

PROCESSOR-CONTROLLED TEST (PCT) FAST FLASH PROGRAMMING

PCT Fast Flash Programming is an alternative to the Standard Flash Programming function provided in the Interactive mode of the PCT Test Toolkit software. Fast Flash Programming offers significant speed increases for certain processor types.

Depending on the type of flash and the way it is implemented on the board, programming times can be less than 10 seconds per megabyte. However, to use the Fast Flash Programming method, the user needs to develop a relatively simple, flash-specific programming algorithm in assembly language and a short TSL/1 test script. In contrast, no programming is required to use the standard Flash Programming function in PCT.

The main application for Fast Flash Programming is in board manufacturing or batch board upgrades. In these situations, the significant time saved in the flash programming will far out-weigh the development time.

Note: Fast Flash Programming is only supported on specific processors. The current list includes:

- Freescale™ PowerArchitecture™ 7447, 7447A, 7450, 7455 (Apollo 2.0 and 3.0), 7457
- Freescale™ PowerQUICC™ 8540, 8541, 8543, 8545, 8547, 8548, 8555, 8560
- Intel® Pentium®, Celeron® and Xeon™ processor families

This range will be increased but older Freescale™ processors are unlikely to be supported due to limitations in their command set.

The benefit of using Fast Flash Programming over Standard Flash Programming is far greater for Freescale™ processors than for Intel® processors due to the reduction of long boundary scan chain transfers, and the larger block size that can be downloaded via their debug ports. However, there should still be a 3 to 4-fold speed increase with Intel® processors.

The differences between PCT Fast Flash Programming and PCT Standard Flash Programming function are as follows:

- PCT Standard Flash Programming stores the binary image on the host PC. Each byte of this image is sequentially written to flash memory via the processor's serial debug port (the flash address width and the number of parallel flash devices determines how many bits need to be transferred on each cycle; this could be a half-word or word instead of a byte). Writing speed is therefore determined by the speed of data transfer across the serial debug port and also the overhead of the extended-JTAG command cycles.
- In contrast, PCT Fast Flash Programming downloads the complete binary image into the RAM of the board being programmed. Additionally, a flash-programming algorithm is also downloaded into the board's RAM. These files are downloaded in blocks rather than single bytes, saving considerable time. Having downloaded both the binary flash image and the algorithm, a TSL/1 test script is run from the PCT interface. This passes relevant parameters to the algorithm and then commands it to execute. The algorithm then runs independently, at full CPU speed. When flash programming and verification are

complete, the algorithm returns status codes to PCT via the debug port to indicate whether or not the flash programming was successful.

The command set and procedure to erase and program flash memory differs from one device to another. It is therefore necessary to create a flash-programming algorithm and TSL/1 script for each type of flash (some use the same method). This document gives an outline of how to write these, and also provides some examples.

Note: The assembly examples are intended only as a guide. Optimal code efficiency was not the prime goal, and register restoration was not considered (PCT stores all status registers on entering debug mode, restoring them on exit). Some commands are assembler-specific: details of the assemblers are given later in this document. Examples for both Freescale™ and Intel® processors are provided.

THE PROGRAMMING ALGORITHM

The algorithm that performs the flash erase, programming and verification is written in assembly language. This is then compiled into a binary file (or an Intel® Hex file), which is downloaded together with the flash image by the TSL/1 script.

The algorithm is written according to the specifications laid down in the flash datasheet. The flash type used in these examples is AMD® (Spansion™) Am29F010B, which is an 8-bit device. The datasheet for this device is available on the Spansion website:

<http://www.spansion.com/products/Am29F010B.html>

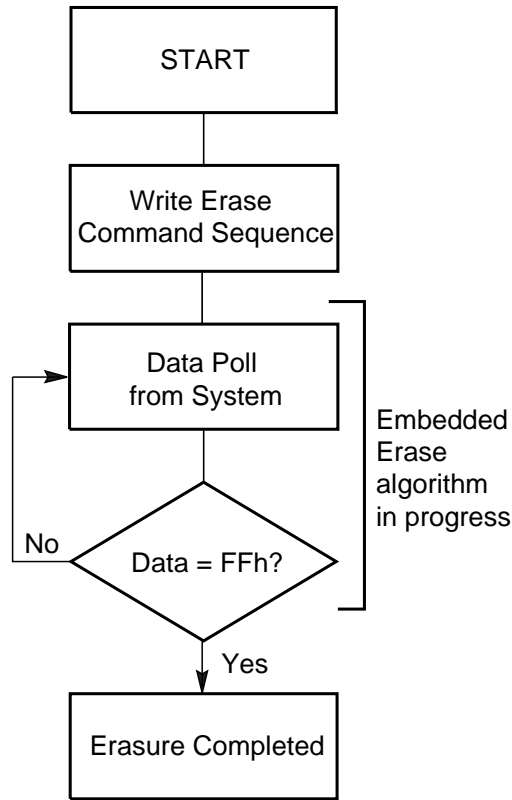
If the above link is no longer active, search for Am29F010B on the Spansion site.

The datasheet contains flow diagrams that explain Program and Erase Operations. If a flash already contains data, it is necessary to erase it before it can be re-programmed. The Erase operation sequence is shown below.

ERASE OPERATION

Note: In order to erase or program some types of flash devices, it is necessary to disable write protection. There are many methods used to write protect flash, some of which are by means of external circuitry. In other cases the flash may provide a Write Protect pin that has to be changed in state to allow writing. The Am29F010B device does not have a Write Protect pin. See our application note #8 for more details on disabling write protection:

http://www.asset-intertech.com/download/App8_Flash_Write_Protect.pdf



1. Erase Operation

The Command Definitions table in the datasheet indicates that there are two erase operations: a Chip Erase and a Sector Erase. Both of these require 6 bus cycles to perform. These two command sequences differ only in the last bus cycle:

Chip Erase

1	2	3	4	5	6
Addr	Data	Addr	Data	Addr	Data
555	AA	2AA	55	555	80
555	AA	2AA	55	555	10

Sector Erase

1	2	3	4	5	6
Addr	Data	Addr	Data	Addr	Data
555	AA	2AA	55	555	80
555	AA	2AA	55	SA	30

SA = Sector Address

Sample assembly language programs to erase either Chip or Sector of the Am29F010B are available here:

Freescale™

http://www.asset-intertech.com/download/erase_sample_Am29F010B.s

Intel®

http://www.asset-intertech.com/download/ip_erase_sample_Am29F010B.as

If you open these files in a text editor you will see that 2 parameters need to be passed into them: the start address of the sector or the device, and either 0 (chip erase) or 1 (sector erase) to indicate the erase mode.

During erase and programming operations it is necessary to carry out data polling to verify that the algorithms embedded in the flash have completed before proceeding. A data polling flow diagram and description is also included in the datasheet. DQ7 is the data polling bit, which is checked in the sample code.

PROGRAMMING OPERATION

The programming operation is shown in the flow diagram below, copied from the Am29F010B datasheet.

The Command Definitions table in the datasheet gives the following 4-cycle command sequence:

1		2		3		4	
Addr	Data	Addr	Data	Addr	Data	Addr	Data
555	AA	2AA	55	555	A0	PA	PD

PA = Address to be programmed PD = Data to be programmed

Sample assembly language files to program the Am29F010B are available here:

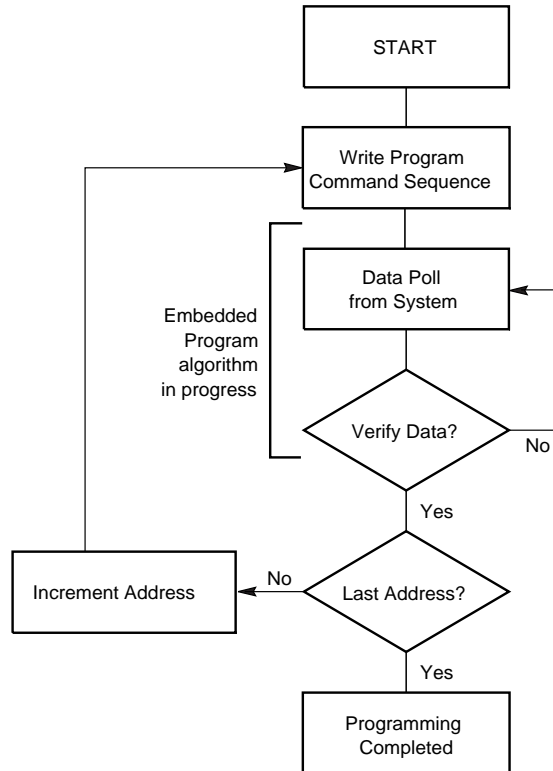
Freescale™

http://www.asset-intertech.com/download/program_sample_Am29F010B.s

Intel®

http://www.asset-intertech.com/download/ip_program_smp_Am29F010B.as

If you open these files in a text editor you will see that 3 parameters need to be passed into them: the start address of the flash image in RAM, the start address of the flash to be programmed, and the number of bytes to be programmed.



1. Programming Operation

It is necessary to carry out data polling during the programming to verify that the algorithm embedded in the flash has completed before proceeding. See the data polling flow diagram and description in the datasheet for more details. DQ7 is the data polling bit, which is checked in the sample code.

CRC CHECK

Another useful algorithm is a CRC Check, which can be used to check the integrity of an image, to check that the data in a flash is identical to an image, or to check that a flash has been erased. Examples of the assembly language code for a simple CRC Check for a 32-bit wide flash bus are given in the following files:

Freescale™
http://www.asset-intertech.com/download/crc_sample_32bit_wide.s

Intel®
http://www.asset-intertech.com/download/ip_crc_sample_32bit_wide.asm

In fact, the TSL/1 scripting command **ROMTest** provides a more reliable CRC check because it is based on a polynomial algorithm, rather than a simple XOR operation. This is a fast routine so it could be used instead of creating a special algorithm.

COMBINED OR SEPARATE ALGORITHMS?

The erase, programming and CRC check algorithms could be combined into a single algorithm that is downloaded into the RAM of the board being programmed. Alternatively, it may be more flexible to keep them separate and download them to different RAM addresses if and when they are required.

16/32-BIT AND MULTIPLE DEVICE CONFIGURATIONS

The above sample erase and programming algorithms are for single 8-bit flash devices. Changes in the command syntax and programming method will be needed to accommodate 16-bit, 32-bit and multiple device configurations. Example algorithms for erasing 4 parallel Am29F010B devices (32-bit flash bus width) are given in the following files:

Freescale™

http://www.asset-intertech.com/download/erase_sample_Am29F010B_x4.s

Intel®

http://www.asset-intertech.com/download/ip_era_sam_Am29F010B_x4.asm

Example algorithms for programming 4 parallel Am29F010B devices (32-bit flash bus width) are given in the following files:

Freescale™

http://www.asset-intertech.com/download/program_smp_Am29F010B_x4.s

Intel®

http://www.asset-intertech.com/download/ip_prg_smp_Am29F010B_x4.asm

COMPILING THE FREESCALE™ ALGORITHMS

If you don't have a suitable PowerArchitecture™ assembler and linker to obtain a binary file, there are number of Software Development Tools provided on the Freescale™ website in the "Support" "Design Tools" section:

<http://www.freescale.com>

Alternatively, you can download a GNU cross-compiler free-of-charge, for use on x86 Win32 machines from the following location:

<http://www.agelectronics.co.uk/download.html>

The tool required is in the "GNU tools for PowerArchitecture™ products" section, called "cross-compiler for x86 Win32 hosts". This compiler produces object files, which need to be converted to binary files.

Run the compiler from the DOS command line using the following command syntax:

```
as filename.s -o filename.out
```

as.exe is the compiler name (this name may differ), the filename.s is the name of the assembly language file to be compiled, including its extension, filename.out is the name of the resulting object file, including its extension.

We have created a command-line utility for converting the object files produced by this compiler into binary. It is available here:

<http://www.asset-intertech.com/download/PowerPCOConverter.zip>

After un-zipping the executable, run the following command from a DOS prompt (assuming the PowerPCOConverter.exe executable is in the current directory):

```
PowerPCOConverter object_filename
```

Note: The object code filename is given without the extension, and it needs to include the path if the file is not in the current directory.

COMPILING THE INTEL® ALGORITHMS

The Intel® algorithms were written for use with the Phar Lap 386|ASM assembler and 386|LINK linker. These are tools that are part of the Phar Lap TNT DOS-Extender™ SDK (Formerly 386|DOS-Extender), which is available on the following sites:

<http://www.dinigroup.com/pharlap.php>

However, there are many free assemblers and linkers available, such as MASM, NASM, etc. The latest MASM software can be downloaded from the following site:

<http://www.masm32.com/>

The Intel® code examples will need to be revised for use with alternative assemblers and linkers.

WRITING THE TSL/1 SCRIPTS

Having written the Erase, Programming and CRC Check algorithms, it is now necessary to write TSL/1 scripts, which will download the programming algorithms into the RAM of the board being programmed, pass some parameters and receive status information on completion of the process.

The TSL/1 commands used in the PCT versions for Freescale™ and Intel® processors differ slightly, so each version is dealt with separately.

Note: These script examples are incomplete – it is necessary to initialise the memory on the board being programmed (after a reset to ensure known CPU state) before these scripts can be run. Memory set up will differ for each board, so this is beyond the scope of this document. If register-based write protection has been implemented in a flash controller (e.g. CPLD), commands will have to be included in the script to disable this. See Application Note #8 for methods used to write protect flash memory:

http://www.asset-intertech.com/download/App8_Flash_Write_Protect.pdf

Freescale™ TSL/1 scripts

The important commands for Fast Flash Programming with Freescale™ processors are as follows:

DownloadBinaryFile – downloads the algorithm and flash image into RAM. This command is followed by a memory location and then a speed parameter, which is **1** for Fast Flash Programming.

ExecuteUserDiag – runs the algorithm.

WriteGPR – this writes into the CPU's general purpose registers, allowing parameters to be passed to the algorithms.

ReadGPR – this reads the CPU's general purpose registers allowing status information to be returned by the algorithms.

For more details on how to use these commands, see the Help file in the PCT Test Toolkit software.

Sample TSL/1 scripts to separately control the flash erase, CRC check and programming via one of the supported Freescale™ processors are available as follows:

Erase single 8-bit device

http://www.asset-intertech.com/download/TSL_sample_for_flash_erase.rtf

Erase 4 parallel 8-bit devices

http://www.asset-intertech.com/download/TSL_sample_for_flash_erase_x4.rtf

CRC check to verify flash erased

http://www.asset-intertech.com/download/TSL_sample_for_CRC_0.rtf

Program single 8-bit device

http://www.asset-intertech.com/download/TSL_sample_for_flash_program.rtf

Program 4 parallel 8-bit devices

http://www.asset-intertech.com/download/TSL_sample_for_flash_program_x4.rtf

CRC check to verify correct programming

http://www.asset-intertech.com/download/TSL_sample_for_CRC_compare.rtf

These files are well commented so they should be self-explanatory.

The scripts could of course be combined together, and the algorithms could be downloaded into different locations in the RAM so that they can be called separately from the TSL/1 script. Here is an example of a combined script for erasing, CRC checking and programming:

http://www.asset-intertech.com/download/TSL_sample_combined_program.rtf

Intel® TSL/1 scripts

The important commands for Fast Flash Programming with Intel® processors are as follows:

DownloadUserDiag – downloads an Intel® HEX format algorithm or flash image into RAM.

DownloadBinaryFile – downloads a binary format algorithm or flash image into RAM. This command is followed by a memory location but there is no speed parameter as in the Freescale™ version of this command.

ExecuteUserDiag – runs the algorithms.

WriteEAX, WriteEBX, WriteECX, WriteEDX, WriteEBP, WriteEDI, WriteESI, WriteESP – these write into the CPU's general purpose registers, allowing parameters to be passed to the algorithms.

ReadEAX, ReadEBX, ReadECX, ReadEDX, ReadEBP, ReadEDI, ReadESI, ReadESP – these read the CPU's general purpose registers allowing status information to be returned by the algorithms.

For more details on how to use these commands, see the Help file in the PCT Test Toolkit software.

Sample TSL/1 scripts to separately control the flash erase, CRC check and programming via one of the supported Intel® processors are available as follows:

Erase single 8-bit device

http://www.asset-intertech.com/download/ip_TSL_sample_for_flash_erase.rtf

Erase 4 parallel 8-bit devices

http://www.asset-intertech.com/download/ip_TSL_sample_for_flash_erase_x4.rtf

CRC check to verify flash erased

http://www.asset-intertech.com/download/ip_TSL_sample_for_CRC_0.rtf

Program single 8-bit device

http://www.asset-interech.com/download/ip_TSL_sample_for_flash_program.rtf

Program 4 parallel 8-bit devices

http://www.asset-interech.com/download/ip_TSL_sample_for_flash_program_x4.rtf

CRC check to verify correct programming

http://www.asset-intertech.com/download/ip_TSL_sample_for_CRC_compare.rtf

These files are well commented so they should be self-explanatory.

As for the Freescale™ versions, the scripts could be combined together, and the algorithms could be downloaded into different locations in the RAM so that they can be called separately from the TSL/1 script.