Application Note

Report No:

AN101

Title:

In-System Programming (ISP) of the Atmel AVR FLASH Microcontroller Family using the SPI Programming Interface

Author:	Date:	Version Number:
John Marriott	11th Feb 09	1.17

Abstract:

This application note describes how to develop and implement In-System Programming (ISP) support for the Atmel AVR microcontroller family using the 'SPI Programming Interface'. The document details how to make a 'Programming Project' which will operate on any Equinox ISP programmer. A full description of how to implement In-System Programming (ISP) of the Atmel AT89S, AT90S, AT90USB, ATmega and ATtiny AVR FLASH Microcontroller is also included. The application note 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.

Contents

1.0 Introduction 4
1.1 Programmers supported 4
1.2 Device Support 5
1.3 SPI Algorithm Overview 6
1.4 JTAG Algorithm Overview 7
2.0 SPI Programming Algorithm 8
2.1 Overview of the SPI Programming Interface 8
2.2 Atmel AVR Microcontroller - SPI Implementation 9
2.3 Atmel AT90S AVR Microcontrollers 10
2.4 Atmel ATmega AVR Microcontrollers 11
2.4.1 Overview of possible ATmega pin-outs 11
2.4.2 ATmega AVR - Standard SPI Pin-out 12
2.4.3 ATmega AVR - UART SPI Pin-out 13
2.5 ISP Header Selection Chart (by header) 14
3.0 Creating an SPI Programming Project 16
3.1 Overview 16
3.2 Information required to create an SPI Project 16
3.3 Creating an EDS (Development project) 17
3.3.1 Launching EDS and selecting a Target Device 17
3.3.2 Target Oscillator Settings 18
3.3.3 Target System – Power Supply Settings 19
3.3.4 Specifying the FLASH (Code) File 20
3.3.5 Specifying the EEPROM (Data) File 21
3.3.6 Launching EDS at the end of the EDS Wizard 22
3.4 Testing an SPI Project in Development (EDS) Mode 23
3.5 SPI Programming Mode 24
3.5.1 Overview 24
3.5.2 Hardware SPI Mode 24
3.5.3 Software SPI Mode 24
3.6 Using 'Hardware SPI' Mode 25
3.6.1 Overview 25
3.6.2 Optimum SPI Frequencies for each programmer 26
3.7 Using 'Software SPI' mode 27
3.8 Choosing the fastest possible SPI frequency 28
3.8.1 Overview 28
3.8.2 Programming the CKSEL Fuses to select a faster Oscillator Frequency 28
3.8.3 Creating a Standalone Project which programs the fuses before programming the FLASH / EEPROM 29
3.9 Testing SPI communication with the Target Chip 30
3.10 Programming the FLASH Area 31
3.11 Programming the EEPROM Area 33
3.12 Erasing the FLASH / EEPROM area 35
3.12.1 Erasing the FLASH area 35
3.12.2 Erasing the EEPROM area – special considerations 35
3.13 Programming the Configuration Fuses 36
3.13.1 Overview 36
3.13.2 Reading the Fuses from a Target Device 36
3.13.3 Verifying the Fuses of a Target Device 37
3.13.4 Writing the Fuses into a Target Device 38
3.13.5 Using a 'Fuse File' to import Fuse settings into a project 38
3.13.6 Importing Fuse Settings in HEX format from AVR Studio 38
3.14 Programming the Security Fuses 39

3.14.1 Overview 39
3.14.2 Reading the Security Fuses from a Target Device 40
3.14.3 Verifying the Security Fuses of a Target Device 40
3.14.4 Writing the Security Fuses into a Target Device 41
3.14.5 Erasing the Security Fuses 41
3.14.6 Using a 'Fuse File' to import Security Fuse settings into a project 41
3.14.7 Importing Security Fuse Settings in HEX format from AVR Studio 41
3.15 Exporting an EDS Project to a Standalone Project 42
4.0 Exporting / Importing Fuse Settings to / from File 43
4.1 Overview 43
4.2 Exporting the Fuse Settings to a Fuse File 43
4.3 Copying the Fuses from a Target Device 43
4.4 Importing the Fuse Settings from a Fuse File 43
5.0 Importing Fuse Settings in HEX format from AVR Studio 44
5.1 Overview 44
5.2 Finding the AVR Studio 'Hex Fuse Values' 44
5.3 Importing the AVR Studio 'Hex Fuse Values' into EQTools 45
5.4 Importing the AVR Studio 'Hex Security Fuse Values' into EQTools 48
6.0 Creating a Standalone Project 51
6.1 Overview 51
6.2 Creating a Standalone Project from EDS (Development Mode) 51
6.3 Add Project File to a new Project Collection 51
6.4 Uploading a Project to a programmer 52
6.5 Re-testing a Project in EDS (Development mode) 53

1.0 Introduction

This application note describes how to develop and implement ***In-System Programming (ISP)*** support for the Atmel AVR microcontroller family using the ***'SPI Programming Interface'***. The document details how to make a ***'Programming Project'*** which will operate on any Equinox ISP programmer. A full description of all connection methods required to implement In-System Programming (ISP) of the Atmel AT89S, AT90S, AT90USB, ATmega and ATtiny AVR FLASH Microcontroller is also discussed. 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.

Please note:

- Programming of the Atmel AVR microcontroller family using the ***'JTAG Programming Interface'*** is covered in Application Note – AN105.
- The ***Atmel ATtiny AVR Family*** features both a ***'Low Voltage'*** and ***'High Voltage'*** Serial Programming Modes. Please refer to ***Application Note - AN104*** for further details.

1.1 Programmers supported

Most Equinox ISP Programmers support programming of Atmel AVR microcontrollers using the ***'SPI Programming Interface'*** as standard – see table below. It is also possible to upgrade many of these programmers to support programming via the ***'JTAG Programming Interface'.***

Fig. 1.1 Equinox Programmer – SPI and JTAG ISP Support

Programmer	SPI algorithms	JTAG algorithms	Upgrade Order Code
EPSILON5	YES	UPGRADE	EPSILON5-UPG3
EPSILON5(AVR-JTAG)	NO	YES	N/A
FS2003	YES	UPGRADE	FS2003-UPG7
FS2009	YES	UPGRADE	FS2009-UPG7
FS2009(AVRJTAG)	NO	YES	N/A
PPM3 MK2	YES	UPGRADE + IO-CON-3 JTAG Connector Module + SFM-MAX-V1.3 Special Function Module	PPM3A1-UPG7
PPM4 MK1	YES	UPGRADE + IO-CON-3 JTAG Connector Module + SFM-MAX-V1.3 Special Function Module	PPM4MK1-UPG7
ISPnano	UPGRADE - TBC	UPGRADE - TBC	TBC

Key:

- YES - Enabled as standard
- UPGRADE – Chargeable license upgrade required
- TBC – Details to be confirmed (please e-mail sales@equinox-tech.com for further details.)

1.2 Device Support

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 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. Go to *Programmer* ; *Create a Device Support*.
  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.

Some ATmega devices such as the ATmega8(L) and ATmega161(L) do not have a JTAG port and so cannot support JTAG programming.

Please note:

- Devices with greater than 128kb of FLASH memory require a firmware upgrade to version 3.01 or above in order to support programming of the upper 128kb.
- It is possible to program devices connected in a 'JTAG Chain' using firmware 3.05 or above.
- Please see ***Application Note – AN112*** for instructions on updating your programmer firmware.
- As a rule of thumb, only Atmel Atmega AVR devices with 16k bytes of FLASH or greater will feature the JTAG Programming Interface.

1.3 SPI Algorithm Overview

The SPI algorithm is a simple 3-wire interface which can be used to program most AVR Microcontrollers. The advantages and disadvantages of this algorithm are detailed below.

Advantages

- The SPI algorithm is supported by almost all Atmel AVR microcontrollers including AT90S, AT90CANxxx, ATtiny and ATmega devices. This means that the same Programming Interface can be used on any products containing any AVR microcontroller.
- The SPI Programming Interface uses only 3 SPI pins (MOSI, MISO, SCK) and the RESET pin.
- The SPI pins can be used to drive other circuitry such as LED's and switches on the Target Board as well as being used for ISP purposes. However, this will require careful design on the Target Board to ensure that the programming signals are not compromised.
- In SPI Mode, it is possible to reprogram a single byte of the EEPROM area without having to perform a Chip Erase first.
- The SPI algorithms are supported as standard on all Equinox ISP Programmers.

Disadvantages

- In general terms, the SPI algorithm is 3-4 times slower than the JTAG algorithm.
- When using the SPI algorithm, the clock used during programming is supplied from either the AVR Internal RC Oscillator or from an external crystal / resonator. The programming SPI speed is completely dependent on the speed of this oscillator.
- If the oscillator speed is slow, then the maximum SPI speed is seriously limited and the overall programming will be very slow.
- If the Clock Selection Fuses are incorrectly programmed in SPI mode, then the chip may no longer have a valid oscillator and so will not respond to the programmer. This can render the chip unprogrammable except by physically removing it from the Target Board and using either a JTAG or Parallel programmer to resurrect the correct Fuse Settings.

1.4 JTAG Algorithm Overview

The JTAG algorithm provides a method of performing high-speed programming of an Atmel Atmega AVR microcontroller. The same JTAG port can also be used for on-chip debugging of code using the Atmel JTAG-ICE Debugger. The advantages and disadvantages of the JTAG algorithm are detailed below.

Advantages

- The JTAG algorithm is approximately 3-4 times faster at programming compared to the SPI algorithm.
- The programming time using JTAG for the EEPROM is significantly faster than the SPI algorithm because in JTAG mode a 'Page' of EEPROM is programmed at a time rather than a single byte. Each byte may take e.g. 9ms to program in SPI mode, where as a whole page of e.g. 4 bytes may take 9ms to program in JTAG mode.
- 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.
- With the JTAG algorithm, the programming clock is supplied by the programmer and JTAG logic inside the Target AVR device does not require any other clocking. This means that the chip is not dependent on the settings of the 'Clock Selection Fuses' in JTAG Mode.
- In JTAG mode is it possible to change the 'Clock Selection Fuses' to any value and still program the chip. (with the exception of the 'JTAGEN' Fuse)
- 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. This functionality is now supported by Equinox programmers running firmware 3.05 and above.

Disadvantages

- The JTAG Programming Interface uses 5 pins: TCK, TDI, TDO, TMS and RESET.
- 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.
- In JTAG mode the EEPROM is divided into 'Pages' rather than 'Single Bytes'. It is therefore more complicated to program a single byte in the EEPROM as the entire page (usually 4 or 8 bytes) must be read back and then the single byte overlaid on top of this data and finally the entire page is then re-programmed back into the EEPROM.
- In JTAG Mode, it is not possible to re-program any location in the EEPROM which is not 0xFF without first performing a Chip Erase operation. This means that if the EEPROM already contains any data, it is not possible to re-program this data without erasing the entire chip first.

2.0 SPI Programming Algorithm

2.1 Overview of the SPI Programming Interface

The SPI Programming Interface is a simple synchronous 3-wire communications bus which is commonly used for control / data transfer between a Master Processor and a Slave Peripheral such as an external SPI memory device – see figure 2.1.

Fig 2.1a – SPI Master / Slave example

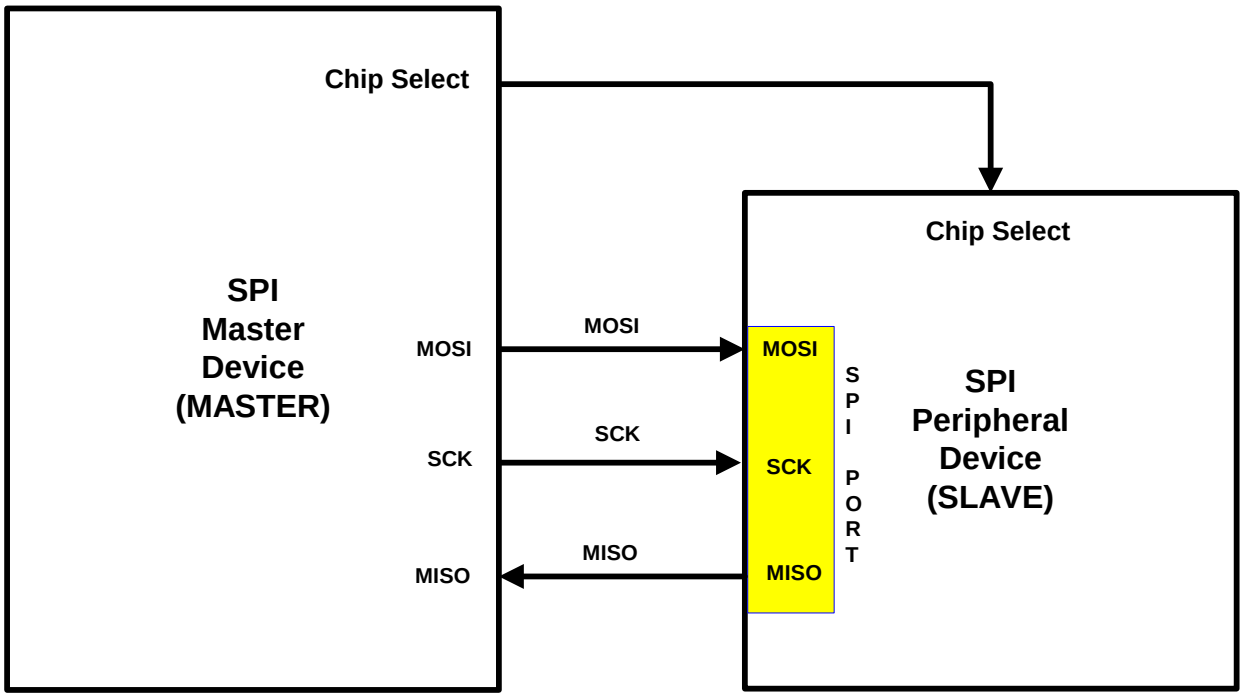


Fig 2.1b – SPI Signal names and directions

Signal Name	Signal description	Signal direction (from Master)
MOSI	Master OUT, Slave In	Output
MISO	Master IN, Slave OUT	Input
SCK	Serial Clock	Output
Chip Select (CS)	Chip Select	Output

Data is transferred from the Master to the Slave using the ***MOSI (Master OUT, Slave In)*** signal line. The Slave transfers data back to the Master using the ***MISO (Master IN, Slave OUT)*** signal line. The data transfer is clocked by the ***SCK (Serial Clock)*** signal line which is generated by the Master on the SPI bus. The Slave uses the ***SCK*** signal to know when to sample the ***MOSI*** signal for valid data and when to output valid data on the ***MISO*** signal line.

Most SPI Slave devices have a ***'Chip Select'*** signal which the Master asserts to select a particular Slave device on the SPI bus. In the example above with only one Slave SPI device, the Master would still have to assert the ***Chip Select*** line in order to communicate with the Slave device.

2.2 Atmel AVR Microcontroller - SPI Implementation

Atmel have chosen the SPI interface to implement fast In-System Programming (ISP) of their AT90S, ATmega and ATtiny AVR Microcontroller families. This implementation allows the on-chip FLASH, EEPROM, Configuration Fuses and Security Fuses of a target AVR Microcontroller to be In-System Programmed using a suitable external ISP Programmer or an-board SPI Master Controller – see fig 2.2a.

Fig 2.2a – ISP Programming Implementation of Atmel AVR Microcontrollers

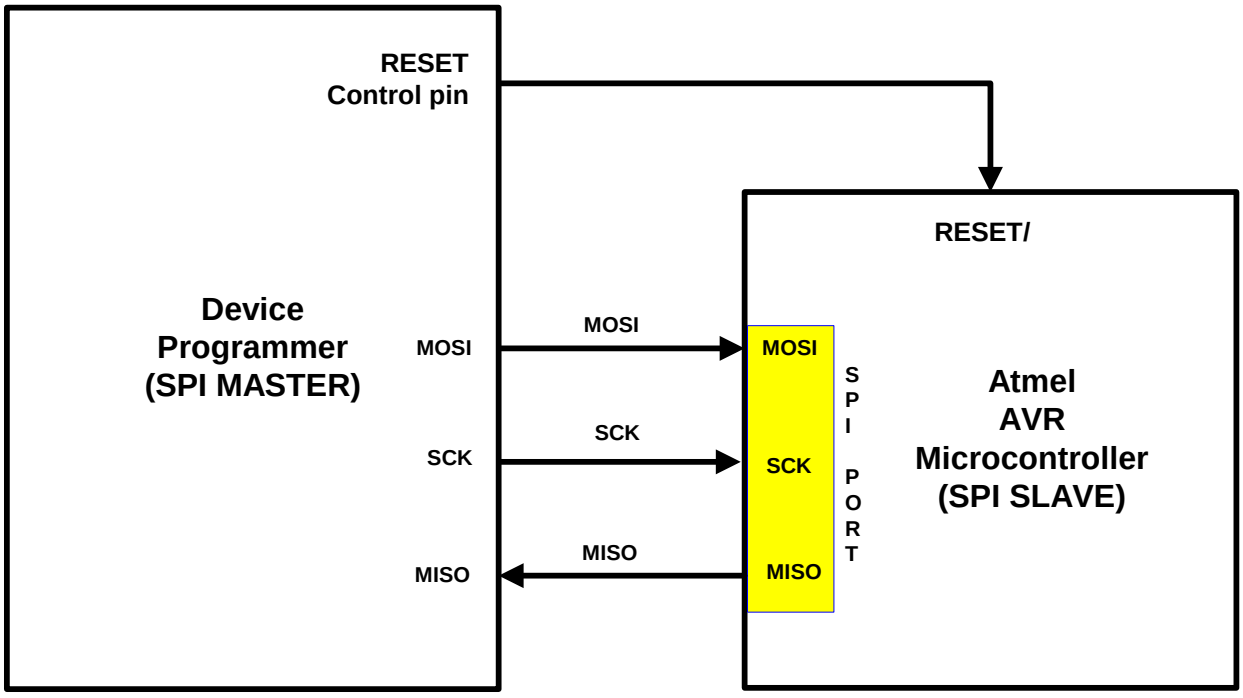


Fig 2.2b – Programmer / Microcontroller - SPI Signal names and directions

Signal Name	Signal description	Signal direction (from Programmer)	Signal direction (from Microcontroller)
MOSI	Master OUT, Slave In	Output	Input
MISO	Master IN, Slave OUT	Input	Output
SCK	Serial Clock	Output	Input
RESET	Chip Select	Output	Input

The external Device Programmer is the ***SPI Bus Master*** and the AVR Microcontroller on the Target System is the ***SPI Slave***. The ***RESET*** control signal from the programmer is used to force the Target Microcontroller to enter the so-called AVR ***'Serial Programming Mode'***. For Atmel AVR Microcontrollers, the programmer must drive the ***RESET*** pin LOW and then send a command on the SPI bus to enter programming mode. This has the effect of resetting the target Microcontroller so it is no longer running firmware (i.e. the user application ceases to execute).

Once the Target Device is in ***'Serial Programming Mode'***, the external programmer can transfer data to / from the target AVR device across the SPI bus. At the end of the programming cycle, the programmer simply creates a RESET pulse and then the device should start to run the firmware which has been programmed into it.

2.3 Atmel AT90S AVR Microcontrollers

The Atmel AT90S AVR Microcontroller Family use the standard SPI pins (MOSI, MISO, SCK) for In-System Programming (ISP) – see fig. 2.3.

Fig 2.3a AT90Sxxxx AVR – ISP Connections

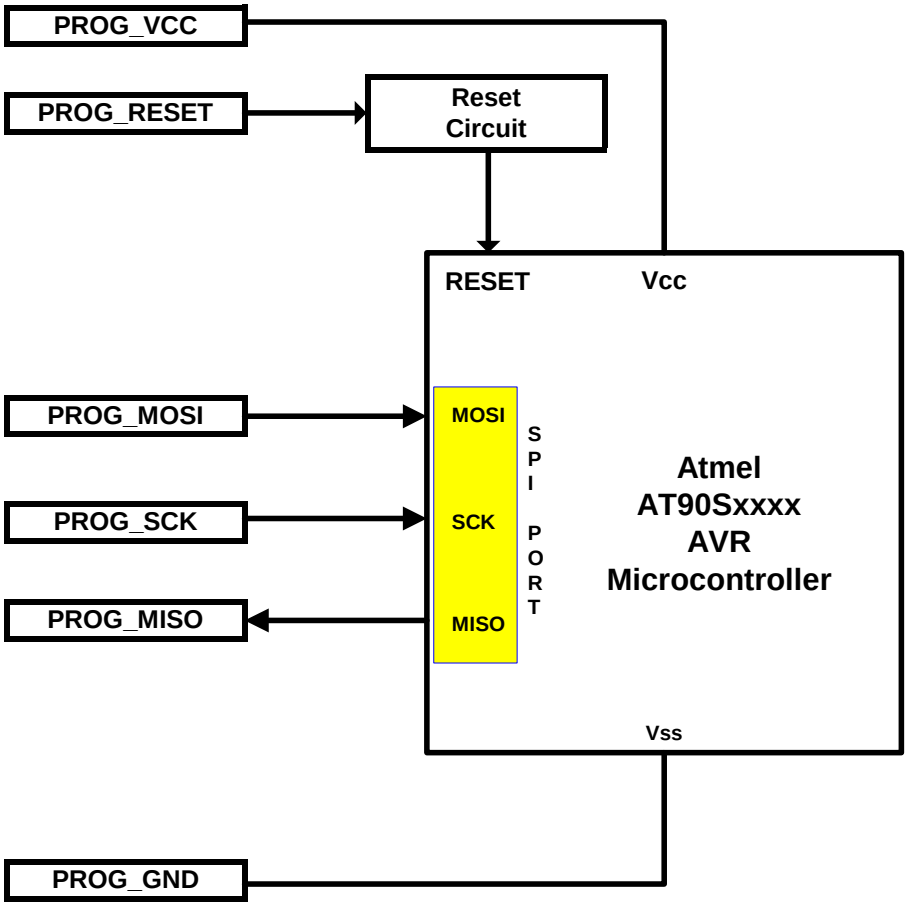


Fig 2.3.b – AT90S AVR Microcontroller - SPI Signal names and directions

Programmer Signal Name	Signal description	Signal direction (from Programmer)	Connect to AVR Microcontroller Pin	Signal direction (from Microcontroller)
PROG_MOSI	Master OUT, Slave In	Output	MOSI	Input
PROG_MISO	Master IN, Slave OUT	Input	MISO	Output
PROG_SCK	Serial Clock	Output	SCK	Input
PROG_RESET	RESET	Output	RESET	Input

2.4 Atmel ATmega AVR Microcontrollers

2.4.1 Overview of possible ATmega pin-outs

The majority of devices in the Atmel ATmega AVR family conform to the standard SPI pin-out for In-System Programming using the MOSI, MISO and SCK pins of the target device. However, there are also a few devices which use the TXD pin as MISO and the RXD pin as MOSI during In-System Programming. These derivatives are referred to as ***'UART SPI Pin-out'*** devices. Special care must be taken to route the programmer ***MOSI / MISO pins*** to the correct pins of the target AVR device otherwise during In-System Programming will not function.

Please refer to the ***'Device Support'*** table in section 1.2 for details of which ATmega devices feature either the ***'Standard'*** or ***'UART'*** SPI pin-out. Please look up the device you are trying to program in the table and then refer to relevant section for the correct ISP pin-out.

2.4.2 ATmega AVR - Standard SPI Pin-out

This pin-out is compatible with all ATmega devices which use the MOSI and MISO pins for In-System Programming.

Fig. 2.4.2a ATmega – Standard Pin-out - ISP connections

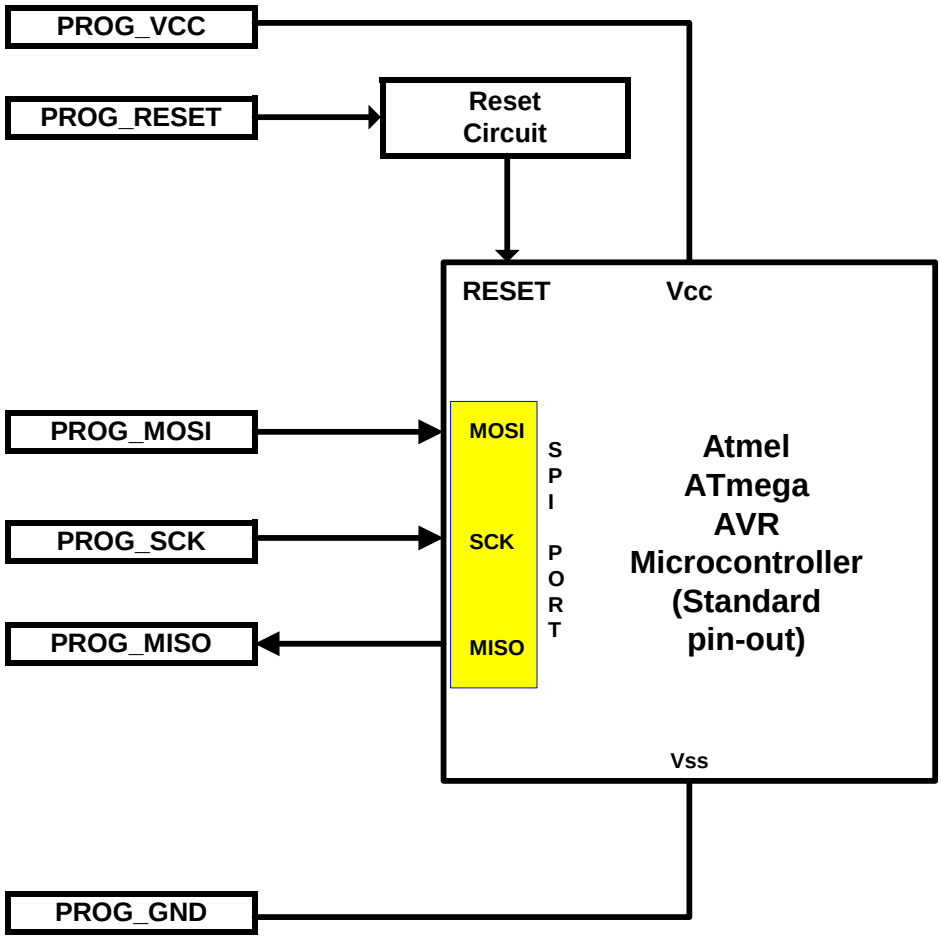


Fig 2.4.2b – ATmega AVR – Standard Pin-out - SPI Signal names and directions

Programmer Signal Name	Signal description	Signal direction (from Programmer)	Connect to AVR Microcontroller Pin	Signal direction (from Microcontroller)
PROG_MOSI	Master OUT, Slave In	Output	MOSI	Input
PROG_MISO	Master IN, Slave OUT	Input	MISO	Output
PROG_SCK	Serial Clock	Output	SCK	Input
PROG_RESET	RESET	Output	RESET	Input

2.4.3 ATmega AVR - UART SPI Pin-out

This pin-out is compatible with the Atmel ATmega 64(L), ATmega103(L), ATmega128(L), ATmega1281, ATmega2561 and ATmega169(L) devices which **use the *TXD* and *RXD* pins for SPI during In-System Programming.** The standard SPI MOSI and MISO pins are not used at all during In-System Programming and can be freely connected to other SPI devices.

2.4.3a ATmega – Standard Pin-out - ISP connections

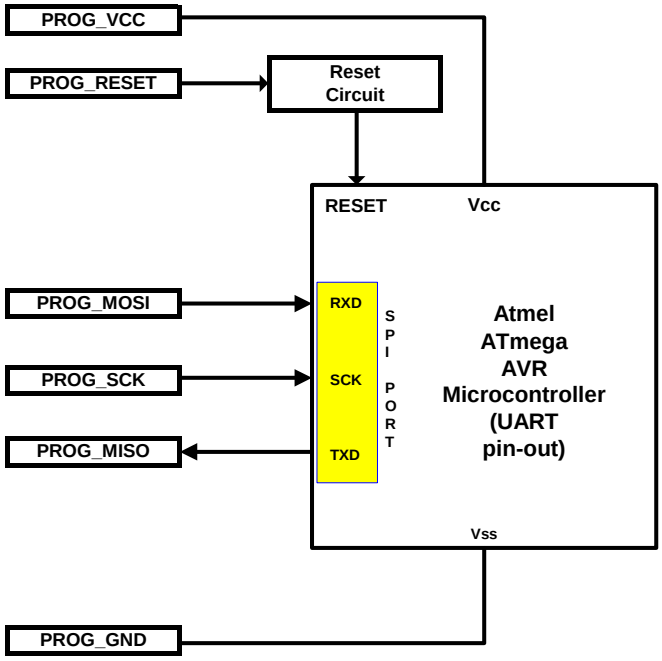


Fig 2.4.3b – ATmega AVR – Standard Pin-out - SPI Signal names and directions

Programmer Signal Name	Signal description	Signal direction (from Programmer)	Connect to AVR Microcontroller Pin	Signal direction (from Microcontroller)
PROG_MOSI	Master OUT, Slave In	Output	**RXD***	Input
PROG_MISO	Master IN, Slave OUT	Input	**TXD***	Output
PROG_SCK	Serial Clock	Output	SCK	Input
PROG_RESET	RESET	Output	RESET	Input

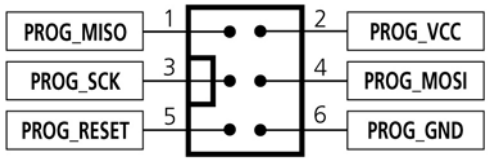
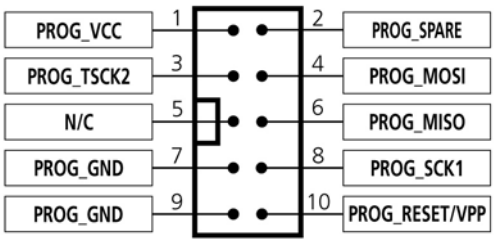
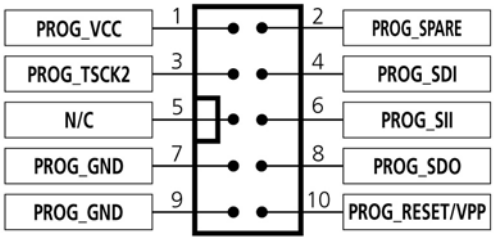
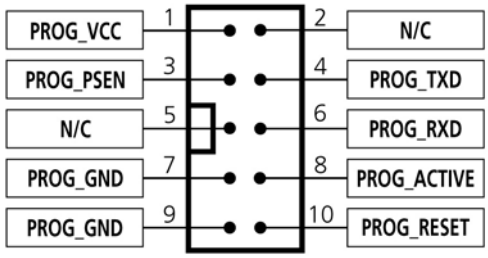
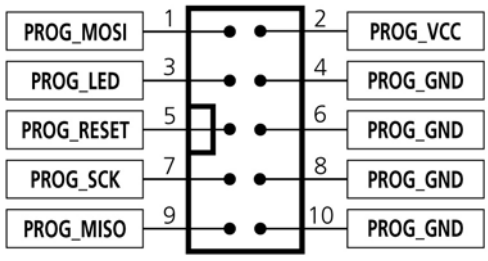
* Please note – The TXD and RXD pins must be used for ISP instead of the MISO and MOSI pins.

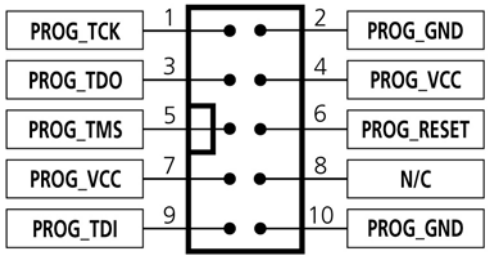
Special Considerations

1. The TXD and RXD pins must be used for ISP instead of the MISO and MOSI pins.
2. For the ATmega103(L) device only, the ATmega103 MISO pin (pin 13) is active during In-System Programming even though this pin is not actually used for programming. If this pin is used as an output, make sure that whatever it is connected to can cope with the pin toggling during ISP. It may also be necessary to insert a current limiting resistor in this line.

2.5 ISP Header Selection Chart (by header)

The FOUR ISP Headers featured on the most Equinox ISP Programmers are detailed in the table below. The Atmel 6-way (1) and Atmel 10-way header (3) connectors are also found on the Atmel STK500 Target Board. The Atmel JTAG connector (4) is found on all Equinox ISP programmers except for the FS2000A and the same connector is used on the Atmel JTAG ICE and many Atmel Target Boards which feature a JTAG programming interface.

#	ISP Header	Description / Function	ISP Header Pin-out
1	J3	Atmel 6-way ISP Header	
Header J6 can have THREE different pin-outs depending on which Target Device is to be programmed. See (2a), (2b) and (2c).			
2a	J6(a)	**Equinox 10-way Header(a)** **Device support:** Atmel AT90S, ATmega, ATtiny, AT89S devices	
2b	J6(b)	**Equinox 10-way Header(a)** **Device support:** Atmel ATtiny11/12/15 High Voltage (+12V Vpp) Programming Mode	
2c	J6(c)	**Equinox 10-way Header(b)** **Device support:** Atmel Wireless T89C51Rx2 Philips P89C51Rx2 / 66x	
3	J7	**Atmel 10-way Header** **Device support:** Atmel AT90S, ATmega, ATtiny, AT89S devices	

4	J8	**Atmel 10-way JTAG Header** **Device support:** Atmel ATmega32/128 + any new devices with JTAG port	

3.0 Creating an SPI Programming Project

3.1 Overview

This section describes how to create a ***'Programming Project'*** for an Atmel AVR microcontroller using ***'SPI – 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.

3.2 Information required to create an SPI Project

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

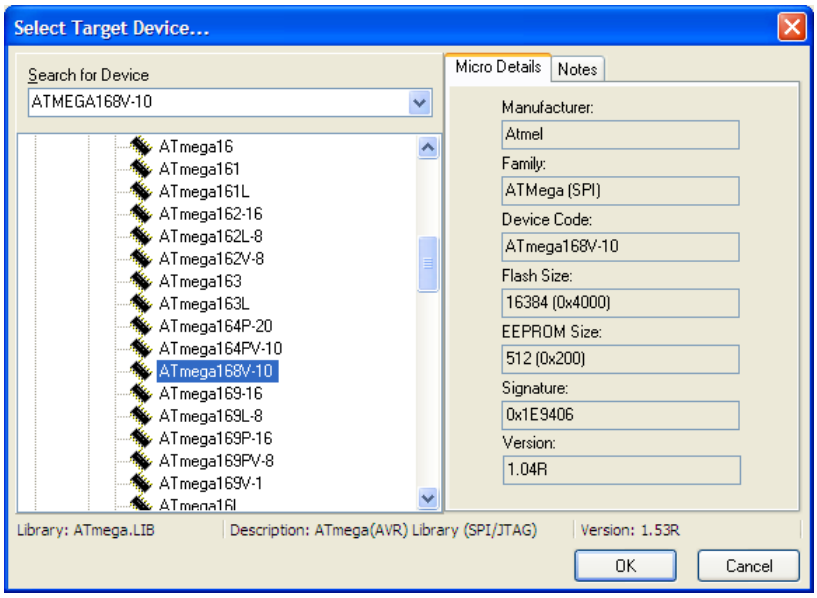
#	Information / data required	Example
1	AVR Device part number	ATmega2561
2	SPI connections / connector on Target board	Atmel 10-way IDC connector
3	SPI Programming configuration	i. Single AVR device or ii. Multiple AVR devices with a different RESET or Chip Select for each device
4	Target device oscillator speed (before programming)	Many AVR devices run from an INTERNAL e.g. 1MHz oscillator when you first receive the chip from Atmel. This means that the programmer SPI speed is limited to a maximum of approx. 250 kHz.
5	Target device oscillator frequency (final frequency during / after programming)	The programmer can instruct the Target AVR device to run from a different faster oscillator during programming. This is usually a faster external oscillator. e.g. 12 MHz crystal
6	Target System Vcc voltage	e.g. 3.3V
7	Target System maximum current consumption	e.g. 100mA
8	FLASH area 'Program File'	Binary (*.bin) or Intel Hex (*.hex)
9	EEPROM area 'Data File'	Binary (*.bin) or Intel Hex (*.hex)
10	Configuration Fuse values These fuse values describe how the 'Configuration Fuses' in the AVR device are to be programmed.	i. Boolean fuse values: e.g. SPIEN=0, CKSEL=1, CKSEL2=0 etc ii. Fuse ***Hex values*** from 'AVR Studio' e.g. 0x22 0x45 0x34
11	Reset circuit parameters	• Capacitor / Resistor circuit • Watchdog supervisor circuit • Voltage monitoring circuit

3.3 Creating an EDS (Development project)

The simplest way to create a Programming Project for an SPI device is to use the EDS (Development Mode) Wizard as follows:

3.3.1 Launching EDS and selecting a Target Device

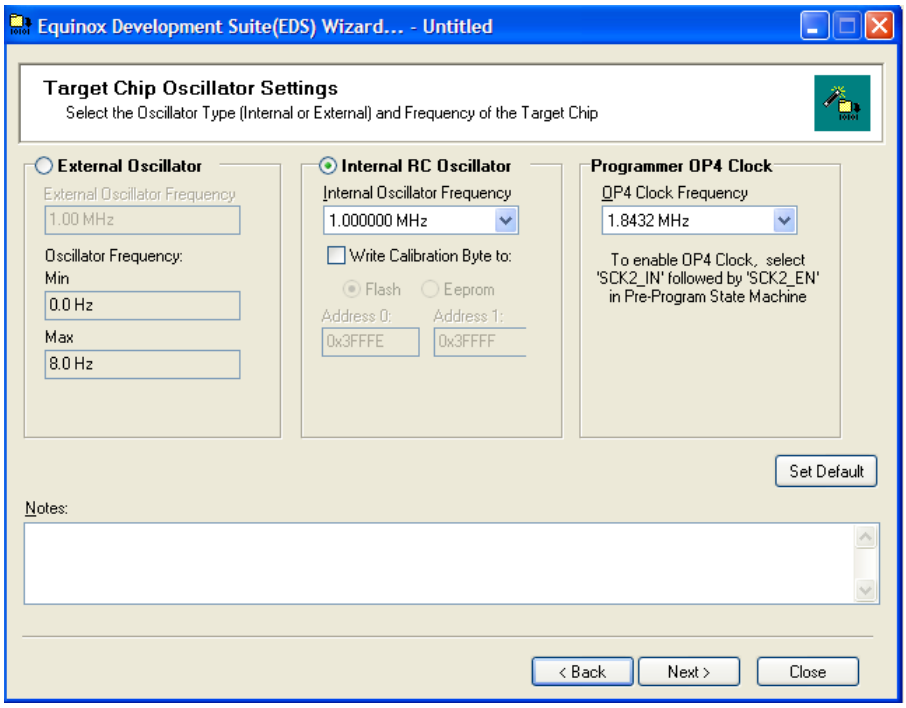
- Launch EQTools
- Select **<Create a new Development (EDS) Project**> → the EDS (Development) Wizard will launch
- Click **<Next>** → the **<Select Target Device>** screen will be displayed.



- Type the required device into the ***'Search for Device'*** field → a list of matching devices are displayed in the box below.
- Select the required device from the list and then click **<OK>** → the device is now selected.
- On the next screen, check that the device selection and all other device parameters are correct
- Click **<Next>** to advance to the next screen

3.3.2 Target Oscillator Settings

This screen allows you to set up the ***'Target Oscillator frequency'***.
The ***'Target Oscillator Frequency'*** is the frequency which the Target Device is being clocked at during the In-System Programming (ISP) Process.



- Many Atmel AVR devices feature both an ***INTERNAL*** on-chip oscillator and also the ability to run from an ***EXTERNAL*** crystal or Ceramic Resonator.
- When a virgin device from Atmel is programmed for the first time, it will usually be running from an ***INTERNA***L Oscillator. The frequency of the oscillator is usually set at the factory to be approximately 1MHz. At this frequency both SPI In-System Programming (ISP) and JTAG will operate but SPI ISP will be very slow.
- In SPI programming mode, the Target AVR device must be clocked by a stable oscillator during programming. e.g. Internal Oscillator, external crystal or ceramic resonator.

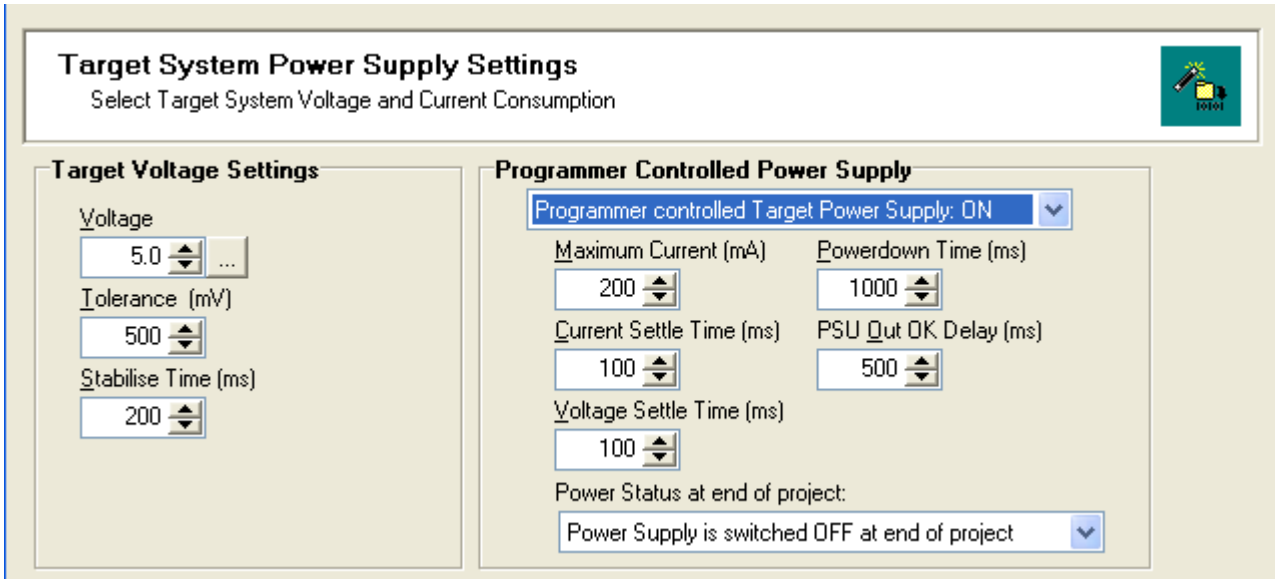
Instructions:

- If the Target Device is running from an ***INTERNAL*** oscillator e.g. 1MHz internal, select 'Internal Oscillator' and select the internal oscillator frequency from the drop-down list.
- If the Target Device is going to be running from an ***EXTERNAL*** oscillator e.g. crystal or ceramic resonator once the CKSEL Fuse bits have been programmed, select ***'External Oscillator'*** and enter the oscillator frequency. This should be written on the oscillator component itself or on the circuit schematic.
- The ***'Programmer OP4 Clock'*** can be used to clock a device which has no oscillator.

Please note:
The Target Oscillator speed is not technically required for SPI programming. It simply allows the GUI to work out what the maximum allowed SPI speed should be for the Target Device.

3.3.3 Target System – Power Supply Settings

This screen allows you to set up the Power Supply characteristics of your Target System.



i. Select the Target Voltage

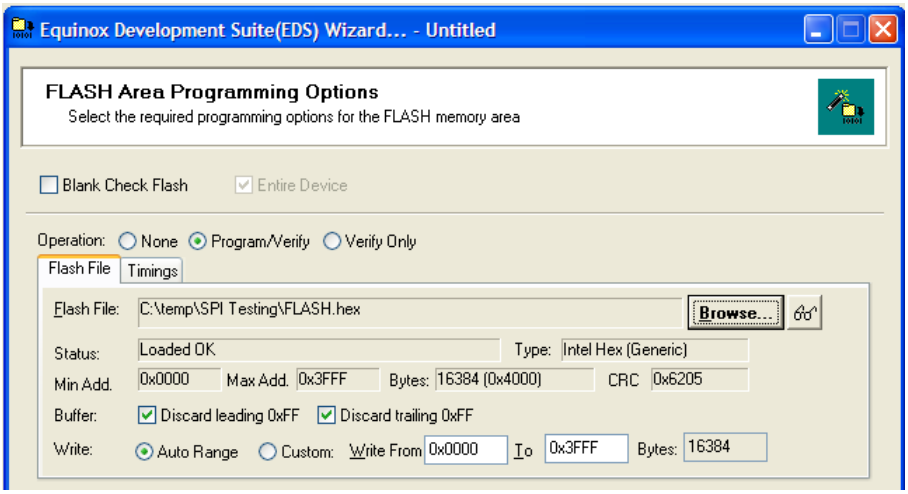
- This should be the voltage at which the Target System is being powered during the programming operation.
- Set the 'Voltage Tolerance' to be as wide as possible e.g. 500mV to allow for power supply variations. If the programmer is powering the Target System, this will also give a faster power-up time.
- It may be possible to power just the Target Microcontroller rather than the entire Target System.

ii. Set up the Target Powering and current parameters

- This option is only available for the PPM3-MK2 / PPM4-MK1 programmers.
- If the programmer is to power the Target System, select ***<Programmer controlled Target Power Supply: ON>***
- Set the ***'Maximum Current'*** to the maximum possible current which the Target System could draw from the programmer.
- Leave all other settings as default.

3.3.4 Specifying the FLASH (Code) File

This screen allows you to specify the Code (firmware) file which is to be programmed into the FLASH area of the Target Device. This is an optional step – you can also specify the file once you are in the Development Suite (EDS).



i. Blank Check the FLASH

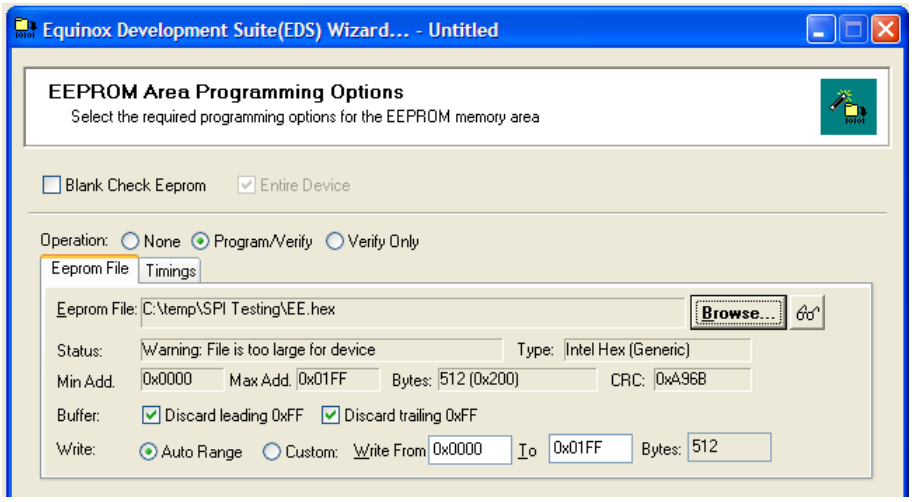
- If the chip has been erased at the start of the programming cycle, then the FLASH should already be blank (i.e. all locations contain the value 0xFF).
- If you want to be absolutely sure the FLASH is blank, you can enable the 'Blank Check Flash' option. This will perform a full Blank Check of the FLASH area to check that all locations are set to 0xFF.
- Warning – this check can be time-consuming and will increase the overall programming time!

ii. Selecting the FLASH File

- Click the **<Browse>** button
- Browse to and select the file you wish to load and then select **<OK>**
- If the input file is a BINARY file then the wizard will load the data in from file starting at address 0x0000 and continuing contiguously to the end of the file.
- If the input file is an INTEL HEX file then the wizard will load in from file from the start address specified in the file to end address specified in the file.

3.3.5 Specifying the EEPROM (Data) File

This screen allows you to specify the EEPROM (data) file which is to be programmed into the EEPROM area of the Target Device. This is an optional step – you can also specify the file once you are in the Development Suite (EDS).



i. Blank Check the EEPROM

- If the chip has been erased at the start of the programming cycle, then the EEPROM should already be blank (i.e. all locations contain the value 0xFF).
- However, if the Target Device has an 'EESAVE' fuse and this fuse is ENABLED (EESAVE=0), then the EEPROM will not be erased during the Chip Erase operation.
- If you want to be absolutely sure the EEPROM is blank, you can enable the 'Blank Check EEPROM' option. This will perform a full Blank Check of the EEPROM area to check that all locations are set to 0xFF.
- Warning – this check can be time-consuming and will increase the overall programming time!

ii. Selecting the EEPROM File

- Click the **<Browse>** button
- Browse to and select the file you wish to load and then select **<OK>**
- If the input file is a BINARY file then the wizard will load the data in from file starting at address 0x0000 and continuing contiguously to the end of the file.
- If the input file is an INTEL HEX file then the wizard will load in from file from the start address specified in the file to end address specified in the file.
- In the example above, the EEPROM file contains more data than the physical EEPROM size of the Target Chip. The input data will therefore only read the data in to the address range of the EEPROM.
- In SPI Mode, the granularity of the EEPROM Memory can be 1, 4 or 8 bytes. This is known as the 'EEPROM Page Size'. This means that the programmer will always program in pages of 1, 4 or 8 bytes. Your input file will therefore be rounded up to the nearest block of 1, 4 or 8 bytes.

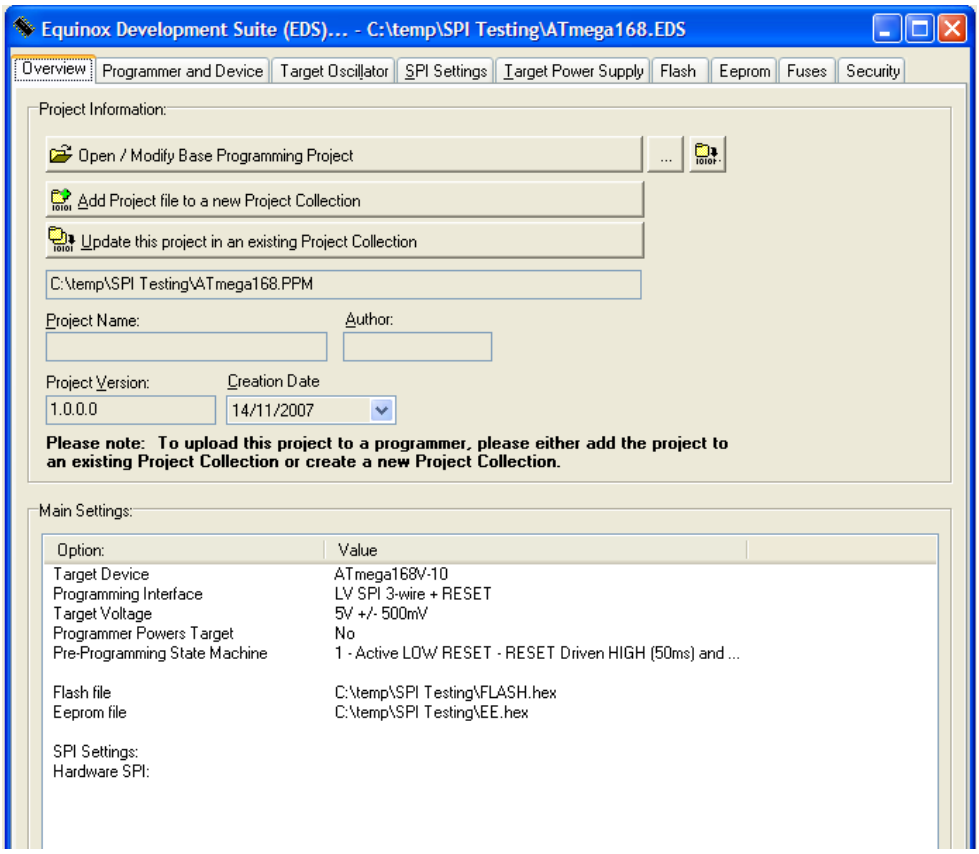
3.3.6 Launching EDS at the end of the EDS Wizard

Once you reach the end of the EDS Wizard, click the **<Test>** button to launch the project in the Equinox Development Suite (EDS).

Enter a name for the EDS project e.g. ATmega168 and click **<Save>**
→ your project will now launch in EDS Mode.

3.4 Testing an SPI Project in Development (EDS) Mode

If you have clicked the **<Test>** button at the end of the EDS Wizard, then an EDS (Development Mode) session will now launch.



The following default settings will be used:

- ***'Hardware SPI'*** is selected by default for all AVR microcontrollers using 'Low-voltage Serial Programming Mode'. This is usually much faster than **'Software SPI'**.
- ***'Software SPI'*** is selected by default for all ATtiny AVR microcontrollers when using the 'High-voltage Serial Programming Mode'. This is because this algorithm does not use true SPI.
- The ***'SLOW SPI'*** speed is used for all operations except for programming the FLASH and EEPROM blocks
- The ***'FAST SPI'*** speed is used for programming the FLASH and EEPROM blocks only.
- Target System not powered by programmer (unless enabled during the EDS Wizard)
- The default ***'SPI pre-programming state machine'*** will be used.
- The ***'Configuration Fuse Write'*** operation is disabled (can be enabled in EDS)
- The ***'Security Fuse Write'*** operation is disabled (can be enabled in EDS)

At this stage there are still a few parameters which may need to be set up / checked before the programmer will communicate with the Target Device on the Target Board.

Please follow the instructions in the next sections which explain how to set up the:

- SPI Mode
- SPI Frequency

3.5 SPI Programming Mode

3.5.1 Overview

It is possible to program most Atmel AVR devices using either **'Software SPI'** or **'Hardware SPI'** mode. The fastest ***'SPI Mode'*** for programming purposes is **'Hardware SPI'** and so this mode is automatically selected for all Atmel AVR devices with the exception of the ATtiny family when programming in 'High-voltage Serial Programming Mode'. The 'Software SPI' mode should only be used if you need to program at very slow SPI frequencies (<100kHz) or the SPI clock timing requires manual adjustment to compensate for an inductive or capacitive load on the SPI pins.

The table below details the available 'SPI Modes' for each Device Family:

#	Device family	Software SPI	Hardware SPI
1	AT89S	Yes	Yes (default)
2	AT90S	Yes	Yes (default)
3	AT90CAN	Yes	Yes (default)
4	AT90USB	Yes	Yes (default)
5	AT90PWM	Yes	Yes (default)
6	ATtiny (LV SPI algorithm)	Yes	Yes (default)
7	ATtiny (HV SPI algorithm)	Yes	Not supported
8	Atmega	Yes	Yes (default)
8	Atmega 'P' PICO	Yes	Yes (default)

The 'default' setting is 'Hardware SPI' for all devices except for the ATtiny devices in 'High-voltage Serial Programming Mode'.

3.5.2 Hardware SPI Mode

When **'Hardware SPI'** mode is selected, the programmer will use a **'Hardware SPI Port'** to generate the SPI waveforms which are used to program the Target Device. This has the advantage that the SPI Clock waveform has a 50:50 mark:space ratio and so is much more reliable when programming many Atmel AVR microcontrollers. It can also run at much higher frequencies than **'Software SPI'**. The disadvantages of the **'Hardware SPI'** are that only a few designated frequencies are available and also that the lowest frequency is 115.2kHz for the PPM3/4 and FS2003/9 and 57.6 kHz for the Epsilon5 programmer.

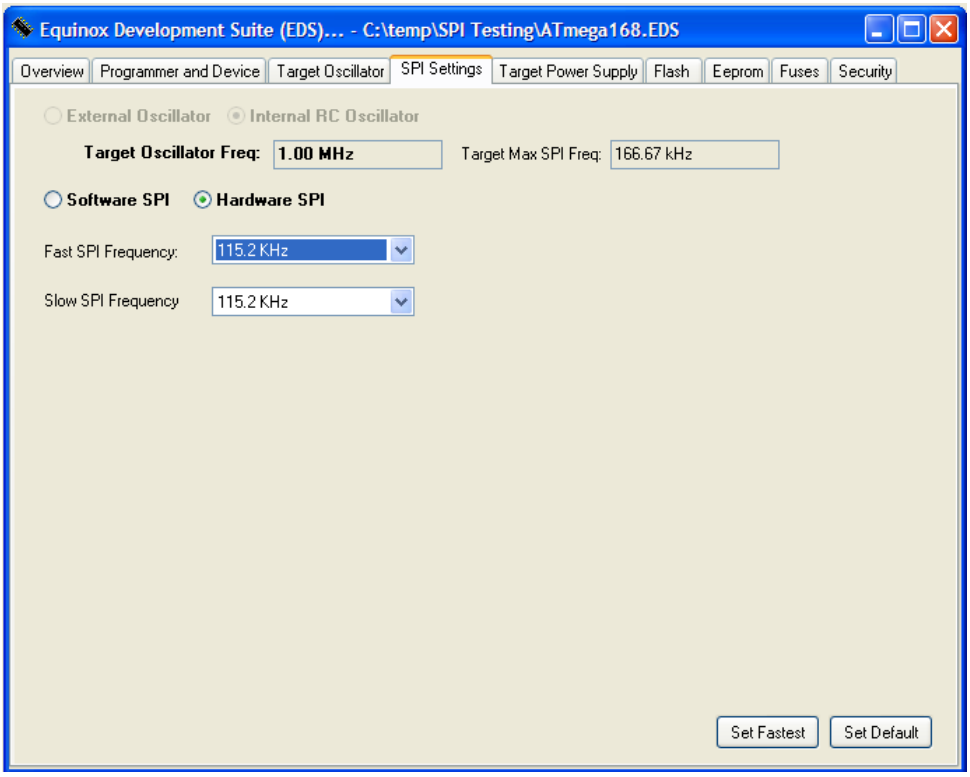
3.5.3 Software SPI Mode

When 'Software SPI' mode is selected, the programmer will use bit-toggling software algorithms to generate the SPI waveforms which are used to program the Target Device. This results in the SPI waveforms having an uneven mark:space ratio which can lead to unreliable programming of some AVR microcontrollers. The advantages of ***'Software SPI'*** are that very low SPI speeds can be supported and the timings of the SPI waveform can be adjusted to suit inductive or capacitive loads.

3.6 Using 'Hardware SPI' Mode

3.6.1 Overview

To select **'Hardware SPI'** mode, click the **<SPI Settings>** tab and then click the **'Hardware SPI'** radio button.



There are two different SPI frequency settings – **SLOW** and **FAST** – which are used during different parts of the programming algorithm.

SLOW SPI Frequency

- The 'Slow SPI Frequency' is used for all programming operations except for programming the FLASH and EEPROM memories.
- This frequency should be left at the lowest selectable value. This means that the programmer should always be able to communicate with an AVR microcontroller, even if the micro is running from a slow internal oscillator.
- The lowest selectable SPI frequencies are 115.2kHz for the PPM3/4 and FS2003/9 and 57.6 kHz for the Epsilon5 programmer.

FAST SPI Frequency

- The ***'Fast SPI'*** speed is used for programming the FLASH and EEPROM blocks only. It should be set to the fastest frequency supported by the Target Device.
- For AVR devices, the maximum SPI frequency must be at least ¼ of the frequency that the Target AVR Microcontroller is running at.

3.6.2 Optimum SPI Frequencies for each programmer

The tables below show the optimum **'FAST SPI'** frequency settings for the Epsilon5, FS2003, FS2009, PPM3-MK2 and PPM4-MK1 programmers. The **'Target AVR Oscillator Frequency'** is the frequency of the oscillator which the AVR device is running from during the programming procedure.

i. For the Epsilon5 programmer:

#	Target AVR Oscillator Frequency	Slow SPI frequency range	Maximum possible SPI frequency	Suggested FAST SPI frequency
1	1 MHz	50 - 150 kHz	<250 kHz	115.2 kHz
2	4 MHz	50 - 150 kHz	< 1MHz	921.6 kHz
3	8 MHz	50 - 150 kHz	< 2 MHz	921.6 kHz
4	16 MHz	50 - 150 kHz	< 8 MHz	921.6 kHz

ii. For the FS2003, FS2009 and PPM3-MK2 and PPM4-MK1 programmers (fitted with the standard EQ-SFM I/O Driver Module):

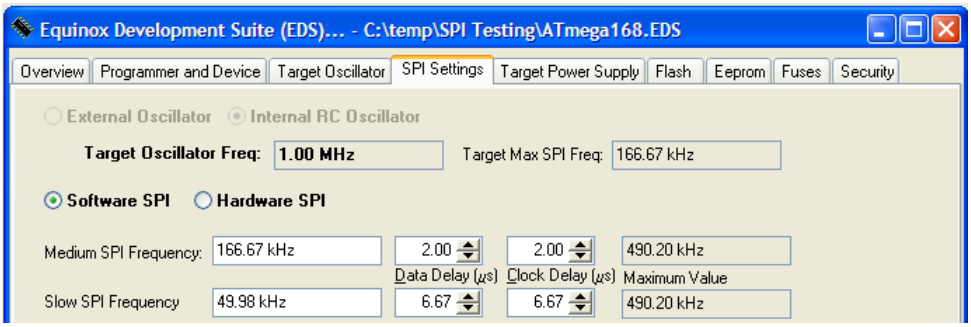
#	Target AVR Oscillator Frequency	Slow SPI frequency range	Maximum possible SPI frequency	Suggested FAST SPI frequency
1	1 MHz	50 - 150 kHz	<250 kHz	115.2 kHz
2	4 MHz	50 - 150 kHz	< 1MHz	921.6 kHz
3	8 MHz	50 - 150 kHz	< 2 MHz	921.6 kHz
4	16 MHz	50 - 150 kHz	< 8 MHz	921.6 kHz

iii. For the PPM3-MK2 or PPM4-MK1 programmers fitted with the EQ-SFM-MAX-V1.2 or V1.3 High-speed I/O Driver Module:

#	Target AVR Oscillator Frequency	Slow SPI frequency range	Maximum possible SPI frequency	Suggested FAST SPI frequency
1	1 MHz	50 - 150 kHz	<250 kHz	115.2 kHz
2	4 MHz	50 - 150 kHz	< 1MHz	921.6 kHz
3	8 MHz	50 - 150 kHz	< 2 MHz	1.8432 MHz
4	16 MHz	50 - 150 kHz	< 8 MHz	3.6864 MHz

3.7 Using ‘Software SPI’ mode

To select ‘Software SPI’ mode, click the **<SPI Settings>** tab and then click the **‘Software SPI’** radio button.



There are two different SPI frequency settings – **SLOW** and **MEDIUM** – which are used during different parts of the programming algorithm.

SLOW SPI Frequency

- The ‘Slow SPI Frequency’ is used for all programming operations except for programming the FLASH and EEPROM memories.
- This frequency should be left at a low value e.g. 50 – 100 kHz. This means that the programmer should always be able to communicate with an AVR microcontroller, even if the micro is running from a slow internal oscillator.

MEDIUM SPI Frequency

- The ***‘Medium SPI’*** speed is used for programming the FLASH and EEPROM blocks only. It should be set to the fastest frequency supported by the Target Device.
- For AVR devices, the maximum SPI frequency must be at least ¼ of the frequency that the Target AVR Microcontroller is running at.

3.8 Choosing the fastest possible SPI frequency

3.8.1 Overview

The programming time of an Atmel AVR microcontroller can be dramatically reduced by using the fastest possible SPI frequency for programming the FLASH and EEPROM areas of the chip. As a general rule of thumb, the fastest possible SPI frequency must be less than ¼ of the Target Microcontroller Oscillator Frequency.

The table below shows the typical frequency ranges for both the **'SLOW SPI'** and **'MEDIUM / FAST'** SPI settings for a given Target Oscillator frequency.

#	Target AVR Oscillator Frequency	Slow SPI frequency range	MEDIUM or FAST SPI frequency range
1	1 MHz	50 - 150 kHz	<250 kHz
2	4 MHz	50 - 150 kHz	< 1MHz
3	8 MHz	50 - 150 kHz	< 2 MHz
4	16 MHz	50 - 150 kHz	< 8 MHz

Example:
If the microcontroller is running from e.g. an internal 1MHz oscillator, then the maximum SPI frequency is approximately 250kHz. It would be prudent to use 200kHz to be on the safe side as the internal oscillators of AVR devices are not calibrated and so can be running at a much lower frequency than the nominal value quoted in the datasheet.

3.8.2 Programming the CKSEL Fuses to select a faster Oscillator Frequency

When you first program a virgin AVR microcontroller which has an Internal Oscillator, then the device will have been set up by Atmel to run from the Internal Oscillator at e.g. 1MHz. This means the maximum SPI frequency is limited to < 250 kHz which would lead to very slow programming of the FLASH and EEPROM areas. To speed up the programming cycle of the AVR device, it is necessary to set the clock source of the AVR to use either a faster Internal Oscillator or to use a faster External Oscillator. This can be achieved by programming the **'CKSEL – Clock Selection Fuses'** to select the faster oscillator **BEFORE** programming the FLASH and EEPROM areas.

To achieve the fastest programming time in EDS (Development Mode):

- Select the **EDS – Fuses** tab (this will default to Post-erase Fuses)
- Enable the **'Fuse Write'** operation
- If you are only using the AVR Internal Oscillator, set the CKSEL fuses to the fastest Internal Oscillator setting e.g. 8MHz
- If you are only using the AVR External Oscillator, set the CKSEL fuses to the fastest External Oscillator setting e.g. 8MHz. This should be the frequency of your external oscillator signal, crystal or ceramic resonator

- Set the **'Fuse Action'** field to be **'Program → RESET → Verify'**. This will program the fuses and then perform a reset cycle of the AVR device forcing it to run from the new oscillator settings.
- Click the **<Write>** button to program these fuse values into the AVR device.
  → the AVR microcontroller should now start to run from the faster Internal or External oscillator.
- If you now try to program some data into the FLASH or EEPROM area, it should program OK at the SPI Frequency you have selected.
- If the programming fails then it is probable that the chip is still running from a slower oscillator e.g. 1MHz. Check the fuse settings are correct and then try to program the FLASH / EEPROM again. If it still fails, try reducing the MEDIUM / FAST SPI Frequency in steps of eg. 100 kHz until the programming operation passes.

3.8.3 Creating a Standalone Project which programs the fuses before programming the FLASH / EEPROM

To achieve the fastest possible programming times, it is necessary to program the CKSEL Clock Selection Fuses to select the fastest possible AVR oscillator **BEFORE** the FLASH and EEPROM areas are programmed. If you follow the instructions in section 3.8.2 to create an EDS project and then open the ***'Base Project'*** from the ***EDS – Overview*** tab, the Fuse Settings will have been placed in the ***<Post Erase Fuses>*** tab. If you now compile this project and upload it to the programmer, the fuses will be written immediately after the Erase operation and then the chip will reset forcing it to run from the new oscillator settings.

Here is a how a typical Standalone Programming project will program an AVR device:

- Start
- AVR is running from e.g. **Internal 1MHz oscillator**
- Programmer uses **SLOW** SPI speed of e.g. 50kHz
- Enter programming mode
- Chip Erase
- Program ***CKSEL Fuses*** → selects e.g. the internal 8 MHz oscillator
- Reset cycle the chip → AVR starts to run from the **internal 8 MHz oscillator**
- Programmer uses **FAST** SPI speed of e.g. 1.8432 MHz to program FLASH and EEPROM
- Program FLASH area @ SPI frequency = 1.8432 MHz
- Program EEPROM area @ SPI frequency = 1.8432 MHz
- Program Security Fuses @ SPI frequency = 50kHz
- End

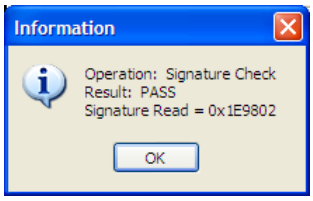
3.9 Testing SPI communication with the Target Chip

To make sure that the programmer can communicate to the Target SPI device, try reading back the Device Signature as follows:

- Select the **<FLASH>** tab
- Locate the **<Check Sig>** button on the right-hand side of the screen and click it.

→ The programmer will now try to communicate with the Target Chip via the SPI Interface

→ If the Target Chip responds correctly, then EDS will report 'Signature Read: Pass'.



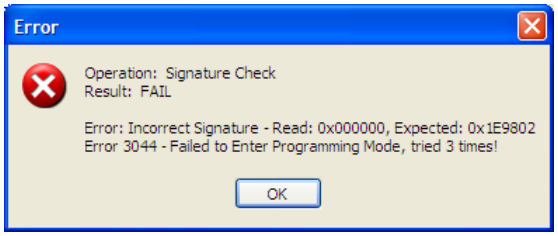
→ If the Target Chip does not respond, then EDS will report either:

i. Cannot enter programming mode

If you receive this error, please check the following:

- The SPI connections between the programmer and the Target System are correct.
- There is definitely power applied to the Target System and to all the SPI devices.
- The ***'SPI Frequency'*** settings are correct for the Target Device being programmed.
- If your project is using ***'Software SPI'***, try slowing down the ***'SLOW SPI Frequency'*** and then try to check the Device Signature again.
- If your project is using ***'Hardware SPI'***, try slowing down the ***'SLOW SPI Frequency'*** and then try to check the Device Signature again.
- Check that the ***'Reset State machine'*** is providing a suitable reset signal to force the Target Device into programming mode.

ii. 'Signature Read: Fail'.



If you receive this error, please check the following:

- Make sure there are no series resistors in-line with any of the SPI signal lines
- Make sure there are no capacitors on any of the SPI signal lines
- Make sure the total length of the SPI ISP cabling is no more than 200 cm
- Try slowing down the 'SLOW SPI Frequency' and then try to check the Device Signature again.

3.10 Programming the FLASH Area

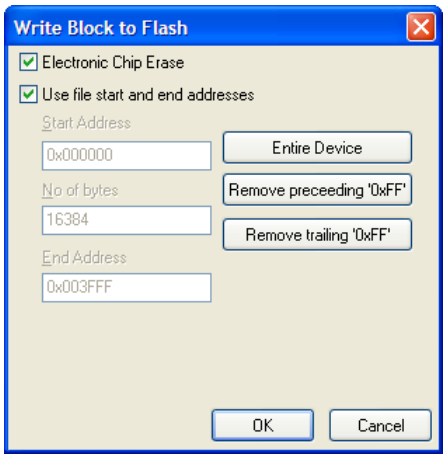
These instructions describe how to program the contents of a file into the FLASH area of the Target Device.

Please note:

- If you are using ***'Software SPI'***, then the ***'MEDIUM' SPI speed*** is used for writing the FLASH area.
- If you are using ***'Hardware SPI'***, then ***the 'FAST' SPI speed*** is used for writing the FLASH area.
- For AVR devices, the **maximum value of the SPI Frequency must be < ¼ of the AVR oscillator frequency.**

To program the FLASH area:

- Select the **<FLASH>** tab
- If you have not already selected a data file to program, click the **'Edit buffer'** check box and then click the **<Load>** button to select a suitable Binary of Intel Hex file.
- The contents of the specified file should now be displayed in the Buffer Window.
- Click the **<Write>** button



- EDS will automatically perform a Chip Erase by default which will erase the entire FLASH before programming any data into it.
- Select the address range you wish to program.
- EDS will automatically use the 'Start' and 'End' address of the FLASH input file unless otherwise specified. This reduces the total data actually programmed to the number of bytes in the input file rounded to the end of the nearest FLASH Page.
- If you want to program the entire FLASH range, click the **<Entire Device>** button.
- Click **<OK>** to program the FLASH of the Target Chip.
- The programmer should now start to program the chip.
- The BUSY LED will illuminate on the programmer.
- The programmer will program the contents of the Buffer Window into the FLASH area of the Target Device.

- Each block of data is programmed and then verified so if a failure occurs it will be notified immediately.
- To verify that the data has been programmed correctly, click the **<Verify>** button.

If EDS reports a 'FLASH programming error', please check the following:

- Make sure the FLASH area is definitely erased before attempting to program it.
- The ***'SPI Frequency'*** settings are correct for the Target Device being programmed.
- The Target AVR Device may be running from a slow Internal Oscillator (e.g. @ 1MHz) so the maximum SPI Frequency can only be approximately 250kHz. Try writing the AVR Clock Selection Fuses (CKSEL xxx) to force the chip to run from a faster oscillator.
- If your project is using ***'Software SPI'***, try slowing down the ***'MEDIUM SPI Frequency'*** and then try to program the FLASH again.
- If your project is using ***'Hardware SPI'***, try slowing down the ***'FAST SPI Frequency'*** and then try to program the FLASH again.

3.11 Programming the EEPROM Area

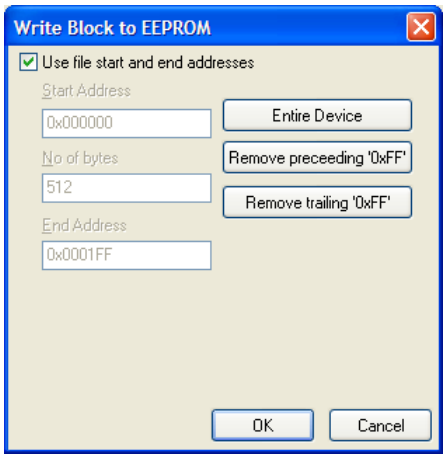
These instructions describe how to program the contents of a file into the EEPROM area of the Target Device:

Please note:

- If you are using ***'Software SPI'***, then the ***'MEDIUM' SPI speed*** is used for writing the EEPROM area.
- If you are using ***'Hardware SPI'***, then ***the 'FAST' SPI speed*** is used for writing the EEPROM area.
- For AVR devices, the **maximum value of the SPI Frequency must be < ¼ of the AVR oscillator frequency.**

To program the EEPROM area:

- Select the **<EEPROM>** tab
- If you have not already selected a data file to program, click the 'Edit buffer' check box and then click the **<Load>** button to select a suitable Binary of Intel Hex file.
- The contents of the specified file should now be displayed in the Buffer Window.
- Click the **<Write>** button



- Select the address range you wish to program
- EDS will automatically use the 'Start' and 'End' address of the FLASH input file unless otherwise specified. This reduces the total data actually programmed to the number of bytes in the input file rounded to the end of the nearest EEPROM Page.
- If you want to program the entire EEPROM range, click the **<Entire Device>** button.
- Click **<OK>** to program the EEPROM of the Target Chip.
- The programmer should now start to program the chip.
- The BUSY LED will illuminate on the programmer.
- The programmer will program the contents of the Buffer Window into the EEPROM area of the Target Device.
- The EEPROM data is programmed in pages of either 1, 4 or 8 bytes and then verified so if a failure occurs it will be notified immediately.

If EDS reports an 'EEPROM programming error', please check the following:

- The ***'SPI Frequency'*** settings are correct for the Target Device being programmed.
- The Target AVR Device may be running from a slow Internal Oscillator (e.g. @ 1MHz) so the maximum SPI Frequency can only be approximately 250kHz. Try writing the AVR Clock Selection Fuses (CKSEL xxx) to force the chip to run from a faster oscillator.
- If your project is ***using 'Software SPI'***, try slowing down the ***'MEDIUM SPI Frequency'*** and then try to program the FLASH again.
- If your project is using ***'Hardware SPI'***, try slowing down the ***'FAST SPI Frequency'*** and then try to program the FLASH again.

3.12 Erasing the FLASH / EEPROM area

It is possible to erase the FLASH and / or EEPROM area of a Target Device by clicking the **<Erase>** button. This will also erase the Security Lock Bits changing all the Lock Bit values from '0' to '1'. The Configuration Fuses are not affected by a Chip Erase operation.

3.12.1 Erasing the FLASH area

The only way to erase the FLASH area of the Target Device is to use the 'Chip Erase' command:

- Select the **<FLASH>** tab
- Click the **<Erase>** button
- This will send the 'Chip Erase' command to the Target Device.
- The Target Device will then erase the FLASH (and EEPROM as long as the EESAVE flag is not set to 0)
- To confirm that the FLASH / EEPROM is definitely blank, you can choose to perform a Blank Check operation.

3.12.2 Erasing the EEPROM area – special considerations

The only way to erase the EEPROM area of the Target Device is to use the 'Chip Erase' command:

- Select the **<EEPROM>** tab
- Click the **<Erase>** button
- This will send the 'Chip Erase' command to the Target Device.
- The Target Device will then erase the FLASH.
- The EEPROM area will only be erased if the EESAVE flag is set to '1'.
- To confirm that the FLASH / EEPROM is definitely blank, you can choose to perform a Blank Check operation.
- If the EEPROM is still not blank after the Erase Operation, check that the EESAVE fuse is definitely set to '1'.

Important note:
In JTAG ISP mode, it is not possible for the programmer to write any bit of EEPROM from a '1' to a '0'. This means that each EEPROM location must contain 0xFF before it can be programmed to any other value. This requires a Chip Erase operation to clear all locations to the value 0xFF.

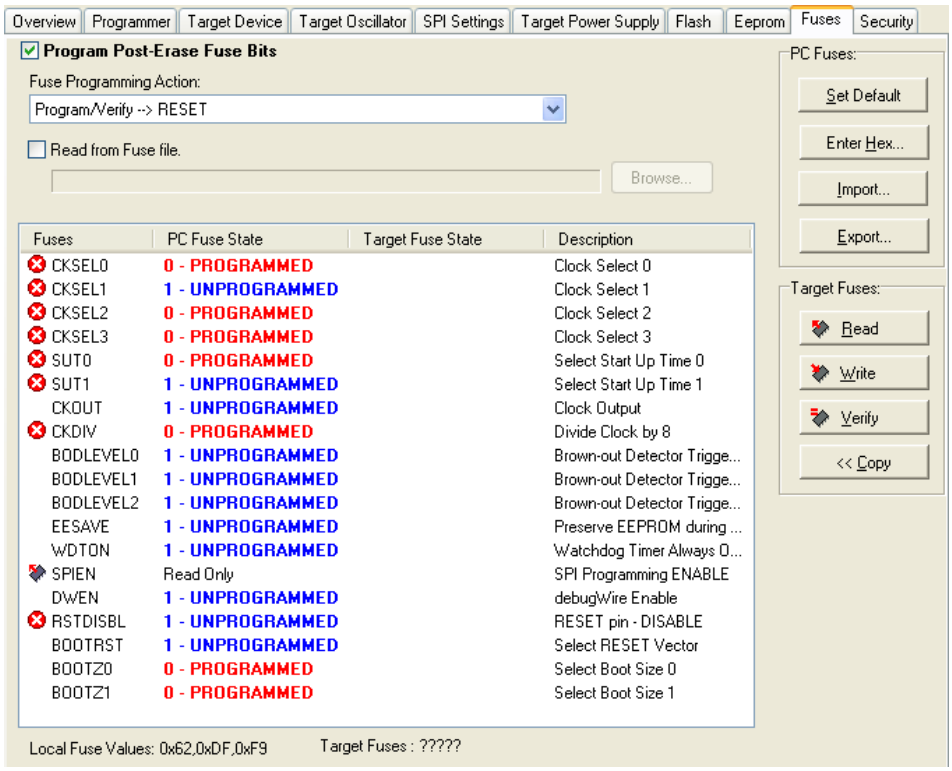
3.13 Programming the Configuration Fuses

3.13.1 Overview

The Configuration Fuses of an Atmel AVR device can be programmed / read using the **<Fuses>** tab.

Instructions:

- Select the **<Fuses>** tab
- If this is a new EDS project, then the Fuses will be disabled.
- Check the ***'Program Post-Erase Fuse Bits'*** box → the Fuses can now be programmed

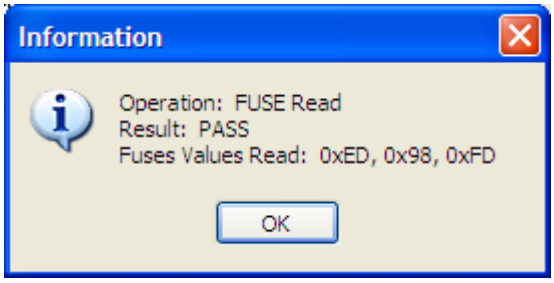


- The values of the Fuses which could be programmed into the Target Chip are shown in the ***'PC Fuse State'*** column. The initial Fuse values are the default Fuse values for a virgin chip.
- The ***'Target Fuse State'*** column displays the current value of the Fuses of the actual Target Device. They are initially set to ***'?'*** until the first read or write operation is performed.
- The ***Fuse Hex values*** are shown for the ***'PC Fuse State'*** at the bottom of the screen.
- A red ***'x'*** next to a fuse indicates a ***'Dangerous Fuse'***. Programming one of these fuses incorrectly could result in the chip no longer responding to the programmer.

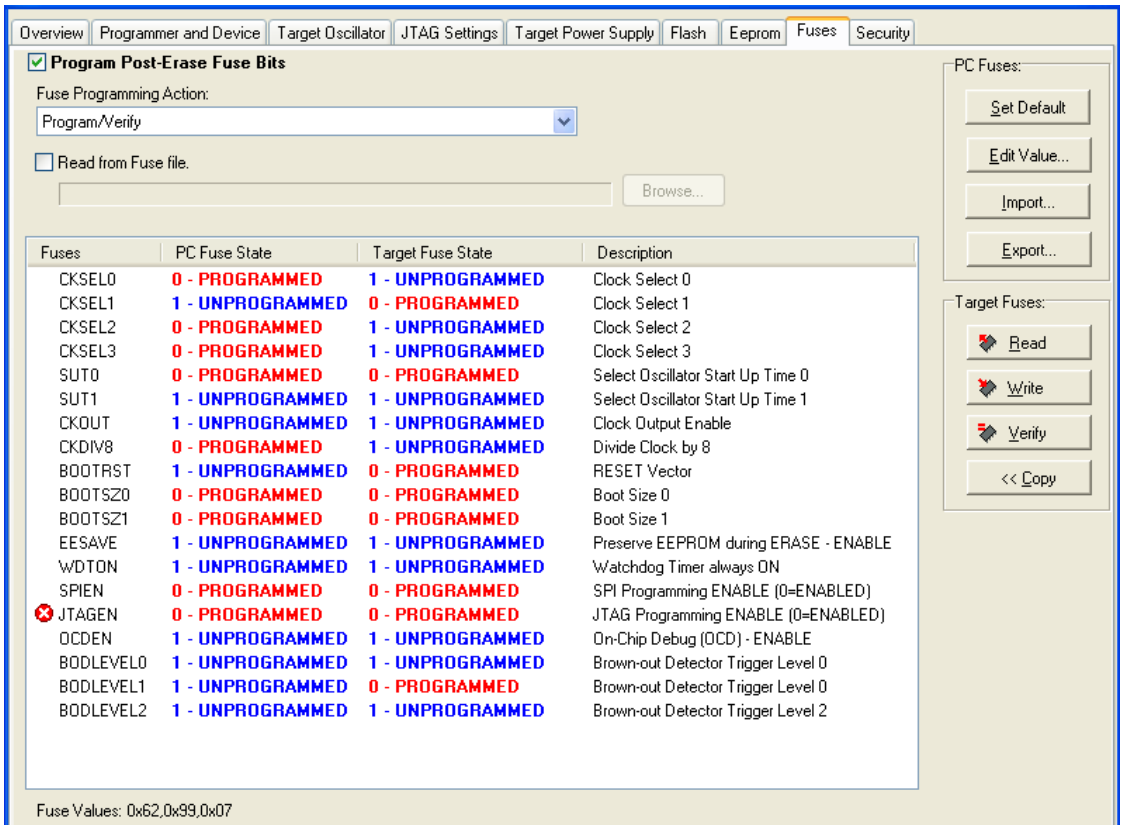
3.13.2 Reading the Fuses from a Target Device

To read the Fuse Values from a Target Device:

- Click the **<Read>** button
- The Hex values of the 'Fuse Bytes' which are read back are displayed as follows:



- The Fuse values from the Target Device are now displayed in the ***'Target Fuse State'*** column.



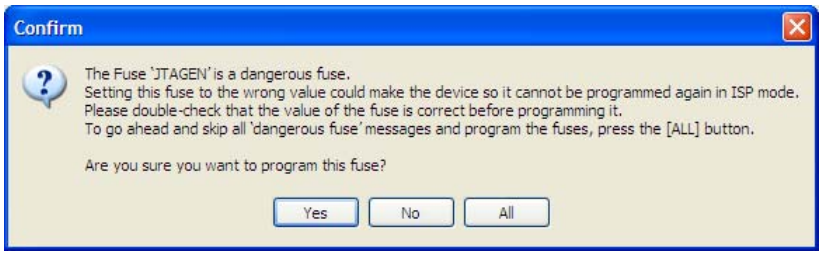
3.13.3 Verifying the Fuses of a Target Device

To verify the Fuse Values in a Target Device with the Fuse Values in the ***'PC Fuse State'*** column:

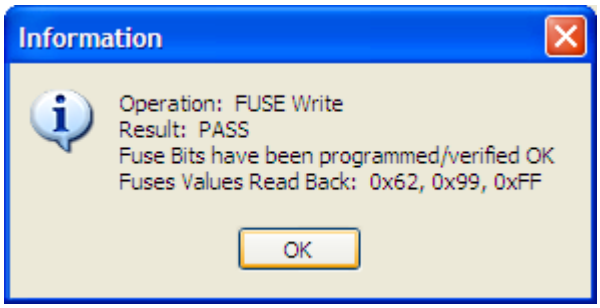
- Click the **<Verify>** button
- Any differences in the Fuse Values between the PC settings and the Target Device setting will now be displayed.

3.13.4 Writing the Fuses into a Target Device

- To program the Fuses values in the 'PC Fuse State' column into the Target Device, click the **<Write>** button
- If there are any 'Dangerous Fuses' in the list, then the following warning will be displayed:



- **!!! WARNING !!!** If you choose to program e.g. the SPIEN (SPI Enable) Fuse to a '1' (unprogrammed), then the chip will no longer respond to SPI ISP programming.
- Click **<Yes>** to allow programming of the selected Fuse
- Click **<All>** to skip all fuse warning messages and program all the fuses
- The programmer will now program all the fuses at the same time and then read them back and verify them with the values in the 'PC Fuse State' column.
- The programmer will then report a PASS or FAIL for programming the Fuses.



3.13.5 Using a 'Fuse File' to import Fuse settings into a project

It is possible to export the ***'Fuse Values'*** for a particular device to a so-called ***'Fuse File'*** so that a single copy of the fuses is stored in one place. This ***'Fuse File'*** can then be shared amongst many projects if required. See section 4 for further details about using ***'Fuse Files'*** .

3.13.6 Importing Fuse Settings in HEX format from AVR Studio

If the original firmware for your project has been developed using the Atmel ***'AVR Studio'*** software, then it is likely that the ***'Fuse Settings'*** are defined as ***'Hex Fuse Bytes'***. Please refer to Section 4 for instructions on how to import these ***'Hex Fuse Bytes'*** into your EQTools project.

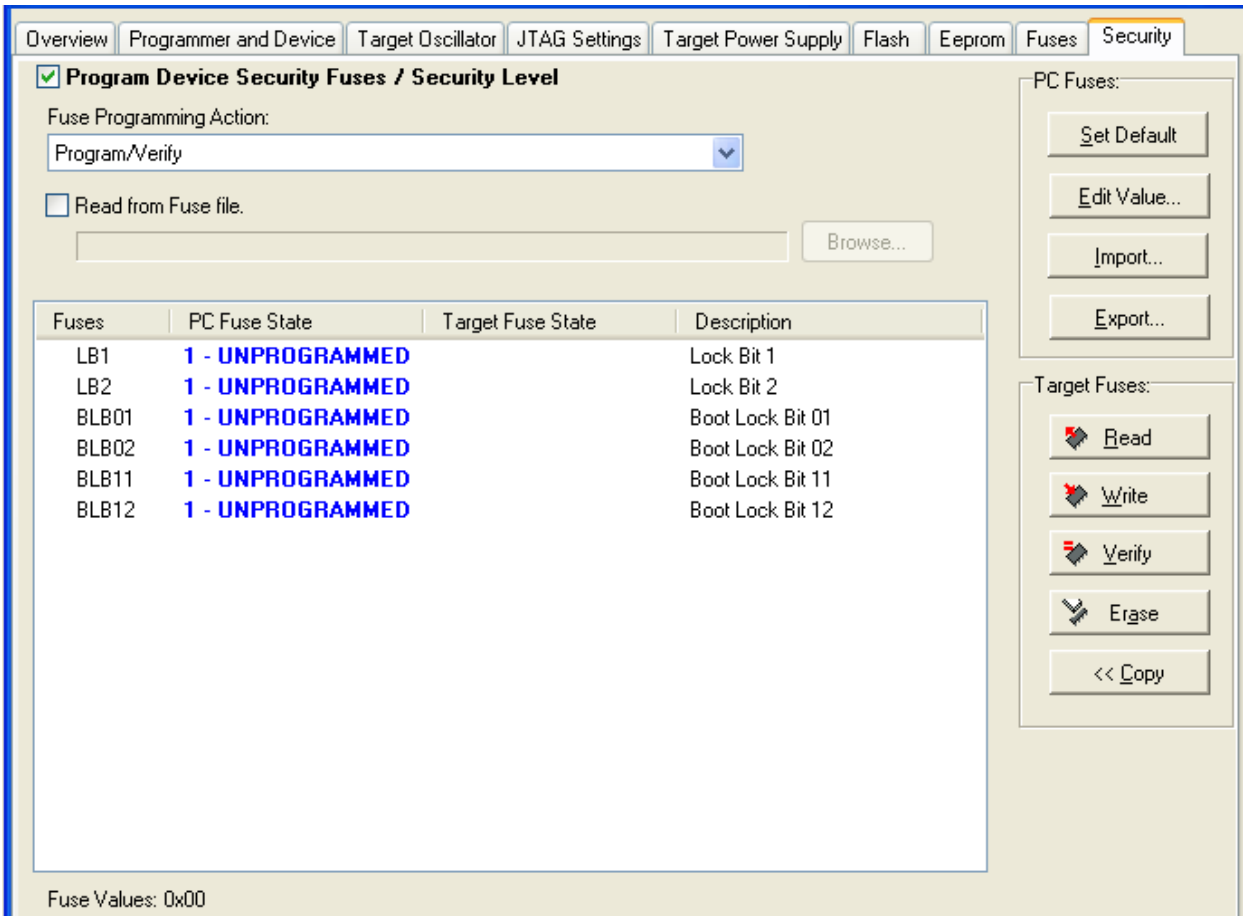
3.14 Programming the Security Fuses

3.14.1 Overview

The Security Fuses of an Atmel AVR device can be programmed / read using the **<Security Fuses>** tab.

Instructions:

- Select the **<Security Fuses>** tab
- If this is a new EDS project, then the Security Fuses will be disabled.
- Check the ***'Program Device Security Fuses'*** box → the Security Fuses can now be programmed

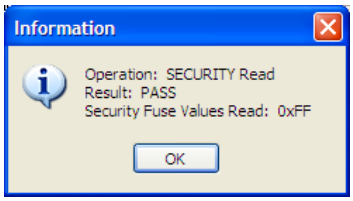


- The values of the Security Fuse values which could be programmed into the Target Chip are shown in ***the 'PC Fuse State'*** column. The initial Fuse values are the default Fuse values for a virgin chip which usually represents an 'unlocked' chip.
- The ***'Target Fuse State'*** column displays the current value of the Fuses of the actual Target Device. They are initially set to ***'?'*** until the first read or write operation is performed.
- The ***'Fuse Hex values'*** are shown for the ***'PC Fuse State'*** at the bottom of the screen.

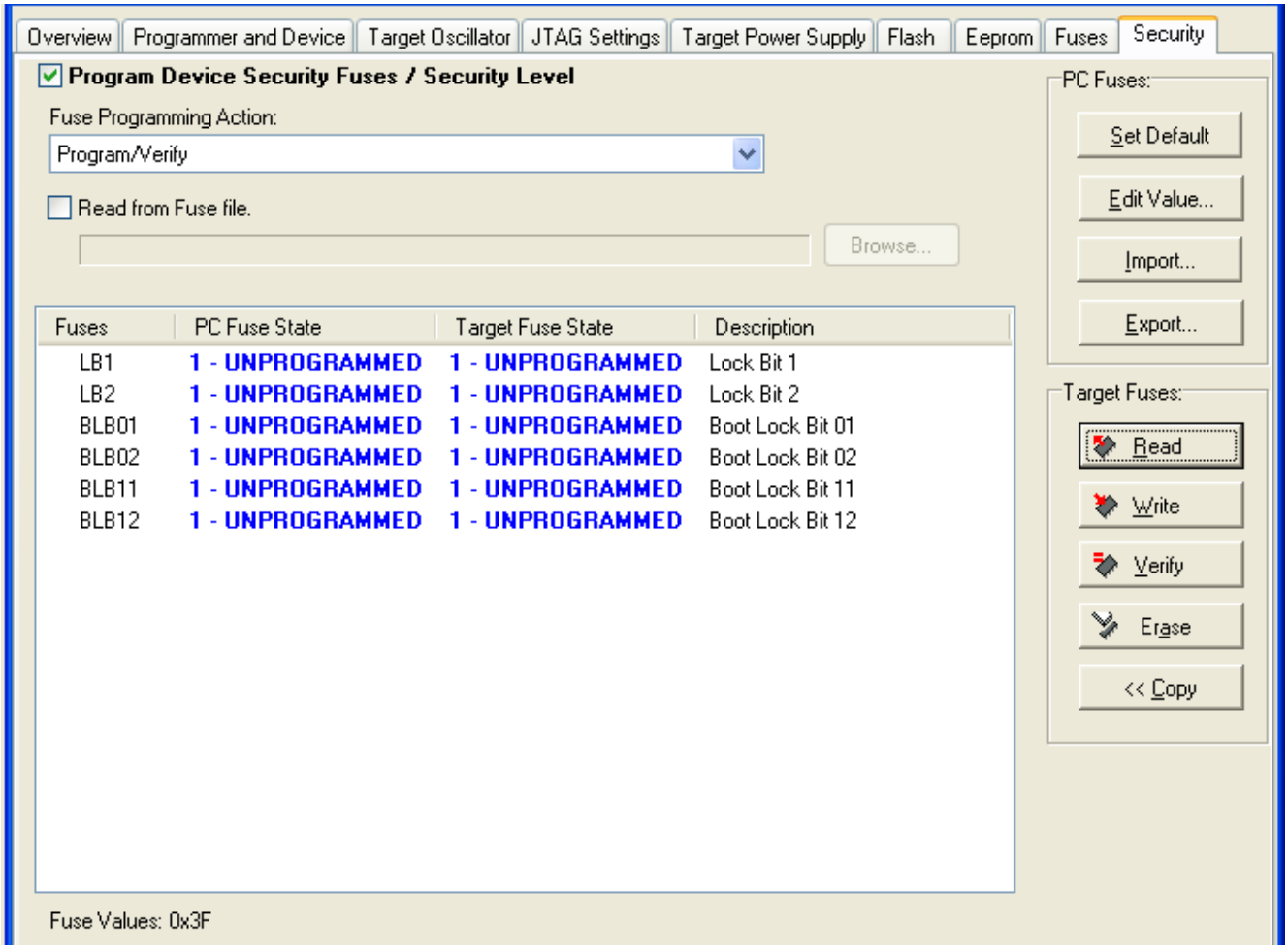
3.14.2 Reading the Security Fuses from a Target Device

To read the Security Fuse Values from a Target Device:

- Click the **<Read>** button
- The Hex values of the 'Fuse Bytes' which are read back are displayed as follows:



- The Security Fuse values from the Target Device are now displayed in the 'Target Fuse State' column.



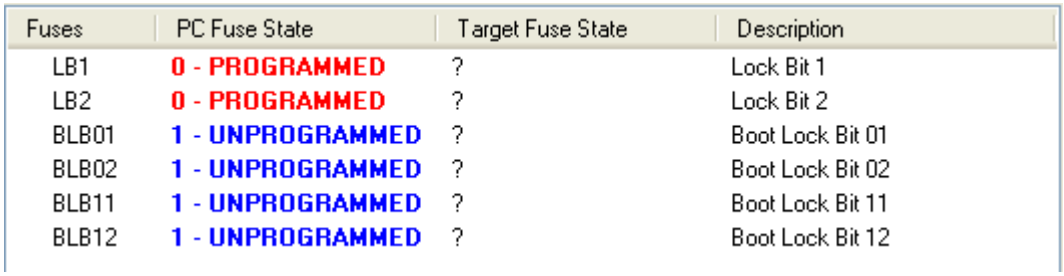
3.14.3 Verifying the Security Fuses of a Target Device

To verify the ***Security Fuse Values*** in a Target Device with the Fuse Values in the ***'PC Fuse State'*** column:

- Click the **<Verify>** button
- Any differences in the Fuse Values between the PC settings and the Target Device setting will now be displayed.

3.14.4 Writing the Security Fuses into a Target Device

- To stop anyone from reading / copying the contents of AVR devices, it is usual practice to 'Lock' the device at the end of the programming sequence.
- To lock the FLASH and EEPROM from being read back, set the 'LB1' and 'LB2' Lock Bits to '0'.
- To program the Fuses values in the 'PC Fuse State' column into the Target Device, click the **<Write>** button



Fuses	PC Fuse State	Target Fuse State	Description
LB1	0 - PROGRAMMED	?	Lock Bit 1
LB2	0 - PROGRAMMED	?	Lock Bit 2
BLB01	1 - UNPROGRAMMED	?	Boot Lock Bit 01
BLB02	1 - UNPROGRAMMED	?	Boot Lock Bit 02
BLB11	1 - UNPROGRAMMED	?	Boot Lock Bit 11
BLB12	1 - UNPROGRAMMED	?	Boot Lock Bit 12

- The programmer will now program all the Security Fuses at the same time and then read them back and verify them with the values in the 'PC Fuse State' column.
- The programmer will then report a PASS or FAIL for programming of the Security Fuses.
- The Target Device is now locked

Please note:

- Once the Lock Bits have been programmed on an AVR Device, it is then no longer possible to read or re-program the FLASH or EEPROM memory areas.
- The Configuration Fuses and Security Fuses can usually still be read from a Target Device even if the device is locked.

3.14.5 Erasing the Security Fuses

The only way to change a Security Fuse from a '0' to a '1' is to perform a 'Chip Erase' operation. This will erase the FLASH / EEPROM and then finally erase the Security Fuses and set them back to a value of '1'.

3.14.6 Using a 'Fuse File' to import Security Fuse settings into a project

It is possible to export the ***'Fuse Values'*** for a particular device to a so-called ***'Fuse File'*** so that a single copy of the fuses is stored in one place. This ***'Fuse File'*** can then be shared amongst many projects if required. See section 6 for further details about using ***'Fuse Files'*** .

3.14.7 Importing Security Fuse Settings in HEX format from AVR Studio

If the original firmware for your project has been developed using the Atmel ***'AVR Studio'*** software, then it is likely that the ***'Security Fuse Settings'*** are defined as ***'Hex Fuse Bytes'***. Please refer to Section 5 for instructions on how to import these ***'Hex Fuse Bytes'*** into your EQTools project.

3.15 Exporting an EDS Project to a Standalone Project

Once you have fully tested your EDS Development Project, it is possible to add the project to a Project Collection and then upload it to a programmer as a so-called ***'Standalone Project'***. The project can then be executed on a programmer without requiring any form of PC control.

Please follow the instructions detailed in Section 6 to upload your EDS project to a programmer.

4.0 Exporting / Importing Fuse Settings to / from File

4.1 Overview

One of the new power features of EQTools is the ability to export the ***'Fuse Settings'*** for a Programming Project to a ***Fuse File (*.eff)***. This allows the settings for all the fuses to be contained in one Fuse File which can then be imported into any of the Fuse tabs in Project Manager or in the ***<Fuses>*** tab in EDS. In this way, the values of all the Fuses for a particular project can be shared with other projects. This helps to ensure that the correct fuse values are specified in all projects.

4.2 Exporting the Fuse Settings to a Fuse File

To export the settings of the 'Local Fuses' column to a fuse File:

- Select the EDS ***<Fuses>*** Tab
- Set up the ***'Local Fuses'*** to the correct values for your Target Device.
- Click the ***<Export>*** button → a file browser is displayed.
- Enter a suitable name for your Fuse File eg. ***project_fuses.eff***
- Click ***<Save>*** → The ***'Local Fuses'*** column is transferred to your specified ***Fuse File (*.eff)***.

4.3 Copying the Fuses from a Target Device

To copy the Fuses from the Target Device to a Fuse File:

- Select the EDS ***<Fuses>*** Tab
- Click the ***<Read>*** button
  → the Fuses are read from the Target Device and are then displayed in the ***'Target State'*** column.
- Click the ***<<Copy*** button
  → the Fuse settings read from the Target Device are copied into the ***'Local State'*** fuse column.
- Click the ***<Export>*** button → a file browser is displayed.
- Enter a suitable name for your Fuse File eg. ***project_fuses.eff***
- Click ***<Save>*** → The ***'Local Fuses'*** column is transferred to your specified ***Fuse File (*.eff)***.

4.4 Importing the Fuse Settings from a Fuse File

To import the settings of the ***'Local Fuses'*** column from a Fuse File:

- Select the EDS ***<Fuses>*** Tab
- Click the ***<Import>*** button → a file browser appears
- Browse to and select your ***Fuse File (*.eff)***
  → The Fuse settings are then automatically copied from the ***Fuse File*** to the ***'Local Fuses'*** column.
- To program these settings into a Target Device, click ***<Write>.***

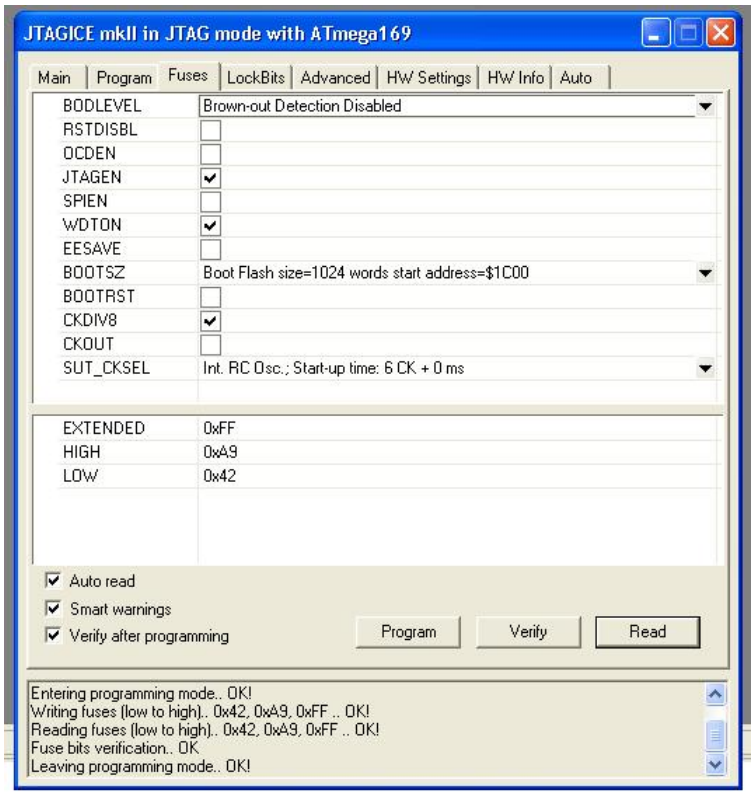
5.0 Importing Fuse Settings in HEX format from AVR Studio

5.1 Overview

If the original firmware for your project has been developed using the Atmel ***'AVR Studio'*** software then it is likely that both the AVR ***'Configuration Fuse settings'*** and the ***'Security Fuse Settings'*** are defined as so-called ***'Hex Fuse Bytes'***. This is the raw version of the fuses where each ***'Hex Fuse Byte'*** can represent up to 8 individual 'Boolean Fuses'. It is possible to import the ***'Hex Fuse Bytes'*** from AVR Studio into an EQTools project by following the instructions in the next section.

5.2 Finding the AVR Studio 'Hex Fuse Values'

In the Atmel 'AVR Studio' software, the ***'Configuration Fuse settings'*** for your project are displayed on the ***<Fuses>*** tab – see screenshot below.



The ***'AVR Studio'*** software displays a high-level overview of the fuses, grouping similar fuses together with more meaningful group names eg. ***'BODELEVEL'*** is made up of two fuses: BODLEVEL0 and BODLEV1 and the fuse ***'SUT_CKSEL'*** actually represents the following six Boolean fuses: SUT0, SUT1, CKSEL0, CKSEL1, CKSEL2, CKSEL3.

These Fuse values are then converted by AVR Studio into ***'Hex Fuse Bytes'.*** For AVR microcontrollers, the ***'Hex Fuse Bytes'*** are called ***'LOW', 'HIGH'*** and ***'EXTENDED'*** – see screenshot from AVR Studio above.

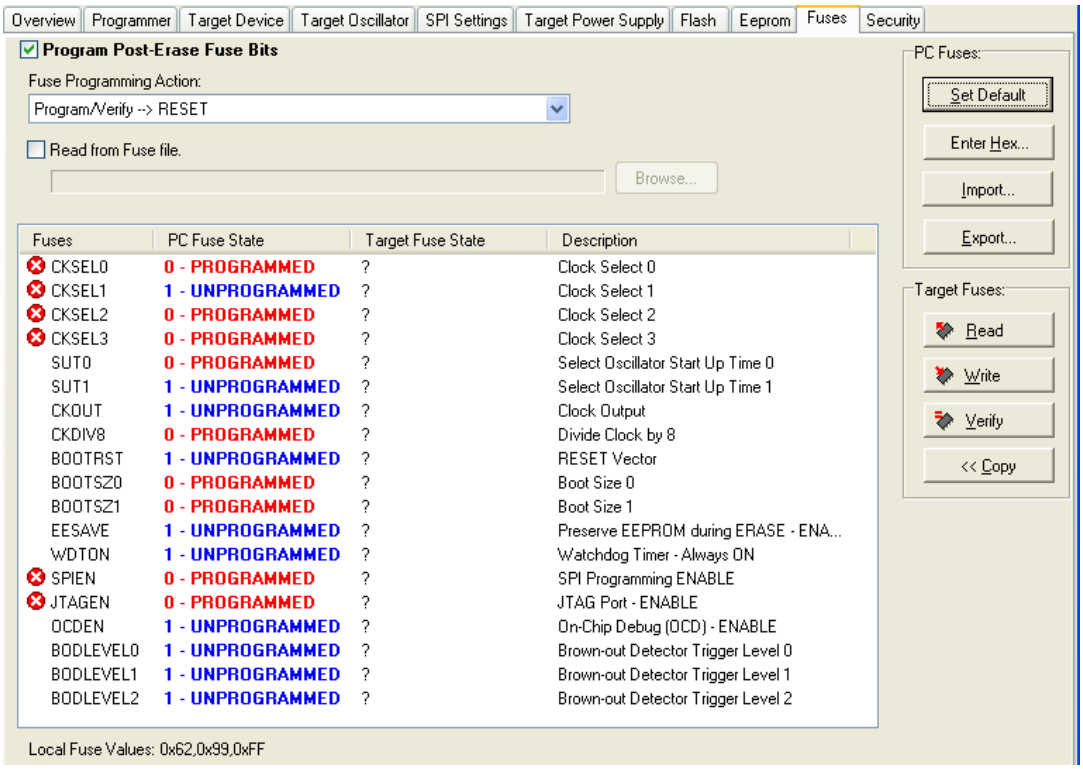
In this example, the ***'Hex Fuse Bytes'*** are as follows:
EXTENDED: 0xFF
HIGH: 0xA9
LOW: 0x42

5.3 Importing the AVR Studio 'Hex Fuse Values' into EQTools

It is possible to import the AVR ***'Hex Fuse Bytes'*** from AVR Studio into EQTools. This functionality requires that EQTools build 927 or above is installed.

Instructions:

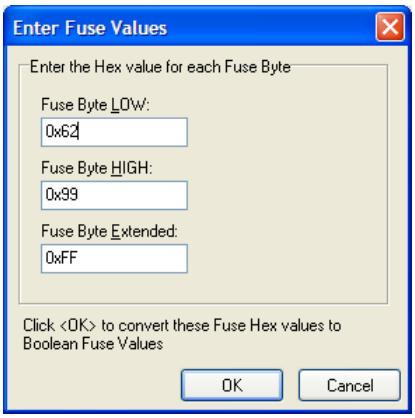
1. **Launch your project in EDS (Development Mode)**
   → Launch your project in EDS (Development Mode) and then select the ***<Fuses>*** tab.



- The default 'library' settings for the Fuses are displayed in the ***'PC Fuse State'*** column. These fuse values represent a virgin AVR device which has never been programmed before.
- The ***'Local Fuse Values'*** in Hex format are displayed at the bottom of the window. These values represent the current settings of the fuses in the 'PC Fuse State' column.
- The ***'Local Fuse Values'*** are displayed in the following order: ***LOW, HIGH, EXTENDED***

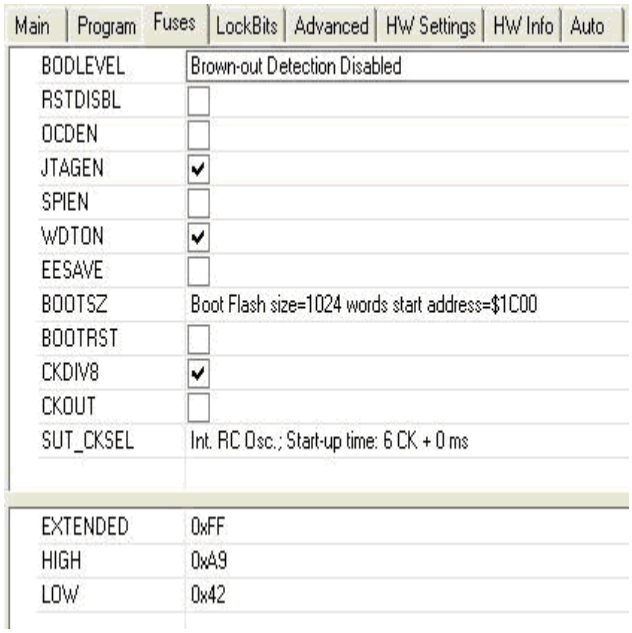
2. Click the <Enter Hex> button

- The ***'Enter Fuse Values'*** dialog box is displayed:



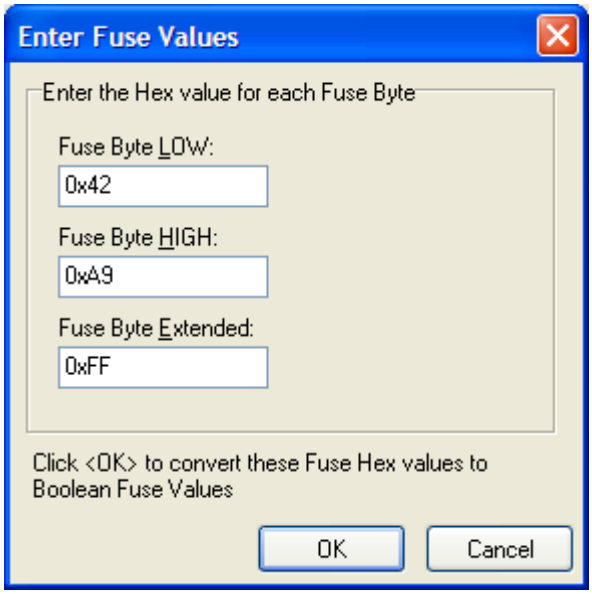
- The Hex values displayed as default are the values corresponding to the default Fuse Settings already specified in EQTools.
- Enter the ***'Hex Fuse Bytes'*** from AVR Studio in the relevant Fuse Value fields***: LOW, HIGH, EXTENDED***. These fields correspond to the same fuse field values in AVR Studio – see example below.

From AVR Studio – Fuses tab:

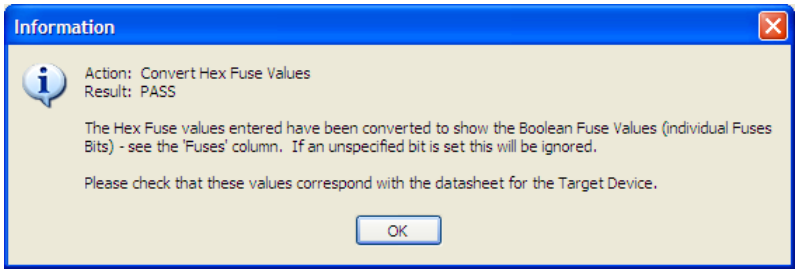


In this example, ***'Hex Fuse Bytes'*** are as follows:
EXTENDED: 0xFF
HIGH: 0xA9
LOW: 0x42

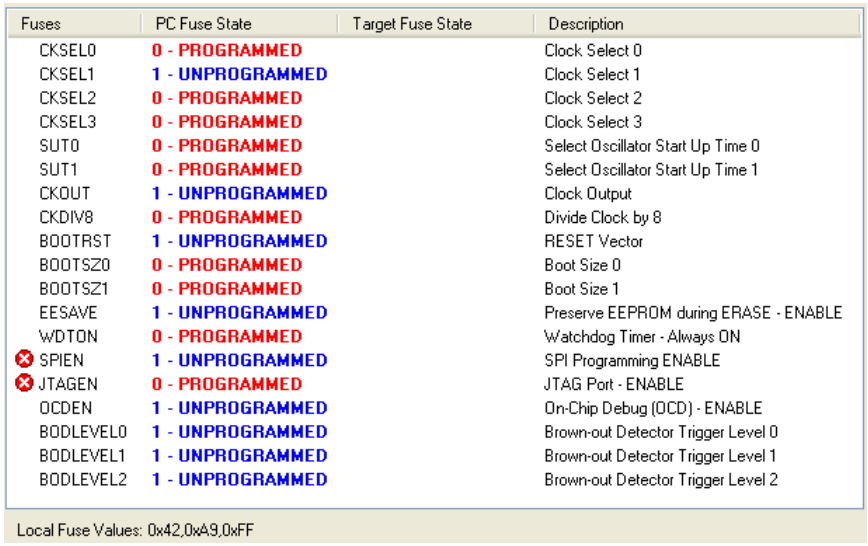
Enter into EQTools – Enter Hex values…..



- Click the ***<OK>*** button to accept the Fuse values
- EQTools will then convert these bytes into the Individual Boolean Fuse values.



→ The Hex values which you entered are now converted to the corresponding individual Boolean fuses for each Fuse Byte.



Fuses	PC Fuse State	Target Fuse State	Description
CKSEL0	0 - PROGRAMMED		Clock Select 0
CKSEL1	1 - UNPROGRAMMED		Clock Select 1
CKSEL2	0 - PROGRAMMED		Clock Select 2
CKSEL3	0 - PROGRAMMED		Clock Select 3
SUT0	0 - PROGRAMMED		Select Oscillator Start Up Time 0
SUT1	0 - PROGRAMMED		Select Oscillator Start Up Time 1
CKOUT	1 - UNPROGRAMMED		Clock Output
CKDIV8	0 - PROGRAMMED		Divide Clock by 8
BOOTRST	1 - UNPROGRAMMED		RESET Vector
BOOTSZ0	0 - PROGRAMMED		Boot Size 0
BOOTSZ1	0 - PROGRAMMED		Boot Size 1
EESAVE	1 - UNPROGRAMMED		Preserve EEPROM during ERASE - ENABLE
WDTON	0 - PROGRAMMED		Watchdog Timer - Always ON
SPIEN	1 - UNPROGRAMMED		SPI Programming ENABLE
JTAGEN	0 - PROGRAMMED		JTAG Port - ENABLE
OCDEN	1 - UNPROGRAMMED		On-Chip Debug (OCD) - ENABLE
BODLEVEL0	1 - UNPROGRAMMED		Brown-out Detector Trigger Level 0
BODLEVEL1	1 - UNPROGRAMMED		Brown-out Detector Trigger Level 1
BODLEVEL2	1 - UNPROGRAMMED		Brown-out Detector Trigger Level 2

Local Fuse Values: 0x42,0xA9,0xFF

- The ***'Local Fuse Values'*** represent the ***'Hex Fuse Values'*** and they should have the same values as the Fuse Bytes specified in the 'AVR Studio' project.

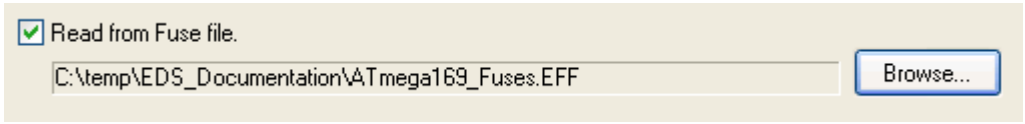
3. Export the 'PC State Fuses' to a Fuse File

It is possible to export these Fuse Settings to a ***'Fuse File'*** as follows:

- Click the ***<Export>*** button
- Save the fuse settings with a suitable name e.g. ***ATmega169_SPI_Fuses.eff***

4. Read the fuses from the Fuse file

- Once you have exported the Fuse Settings to a ***Fuse File***, you can then include these Fuse Settings in any project.
- In EDS, on the **<Fuses>** tab, tick the ***'Read from Fuse File'*** check box and then browse to and select your Fuse File.



- The project will then automatically use the Fuse Settings in the specified ***Fuse File***.
- The ***Fuse File*** can also be used by any other project allowing the fuse values to be shared between many projects if required.

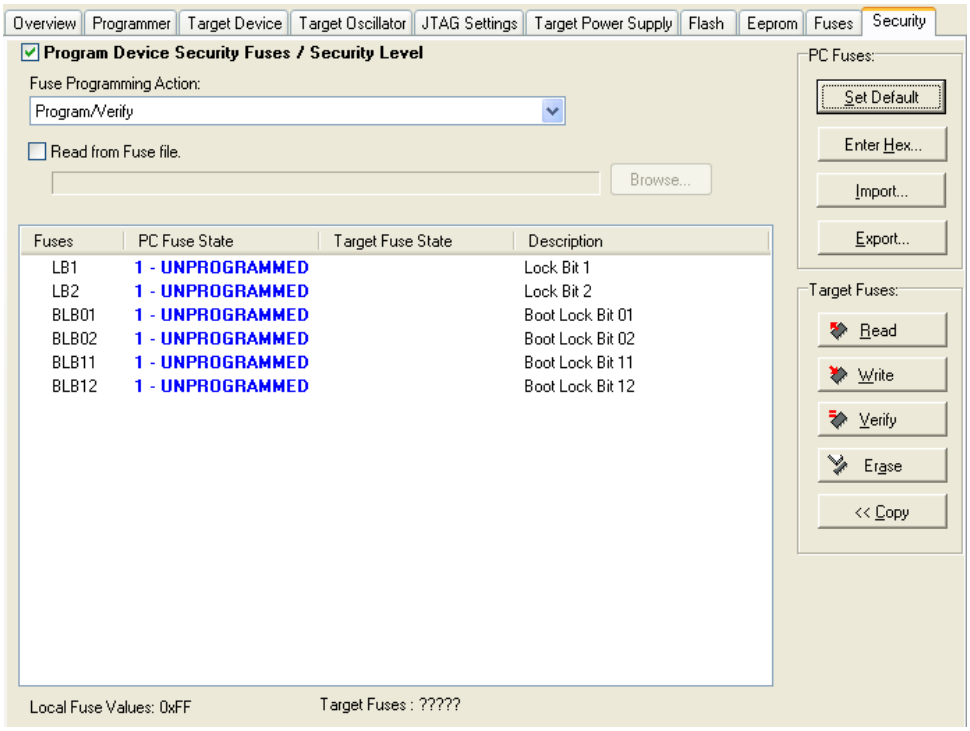
5.4 Importing the AVR Studio 'Hex Security Fuse Values' into EQTools

In AVR Studio, the ***'Security Fuse settings'*** for your project are displayed on the ***<Lock Bits>*** tab and will be defined as one or more ***'Hex Security / Lock Bytes'.*** This is the raw version of the fuses where each ***'Hex Security Fuse Byte'*** can represent up to 8 individual ***'Boolean Fuses'***.

It is possible to import the AVR ***'Hex Security Fuse Bytes'*** from AVR Studio into EQTools. This functionality requires that EQTools build 927 or above is installed.

1. Launch your project in EDS (Development Mode)

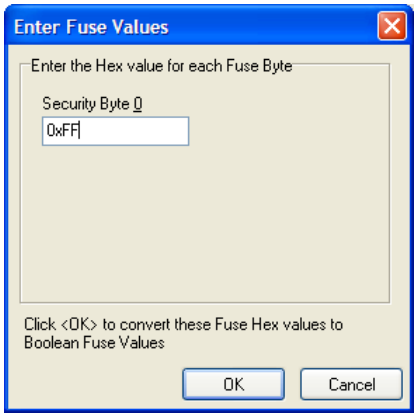
- Launch your project in EDS (Development Mode) and then select the ***<Security>*** tab.



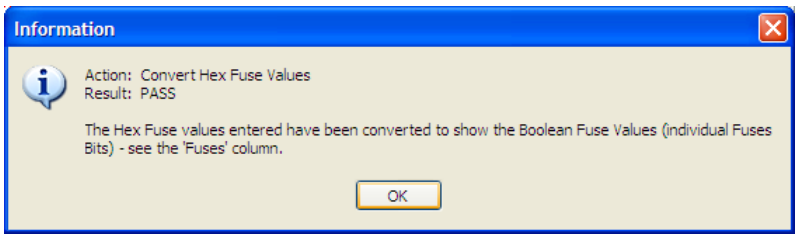
- The default 'library' settings for the Fuses are displayed in the 'PC Fuse State' column.
- These fuse values represent a virgin AVR device which has never been programmed before which should be "unlocked" ie all Lock Bits are set to '1'.

2. Click the <Enter Hex> button

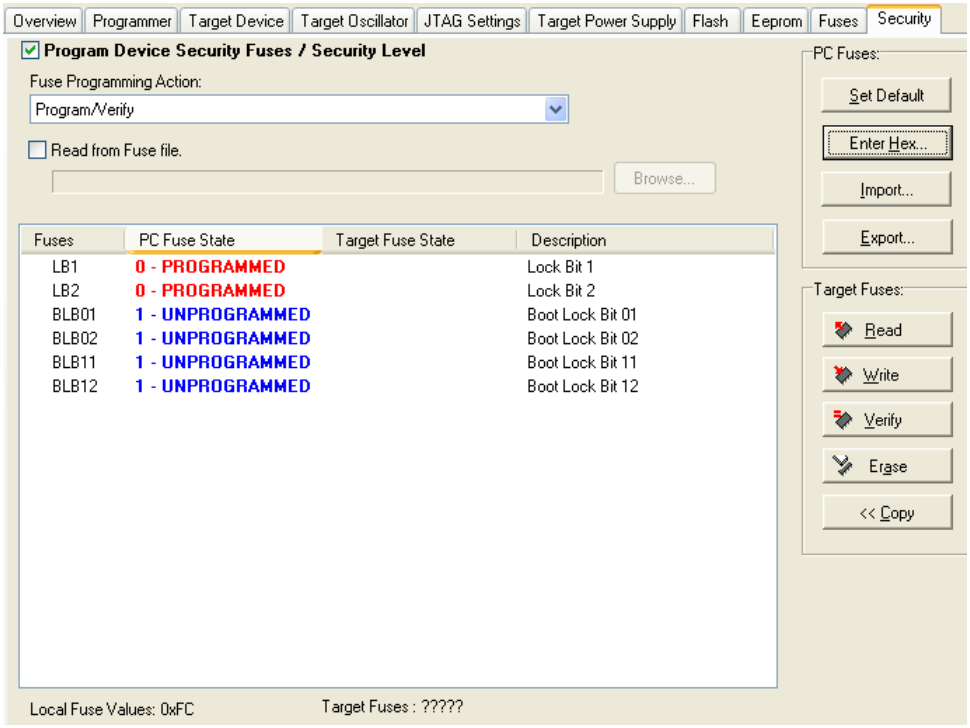
- The ***'Enter Fuse Values'*** dialog box is displayed:



- The value(s) displayed as default are the values corresponding to the individual Boolean Fuse Settings already specified in EQTools.
- Enter the ***'Security Hex Fuse Byte(s)'*** from AVR Studio in the relevant Fuse Value field(s)***.***
- Click ***<OK>*** to convert the Hex value(s) to Boolean fuses.



- Click ***<OK>*** → the individual Boolean Security fuses are now displayed:

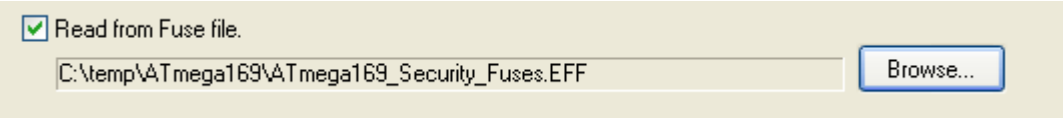


3. Export Security Fuses to a Fuse File

- Click ***<Export>*** and then save the Security Fuses to a Fuse File called eg. ***ATmega169_Security.eff***.
- This file can then be automatically read back into your programming project by selecting ***'Read from Fuse File'*** and then specifying the relevant ***Fuse File***.

4. Reading the Security Fuses from a Fuse file

- Once you have exported the Security Fuse Settings to a ***Fuse File***, you can then include these Fuse Settings in any project.
- In EDS, on the **<Security>** tab, tick the ***'Read from Fuse File'*** check box and then browse to and select your Fuse File.



- The project will then automatically use the Fuse Settings in the specified Fuse File.
- The '***Security Fuse File'*** can also be used by any other project allowing the fuse values to be shared between many projects if required.

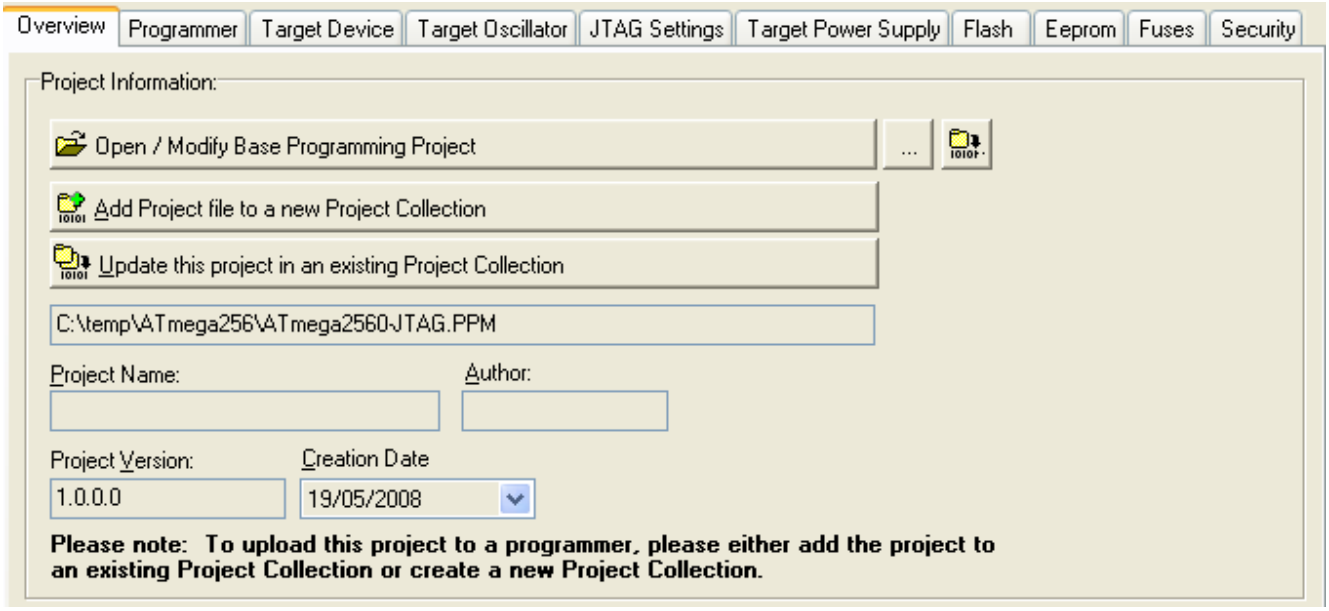
6.0 Creating a Standalone Project

6.1 Overview

Once you have tested your project fully in ***EDS*** (Development Mode), it is possible to then make this project into a ***'Standalone Project'*** which can be uploaded to a programmer. This single standalone project file (*.prj) will contain all the information required to program the Target Device including FLASH file, EEPROM file, Fuse settings, Security Settings etc.

6.2 Creating a Standalone Project from EDS (Development Mode)

In EDS (Development Mode), select the ***<Overview>*** tab

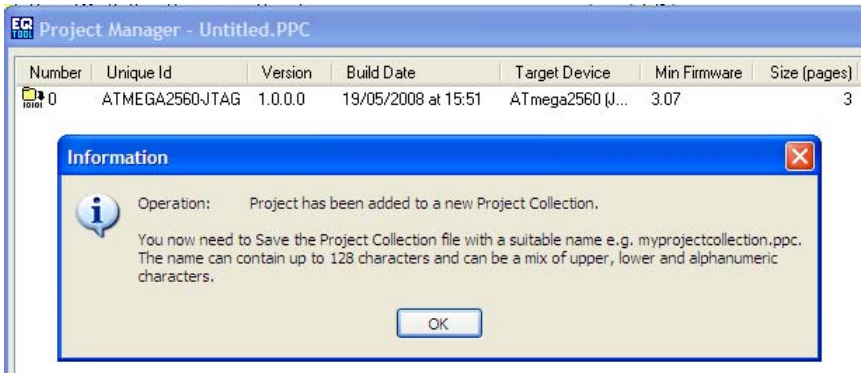


- If this is the first time the EDS Project has been uploaded to a programmer, click the ***<Add Project File to a new Project Collection>*** button.
- If the EDS Project has already been uploaded to a programmer before, click the ***<Update this project in an existing Project Collection>*** button.

6.3 Add Project File to a new Project Collection

When the ***<Add Project File to a new Project Collection>*** button is pressed, the EDS project will be automatically added to a new ***'Project Collection'***.

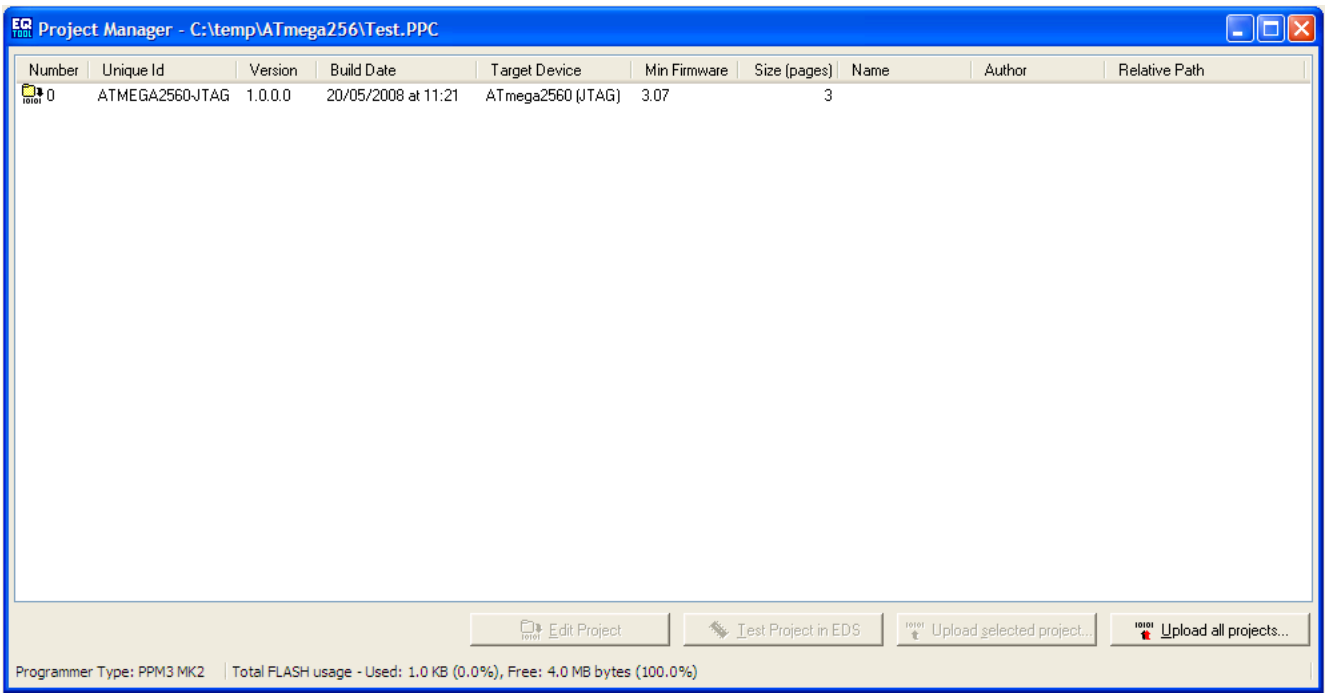
- The EDS Project will appear in a ***'Project Manager'*** window.
- You will then be prompted to save the ***'Project Collection'***. Choose a suitable name eg. ***Test.ppc*** and click the ***<Save>*** button.



The Project Manager window is now displayed – see section 6.4.

6.4 Uploading a Project to a programmer

The ***Project Manager*** window displays all the projects in your ***Project Collection***.



In this example we have only one project called ***'ATMEGA2560-JTAG'***.
The ***'Unique ID'*** is the ***'Project Name'*** which is also the file name you saved the project with in EDS.

To upload the project to the programmer:

- Click the ***<Upload all projects>*** button
  → uploads all the projects in the collection to the programmer.

or

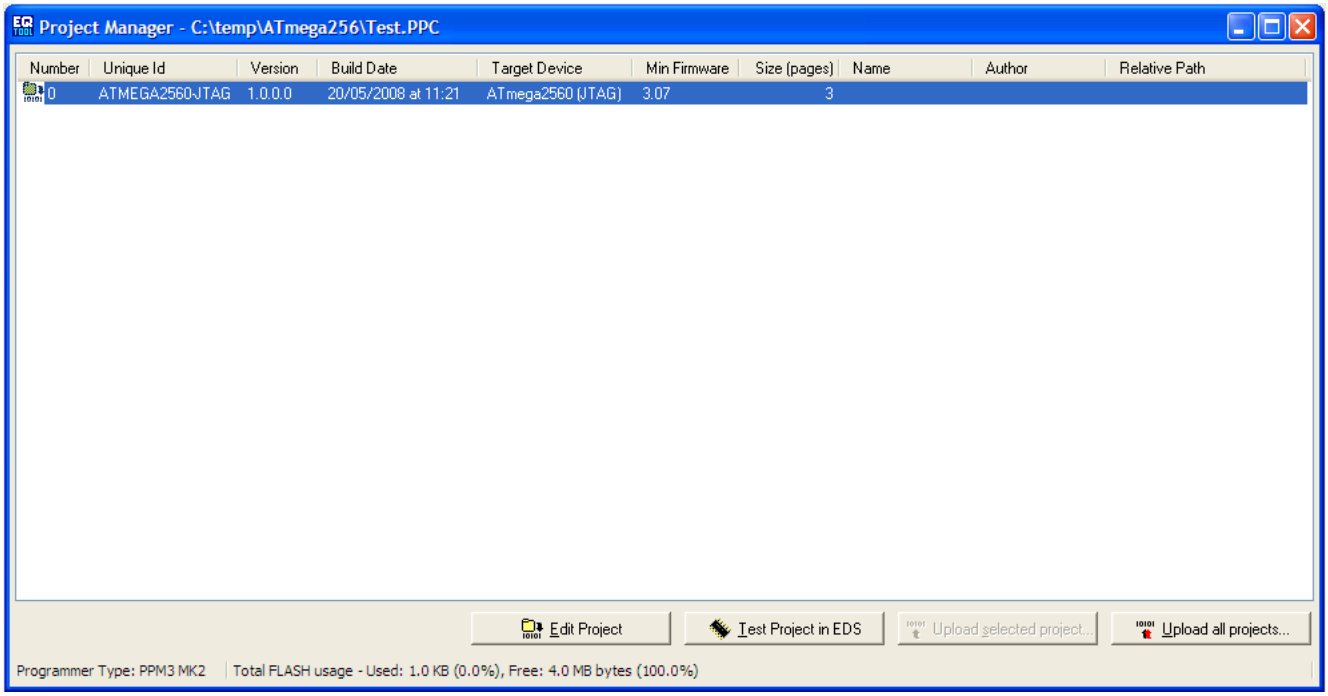
- Click once on the project you wish to upload in the ***Project Manager*** window and then click the ***<Upload selected project>*** button
  → uploads only the selected project in the collection to the programmer.

Follow the on-screen ***Upload Wizard*** instructions to complete the uploading of the project(s) to the programmer(s).

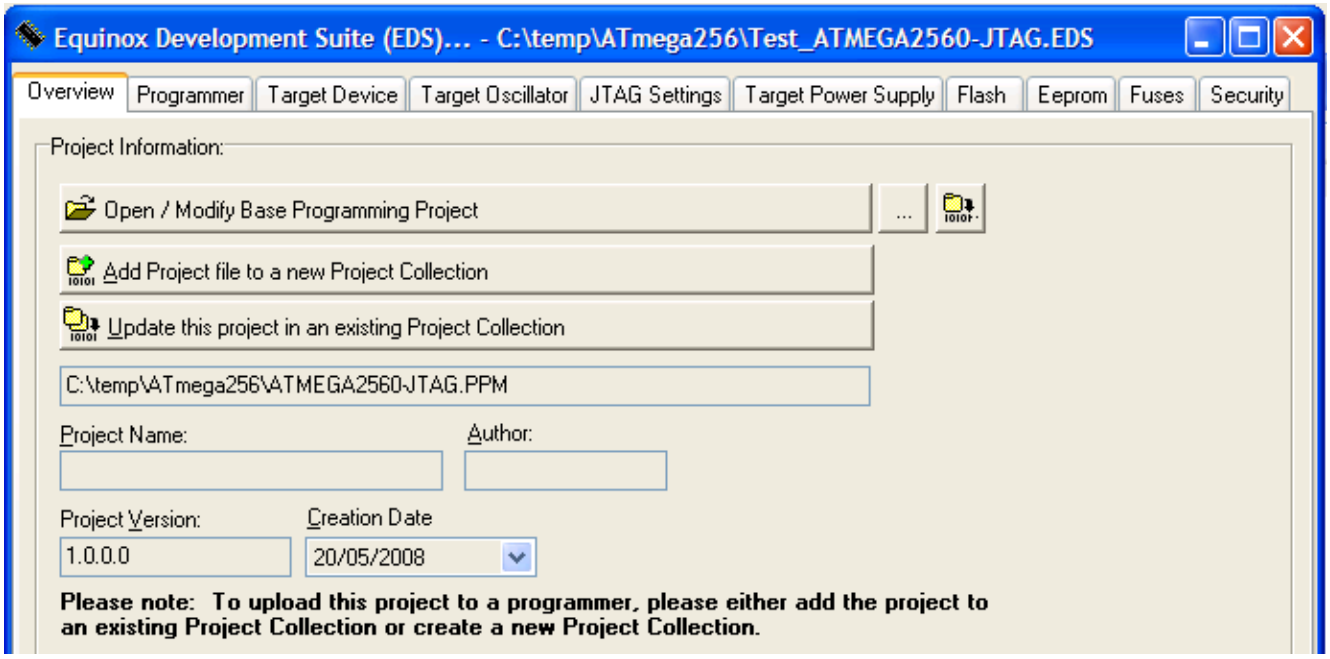
6.5 Re-testing a Project in EDS (Development mode)

If you want to re-test a Programming Project in EDS (Development Mode), the simplest method to do this is as follows:

- Use ***Project Manager*** to open your ***Project Collection (*.ppc file)***
- Click once on the project which you wish to test in EDS mode. This will select the project.



- Click the ***<Test in EDS>*** button
  → The selected project will now be opened in EDS (Development Mode).



- You can now test your project in EDS (Development Mode).