Serial SPI Memory Programming for ARM® devices

USER GUIDE

ARM CONNECTED

Purchase Agreement

P&E Microcomputer Systems, Inc. reserves the right to make changes without further notice to any products herein to improve reliability, function, or design. P&E Microcomputer Systems, Inc. does not assume any liability arising out of the application or use of any product or circuit described herein.

This software and accompanying documentation are protected by United States Copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted.

All the software described in this document is copyrighted by P&E Microcomputer Systems, Inc. Copyright notices have been included in the software.

P&E Microcomputer Systems authorizes you to make archival copies of the software and documentation for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this software or documentation for the purpose of distribution to others. Under no conditions may you remove the copyright notices from this software or documentation.

This software may be used by one person on as many computers as that person uses, provided that the software is never used on two computers at the same time. P&E expects that group programming projects making use of this software will purchase a copy of the software and documentation for each user in the group. Contact P&E for volume discounts and site licensing agreements.

P&E Microcomputer Systems does not assume any liability for the use of this software beyond the original purchase price of the software. In no event will P&E Microcomputer Systems be liable for additional damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use these programs, even if P&E Microcomputer Systems has been advised of the possibility of such damage.

By using this software, you accept the terms of this agreement.

P&E Microcomputer Systems, Inc.
98 Galen St.
Watertown, MA 02472
617-923-0053
http://www.pemicro.com

Manual version 1.01

December 2018

1 Introduction

PEmicro provides algorithms for programming SPI serial Flash, EEPROM, FRAM, nvRAM, MRAM, RAM, and other memories using PEmicro's GUI programming software PROGACMP [3.] or command-line programming software CPROGACMP [20.] via any of PEmicro's hardware interfaces for ARM® microcontrollers. See PEmicro's website, www.pemicro.com, for various interface cables and standalone programming hardware which support ARM controllers. PEmicro already provides support for the internal flash memory of many ARM microcontrollers from various manufacturers. In one simple procedure, the internal flash (for supported manufacturers) as well as any external memories connected via an SPI interface can be programmed.

PEmicro's SPI serial algorithms are designed to use any four I/O port pins to talk to SPI serial memory devices. This makes it easy for the user to configure these algorithms for their particular hardware set up. The user simply specifies the port addresses and pin numbers for each of the four SPI interface signals. The user configures their device using 8, 16, and 32 bit memory write commands that are inserted into the programming algorithm header. These commands are used to turn off the watchdog, set up the clock, and turn on and initialize I/O pins. The information required to do this is readily available from the code used to configure the device during normal operation. Examples of this procedure are included in this manual.

PEmicro's SPI serial ARM programming algorithms have been tested on ARM Cortex-M0, M0+, M3, and M4 controllers from various vendors. They should also work on other ARM processors which have the same minimum instruction set and debug interface. In addition to this basic requirement, the Cortex-M microcontroller connected to the external memory must have a minimum of 2 kB of RAM to load and then execute the programming routines.

All the algorithms supplied by PEmicro for programming SPI memory devices automatically verify that the data is programmed properly during the programming process, hence it is not necessary to do a separate verify command. However, a separate verify command is still available if desired.

Currently, PEmicro has well over 1100 algorithms for various vendors' serial SPI parts. If you do not see what you need, simply create a Support Request on PEmicro's website for a new or modified algorithm.

2 Serial Peripheral Interfaces.

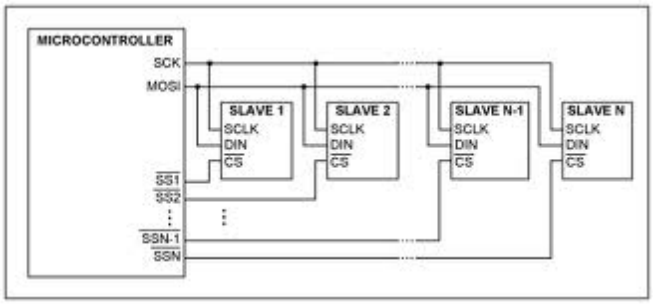
The Serial Peripheral Interface (SPI) is a very simple Master/Slave serial protocol used to talk to external devices attached to processors. The interface consists of only four pins:

- Clock line (CLK)
- Serial output (MOSI)
- Serial input (MISO)
- Slave select (/SS)

The master controls the clock line, CLK, which is used to time the transfer of data - usually eight bits at a time, most significant bit first, from the master to the slave and from the slave to the master. Data from the master to the slave is output on the MOSI high line and data from the slave to the master is on the MISO line. The slave select line (/SS) is used by the master to turn on the slave device and control the framing of messages to and from the slaves.

In the case of Flash/EEPROM programming, the master is always the local processor/microcontroller and the slave device is always the serial Flash/EEPROM to be programmed. Shown below is a typical setup diagram with the master and several slaves.

Figure 2-1: Master/Slave Setup Example

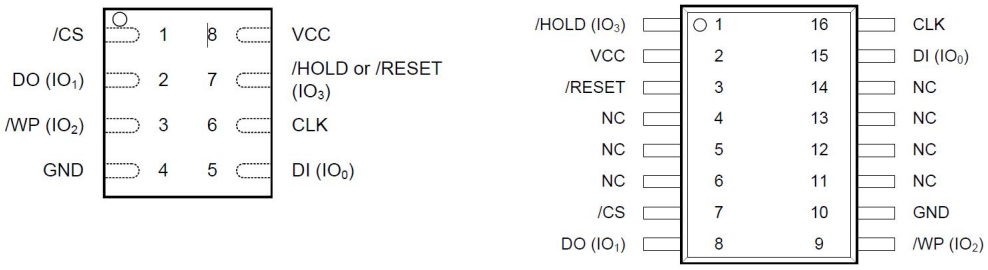


Historically, the SPI interface has been used to transfer a single bit per clock pulse, to and from the master and slave. Recently the SPI interface has been expanded to provide for multiple bits to be transferred on a single clock pulse. These interfaces are called Dual and Quad SPI.

3 SPI Serial Memory Devices

These memory devices usually consist of Flash, EEPROM, nvRAM, FRAM, or RAM. In most cases, they come in either 8- or 16-bit packages as shown below [1.]:

Figure 3-1: 8- and 16-Bit Packages



Both the 8- and 16-bit packages can support Dual and Quad SPI transfers. However, all Flash and EEPROM devices can be programmed using just single-bit SPI transfers. The Dual and Quad SPI interfaces use different commands than the single SPI interface when programming the devices. The Dual and Quad SPI provide for higher-speed transfers, but as stated, all serial devices can be programmed using the single SPI configuration, even though the device is capable of either Dual or Quad communications. Notice that both the 8- and 16-pin packages have a common set of signals:

- /CS - Not Chip Select, active low, connected to /SS of the microcontroller
- D0 - Data Output, connected to MISO of the processor/microcontroller
- /WP - Not Write Protect, active low, pulled high during programming
- GND - Ground
- VCC - Power, voltage dependent on chip type
- /HOLD - /RESET on some parts, active low, pulled high during programming
- CLK - Clock, connected to SCLK of the processor/microcontroller
- DI - Data Input, connected to MOSI of the processor/microcontroller
- IO1, IO0 - Alternate designations used by Dual and Quad SPI devices

- IO3, IO2 - Alternate designations used by Quad SPI devices

The master processor/microcontroller programs the slave Flash/EEPROM slave device by sending a set of commands, addresses, and data to the slave. The processor/microcontroller reads the slave Flash/EEPROM device by sending a command and address to the slave and receiving back data. The commands used and their sequences are dependent on the particular Flash/EEPROM device and vary by manufacturer. In addition, there are commands, usually called page, sector, or block erase, that erase the entire Flash/EEPROM or erase only certain subdivisions of the flash. Names of the subsections of the flash vary greatly between different manufacturers. To prevent confusion, PEmicro uses the terms “A Blocks”, “B Blocks” and, “C Blocks” which are defined by quantity and block size in each individual programming algorithm. Many Flash/EEPROM devices such as the GigaDevice GD25Q10 [2.], have A Blocks which are uniform 4K bytes, B Blocks which are uniform 32K bytes, and C Blocks which are uniform 64K bytes.

4 PEmicro’s Serial SPI Algorithms

The programming algorithms (labeled with the extension .ARP) are composed of two parts, a header and a set of S-Records for the algorithm code. The header contains the manufacturer, part number, size, and where things are located on the microcontroller. The algorithms have structured names so that you can identify what part each algorithm programs.

For example, the file name: “Micron_M25P16A_8x2Meg_sw_spi.arp” means:

- Micron is the manufacturer
- M25P16A is the part number
- 8x2Meg is the part size, 2 megabytes, i.e. 16 megabits
- sw_spi indicates that it is a software SPI algorithm
- .arp indicates it is for an ARM processor

If you set up an algorithm to run on the particular microcontroller configuration, PEmicro suggests that you append the name of the microcontroller to the source file name to indicate this change. This is important since you may use the same algorithm in different microcontroller setups.

The header has additional information about the part to be programmed and its size. In addition, the header has sections where the user inputs information used in configuring the part for their application. The header is readable using a standard ASCII text editor/reader. For example, the range of parameters for different commands is usually specified with comments in the header file.

The header may include descriptions of up to three block erasing commands. When a block erase command is selected, it prompts the user for a beginning block number and an ending block number. Block numbers are input by the user in hex. These block numbers must range between zero and the total number of blocks -1. For example, if there are 32 blocks the user could assign beginning and ending block numbers from 0-1F.

This command will erase all blocks between the selected beginning block number and ending block number, inclusive. The programming software translates these hex block numbers into addresses used to erase the corresponding blocks in the device. A part’s block numbers are described in the manufacturer’s part specification sheet for the part and in a comment line just before the erase command. The header, shown below in **blue**, uses the erase commands and block structure for the GigaDevice GD25Q10 device [2.].

The S-Records contain the code and constant data for the programming algorithm which gets loaded into the RAM of the target ARM microcontroller. The S-Records appear at the end of the algorithm file as indicated below.

```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com [GigaDevice_GD25Q10]
;device GigaDevice, GD25Q10, 8x128k, desc=sw_spi
;begin_cs device=$00000000, length=$00020000, ram=$20000000
;
;*********************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
;
;*********************************************************************************
;Set up system clock source and frequency if different from defaults
;
;*********************************************************************************
;Turn on clock distribution and power to used I/O modules
;
;*********************************************************************************
;Set up serial part /WP and /HOLD to HIGH if necessary
;
;*********************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
;
;*********************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*********************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/
;
;A Blocks 32-4k  (0..1F)
USER=E1 Erase A blocks     2First A  >/00000000/0000001F/2Last A   >/00000000/0000001F/
;
;B Blocks 4-32k  (0..3)
USER=E2 Erase B blocks     1First B  >/00000000/00000003/1Last B   >/00000000/00000003/
;
;C Blocks 2-64k  (0..1)
USER=E3 Erase C blocks     1First C  >/00000000/00000001/1Last C   >/00000000/00000001/
```

```
;
       {   S-Records shown here are only partial   }
S023000047696761446576696365F474432355131305F73775F7370695F6D302E73313916
S315200000004406002060070020800000000000000059
S315200000100000200A30400200000000000000000F1
S31520000020000000008B040020DB04002000000000FC
…
S31520000650FC430040FC430040FC43004002030405E9
S70520000000DA
;
&#$@
```

5 Hardware Setup

For consistency and ease of operation, PEmicro's programming approach does not use any special on-chip hardware on your microcontroller. Instead, it assumes that your SPI serial memory device is connected to port pins of your microcontroller and a bit-banging algorithm is used to program the SPI memory device. This means you do not have to setup the SPI module of your microcontroller, but instead setup your port pins as if for normal GPIO operations. This is not a problem if you used internal microcontroller SPI hardware in your design. All port pins associated with the on-chip hardware are also configurable as I/O port pins. It is necessary to insert, in the programming algorithm header, definitions of which ports and port pins are used to connect your SPI serial memory to the microcontroller. This is accomplished by specifying the port addresses and pin numbers.

Port pins must be configured in the algorithm header with the correct direction, either input or output. The default direction of the I/O port pin may be incorrect for the associated SPI signal. For example, the MOSI signal needs to be configured as an output, and the MISO needs to be configured as an input. If the port pin direction is not properly configured, the transmission will be unsuccessful.

If necessary, any watchdog timers which are running after a reset need to be turned off by putting appropriate statements in the algorithm's header. For many microcontrollers this is not necessary since the watchdogs are not enabled after a reset. Some watchdogs can be disabled by a simple write to a memory register. In some circumstances, code must be run on the microcontroller to disable watchdog. A provision is made to do this within the header of the algorithm.

On some microcontrollers it is necessary to supply power or clock to some submodules before they can be used. This is accomplished by simple memory writes inserted in the header of the algorithm file.

Although it is not necessary, it may be desirable to set up the PLL (phase locked loop) or FFL (frequency locked loop) of the microcontroller since the data transfer rate between the microcontroller and the serial memory device increases with processor CPU clock speed. Completing this setup can significantly reduce the programming time for large serial memory devices. However, this setup is usually not worth the effort for small-to-medium serial memory devices. As mentioned later in this manual, it is important to note that in some cases the clock that is output to the serial device may actually be too fast. In this case you must slow down the serial clock so it does not exceed the maximum clock allowed for the serial device.

Before attempting to erase or program your SPI memory device, it may be necessary to un-protect the device. Please refer to the documentation with your SPI memory device for more details on how to do this.

This is all that is needed to set up a particular device-programming algorithm for your microcontroller-based hardware configuration. At first this may seem a little daunting, however if you look at the examples in this manual you will see that it is relatively straightforward. If you have any problems getting your setup to work, create a Support Request via PEmicro's website. If necessary, PEmicro can configure an algorithm for your hardware. You will need to provide the processor pins you are using, the name of the microcontroller, and the name of the serial memory device you are using. If you supply PEmicro with a test board, we can guarantee that it works properly.

6 Algorithm Configuration

In order to use these algorithms, the user must insert some configuration information into the header section of the algorithm file. The different sections of the header file are detailed below, section by section. The text from a typical header file is shown in `blue` below followed by an explanation of what they are for and what must be provided. Configuration information is sometimes written directly into the header form. Simple hexadecimal (hex) characters are input for data, addresses, and bit numbers, either directly or as memory writes, as shown below. There are three forms of memory writes. They are highly structured statements which must be entered exactly as shown including the exact number of hex digits and constant characters. Data ('d' characters) contain 8, 4, or 2 hex digits. Addresses ('a' characters) always have 8 hex characters. You can use these commands to write code into RAM memory if necessary to initialize the target device. In addition there is a command called SET_PC_AND_RUN shown below which allows you to execute such code at a given address. Your code should end in a breakpoint instruction, bkpt #0x00 (0xBE00). Another command, DELAY, causes a delay in milliseconds specified by the data. This is used when the microcontroller needs to wait for some period of time after initializing a piece of hardware. There are many other commands which could be placed in the header. These are documented in the individual programming software manuals. However, these are not needed to configure the serial SPI Flash/EEPROM algorithms.

```
WRITE_LONG=dddddddd/aaaaaaaa/
WRITE_WORD=dddd/aaaaaaaa/
WRITE_BYTE=dd/aaaaaaaa/
SET_PC_AND_RUN=aaaaaaaa/
DELAY=dddd/
```

These memory writes should begin in column 1 and not be preceded by any other characters. Comments can be added at the end of the memory writes lines by proceeding them with a ; character (semicolon).

6.1 Identification Section

```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com [GigaDevice_GD25Q10]
;device GigaDevice, GD25Q10, 8x128k, desc=sw_spi
;begin_cs device=$00000000, length=$00020000, ram=$20000000
;
```

The identification section contains only comments which describe the algorithm. All lines beginning with a ; (semicolon) are strictly comments and do not affect the operation of the algorithm.

`Version 1.00` shows the current version number of the algorithm.

`03/15/2014` shows the date the algorithm was created.

`Copyright P&E Microcomputer Systems, www.pemicro.com` is PEmicro's copyright information.

`[GigaDevice_GD25Q10]` is the source file name used by PEmicro to create the algorithm.

The `;device` line shows the manufacturer's name, part number, memory size, and the description. `desc=sw_spi`, indicates that it is a software SPI algorithm.

6.2 Turn Off Watchdog Timer Section

```
;*********************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
;
```

Here, the user should enter **memory writes** which turn off the watchdog timer, if one exists and is enabled at reset. If no watchdog timer is turned on at reset or there is no watchdog timer at all, nothing needs to be

entered.

6.3 Set Up System Clock and Frequency Section

```
;*********************************************************************
;Set up system clock source and frequency if different from defaults
;
```

If necessary, the user enters **memory writes** to choose the system clock source and set its frequency. The system clock frequency controls the speed of the SPI interface. Care should be taken such that the system clock frequency is not too high, depending on the particular memory part. The clock frequency can be tested using the test pattern function described below. You should make sure that the CLK signal to the SPI does not exceed the maximum frequency of the serial memory part. PEmicro suggests that you set the clock to a low frequency in your initial setup. This is often the default after a reset. Once you have the algorithm working on your hardware, then speed up the clock frequency until you reach either the maximum clock frequency of your microcontroller or the maximum SPI clock frequency of your SPI memory device.

6.4 Clock Distribution / Power Section

```
;*********************************************************************
;Turn on clock distribution and power to needed microcontroller modules
;
```

On some microcontrollers, the clock or power is turned off to certain sections and must be turned on during system initialization. If this is necessary, enter **memory writes** here to turn on power to those microcontroller modules such as ports, RAM, etc. which are necessary for the system to operate properly.

6.5 Write Protect and Hold Section

```
;*********************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
```

Many SPI serial memory devices have write protect and hold pins. If they are connected to the microcontroller then they must be initialized to the inactive high state so that the device can be programmed or erased. In many cases, the write protect and hold pins are tied high through pull-up resistors on the target hardware. In this case, there is nothing to do to set up these pins. However, if these pins on the SPI device are connected to port pins of the host microcontroller, it is necessary to set those port pins to outputs and set their values appropriately. This can be done using memory write statements in the header of the programming algorithm.

6.6 Ports and Direction Section

```
;*********************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
;
```

In some microcontrollers, the pins must be mapped to the I/O ports. In addition, most microcontrollers have their I/O pins initially set for input. Those pins which are outputs - CLK, /CS, and MOSI - must be configured as outputs. This is accomplished by inserting memory writes at this location.

6.7 Port Addresses and Pin Number Section

```
;*********************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
```

```
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
;
```

The programming algorithm software needs to know which microcontroller pins are connected to the serial SPI memory part. This is done by specifying the port address and port pin number for each pin CLK, /SS, DI, and D0 of the serial memory device. These statements are very strongly typed and need all the constant characters and number of characters exactly as shown.

6.8 Programmer Control Section

```
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
```

`;end_cs` represents the end of the chip set up area. The rest of the statements are needed by the programming software to set up proper operation for programming serial SPI memory devices and to assure that the necessary revision of the programming software is being used.

Warning: Do not remove or change these statements!

These statements are used to tell the PROG software how to set itself up to properly talk to the microcontroller and the attached SPI device. If you remove or change these statements, the programming algorithms will not operate properly.

`REQUIRES_PROG_VERSION=5.00/`

This is the revision number of PEmicro's programming software that is required for this algorithm to operate properly. If you have an earlier version number you should contact PEmicro to upgrade your programming software.

`NO_TIMING_TEST`

This tells the programming software that it is not required to test the speed at which the Microcontroller is running.

`PROGRAMMING_ALSO_DOES_VERIFY`

This tells the programming software to put a message on the screen that programming also just a verify the data as it is programed. This is always the case for serial flash devices and hence it is not necessary to do a separate verify unless It is required to double-check the programming operation.

`NO_BASE_ADDRESS`

This specifies that addressing always starts at address 0 within the serial memory device.

```
FIXED_ADDRESS_READ
```

This indicates that the serial memory device is not memory mapped within the address space of the microcontroller. It is always read at a fixed address.

6.9 Test Pattern Command Section

```
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
```

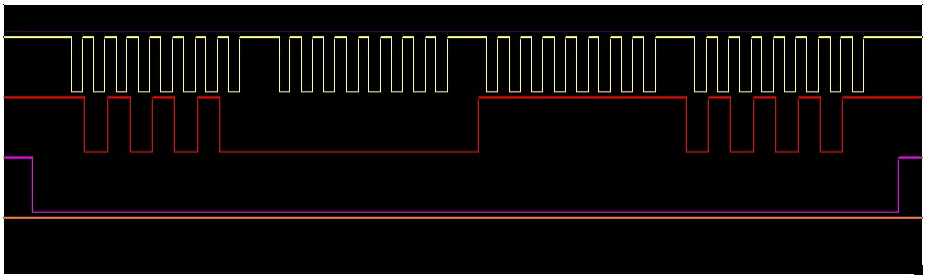
This command causes the programmer to send a sequence of four bytes out the SPI interface. The command has two parameters.

The first parameter consists of up to four bytes which are sent out sequentially, most significant byte first. The bytes are sent in a frame with the /CS line being pulled low, then the 4 bytes are sent, and then the /CS line is pulled inactive high.

The second parameter is a count of the number of times this pattern is repeated.

If you use this command with a large count, you can see the pattern repeat by using a logic analyzer or an oscilloscope and syncing on the /CS line. This allows you to determine the frequency of the CLK transfer rate. You can then use this to change the settings in your header to adjust the frequency. A sample picture of this frame which repeats the pattern AA00FF55 is shown below. The lines on the display are top to bottom yellow-CLK, red-MOSI(DI), pink-/SS(/CS), orange-MISO(DO). The MISO(DI) signal is always high since it was not connected and floats high.

Figure 6-1: AA00FF55 Test Pattern



Some SPI flash devices, such as the Atmel/Adesto 45DBxxx series, have special 4-byte commands which may not be implemented in PEmicro's standard programming algorithms. The test pattern command can be used to implement these commands. Even though the test pattern command always sends four bytes, it can be used to implement commands with less than four bytes. Simply send the command bytes first, followed by either FF (or 00) bytes. FF (or 00) bytes are not valid commands for SPI memory devices and hence they are ignored. For example, to send a typical write enable command (0x06) followed by a typical chip erase command (0xC7), use the following test commands with a repeat 1 time for each: 06FFFFFF followed by C7FFFFF. The reason that these commands need to be in separate test pattern commands is that both commands need to be separate transmissions with the chip select framing each separate transmission. Note that on most serial SPI parts you must do a write enable command before you can do a chip erase command.

6.10 Disable Protection Command Section

Most SPI memory devices have a command which allows programming protection to be turned off. Before programming or erasing the device you should usually make sure that any protections are turned off, otherwise sections of the device may not erase or program properly.

Warning: PEmicro does not automatically turn off protection because in some circumstances it may be required to leave certain sections protected.

On most devices, protection is turned off by writing 0 to the status register. However, some devices such as 45DBxx require a special sequence. This command uses the type of diable protection needed for the particular device.

```
; Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;;
```

6.11 Write Status Command Section

Most SPI memory devices have a command which allows the Status Register to be written. This is often used to turn on/off protection bits for the device. Before programming or erasing the device you should usually make sure that any protection bits are turned off, otherwise sections of the device may not erase or program properly.

Warning: PEmicro does not automatically turn off protection because in some circumstances it may be required to leave certain sections protected.

Some devices may not have this command.

```
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/;
```

6.12 Block Erase Commands Section

Many devices have the ability to erase “pages”, “sectors”, or “blocks” of memory - names for these various regions are not consistent across different manufacturers. To avoid naming problems, PEmicro refers to these region types as A, B, and C Blocks if they exist. Comments are given for each type of block available which indicate the region size and the allowed range for region numbers. The user can specify a whole range of blocks by providing a beginning and ending block number within the specified range when prompted. In the text below, the module has a size of 128 kB. By calling the Erase A Block user function, you can erase the external memory in blocks of size 4k, and naturally there are 32 of those blocks (as indicated by the range of 00000000 to 0000001F). All of the blocks are numbered starting with 0. A whole range of blocks can be erased by providing a first block number and a last block number in hexadecimal.

```
;A Blocks 32-4k  (0..1F)
USER=E1 Erase A blocks     2First A  >/00000000/0000001F/2Last A   >/00000000/0000001F/
;
;B Blocks 4-32k  (0..3)
USER=E2 Erase B blocks     1First B  >/00000000/00000003/1Last B   >/00000000/00000003/
;
;C Blocks 2-64k  (0..1)
USER=E3 Erase C blocks     1First C  >/00000000/00000001/1Last C   >/00000000/00000001/
;
```

6.13 S-Record Code Section

The S-Records, only partially shown here, represent the code, constants and tables of the algorithm.

```
S315200000004406002060070020800000000000000059
S315200000100000200A304002000000000000000000F1
S31520000020000000008B040020DB04002000000000FC
 …
S70520000000DA
```

6.14 Checksum Section

There is a checksum after the end of file mark for the algorithm. The checksum is done over all the characters of the programming algorithm including the S-Records. It is used by PEmicro to determine if the algorithm has been modified. If the algorithm has been changed in any way, PEmicro's programming software gives a warning message. However, the algorithm is still allowed to execute.

```
;
&*$@
```

7 Algorithm Header Setup Examples

Shown in this section are the headers for various PEmicro programming algorithms modified for different vendors' hardware platforms. The header of the original algorithm is shown in **blue**. Changes and additions made to support a particular platform setup are indicated in **red**. The setups range from easy to rather complex. However, they are all relatively simple to implement as can be seen below. There is nothing else to do to get going. If you are creating your own set up you can follow these examples to see how various microcontroller clocks and I/O pins are initialized. These examples also show you how various port configurations for the SPI flash are set up.

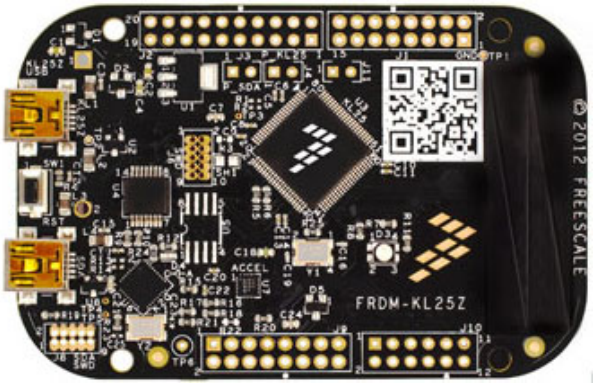
Section 7.4 - Toshiba TMPM364F10FG / Adesto AT25DF041A is a case where all the SPI pins are connected to different microcontroller ports. It is important to note that this choice of ports does not impact the performance of the programming algorithm.

Section 7.7 - Spansion MB9AF312 / Spansion S25FL164K also shows the use of the command-line programmer, CPROGACMP, to program both the microcontroller and the attached SPI Flash in one simple step.

7.1 NXP PKL25Z128VLK4 / Atmel AT25640B

This section shows the header modifications needed to run the algorithm for an Atmel AT25640B Serial SPI EEPROM Memory [4.] attached to the slave NXP Kinetis® MKL25Z128 microcontroller [5.] on a FRDM-KL25Z development Platform [6.].

Figure 7-1: FRDM-KL25Z Development Platform



Shown below is the original Atmel AT25640B header in **blue**. The changes made to attach to the MKL25Z128 microcontroller are shown in **red**. First, the watchdog is disabled. Then the internal clock oscillator is set up for 48 MHz. The clock is turned on for Port C. The address of Port C and the pin numbers are provided for all 4 SPI pins. All the SPI signals are on the same port. However, the output bit port address is different than the input bit port address since, on this microcontroller, the same port is read and written at different addresses. The Port C Data Direction Register (DDR) is set up for the bits which are outputs. PEmicro's programming software can now use this algorithm to talk to the KL25Z microcontroller with an Atmel AT25640B serial SPI EEPROM attached as indicated.

Before trying to program or erase this serial device you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

Actual measurements indicate that the bit rate between the microprocessor and the EEPROM is approximately 1.86 MegaBits per second. Since there is a small overhead delay between bytes, the actual byte rate turns out to be approximately 204 KiloBytes per second.

```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Atmel, AT25640B, 8x8k, proc=Freescale_Freedom_MKL25Z128VLK4, desc=sw_spi
;begin_cs device=$00000000, length=$00002000, ram=$20000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
WRITE_LONG=00000000/40048100/   ;SIM_COPC=0
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_BYTE=A0/40064003/             ;MCG_C4 48MHz
;
;*****************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
WRITE_LONG=00040982/40048038/   ;scgc5=portc clock on
;
;*****************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*****************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00000143/4004B008/   ;Set pin as port ptc2 - /ss
WRITE_LONG=00000143/4004B014/   ;Set pin as port ptc5 - clk
WRITE_LONG=00000143/4004B01C/   ;Set pin as port ptc7 - sdi
WRITE_LONG=00000143/4004B018/   ;Set pin as port ptc6 - sdo
WRITE_LONG=00000064/400FF094/   ;Set portc pins 2, 5, and 6 as outputs
```

```
;
;*****************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=3/400FF080/                ;SPI_/SS  pin Port  - ptc2 - /ss
PARAM=2/400FF080/                ;SPI_CLK  pin Port  - ptc5 - clk
PARAM=4/400FF090/                ;SPI_MISO pin Port  - ptc7 - sdi
PARAM=5/400FF080/                ;SPI_MOSI pin Port  - ptc6 - sd0
PARAM=6/06070205/                ;port pin number definitions
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/;
```

7.2 Texas Instruments® LM3S301 / Atmel AT25F1024

This section shows the header modifications needed to run the algorithm for an Atmel AT25F1024 Serial SPI Flash Memory [7.] attached to a Texas Instruments (Luminary Micro) LM3S301microcontroller [8.] on a Stellaris® Family Development Board [9.].

Shown below is the original Atmel AT25F1024 header in `blue`, and the changes made to attach to the LM3S301 microcontroller are shown in `red`. First, the watchdog is turned off. Then, the internal clock oscillator is set up for 12 MHz. The clock is turned on for Port A. The address of Port A and the pin numbers are provided for all 4 SPI pins. All the SPI signals are on the same port which corresponds to pins which could be used as a hardware SPI. The Port A Data Direction Register (DDR) is set up for the bits which are outputs. PEmicro's programming software can now use this algorithm to talk to the LM3S301 microcontroller with an Atmel AT25F1024 serial SPI Flash attached as indicated. Actual measurements indicate that the bit rate between the microprocessor and the Flash is approximately 476 KiloBits per second. Since there is an overhead delay between bytes, the actual byte rate turns out to be approximately 51 KiloBytes per second.

Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

Figure 7-2: Stellaris Family Development Board

```
;version 1.00, 11/05/2011, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Atmel, AT25F1024, 8x128k, proc=TI_LM3S301_Stellaris_DevBoard, desc=sw_spi
;begin_cs device=$00000000, length=$00020000, ram=$20000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
WRITE_LONG=00000000/400F2000/      ;Turn off watchdog
WRITE_LONG=00000000/400F2000/      ;Turn off watchdog
WRITE_LONG=000000B1/400F2004/
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_LONG=078E3AD0/400FE060/      ;Set internal default clock to 12MHz
;
;*****************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
WRITE_LONG=00000001/400FE108/      ;Turn on port A
;
;*****************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*****************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=0000002C/40004400/      ;Port A pins 2,3,5 outputs, pin 4 input
;
;*****************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used..
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                 ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                 ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                 ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=2/400043FC/                 ;SPI_CLK  pin Port  - PTA5
PARAM=3/400043FC/                 ;SPI_/SS  pin Port  - PTA2
PARAM=4/400043FC/                 ;SPI_MISO pin Port  - PTA7
PARAM=5/400043FC/                 ;SPI_MOSI pin Port  - PTA6
```

```
PARAM=6/06070205/                ;port pin number definitions
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/;
;A Blocks 4-32k (0..3)
USER=E1 Erase A            1First A  >/00000000/00000003/1Last A   >/00000000/00000003/
;
```

7.3 NXP K20DX128VFM5 / Adesto AT45DB161E-512

This section shows the header modifications needed to run the algorithm for an Adesto / Atmel AT45B161E-512 Serial SPI Memory [4.] attached to the slave NXP Kinetis K20DX128VFM5 microcontroller [10.] on a FRDM-KL25Z development Platform [6.]. Before trying to program or erase this serial device, you should unprotect it using the DP (Disable Protection) USER Command.

Notice that all the I/O pins used are on Port C which could also be mapped as SPI hardware pins. This represents probably the most complex setup because special run time code must be used to turn off the watchdog timer on the K20 processor. This example was run on the K20 control processor of the same FRDM-KL25Z development Platform [6.] used in **Section 7.1 - Freescale PKL25Z128VLK4 / Atmel AT25640B**.

Shown below is the original Adesto AT45DB161E-512 header in `blue`, and the changes made to attach to the K20DX128VFM5 microcontroller in `red`. In this example, Cortex–M0 code is first written into the processor RAM at address 1FFFFF00. It is then executed. This code Is necessary to turn off the watchdog timer. After turning off the watchdog timer, the clock speed is increased to 48 MHz. Finally, the Port C pins as set up to access the SPI flash. These port pins correspond to one of the on-chip hardware SPIs.

Before trying to program or erase this serial device you should unprotect it using the DP USER Command.

```
;version 1.00, 04/01/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Adesto, AT45DB161E-512, 8x2Meg, proc=Freescale_Freedom_K20DX128VFM5, desc=sw_spi
;begin_cs device=$00000000, length=$00200000, ram=$20000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
WRITE_LONG=49064805/1FFFFF00/ ;          ldr  r0,unlock  ;get wdog unlock addr
                              ;          ldr  r1,control ;get wdog control addr
WRITE_LONG=4B074A06/1FFFFF04/ ;          ldr  r2,const   ;get constants
                              ;          ldr  r3,const+4
WRITE_LONG=80022402/1FFFFF08/ ;          movs r4,#0X2    ;const 0x02
                              ;          strh r2,[r0]    ;unlock wdog
WRITE_LONG=BF008003/1FFFFF0C/ ;          strh r3,[r0]    ;  "
                              ;          nop             ;short delay
```

```
WRITE_LONG=2200800C/1FFFFF10/ ;          strh r4,[r1]     ;turn wdog off
                              ;          movs r2,#0x0     ;return no error
WRITE_LONG=0000BE00/1FFFFF14/ ;          bkpt #0x0        ;cause breakpoint
                              ;          allign data
WRITE_LONG=4005200E/1FFFFF18/ ; unlock:  dc32 0x4005200E ;wdog unlock addr
WRITE_LONG=40052000/1FFFFF1C/ ; control: dc32 0x40052000 ;wdog control addr
WRITE_LONG=0000C520/1FFFFF20/ ; const:   dc32 0x0000C520
WRITE_LONG=0000D928/1FFFFF24/ ;          dc32 0x0000D928
SET_PC_AND_RUN=1FFFFF00/      ; run code to disable software watchdog
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_BYTE=A0/40064003/         ;MCG_C4 48MHz
;
;*****************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
WRITE_LONG=00040982/40048038/   ;scgc5=portc clock on
;
;*****************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*****************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00000143/4004B004/   ;Set pin as port ptc1 - /reset
WRITE_LONG=00000143/4004B008/   ;Set pin as port ptc2 - /ss
WRITE_LONG=00000143/4004B014/   ;Set pin as port ptc5 - clk
WRITE_LONG=00000143/4004B01C/   ;Set pin as port ptc7 - sdi
WRITE_LONG=00000143/4004B018/   ;Set pin as port ptc6 - sdo
WRITE_LONG=00000066/400FF094/   ;Set portc pins 1,2,5, and 6 as outputs
WRITE_LONG=00000002/400FF080/   ;ptc1 - make /reset high
;
;*****************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parametes will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=3/400FF080/                ;SPI_/SS  pin Port  - ptc2 - /ss
PARAM=2/400FF080/                ;SPI_CLK  pin Port  - ptc5 - clk
PARAM=4/400FF090/                ;SPI_MISO pin Port  - ptc7 - sdi
PARAM=5/400FF080/                ;SPI_MOSI pin Port  - ptc6 - sd0
PARAM=6/06070205/                ;port pin number definitions
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
```

```
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;A Blocks 4k-512  (0..FFF)
USER=E1 Erase A            3First A  >/00000000/00000FFF/3Last A   >/00000000/00000FFF/
;
;B Blocks 512-4k  (0..1FF)
USER=E2 Erase B            3First B  >/00000000/000001FF/3Last B   >/00000000/000001FF/
;
;C Blocks 1-4k,1-124k,15-128k  (0..10)
USER=E3 Erase C            2First C  >/00000000/00000010/2Last C   >/00000000/00000010/
;
```

7.4 Toshiba TMPM364F10FG / Adesto AT25DF041A

Shown below is the original Adesto AT25DF041A header in **blue** and the changes made to attach to the TMPM364F10FG microcontroller [11.][12.] are shown in **red**. First, the watchdog is disabled. The internal clock oscillator is set up for approximately 40 MHz using the PLL. Addresses of the I/O ports and associated pin numbers are provided for all 4 SPI pins. PEmicros programming software can now use this algorithm to talk to a Toshiba microcontroller with an Adesto AT25DF041A serial SPI Flash attached as indicated. Actual measurements indicate that the bit rate between the microprocessor and the Flash is approximately 1.408 MegaBits per second. Since there is a small overhead delay between bytes, the actual byte rate turns out to be approximately 147 KiloBytes per second.

Notice that all of the pins used to implement the SPI are on different ports and do not correspond to any hardware SPI signals.

Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

Figure 7-3: Keil Development Board


```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Adesto, AT25DF041A, 8x512k, desc=sw_spi, proc=Keil_Toshiba_TMPM364F10FG_Board
;begin_cs device=$00000000, length=$01000000, ram=$20000000
;
;*************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
WRITE_LONG=00000000/400F2000/      ;WDMOD-Disable watchdog timer.
WRITE_LONG=000000B1/400F2004/      ;WDCR -Set disable code
;
;*************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_LONG=0000701E/400F400C/ ;CGPLLSEL - set pll rate rs/is/cD2s/nd
```

```
WRITE_LONG=80000334/400F4004/ ;CGOSCCR  - set pllon
WRITE_LONG=0000701F/400F400C/ ;CGPLLSEL - set pllsel
;
;*************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
;
;*************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00000008/400C0204/  ;PC3 - CLK  - Output
WRITE_LONG=00000010/400C0604/  ;PG4 - /SS  - Output
WRITE_LONG=00000040/400C0B38/  ;PL6 - MISO - Enable Input
WRITE_LONG=00000004/400C0D04/  ;PN2 - MOSI - Output
;
;*************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=2/400C0200/                ;SPI_CLK  port address - PC3
PARAM=3/400C0600/                ;SPI_/SS  port address - PG4
PARAM=4/400C0B00/                ;SPI_MISO port address - PL6
PARAM=5/400C0D00/                ;SPI_MOSI port address - PN2
PARAM=6/02060403/
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/
;
;A Blocks 128-4k  (0..7F)
USER=E1 Erase A blocks     2First A  >/00000000/0000007F/2Last A   >/00000000/0000007F/
;
;B Blocks 16-32k  (0..F)
USER=E2 Erase B blocks     1First B  >/00000000/0000000F/1Last B   >/00000000/0000000F/
;
```

```
;C Blocks 8-64k  (0..7)
USER=E3 Erase C blocks     1First C  >/00000000/00000007/1Last C   >/00000000/00000007/
;
```

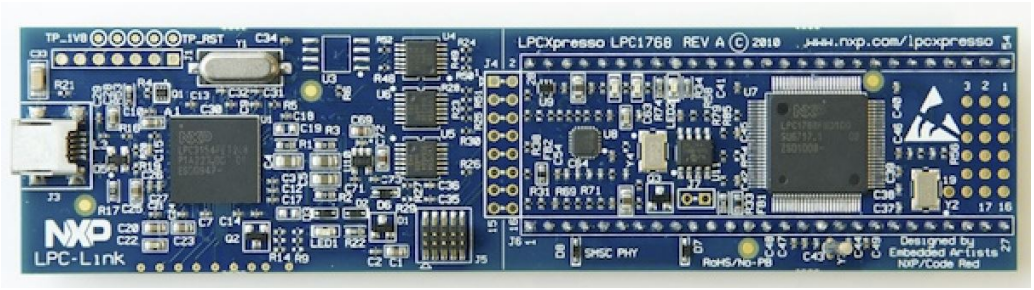
7.5 NXP® LPC1769 / Adesto AT25DL081

Shown below is the original Adesto AT25DL081 header in **blue** and the changes made to attach to the NXP LPC1769 microcontroller on an Embedded Artists LPCXpressoLPC1769revB development board [13.][14.] are shown in **red**. There is no need to disable watchdog since it is off when the processor is reset. PEmicro's programming software automatically does a reset of the microcontroller during startup. The internal clock oscillator is set up for approximately 120 MHz using the PLL and the internal 4 MHz clock. Addresses of the I/O ports and associated pin numbers are provided for all 4 SPI pins. PEmicros programming software can now use this algorithm to talk to the NXP microcontroller with an Adesto AT25DL081 serial SPI Flash attached as indicated. Actual measurements indicate that the bit rate between the microprocessor and the Flash is approximately 6.666 MegaBits per second. Since there is a small overhead delay between bytes, the actual byte rate turns out to be approximately 676 KiloBytes per second.

Notice that all the pins used to implement the SPI are on Port P0 and correspond to on-chip hardware SPI pins. Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the protection bits using the WS (Write Status) USER Command. Also notice that the RAM address is set at 10000000 which is necessary for NXP microcontrollers since they do not have RAM at address 20000000 that most ARM Cortex-M microcontrollers have. In general, if you are using NXP microcontrollers, make sure you received algorithms with the RAM address set at 10000000 when you acquire algorithms from PEmicro. These algorithms have 'sw_spi_nxp' in their names as opposed to just 'sw_spi'.

Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

Figure 7-4: NXP Development Board




```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Atmel, AT25DL081A, 8x1Meg, ,proc=NXP_Xpresso_LPC1769, desc=sw_spi_nxp
;begin_cs device=$00000000, length=$00100000, ram=$10000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
;Set pll for ~ 120Mz
WRITE_LONG=0000003B/400FC084/  ;PLL0CFG, N=1, M=60, Internal RC Clock ~ 4MHz
WRITE_LONG=000000AA/400FC08C/  ;PLL0FEED, Feed Seqence-Make PLL0CFG effective
WRITE_LONG=00000055/400FC08C/
WRITE_LONG=00000001/400FC080/  ;PLL0CON, Enable
WRITE_LONG=000000AA/400FC08C/  ;PLL0FEED, Feed Seqence-Enable PLL0
WRITE_LONG=00000055/400FC08C/
```

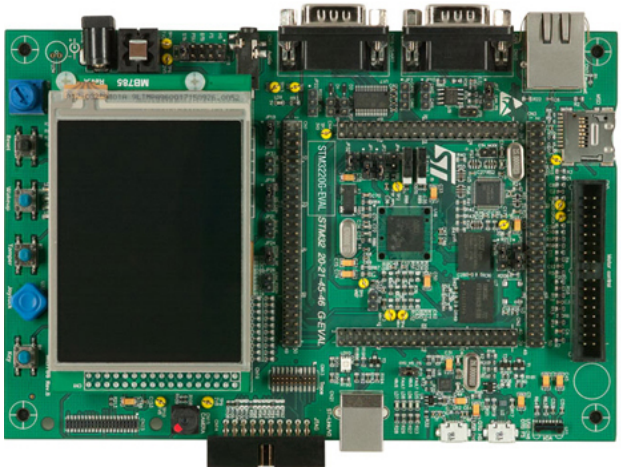
```
WRITE_LONG=00000003/400FC104/  ;CCLKCFG, Divide pll clock by 4
DELAY=0001/                    ;Wait 1 ms for pll stablize
WRITE_LONG=00000003/400FC080/  ;PLL0CON, Connect
WRITE_LONG=000000AA/400FC08C/  ;PLLOFEED, Feed Seqence-Connect PLL0
WRITE_LONG=00000055/400FC08C/
;
;*************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
;
;*************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00058000/2009C000/  ;FIODIR, Make P0.15, P0.16, ad P0.18 Outputs
;
;*************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=2/2009C014/                ;SPI_CLK  port address - P0.15
PARAM=3/2009C014/                ;SPI_/SS  port address - P0.16
PARAM=4/2009C014/                ;SPI_MISO port address - P0.17
PARAM=5/2009C014/                ;SPI_MOSI port address - P0.18
PARAM=6/1211100F/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/
;
;A Blocks 256-4k  (0..FF)
USER=E1 Erase A blocks     2First A  >/00000000/000000FF/2Last A   >/00000000/000000FF/
;
;B Blocks 32-32k  (0..1F)
USER=E2 Erase B blocks     2First B  >/00000000/0000001F/2Last B   >/00000000/0000001F/
;
```

```
;C Blocks 16-64k  (0..F)
USER=E3 Erase C blocks     1First C  >/00000000/0000000F/1Last C   >/00000000/0000000F/
;
```

7.6 STMicroelectronics® STM32F207 / ST 95M02.

The maximum speed of the ST STM32F207 microcontroller [15.] used in this example is 120 MHz using the internal HSI clock and PLL. However, running this microcontroller at the 120 MHz speed generated an SPI clock signal of over 7 MHz. This caused programming errors since the max speed of the ST and M95M02 EEPROM is 5 MHz. Slowing the clock down to 100 MHz worked perfectly. Notice that the ST M95M02 algorithm has no chip erase or sector erase commands. This device is a true EEPROM, hence locations can be programmed without first being erased. Shown below is the original ST M95M02 header in **blue** and the changes made to attach to the STM32F207 microcontroller are shown in **red**. This example was run on a STM3220F-eval board, shown below [16.].

Figure 7-5: STMicroelectronics STM3220F Evaluation Board



On this microcontroller it is not necessary to turn off the watchdog timer since it is off after a reset. The PLL is set up for 100 MHz. There is a 1ms delay included to allow time for the PLL to stabilize before it is selected as the system clock. The clock is enabled for Port F and is used to connect to the external SPI EEPROM. The output pins are then configured for the SPI outputs on Port F. Finally, the port pin addresses and the pin numbers are established. You can see that this is a very simple set up which would be even simpler if the PLL was not used.

Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device ST, M95M02, 8x256k, desc=sw_spi, proc=STM32F207
;begin_cs device=$00000000, length=$00040000, ram=$20000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_LONG=24000C84/40023804/  ;RCC_PLLCFGR enable P=2, N=200, M=4
WRITE_LONG=01000083/40023800/  ;RCC_CR enable pll0 and hsi
DELAY=0001/                    ;wait for pll
```

```
WRITE_LONG=00000002/40023808/  ;RCC_CFGR enable pll as sys clk ~ 120 MHz
;
;*****************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
WRITE_LONG=00000020/40023830/   ;RCC_AHB1ENR enable clock to Port F
;
;*****************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*****************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00000045/40021400/   ;GPIOF_MODER Set as output pins PF0, PF1, PF3
;
;*****************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                ;SPI_MOSI port address
;PARAM=6/eeffgghh/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=2/40021414/                ;SPI_CLK  port address - PF0
PARAM=3/40021414/                ;SPI_/SS  port address - PF1
PARAM=4/40021410/                ;SPI_MISO port address - PF2
PARAM=5/40021414/                ;SPI_MOSI port address - PF3
PARAM=6/03020100/                ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Disable Sector Protection.
USER=DP Disable Protection 0          /00000000/00000000/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/
;
```

7.7 Spansion MB9AF312 / Spansion S25FL164K

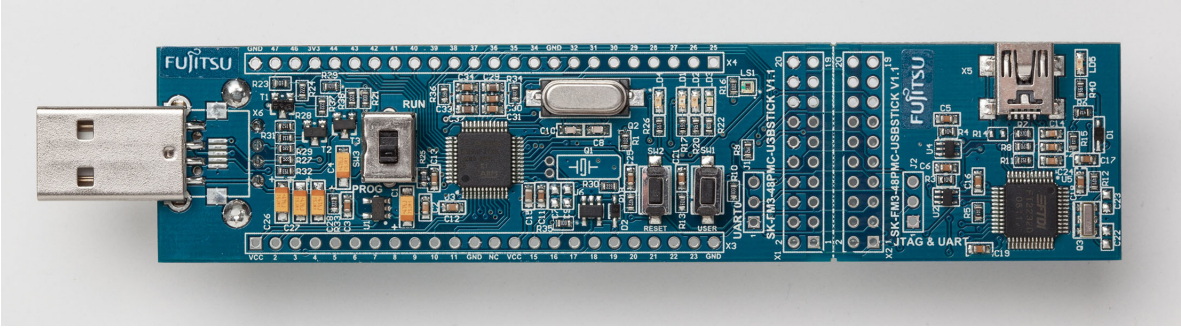
Shown below is the original Spansion S25FL164K [17.] header in **blue** and the changes made to attach to the Spansion MB9AF312 [18.] microcontroller on a SK-FM3-48PMC-USBSTICK [19.] are shown in **red**. First, the watchdog is disabled. Then the I/O ports are set up. The internal clock oscillator is set up for approximately 40 MHz using the PLL and the internal 4 MHz clock. Finally, addresses of the I/O ports and associated pin numbers are provided for all 4 SPI pins. PEmicro's programming software can now use this algorithm to talk to

the Spansion microcontroller with a Spansion S25FL164K serial SPI Flash attached.

Notice that all the pins used to implement the SPI are on Port 3. Before trying to program or erase this serial device, you should unprotect it by writing 00 hex to the status register protection bits using the WS (Write Status) USER Command.

Also shown below is a script file used with CPROGACMP (Command-Line Programmer) to automatically program both the Spansion MB9AF312 microcontroller and an attached Spansion S25FL164K serial SPI Flash in one simple operation.

Figure 7-6: Spansion Development Board




```
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Spansion, S25FL164K, 8x8Meg, desc=sw_spi, proc=Spansion_MB9AF312
;begin_cs device=$00000000, length=$00800000, ram=$20000000
;
;*****************************************************************************
;Disable Watchdog Timer, if one exists and is enabled at Reset
WRITE_LONG=1ACCE551/40011C00/    ;WDG_LCK - Turn off watchdog
WRITE_LONG=E5331AAE/40011C00/
WRITE_LONG=00000000/40011008/    ;WDG_CTL
;
;*****************************************************************************
;Set up system clock source and frequency if different from defaults
WRITE_LONG=00000010/40010034/          ;PSW_TMR  -  PINC=1
WRITE_LONG=00000009/40010038/          ;PLL_CTL1 -  N=10
WRITE_LONG=00000004/4001003C/          ;PLL_CTL2 -  K=1, M=5
WRITE_LONG=00000010/40010000/          ;SCM_CTL  -  PLLE=1
DELAY=0002/                            ;Wait for PLL sync
WRITE_LONG=00000050/40010000/          ;SCM_CTL  -  PLLE=1, RCS=2
;
;*****************************************************************************
;Turn on clock distribution and power to needed microcontroller modules
;
;*****************************************************************************
;Set up serial part /WP and /HOLD to the inactive high state if necessary
;
;*****************************************************************************
;Set up Ports, Data Direction, and Mapping as necessary
WRITE_LONG=00008A00/4003320C/    ;Port3 DDR - Set up outputs on pis 9, B, and F
;
;*****************************************************************************
;Enter SPI signal port addresses and pin numbers for the processor pins, either
; by filling in the symbolic hex parameter values in the ;PARAM statements below
; and remove the leading ; or insert your own formatted PARAM statements. Else,
; default parameters will be used.
```

```
; aaaaaaaa = 8 digit hex address; ee, ff, gg, hh = 2 digit hex port pin numbers
;*****************************************************************************
;PARAM=2/aaaaaaaa/                  ;SPI_CLK  port address
;PARAM=3/aaaaaaaa/                  ;SPI_/SS  port address
;PARAM=4/aaaaaaaa/                  ;SPI_MISO port address
;PARAM=5/aaaaaaaa/                  ;SPI_MOSI port address
;PARAM=6/eeffgghh/                  ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
PARAM=2/4003340C/                  ;SPI_CLK  P35
PARAM=3/4003340C/                  ;SPI_/SS  P3B
PARAM=4/4003330C/                  ;SPI_MISO P3D
PARAM=5/4003340C/                  ;SPI_MOSI P3F
PARAM=6/0F0D0B09/                  ;pin#s MOSI=ee,MISO=ff,/SS=gg,CLK=hh
;
;end_cs
REQUIRES_PROG_VERSION=5.00/
PROGRAMMING_ALSO_DOES_VERIFY
NO_TIMING_TEST
NO_BASE_ADDRESS
FIXED_ADDRESS_READ
;
;Generate a test pattern "CS-ON":"Pattern":"CS-OFF" and repeat Count times.
USER=TP Test Pattern       8Pattern  >/00000000/FFFFFFFF/8Count    >/00000000/FFFFFFFF/
;
;Writes bits to the status register to set/clear protection bits.
USER=WS Write Status byte  2Byte     >/00000000/000000FF/
;
;A Blocks 2k-4k  (0..7FF)
USER=E1 Erase A blocks     3First A  >/00000000/000007FF/3Last A   >/00000000/000007FF/
;
;B Blocks 128-64k  (0..7F)
USER=E2 Erase B blocks     2First B  >/00000000/0000007F/2Last B   >/00000000/0000007F/
;
```

Programming in one simple step with the CPROGACMP command-line programmer typically uses two files. First a .bat (batch) file is used to start the program. This can also be done by typing directly on the command line. However, since this may be done quite often it is simpler to use a batch file. Second is a .cfg (configuration) file which contains the programming commands. Sample files for this example are:

File Spansion.bat

```
C:\pemicro\progacmp\CPROGACMP ? Spansion.cfg bdm_speed 20 interface=USBMULTILINK port=USB1
```

File Spansion.cfg

```
:device=NXP                                                  ;Select Mfg Interface
:useswd 1                                                    ;Select SDI
CM Spansion_MB9AF312K_1x16x64k.arp                           ;Select Algorithm
SS 128k.s19                                                  ;Select Micro Data File
EM                                                           ;Erase Micro
PM                                                           ;Program Micro
CM Spansion_S25FL164K_8x8Meg_Spansion_MB9AF312_sw_spi.arp    ;Select SPI Algorithm
SS 1meg.s19                                                  ;Select SPI Data File
WS 00                                                        ;Disable SPI Protection
PM                                                           ;Program SPI Device
```

Running the batch file produces the following output. Notice that both the microcontroller and the attached SPI

memory are programmed.

```
Pre-processed line 1 Ok ::device=NXP                              ;Select Mfg Interface
Pre-processed line 2 Ok ::useswd 1                                ;Select SDI
PEmicro Interface detected - Flash Version 7.09


REM>:device=NXP                                                   ;Select Mfg Interface

REM>:useswd 1                                                     ;Select SDI
CMD>CM Spansion_MB9AF312K_1x16x64k.arp                            ;Select Algorithm
Initializing. -Initialized.
;version 1.00, 02/02/2013, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Spansion, MB9AF312K, 1x16x64k
;begin_cs device=$00000000, length=$00020000, ram=$20000000
Loading programming algorithm ... Done.
CMD>SS 128k.s19                                                   ;Select Micro Data File
CMD>EM                                                            ;Erase Micro
Erasing. -Module has been erased.
CMD>PM                                                            ;Program Micro
Checking range of S records. -Checked.
Programming and Verifying Address $0001F800. Programmed.
CMD>CM Spansion_S25FL164K_8x8Meg_Spansion_MB9AF312_sw_spi.arp ;Select SPI Algorithm
Initializing. -Initialized.
;version 1.00, 03/15/2014, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Spansion, S25FL164K, 8x8Meg, desc=sw_spi, proc=Spansion_MB9AF312
;begin_cs device=$00000000, length=$00800000, ram=$20000000
Loading programming algorithm ... Done.
CMD>SS 1meg.s19                                                   ;Select SPI Data File
CMD>WS 00                                                         ;Disable SPI Protection
Started. -Done.
CMD>PM                                                            ;Program SPI Device
Checking range of S records. -Checked.
Programming and Verifying Address $000FFC00. Programmed.

REM>

All programming steps completed.
```

7.8 Standalone Programming

In addition to being used for interactive programming through PROGACMP, all of the SPI ARM algorithms can also be used to program in stand-alone mode by using interfaces such as the Cyclone for ARM devices, or Cyclone MAX. Stand-alone programming is ideal for a production environment and offers improved performance without the necessity of a PC. The transition from developing your code by using PROG to production programming your final S19 with a Cyclone is seamless.

The first step in the transition to production programming with a Cyclone is to modify your SPI ARM algorithm and verify its functionality in PROGACMP. Once the algorithm is working correctly, the move to a stand-alone solution simply involves procurement of a Cyclone, followed by these steps:

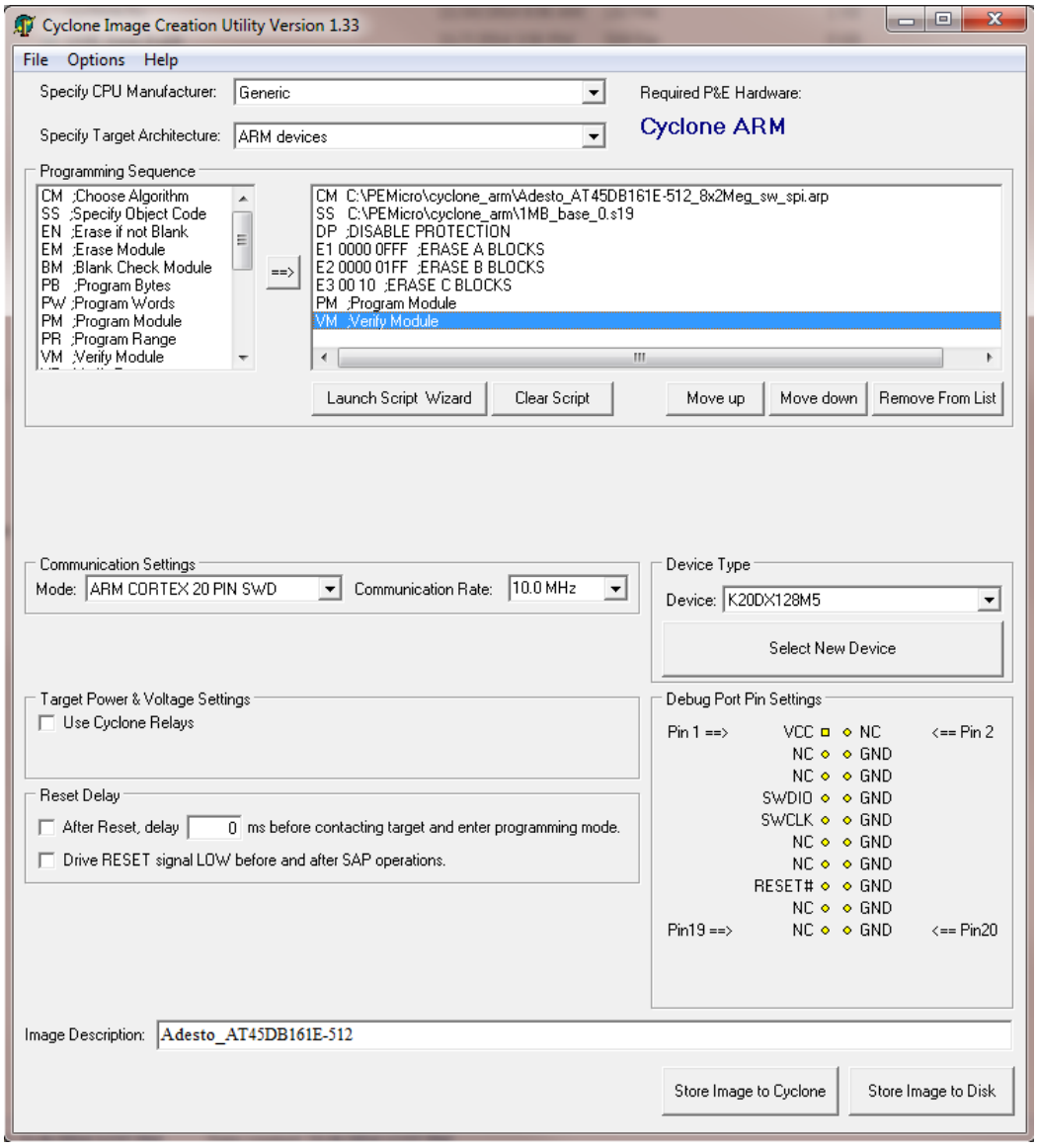
1. Launch the CreateImage.exe software.
2. Use CM (Choose Module) to select your algorithm.
3. Select all of the commands (in the same order) that you would normally select in the GUI PROGACMP
4. Modify all other relevant settings including communication mode, communication rate, Cyclone

relay options (to provide power), and image description

5. Store this programming image on the Cyclone programmer.
6. Once the image is stored on the Cyclone, you are ready to program the internal flash in a production line. Press the Start button on your Cyclone to begin.

The screenshot below shows how you could take the algorithm created in **Section 7.3 - Freescale K20DX128VFM5 / Adesto AT45DB161E-512** and create an image to be used in standalone.

Figure 7-7: Screenshot




```
CM C:\PEMicro\cyclone_arm\Adesto_AT45DB161E-512_8x2Meg_sw_spi.arp
```

- This command selects the algorithm created in **Section 7.3 - Freescale K20DX128VFM5 / Adesto AT45DB161E-512**.

```
SS C:\PEMicro\cyclone_arm\1MB_base_0.s9
```

- This command selects the object file (s19) that contains the code you wish to program.

```
DP ;DISABLE PROTECTION
```

- This command is a special user command this algorithm provides to disable write protection. If the algorithm has special user commands in its header, they will show up as available commands.

```
E1 0000 0FFF ;ERASE A BLOCKS
```

- This command is used to erase the A Blocks in the flash. Blocks 0 through 4095 are erased.

```
PM ;Program Module
```

- This command programs the object file you selected in SS.

```
VM ;Verify Module
```

- This command reads from flash and verifies that the S19 is actually programmed.

8 Conclusions

PEmicro's approach to production programming of SPI memory devices allows you to use our tools on any of your designs. Even those designs which have the SPI memory connected to whatever pins you had left over to implement the SPI interface can easily be set up and programmed in a universal manner. Using PEmicro's tools, you can program the internal microcontroller memory and the external SPI memory in one simple manufacturing step. Even though the algorithms with the SPI memory are implemented with the bit banging software, they run fast enough to not slow down the manufacturing process. From the examples above we see that the transfer speed from the microcontroller to the serial SPI memory is typically faster than the transfer speed from the host processor microcontroller via the SWD/JTAG interface. In addition, the actual programming time on the SPI devices is greater than the data transfer. The following table derived from the prior examples indicates very reasonable data transfer times over the Bit Bang SPI interface. These measured numbers are only approximate since internal clocks and PLLs were used.

Table H-1. Data Transfer Times

Ex	Micro	~Freq MHz	SPI memory	~MBit/S	~KByte/S	~KByte /S/MHz
1	NXPPKL25Z128VLK4	48	Atmel AT25640B	2.381	232	4845
2	Texas Instruments LM3S301	12	Atmel AT25F1024	0.575	051	4278
3	NXP K20DX128VFM5	48	Adesto AT45DB161E	2.373	243	5074
4	Toshiba TMPM364F10FG	40	Adesto AT25DF041A	1.408	147	3676
5	NXP LPC1769	120	Adesto AT25DL081	6.666	676	5630
6	ST STM32F207	100	ST M95M02	3.7113	823	820
7	Spansion MB9AF312K	40	Spansion S25FL164K	0.507	057	1421

The speeds shown vary somewhat by the particular microcontroller implementation, accuracy of the clock, speed of the I/O, and other factors. Even under these variations the kilobytes per second per megahertz CPU clock frequency is very consistently in the 4000 to 5000 range. It should be noted that increasing the clock

speed of the microcontroller is very desirable, since it increases the transfer rate. Hence, if available, you should increase the CPU clock frequency to the maximum allowed for the particular microcontroller by enabling the phase locked loop if available. It should also be noted that these examples all used uncalibrated internal RC clocks whose accuracy was not verified.

9 PEmicro's ARM®-Compatible Hardware Interfaces

9.1 PEmicro's CYCLONE & CYCLONE FX In-System Programmers

PEmicro's Cyclone in-system programmers are thoughtfully designed tools which program ARM Cortex™ devices from NXT and several other manufacturers.A complete listing of supported manufacturers/devices is available at pemicro.com/partners.

Figure 9-1: PEmicro's CYCLONE and CYCLONE FX

By connecting to a debug header on the target, Cyclone programmers can program internal memory on an ARM Cortex-M processor. SWD/JTAG protocol is used for programming, which requires only 2 pins. The processor or memory device can be mounted on the final printed circuit board before programming.

Once loaded with data by a PC a Cyclone can be disconnected and operated manually in a completely stand-alone mode via the LCD menu and control buttons. A Cyclone may come with up to 1GB of non-volatile internal memory, which allows for the onboard storage of multiple programming images. When connected to a PC for programming or loading it can communicate via the ethernet, USB, or serial interfaces.

PEmicro's Cyclones also include Cyclone Control Suite software, which consists of three methods of control/automation. A Cyclone may be operated interactively via Windows based GUI applications, or programming operations can be scripted, and finally customs application scan use .dll commands from a PC. CYCLONE FX models include an advanced version of the suite with additional features such as gang programming with multiple Cyclones. CYCLONE models can be licensed separately to include these advanced features.

Click to view more about PEmicro's Cyclone programmers.

9.2 PEmicro's USB Multilink & Multilink FX Debug Probes

PEmicro's USB Multilink and Multilink FX debug probes allow a PC access to the Background Debug Mode (BDM) or JTAG/SWD interface on many different ARM MCU devices (see complete list below). The USB

interface allows communications between your Windows machine and the standard debug connector on the target. The ARM microcontrollers are supported via the multiple headers located on the Multilink. The headers can be accessed by simply flipping open the plastic case. Various ribbon cables suitable for a variety of connector types.

Figure 9-1: PEmicro's USB Multilink & USB Multilink FX

PEmicro's Multilinks allow the user can take advantage of the background debug mode to halt normal processor execution and use a PC to control the processor. The user can then directly control the target's execution, read/write registers and memory values, debug code on the processor, and program internal or external FLASH memory devices. The USB Multilink FX can also provide power to your target MCU, either 3.3V or 5V.

The USB Multilink FX is an advanced, high-speed model that includes all the features of PEmicro's USB Multilink interfaces, plus these additional benefits:

- Lightning fast! Up to 10X faster communications speed than PEmicro's other Multilinks
- Can provide power to the target MCU
- I/O line clamping for added protection
- Supports legacy NXP device architecutres

Multilinks are natively supported by all current PEmicro software applications, and tool chains from many PEmicro partners including NXP .IAR, Keil, Cosmic, Atollic, and Green Hills (check with the vendor for device compatibility). A list of supported software can be found here.

Multilinks are USB 2.0 compatible, and are backwards compatible with USB 1.1 ports.

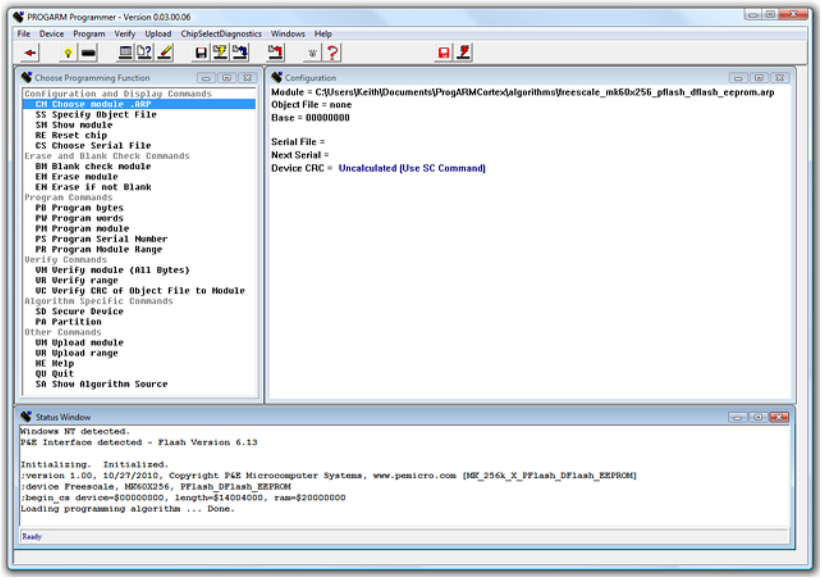
Click to view more info about PEmicro's **Multilink/ Multilink FX debug probes**.

10 PEmicro's ARM Programming Software

10.1 PEmicro's PROG for ARM devices

PEmicro's PROG for ARM® Cortex™ processors flash programmer software allows you to program/reprogram external flash devices in-circuit, via one of PEmicro's hardware debug interfaces. The programmer talks to the processor's Background Debug Mode through an ARM®-compatible hardware interface (sold separately), which connects the parallel port or USB port of a PC to a mini-10 or mini-20 pin JTAG connector on the target system. The programmer supports ARM Cortex-M microcontrollers.

Figure 10-1: PROG for ARM Cortex processors



PROG for ARM Cortex processors also includes the command-line version of the programmer software. CPROGACMP. Together they are perfect for development, production line programming, or field firmware upgrades.

In addition, the programmer comes with PEmicro's entire library of ARM programming algorithms.

PEmicro has a resource page describing different methodologies for programming target flash in both development and production environments: Flash Programming Resource Page

Programmer features include:

- Program
- Verify
- Blank check
- Upload
- Display
- Erase

Click to view **PEmicro's PROG for ARM Cortex processors User Manual.**

10.2 Library Routines for ARM® Cortex™-M processors ARM Cortex Library Routines

The software package consists of an interface DLL and sample code which allows a custom application to be built which can interact with and debug an ARM® Cortex™-M processors (ACMP) based system via one of PEmicro's hardware interfaces. This includes NXP's Kinetis® and other vendor’s devices. The package includes both C/C++ and Delphi example routines as well as detailed calling information for the DLL/SO. The C and Delphi modules in the ACMP Hardware Interface Library Routines come as source code which interfaces with a DLL/SO to interface to the hardware. The ACMP Hardware Interface Library Routines library are very useful for building your own custom application to do such functions as product test, calibration, and update. The demo programs included in the package show you how to initialize the interface, program registers, download code to RAM and step through code. Starting with these programs it is easy to customize them to

your specific requirements. Applications created with the included 32-bit .DLL will also operate on the 64-bit operating systems listed below under System Requirements.

Read more about simplifying product testing in PEmicro's Expert's Corner.

Key Features:

- Works with ARM Cortex-M devices
- Works through PEmicro's USB Multilink & USB Multilink FX debug probes, and CYCLONE/ CYCLONE FX in-system programmers
- Uses Background Debug Mode or JTAG/SWD protocols
- Included in the package are C/C++ and PASCAL modules
- Source and executables provided for both C and PASCAL demos
- Compilers supported:
  - UNITACMP and UNITACMP_DIST: MicrosoftVisual C++ 5.0 or greater, Delphi 2.0 or greater
  - Development platform: Windows 32-bit, Windows 64-bit (XP, Vista, 7, 8, 10).
- Very useful for building production line testers

UNITACMP includes a license to distribute the binaries (DLL/so and Drivers) of UNITACMP to up to five target PCs subject to the conditions of the license agreement. For a license to distribute to an unlimited number of PCs, see UNITACMP_DIST.

Click to read more about the Unit Library SDK for ARM® Cortex™-M processors on our website.

11 Contact PEmicro

PEmicro
98 Galen St – 2nd Fl.
Watertown, MA 02472-4502, USA
www.pemicro.com

- General Information and Sales
- PEmicro Algorithm Downloads
- PEmicro Support Requests
- PEmicro Forums

12 References

1. Winbond Electronics Corp, W25Q256FV SPIFLASH Data Sheet
2. GigaDevice. GD25Q10 Data Sheet
3. PEmicro’s PROGACMP User Guide
4. Atmel , AT25640B SPI Serial Memory Date Sheet
5. NXP, KL25 Sub-Family Reference Manual
6. NXP, FRDM-KL25Z User's Manual
7. Atmel , AT25F1024 SPI Serial Memory Date Sheet
8. Texas Instruments, Stellaris® LM3S301 Microcontroller Data Sheet
9. Texas Instruments, Stellaris® Family Development Board Users Manual

10. NXP, K20 Sub-Family Reference Manual
11. Toshiba, TMPM364F10FG Data Sheet
12. Keil, Toshiba MCBTMPM364 Evaluation Board
13. NXP, LPC176x/5x User manual
14. Embedded Artists, LPCXpressoLPC1769revB Development Board
15. ST Microelectronics, STM32F207 microcontroller reference manual
16. ST Microelectronics, STM3220F-EVAL board manual
17. Spansion S25FL164K, 64 Mbit (8 Mbyte) CMOS SPI Flash Memory
18. Spansion MB9AF312 Microcontroller Specification
19. Spansion SK-FM3-48PMC-USBSTICK, MB9AF312 Eval Board
20. PEmicro's CPROGACMP User Guide