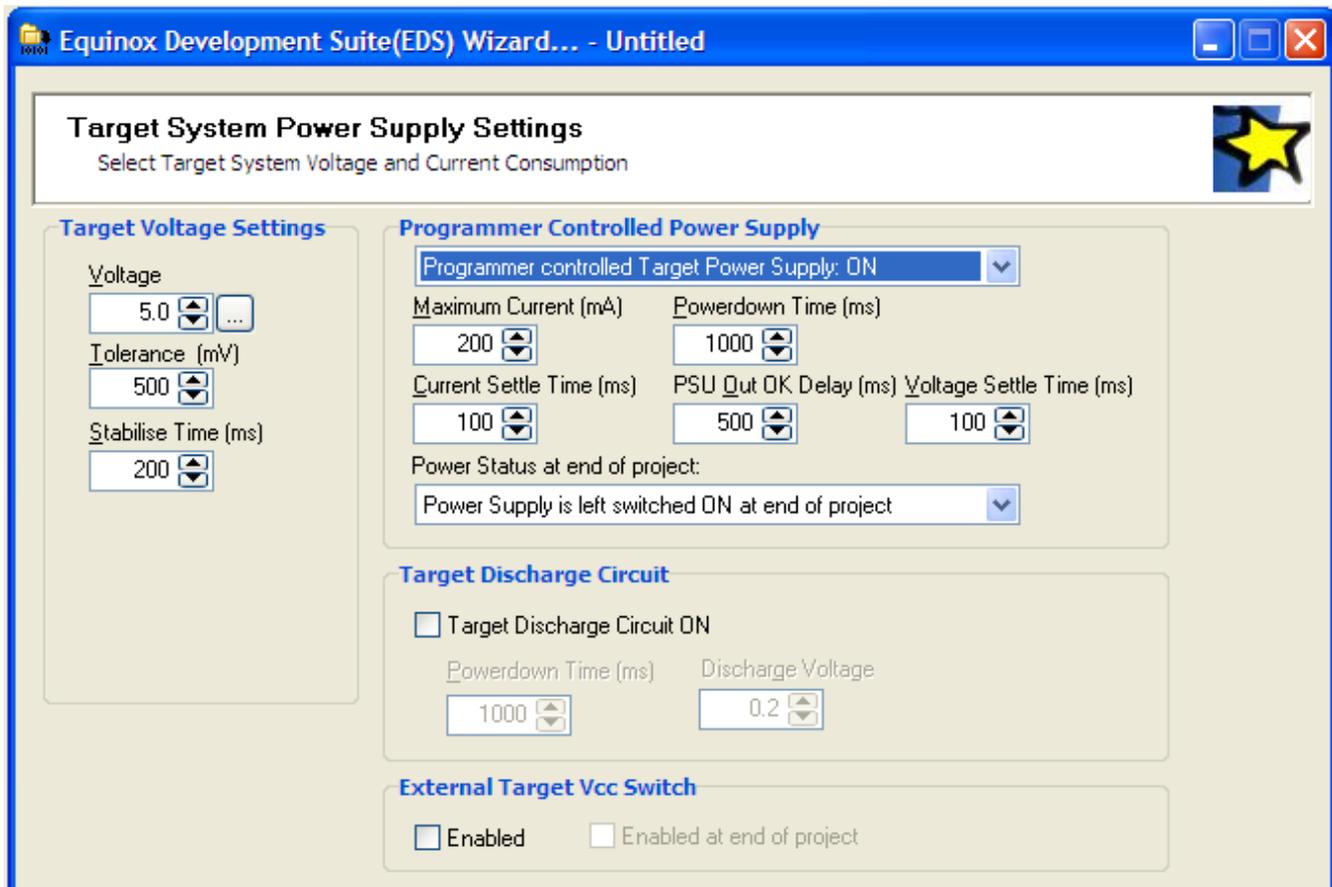
**External Oscillator**  
External Oscillator Frequency: 8.00 MHz  
Oscillator Frequency:  
Min: 0.0 Hz  
Max: 12.0 MHz

**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

- Leave the oscillator source as 'Internal oscillator'. 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 ATtiny TPI programming, the programmer must power the **'Target System'** at 5.0V.

### Important note:

The programming will not work below 5.0V. This is a limitation of the ATtiny devices for programming purposes only. It is possible to run the ATtiny devices at less than 5.0V after programming them.

### To power the Target System from the programmer:

- Set the **'Programmer controlled Target Power Supply'** to **'ON'**
- Set the **'Target Voltage'** to 5.0V

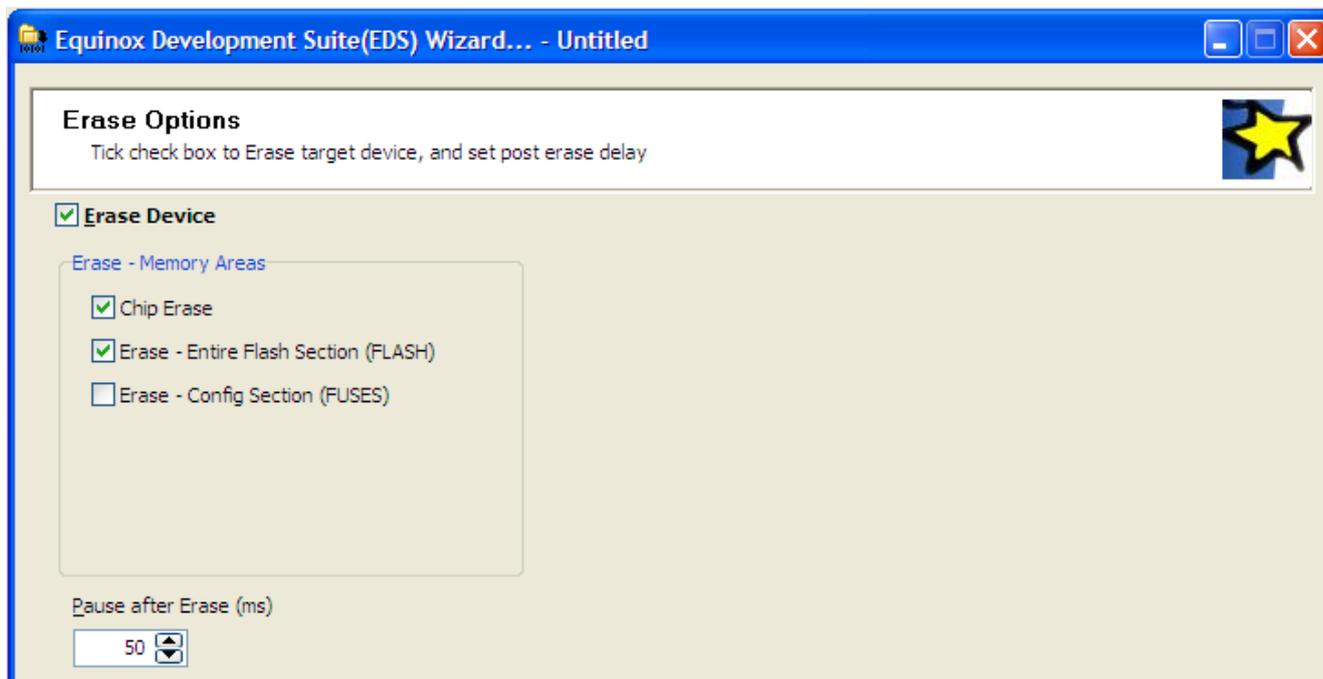
### To power the Target System from an external +5V supply:

- Set the **'Programmer controlled Target Power Supply'** to **'OFF'**
- Set the **'Target Voltage'** to 5.0V

Click **<Next>** → the **<Erase options>** screen is displayed.

## 4.8 Setting up the ATtiny 'Erase Options'

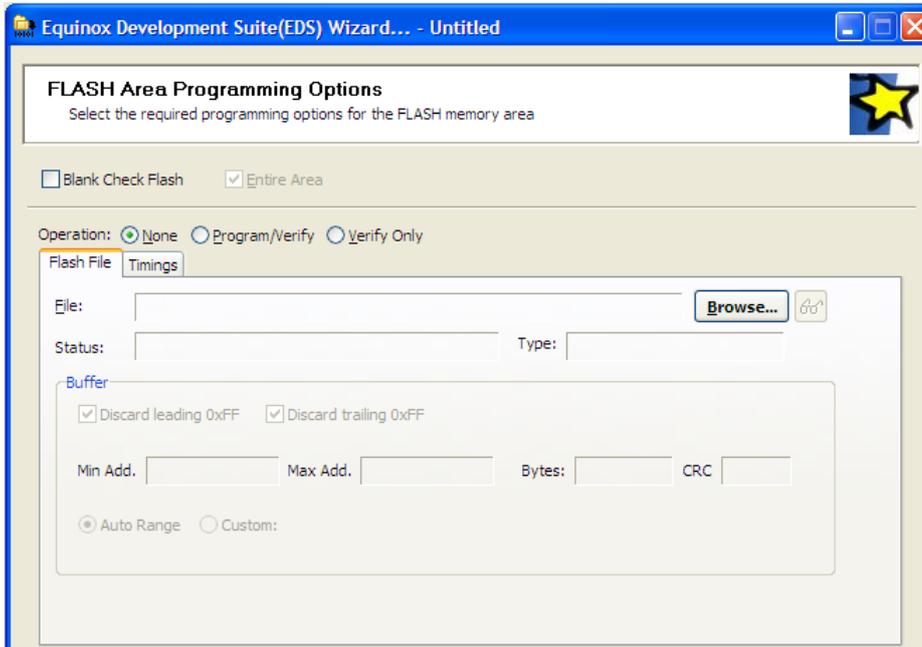
This screen allows you to set up how programmer handles the erasing of an ATtiny device.



- By default, the programmer will perform a '**Chip Erase**' operation which will erase the entire FLASH area but not the '**Config section**' where the fuses are stored.
  - If you plan to re-program the '**Configuration fuses**' in your project, then you need to select '**Erase – Config Section (Fuses)**' in the **Erase Options** menu.
- Click <Next> → the <FLASH Area Programming options> screen is displayed.

## 4.9 Setting up the ATtiny 'FLASH Programming'

This screen allows you to set up how to program the FLASH area of the ATtiny device.



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

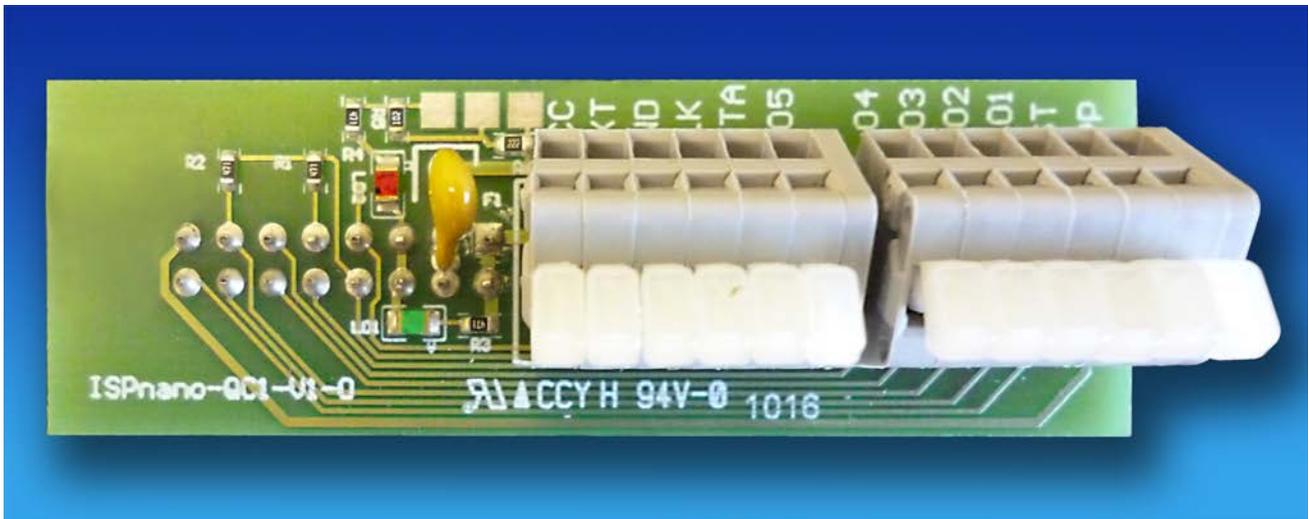
- Click the **<Browse>** button
  - Select the required input file e.g. \*.bin, \*.hex, \*.SREC
- The start and end address of the input file will be displayed.
- The '**Operation**' is automatically set to '**Program/Verify**' which means the programmer will program each page and then verify each page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



## Appendix 1 – 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 ATtiny AVR programming via the 3-wire TPI programming interface. The module plugs into the 16-way '*Target ISP Port*' on the '*ISPnano Series 3*' programmer or '*Series 3 ATE*' programmer. It provides so-called '*Quick connect*' connections allowing both an '*ATtiny TPI*' 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*' programmer.
- Features all circuitry required for programming an Atmel ATtiny AVR microcontroller via the '*ATtiny TPI*' programming 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            | ATtiny TPI Clock   | Output                    | TPICLK   |
| 5      | DATA           | ATtiny TPI Data  | Output                    | TPIDATA  |
| 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"> <li>• SPI – SCK</li> <li>• JTAG – TCK</li> </ul>  | Output                    | <ul style="list-style-type: none"> <li>• SPI – SCK</li> <li>• JTAG - TCK</li> </ul>  |
| 9      | I/O2           | <ul style="list-style-type: none"> <li>• SPI – MISO</li> <li>• JTAG – TDO</li> </ul> | Input                     | <ul style="list-style-type: none"> <li>• SPI – MISO</li> <li>• JTAG - TDO</li> </ul> |
| 10     | I/O1           | <ul style="list-style-type: none"> <li>• SPI – MOSI</li> <li>• JTAG – TDI</li> </ul> | Output                    | <ul style="list-style-type: none"> <li>• SPI – MOSI</li> <li>• JTAG - TDI</li> </ul> |
| 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 Atiny '**TPI Clock**' and '**TPI Data**' signals are marked as '**CLK**' and '**DATA**' respectively and should be connected to the target ATtiny TPICLK pin and TPIDATA pins.

### 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 be connected to the target ATtiny RESET pin

### Note 4

The VPP pin outputs a programmable voltage (+12.0V) which is used for the '**high-voltage TPI mode**'. If the Vpp voltage is required, then the **PROG\_VPP** pin of the programmer must be connected to the **RESET/VPP** pin of the target ATtiny device.

## Appendix 2 – Using ConsoleEDS to program ATtiny TPI devices

### 1.0 Overview

This appendix describes how to use the ConsoleEDS utility to program Atmel ATtiny microcontrollers via the TPI programming interface. It is possible to use ConsoleEDS to program the FLASH, Configuration Fuses and Security Fuses of an ATtiny 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 ATtiny device. For further information about how to use the ConsoleEDS application, 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. ATtiny10
- Target Programming Interface e.g. Low-voltage TPI
- Target Vcc Voltage e.g. +5V
- Target Vpp Voltage e.g. +12V
- Target Power Supply characteristics e.g. Current
- Target Programming speed e.g. 115 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 Creating ATtiny low-voltage and high-voltage base projects

As ATtiny TPI devices can be programmed in both **‘Low-voltage (TPI LV)’** and **‘High voltage (TPI HV)’** modes, it is recommended that two different **‘Base Projects’** are created, one for each mode. This makes it very simple to then switch between modes in the ConsoleEDS environment.

For example, the following Base Projects could be created for the ATtiny10 device.....

#### **i. ATtiny10 – low-voltage mode - ATtiny10LV.prj**

- Select ATtiny10 (LV TPI) device

#### **ii. ATtiny10 – high-voltage mode - ATtiny10HV.prj**

- Select ATtiny10 (HV TPI) device

The table below shows how the two '**Base Projects**' should be set up....

| Base Project name | Device name       | Programming mode | Vcc voltage (V) | Vpp Voltage on RESET pin (V) | Can always enter programming mode?           |
|-------------------|-------------------|------------------|-----------------|------------------------------|--|
| ATtiny10LV        | ATtiny10 (TPI LV) | Low-voltage TPI  | +5.0            | +5.0                         | No – will fail if RSTDIBL fuse is set to '0' |
| ATtiny10HV        | ATtiny10 (TPI HV) | High-voltage TPI | +5.0            | +12.0                        | Yes – will always work                       |

## 1.3 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. **ATtiny10 (TPI LV)**
- 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. ATtiny10LV.prj
- You have now created a '**Base Project**'.

## 1.4 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 usage:**

**ConsoleEDS ATtiny10LV.PRJ /READSIG**

**Typical response:**

**Console EDS - Signature read back: 0x1E9003**

**Possible errors:**

If ConsoleEDS reports '**Error 3039 - Failed to Enter Programming Mode, tried 1 times!**' then it may be that the target device has previously had the **RSTDISBL** fuse set to '0' which has disabled the RESET pin. This means that using the '**Low voltage programming mode**' will always fail to enter programming mode.

*Application Note 132 - In-System Programming (ISP) of the Atmel ATtiny AVR FLASH Microcontroller Family using the TPI Interface* 26

programming mode. The only way around this error is to use the **'High voltage programming mode'** to re-enable the RESET pin.

## 1.5 Erasing the FLASH and Fuses

The ATtiny 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 ATtiny10LV.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.
- iii. The erase operation could also erase the **'Configuration fuses'** back to the value 0xFF if this option has been selected in the **'Base Project'**.

## 1.6 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 ATtiny10LV.PRJ /ERASE /FLASHWRITE=FLASH_DATA.HEX /OFFSET=0x4000
```

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. The **/OFFSET** command is required as the ATtiny FLASH actually starts at address 0x4000.

### 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 ATtiny 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 ATtiny10LV.PRJ /ERASE
```

→ This will erase the entire FLASH area.

### iii. FLASH offset start address

The internal FLASH memory of an ATtiny device does not start at address 0x00000.

It actually starts at 0x4000 (see ATtiny memory map) so it is necessary to use the **/OFFSET=0x4000** command to program the FLASH using ConsoleEDS.

If the FLASH area to be programmed is already erased to 0xFF:

```
ConsoleEDS ATtiny10LV.PRJ /FLASHWRITE=FLASH_DATA.HEX /OFFSET=0x4000
```

## 1.7 Programming the Fuses using the /FUSEWRITE command

The simplest method of programming the '**Configuration fuses**' of an ATtiny 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. It also performs a 'Fuse verify' after programming the fuses.

### Typical example:

**ConsoleEDS ATtiny10LV.PRJ /FUSEWRITE=0xFF**

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

### Important notes:

1. The new '**Configuration fuses**' settings are not actually latched into the device (do not become active) until the ATtiny device has exited programming mode, powered down to 0V and then powered back up to Vcc again.
2. If power is left on the device or the device is able to 'phantom power' from one of its I/O pins, then the '**Configuration fuses**' settings may not become active. A complete power-cycle is required before the fuses become active.

### Warning!

If you program the '**RSTDIBL – RESET Disable**' fuse to a '**0**' then the ATtiny device will disable its RESET pin. This means that the programmer will no longer be able to enter programming mode using '**Low voltage**' programming mode.

## 1.8 Reading the Fuses using the /FUSEREAD command

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

### Typical example:

**ConsoleEDS ATtiny10LV.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 ATtiny 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 ATtiny10LV.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

### Warning!

Once you have locked the device (set both Lock Bits to 0), then you can no longer read / write the FLASH memory or change the value of the ‘*Configuration fuses*’ or the ‘*Security fuses*’. The only way to then re-program the device is to perform a ‘Chip Erase’ operation.

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

The ‘*hex value*’ of the ‘*Security fuses*’ of an ATtiny 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 ATtiny10HV.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 ATtiny10LV.PRJ /AUTOPROGRAM=MAINPROG  
/FLASHWRITE=SerialNumber.HEX /FUSEWRITE=0xFE /SECURITYWRITE=0xFE  
/OFFSET=0x4000***

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

## 1.13 Typical ATtiny 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 ATtiny10LV.PRJ /ERASE /FLASHWRITE=FLASH_DATA.HEX /FUSEWRITE=0xFF  
/SECURITYWRITE=0xFE /OFFSET=0x4000
```

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

- 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 '**FLASH\_DATA.HEX**' into the FLASH area
- Programs and verifies the '**Configuration fuses**'
- Programs and verifies the '**Security fuses**' (Lock Bits)
- Exits '**low-voltage**' TPI programming mode
- Power down the UUT

### Important notes:

3. The new '**Configuration fuses**' settings are not actually latched into the device (do not become active) until the ATtiny device has exited programming mode, powered down to 0V and then powered back up to Vcc again.
4. If power is left on the device or the device is able to 'phantom power' from one of its I/O pins, then the '**Configuration fuses**' settings may not become active. A complete power-cycle is required before the fuses become active.
5. 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.

### Warning!

If you program the fuses to 0xFE then this will program the '**RSTDIBL – RESET Disable**' fuse to a '**0**'. This forces the ATtiny device to disable its RESET pin. This means that the programmer will no longer be able to enter programming mode using 'Low voltage' programming mode. The **/SECURITYWRITE** command will still work as long as the ATtiny device has not yet exited programming mode.

## 1.14 Disabling the ATtiny RESET pin

If you are using the '**Low voltage**' programming mode to program the ATtiny device and want to disable the RESET pin, then this must be done as the final programming action. The ATtiny device does not always latch in the fuses until it has exited programming mode and been completely powered down and back up again.

### i. To disable the RESET pin in '**Low voltage**' programming mode...

[ConsoleEDS ATtiny10LV.PRJ /FUSEWRITE=0xFE](#)

→ The RESET pin is disabled and the ATtiny device will no longer communicate with the programmer.

### ii. To disable the RESET pin in '**Low voltage**' programming mode and also lock the device....

[ConsoleEDS ATtiny10LV.PRJ /FUSEWRITE=0xFE /SECURITYWRITE=0xFE](#)

→ The programmer programs the **RSTDISBL** fuse to '0' but then stays in programming mode and programs the 'Security fuses' which locks the device. It is only when the programmer exits programming mode that the device latches in the new fuse settings. At this point the ATtiny device will no longer communicate with the programmer.

### iii. Checking that the RESET pin has been disabled

The following sequence can be used to check that the RESET pin has been disabled....

- Follow the instructions in (i) or (ii)
  - Use the **/READSIG** command to attempt to communicate with the device.
- This should fail with e.g. Error 40 – Failed to enter programming mode.

## 1.15 Resurrecting a device where the RESET pin is disabled

If the '**RSTDIBL – RESET Disable**' fuse has been set to a '0' then the target ATtiny device will no longer be able to enter programming mode using '**Low voltage**' programming mode. The only way to re-program a device in this state is to use the '**High voltage**' programming mode to re-enable the '**RSTDIBL – RESET Disable**' fuse. Once this is done, the '**Low voltage**' programming mode will work as normal again.

### Typical example:

[ConsoleEDS ATtiny10HV.PRJ /ERASE /FUSEWRITE=0xFF](#)

The '**high voltage**' base project '**ATtiny10HV**' must be used. This will force the programmer to switch a +12V Vpp voltage to the RESET pin of the target ATtiny device. This is used to enter '**high voltage**' programming mode. The **/ERASE** command is required if the ATtiny '**Security fuses**' have been set. It is not possible to re-program the '**Configuration fuses**' if the '**Security fuses**' have been set. The device must be erased first.