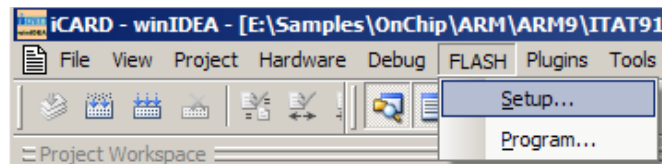


External Flash Programming

1 Introduction

This document describes all settings related to the external flash programming and provides a getting started guide to program the external flash device.

If the debug support features external flash programming, winIDEA will create a FLASH menu through which 'FLASH Programming Setup' and 'FLASH Program' dialogs are available.



2 Options and Settings

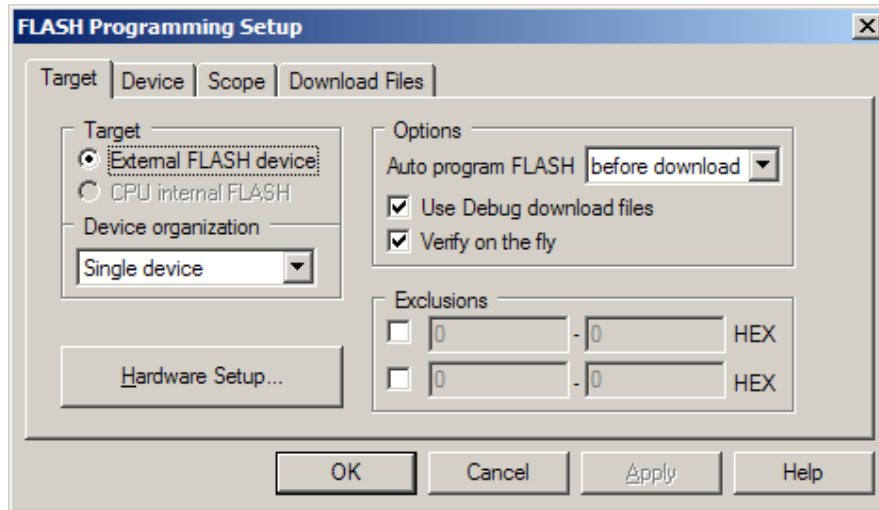
2.1 FLASH Programming Setup

To start the flash programming configuration, select the 'Setup...' command from the FLASH menu. Target tab is visible per default.

Flash programming configuration consists of:

- Target configuration – either on-chip flash or external flash device
- Device configuration – manufacturer and type specification
- Scope configuration – determines whether the entire chip or only parts of it will be used in the operation
- Download files configuration – files the flash will be programmed with.

2.1.1 Target



Target

Typically, this field is not configurable and 'External FLASH device' setting is set per default.

'CPU internal FLASH' setting is available only for specific debug architectures (e.g. Freescale 68HC12), where internal MCU flash programming is not implemented in a standard way through a debug download.

Device Organization

Device Organization determines the target device organization. This is either a single device (16 or 32 bit) or two 16-bit or 32-bit devices in parallel organization (common address bus with microcontroller A2 address line connected to flash device's A0 address line, one device holds even addresses, the other odd addresses).

- **Single device** – select if you wish to program a single standing device;
- **2 devices** – select if you wish to program the device in parallel configuration;
- **4 devices** – select if you wish to program four devices in parallel configuration.
- **8 devices** – select if you wish to program eight devices in parallel.

Options

Several options to automatically program the FLASH can be selected with the 'Auto program FLASH...' option.

The options include:

- **never** – the FLASH is never programmed automatically;
- **after link** – the FLASH is programmed immediately after link;
- **before download** – the FLASH is programmed before download.
- **after download** – the FLASH is programmed after download

The 'Use **Debug download files**' option can be used to override programming files specified in the 'Download Files' tab within the FLASH Programming Setup dialog. When the option is checked, winIDEA programs the

external flash with files from the 'Debug/Files for Download/Download Files' tab, where files for debug download are specified.

Verify on the fly. After flash programming monitor buffer is programmed, the data written is read again and compared with the buffered data. This process is repeated until all download files are programmed to flash devices.

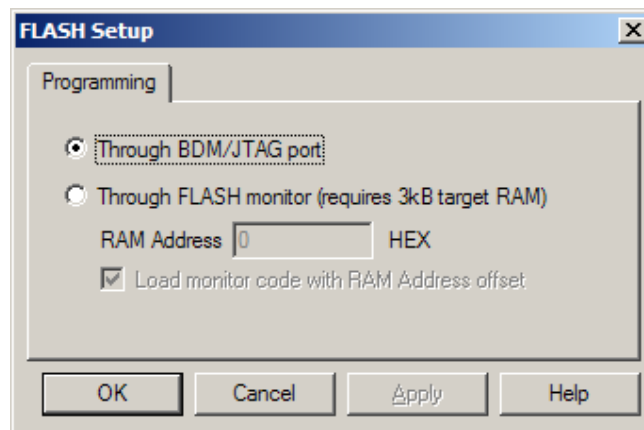
This option has effect only when programming 'Through FLASH monitor' is configured. When programming 'Through BDM/JTAG port' is selected, it's meaningless.

Exclusions

Up to two memory areas to exclude when programming FLASH can be defined here.

Hardware Setup...

Pressing this button opens the 'FLASH Setup' dialog. The external flash programming can be done in two ways: through the BDM/JTAG port or through the FLASH monitor.



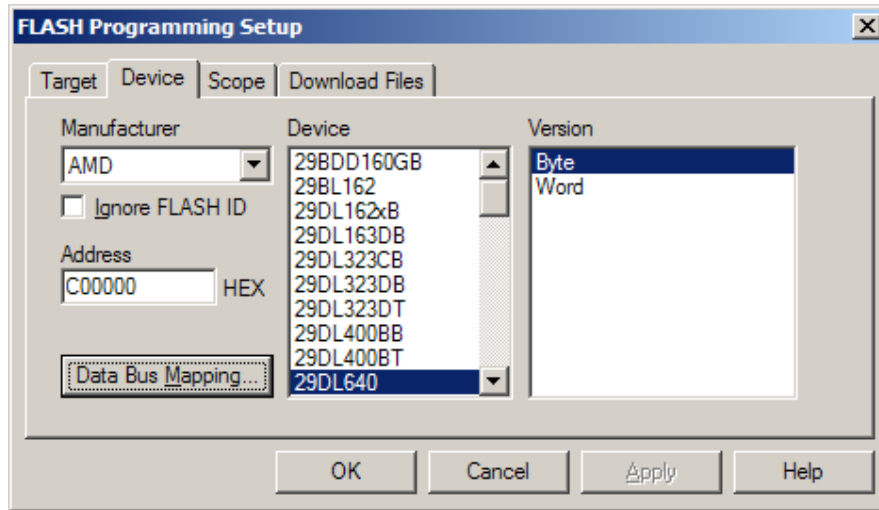
The programming through BDM/JTAG port is considerably slower than through FLASH monitor, but it's more likely to work always since it does not require any target memory resources.

The programming through FLASH monitor is implemented by executing a short monitor (assembler) program. The short FLASH monitor program is copied from the Emulator via BDM/JTAG port to the target where at least 3kB of target RAM is required, either on-chip or external. The user must always assure that RAM on the target is accessible. If RAM is not accessible after the debug reset, appropriate MCU registers must be configured in the 'Hardware/Emulation Options//Initialization' tab. The initialization sequence will be always executed before the flash programming. If the microcontroller features internal watchdog, which is active immediately after the reset, it must be disabled via the initialization sequence too. FLASH monitor is not aware of it and does not service it.

Refer to Technical Notes document for your specific debug architecture for more details on initialization sequence setup and use.

When programming time is an important aspect, which normally is, the programming through FLASH monitor should be used.

2.1.2 Device



Manufacturer

Select the manufacturer of the target flash device to be programmed. The device list will be updated after the manufacturer is selected.

Contact iSYSTEM if target device is not listed. New devices are added on request.

Ignore FLASH ID

One of the first flash programming steps is that the FLASH ID is checked. If this is not desired, please check the 'Ignore Flash ID' option. However, it is recommended to keep it unchecked. If FLASH ID check fails, it means something is wrong with the flash programming configuration.

Device and Version

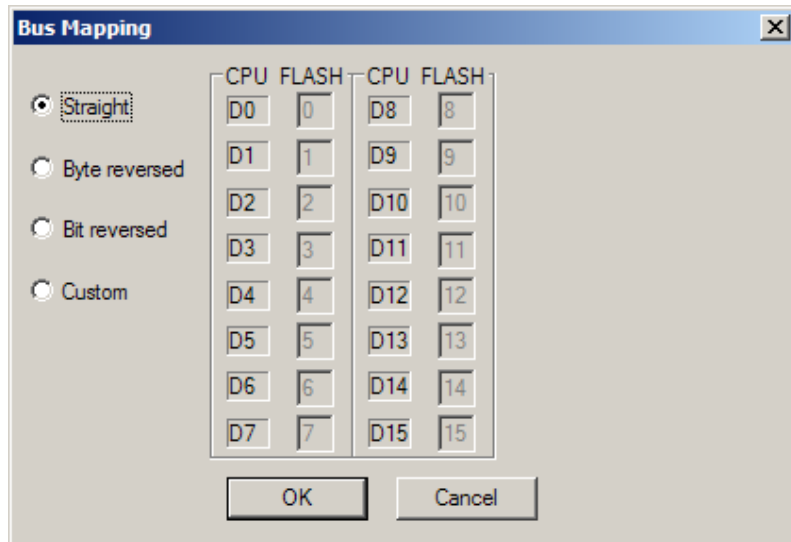
Select the target flash device. If there is only a single version of the device, the 'Version' list will be disabled and will show only 'default' value. Should there be different versions of the device (like 8 and 16 bit data bus versions), select the one that you are using.

Address

Specify the start (base) address of the flash device in the target system – the address that the microcontroller uses to access the device.

Data Bus Mapping

The Bus Mapping dialog is invoked by pressing the 'Data Bus Mapping' button.



The Bus mapping may need to be changed from default setting when the data bus of the flash device is mapped differently than on the microcontroller.

Straight

The bits are mapped directly.

Byte reversed

The bytes are reversed, i.e. the first byte in the FLASH is mapped as the second byte for the CPU, the second FLASH byte is mapped as the first CPU byte.

Bit reversed

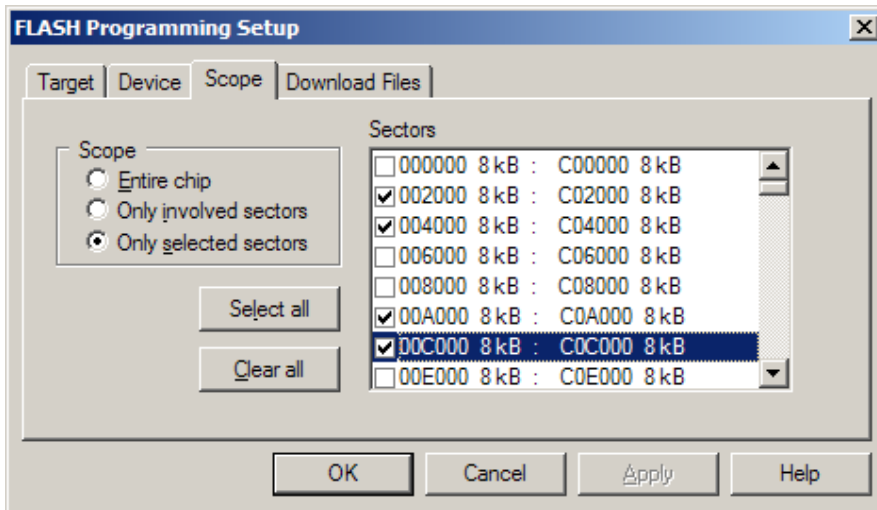
The bits are reversed. The first FLASH bit is the last CPU bit, the last FLASH bit is the first CPU bit.

Custom

Custom mapping can be defined here.

2.1.3 Scope

Settings in the Scope tab determine whether the entire device will be operated on or just certain parts of it.



Entire Chip

When selected, the entire chip will be subject to erase and program operations.

Only involved sectors

When selected, only sectors, where the download file's code is loaded will be erased and/or programmed.

Only selected sectors

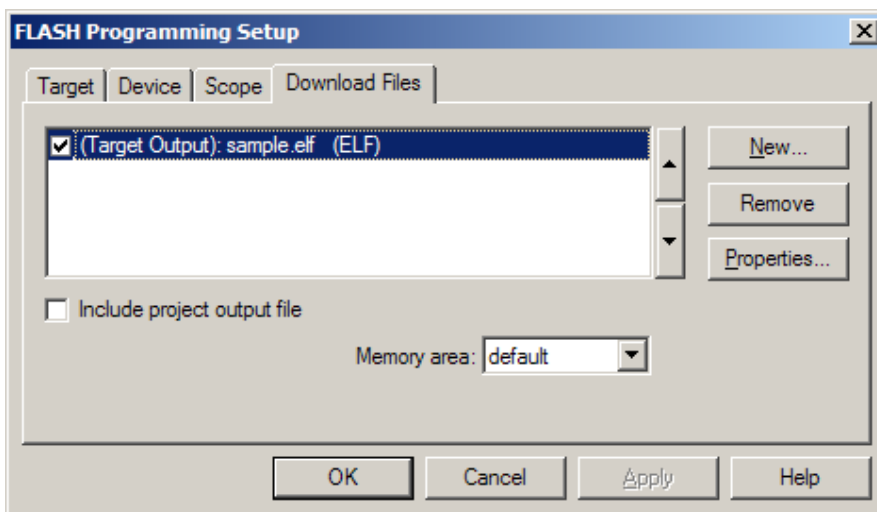
When selected, only sectors that are explicitly checked in the Sectors list will be operated on.

You can use 'Select all' to select all sectors and the 'Clear all' to deselect them.

2.1.4 Download Files

In the 'Download Files' tab, the files to be programmed to the target flash device can be specified.

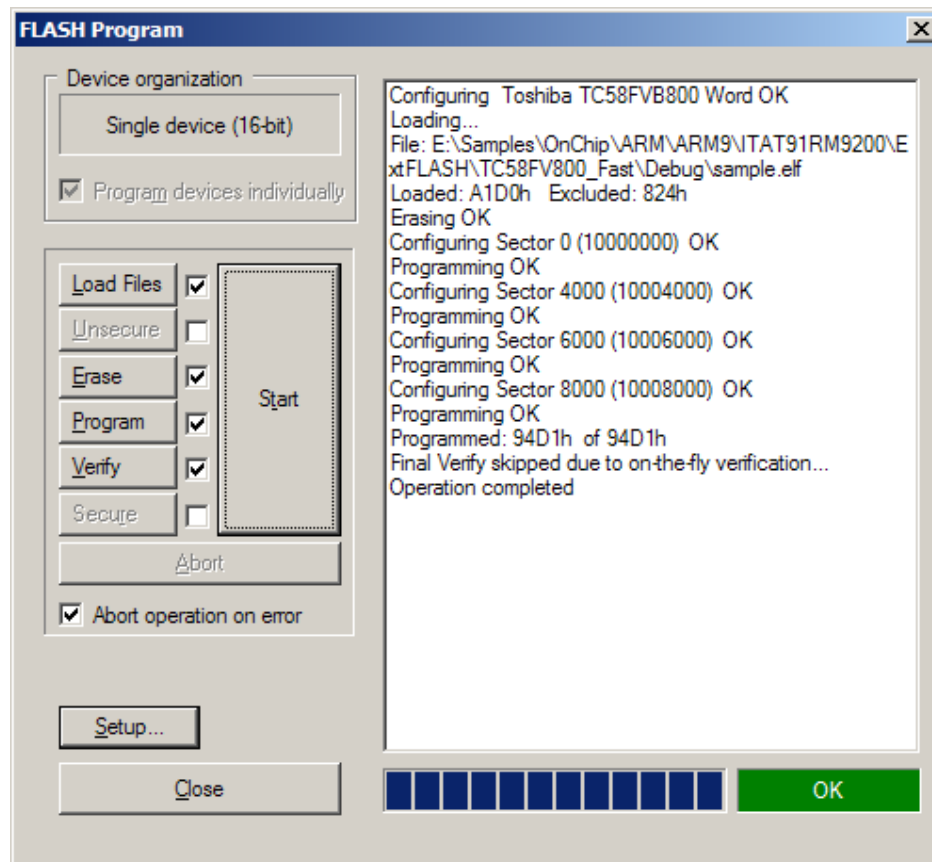
Note: Settings on this page are ignored if 'Use Debug download files' option is checked in the 'Target' tab.



2.2 Device Programming

To open the 'Flash Program' dialog, select the 'Program...' command from the FLASH menu.

After the flash programming configuration is completed, open this dialog to program the external flash device.



Load Files

Click this button to load the download files to winIDEA's internal cache.

Unsecure

Select this option to unsecure the flash device.

Note: This setting is disabled for external flash programming. It's available for specific debug architectures only (e.g. Freescale 68HC12), where flash programming is not implemented in a standard way.

Erase

Click this button to erase the configured scope

Program

Click this button to program the configured scope.

Verify

Click this button to verify if programming was successful.

Secure

Select this option to secure the FLASH.

Note: This setting is disabled for external flash programming. It's available for specific debug architectures only (e.g. Freescale 68HC12), where flash programming is not implemented in a standard way.

Start

Performs all operations checked next to the Load/Erase/Program/Verify buttons.

Abort

Aborts the current operation.

Setup

Opens the 'FLASH Programming Setup' dialog

Results

The Results list shows results of operations performed on the FLASH device.

3 Getting Started

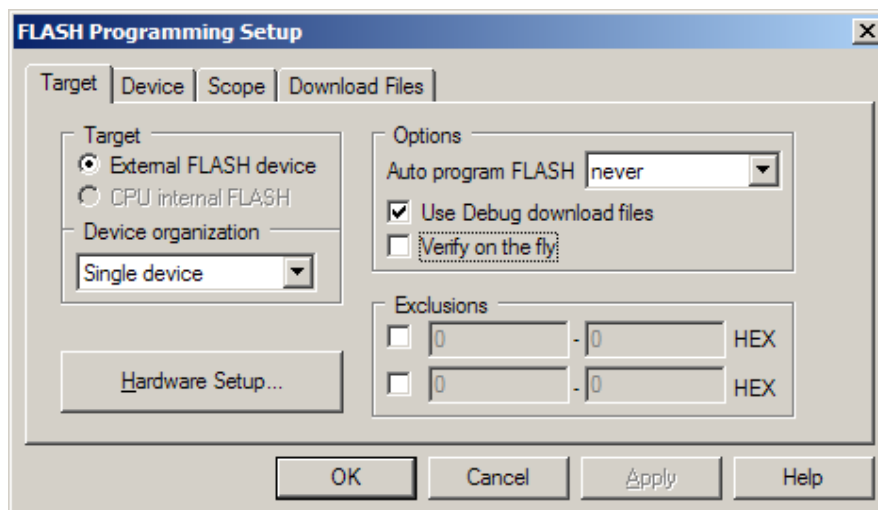
It is presumed that the debug session is operational. The user should be able to perform:

- debug reset
- modify RAM type of memory through the memory window
- single step

If any of the three debug functionalities is not functional, troubleshoot that first.

The following text will explain how to configure the debugger to program an external flash device from Toshiba, type TC58FVB800. It's an 8 Megabit flash memory located at 0x10000000 address in the MCU address space (example only!). The device is connected as a single 16-bit device.

- First, let's open the 'FLASH Programming Setup' dialog (FLASH/Setup...).

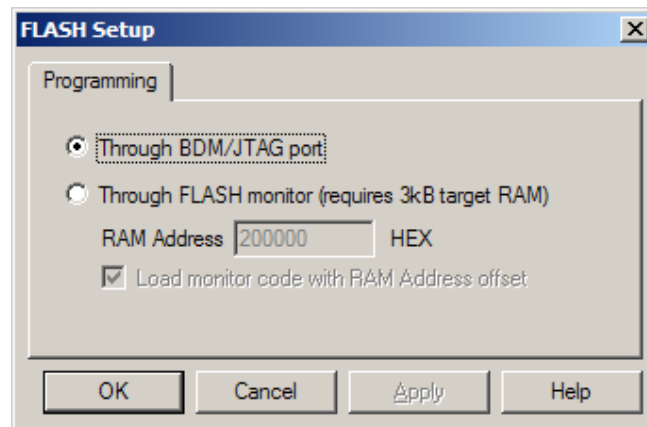


- 'External FLASH device' option should be checked in the Target field.
- Single device should be selected for the Device organization setting.
- Select 'before download' for the 'Auto program FLASH' option.
- Check 'Use Debug download files' option. With this option checked, the debugger will program files defined in the 'Debug/Files for download/Download Files' tab. A file can be either a file including code and symbol information or simply a binary or hex file. Make sure that binary or hex file loads into the target flash address space (add offset address if necessary).

Alternatively, you can keep this option unchecked and define the files to be programmed in the 'Download Files' tab in the 'FLASH Programming Setup' dialog.

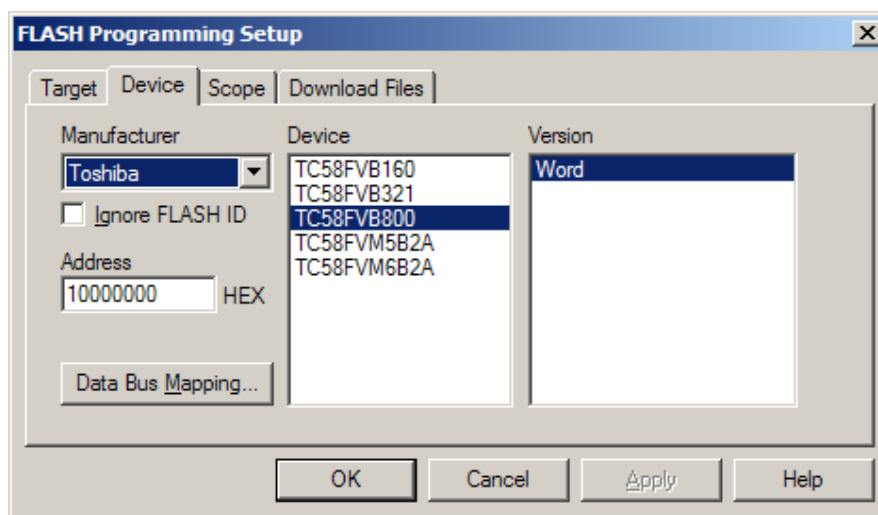
- Next, press 'Hardware Setup...' button, which opens 'FLASH Setup' dialog and let's first select programming 'Through BDM/JTAG port' and close the dialog.

Flash programming through JTAG port is not supported for few flash devices where custom FLASH monitors are written. WinIDEA pops up a warning when programming through JTAG port is not supported.



It is recommended to use 'Through FLASH monitor' programming mode once programming through BDM/JTAG port is operational.

- Next, define the flash device to be programmed and its start address.

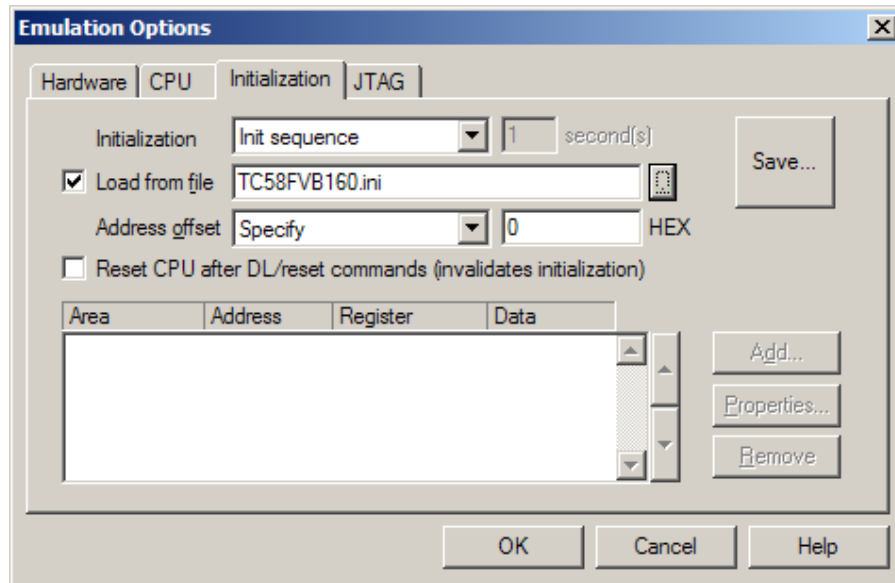


In this particular example, a flash device starts at address 0x10000000 in the MCU address space and is not accessible after the reset. Therefore, appropriate MCU registers must be configured in the 'Hardware/Emulation Options//Initialization' tab in order to make the flash accessible before the flash program process is started. The initialization sequence is always executed before the flash programming.

In this example, following two lines are part of the TC58FVB160.ini file

```
S EBI_CSA L 0x2 //External Bus Interface - Chip Select Assignment  
S EBI_CFGR L 0x0 //External Bus Interface - Chip Configuration
```

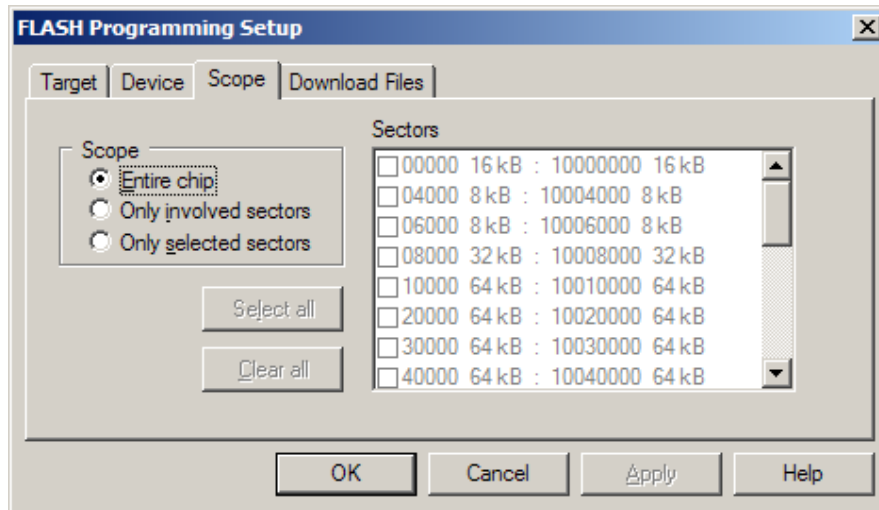
which is then executed as part of the initialization sequence.



Initialization tab

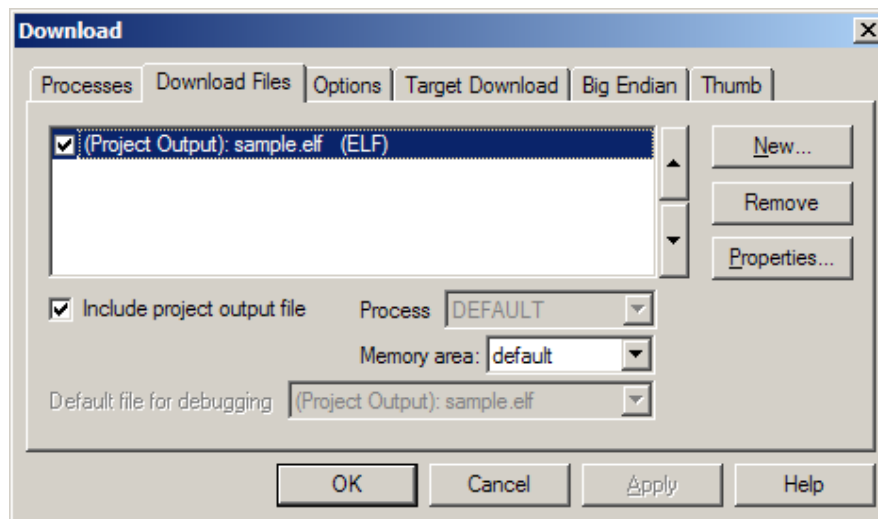
Refer to Technical Notes document for your specific debug architecture for more details on initialization sequence setup and use.

- Settings in the 'Scope' tab typically remains at default setting, which performs flash program on entire flash device. Note that this setting also performs a mass erase before programming the first sector to be programmed. If there are flash sectors, which are not allowed to be erased or reprogrammed, use one of the Scope selections, restricting programming to certain sectors.

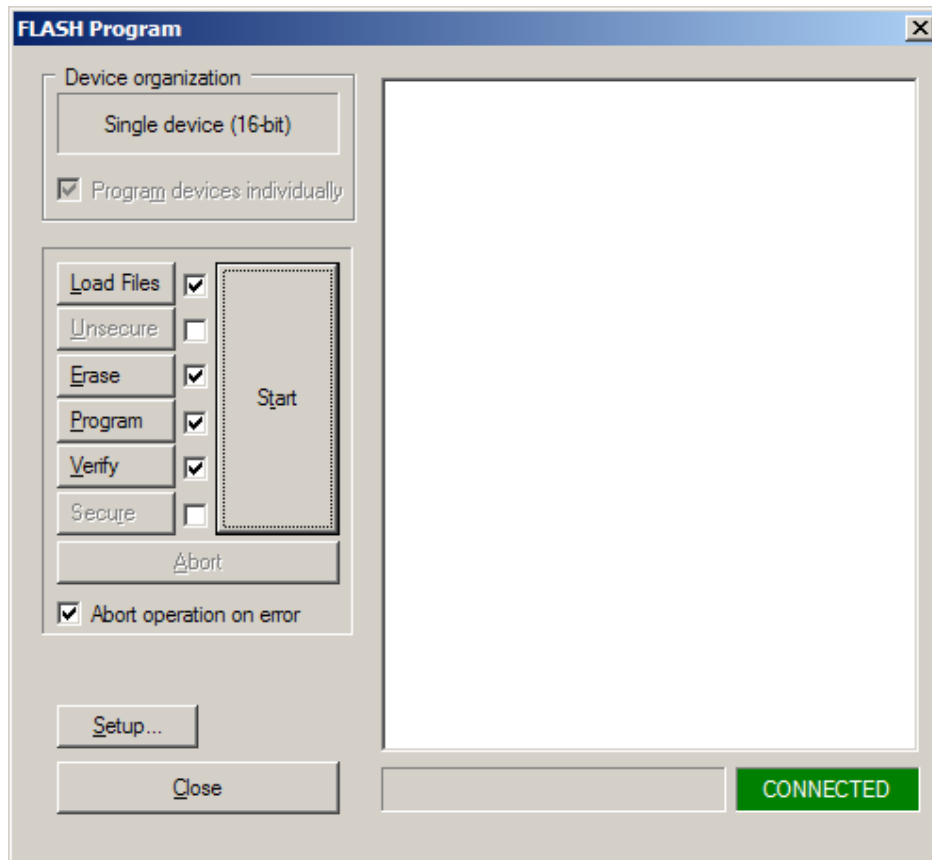


Scope tab

- At the end, the file to be programmed needs to be defined. Add file(s) in the 'Debug/Files for Download/Download files' tab, where files for debug download are specified.

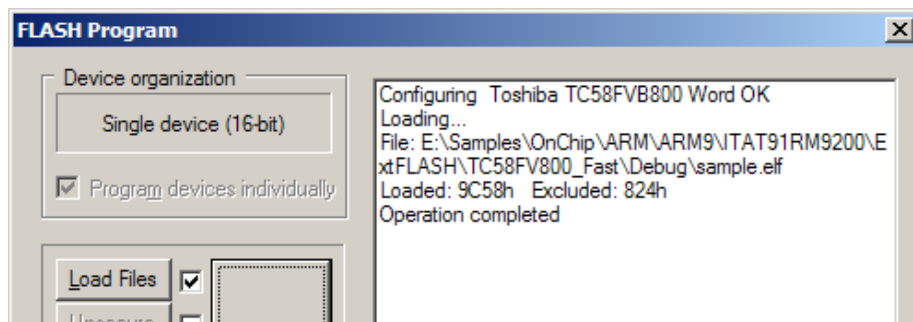


The debugger is configured to program the external flash device. Open the 'FLASH Program' dialog to start the flash programming.

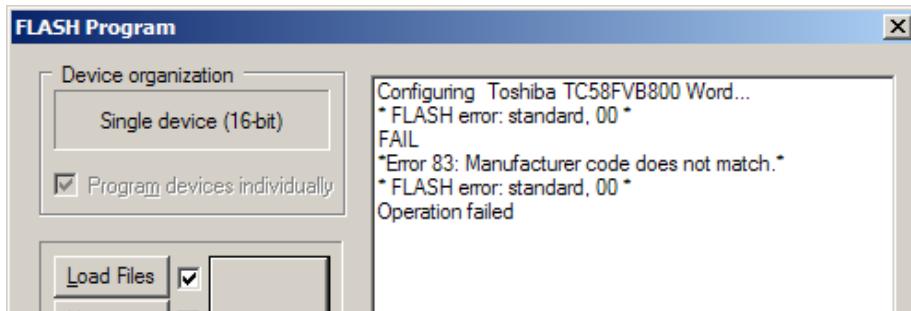


First time, it's recommended to perform flash programming procedure step by step.

- Press Load Files button. In the results list you see important information, how many bytes of the file(s) to be programmed fit into the flash address space and how many are excluded. It may happen that all bytes are excluded and nothing gets programmed then. In such case, check the download file memory mapping. In case of binary or hex file, adding offset to the file may solve the problem already.

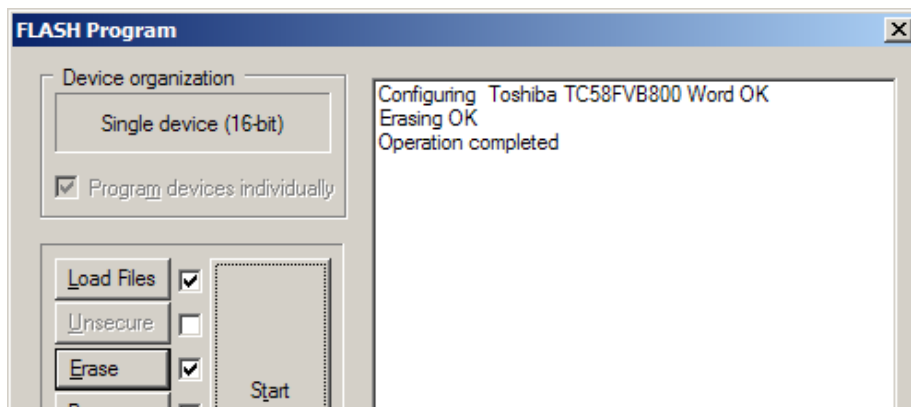


At this step, you may already get manufacturer and/or flash ID mismatch (see next picture). This means that there is something wrong with the flash programming configuration or the target flash device doesn't match with the one selected in the 'FLASH Programming Setup' dialog or it's even a target hardware problem. Most often, the target flash device is not accessible by the microcontroller which means that the microcontroller is not configured properly (yet).

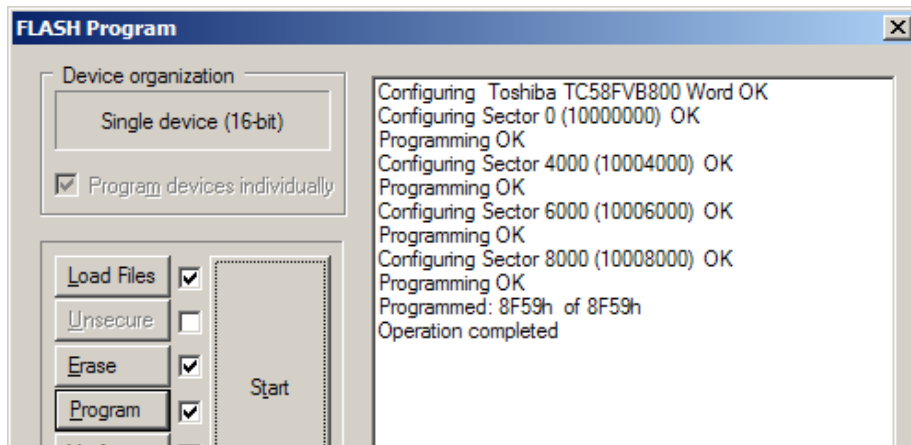


Troubleshoot this step before you proceed to the next Erase step! Read the last 'Troubleshooting - General Guidelines' chapter in the document for assistance.

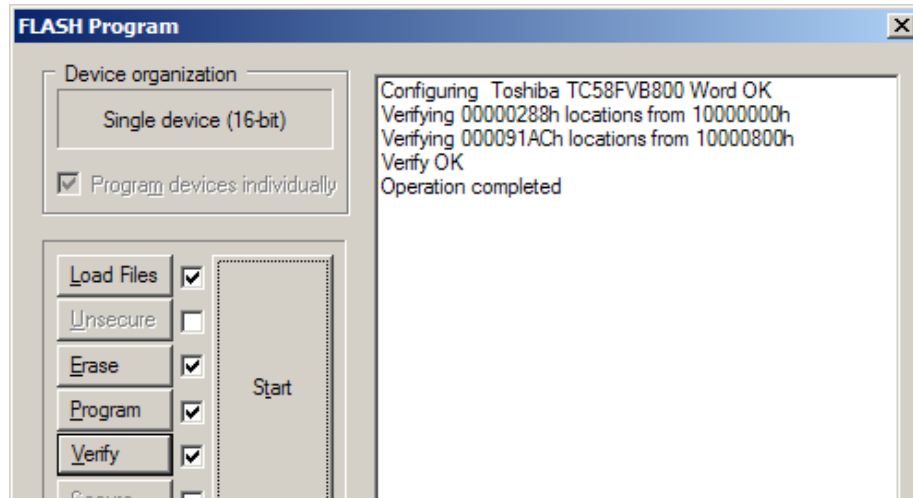
- Press Erase button. The device gets erased.



- Press Program button. The device gets programmed.

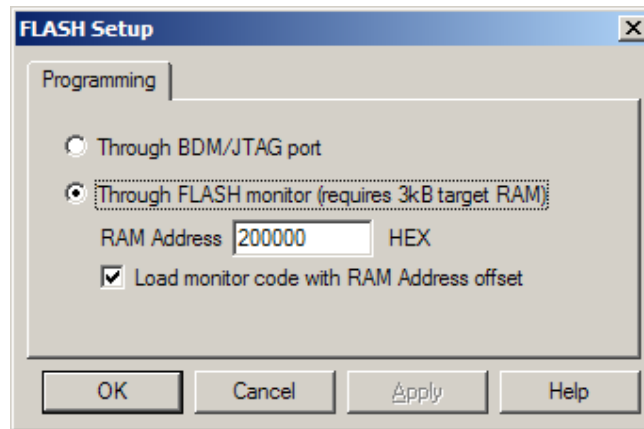


- Finally, press Verify button.



- Once all steps are operational, you may check/uncheck ticks next to the buttons and then simply press Start button, which executes all the selected steps in one run.

Once we have programming through BDM/JTAG port operational, we can try to configure flash programming through FLASH monitor. Press 'Hardware Setup...' button in the 'FLASH Programming Setup' dialog to open the 'FLASH Setup' dialog. Change the setting and additionally enter the address where the RAM in your target application is located.



In our case, external RAM is located at address 0x200000 and is not accessible after the MCU reset. This means we need to additionally configure the MCU via the initialization sequence to make the external RAM accessible before the external flash is programmed. In this example, following lines:

```
//SDRAM configuration
S SDRAMC_MR L 0x2188C150 // SDRAM controller - SDRAMC Configuration
S SDRAMC_MR L 0x2 // SDRAM controller - SDRAMC Mode
S SDRAMC_MR L 0x4 // SDRAM controller - SDRAMC Mode
S SDRAMC_MR L 0x3 // SDRAM controller - SDRAMC Mode
S "Physical":(20000080) L 0x0
S SDRAMC_TR L 0x2E0 // SDRAM controller - Refresh Timer
S SDRAMC_MR L 0x0 // SDRAM controller - SDRAMC Mode
```

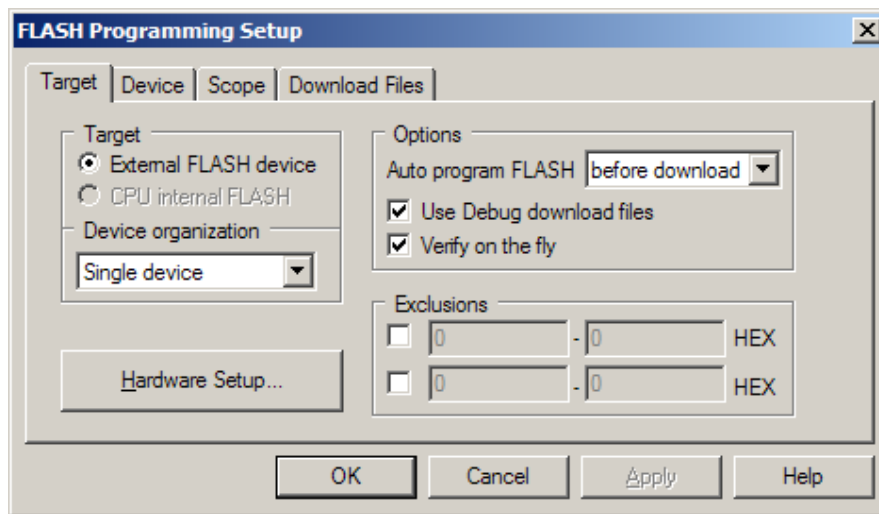
are added to the existing TC58FVB160.ini.

If the MCU has internal RAM, it's recommended to allocate FLASH monitor there since execution is typically faster than from the external RAM.

Before you try the flash programming with this new configuration, perform debug reset with new configuration, open memory window at address where you allocated FLASH monitor and try to modify the content. If that works, you can proceed to flash programming. If it fails, you need to find out why access to the RAM is not operational yet.

Again, the flash should be first programmed step by step (Load files, Erase, Program, Verify). Once that is operational, you can run again all the steps via the Start button.

If all is up and running, we can go back to the FLASH Programming Setup and do the final modifications in the target tab.



Check the 'Verify on the fly' option, which will considerably speed up the Verify phase. In this case, verify is performed by the flash monitor as part of the flash write process and not as a standalone verify process through the debug interface.

It may also be convenient and safer to change the 'Auto program FLASH' setting to 'before download'. With this change, external flash is programmed with every debug download and no mismatch between the download file and the flash content is possible. 'After link' setting would be a good alternative too. In this case, external flash will be programmed only when the source code is changed and recompiled – useful as long as winIDEA is used as editor & build manager too.

4 Troubleshooting – General Guidelines

Typically, first 'Load Files' program flash step fails already when having problems with the flash programming. Most often manufacturer and/or flash ID mismatch is reported. Let's perform some low-level tests, which will help to locate the problem more precisely. Using winIDEA Integrated Script Language (ISL), we will individually read manufacturer ID & device ID, erase flash and program single byte/word. This will help us to better pinpoint the root cause of the problem.

ISL allows us to write and read target memory through a set of predefined script functions, which we use in a text file which has .isl extension. winIDEA can then interpret this file and execute functions from the script file. Refer to winIDEA user's guide for more details on ISL.

The following text is based on troubleshooting programming the Atmel AT49BV1614AT flash device, which can be byte or word programmed, depending on the state of the BYTE pin. Please refer to the Atmel AT49BV1614AT datasheet for more technical details on this particular device.

The described troubleshooting concept can be applied to and used with any other flash device since most flash devices work in the same way except that the command sequences vary from each other. When troubleshooting other flash device, the user may need to modify the command sequence and flash address only.

Command definition for AT49BV1614AT in HEX (excerpt from the flash specification):

Command Sequence	Bus Cycle	Address	Data
Product ID entry	1	555	AA
	2	AAA	55
	3	555	90
Product ID Exit	1	555	AA
	2	AAA	55
	3	555	F0
Chip Erase	1	555	AA
	2	AAA	55
	3	555	80
	4	555	AA
	5	AAA	55
	6	555	10
Byte/Word Program	1	555	AA
	2	AAA	55
	3	555	A0
	4	Address	Din

Manufacturer ID (AT49BV1614AT): 0x1F (byte); 0x001F (word)

Device ID (AT49BV1614A): 0xC0 (byte); 0x00C0 (word)

Device ID (AT49BV1614AT): 0xC2 (byte); 0x00C2 (word)

All examples scripts are written for word organized flash devices. If necessary, adjust script files when troubleshooting byte organized flash device.

When troubleshooting other flash device, check the command sequence of your flash device in the associated datasheet and modify the command sequence in script routines adequately. The user also needs to modify the addr variable in the script file according to his target application. The variable represents a flash base address.

Before running script files, make sure that the flash is not protected from erasing/programming, either by means of software or by means of hardware.

Script routines are self-explanatory and don't need supplement explanation.

The user will notice that the addresses of command sequence from the table and the script file differ. The reason comes from the target application. Typically, flash devices are word organized. Thus, A1 from the CPU is connected to A0 of the flash device (and so on for other addresses: A2->A1, etc.). Consequentially, all commands from the table must be shifted by one, resulting in 555<<1=AAA and AAA<<1=554. Therefore, when the CPU addresses 0xAAA, the flash is actually addressed by 0x555 and when the CPU addresses 0x554, it's addressed by 0xAAA.

As part of the troubleshooting process, the user should run at least Read manufacturer and device ID script. Erase and Program scripts can be run if necessary.

Once .isl script file is created you can run it two ways:

- open it in the editor and press F6
- run it through Run Script command available from the Tools menu,

See the results of script file execution in the output window.

- **Read manufacturer and device ID (e.g. Read.isl)**

```

declare
{
  var int addr;
  var int i;
  var int w;
}

function void main()
{
  //offset - address where FLASH starts – must be set according to the target application
  addr=0x1800000;

  APIPrintString("**** Read FLASH ID");
  APIWriteMemory(0, addr+0xAAAA, 0xAAAA, 2);
  APIWriteMemory(0, addr+0x5554, 0x5555, 2);
  APIWriteMemory(0, addr+0xAAAA, 0x9090, 2);
  APISleep(100); //delay
  //Read ID
  APIReadMemory (0, addr+0x0000, 2);
  APIReadMemory (0, addr+0x0002, 2);
  //software ID exit
  APIWriteMemory(0, addr+0xAAAA, 0xAAAA, 2);
  APIWriteMemory(0, addr+0x5554, 0x5555, 2);
  APIWriteMemory(0, addr+0xAAAA, 0xF0F0, 2);
}

```

The script file should return valid manufacturer and FLASH (device) ID. Double check the microcontroller configuration when wrong ID is read. Most probably you did not set up properly winIDEA initialization sequence. Check values of registers taking care of memory access to the external flash device using Special Function Registers window.

- **Erase flash (e.g. Erase.isl)**

```

declare
{
  var int addr;
  var int i;
  var int w;
}

function void main()
{
  //offset - address where FLASH starts – must be set according to the target application
  addr=0x1800000;

  APIPrintString("**** Erase Sector");
  APIWriteMemory(0, addr+0xAAAA, 0xAAAA, 2);
  APIWriteMemory(0, addr+0x5554, 0x5555, 2);
  APIWriteMemory(0, addr+0xAAAA, 0x8080, 2);
  APIWriteMemory(0, addr+0xAAAA, 0xAAAA, 2);
  APIWriteMemory(0, addr+0x5554, 0x5555, 2);
  APIWriteMemory(0, addr+0xAAAA, 0x1010, 2);
  for (i = 0; i < 30; i = i+1)
  {
    APISleep(1000);
    w=APIReadMemory (0, addr+0x0000, 2);
    if (w==0xFFFF)
    {
      i = 30;
    }
  }
}

```

- ***Program word (e.g. ProgramWord.isl)***

```
declare
{
  var int addr;
  var int program_addr;
  var int i;
  var int w;
}

function void main()
{
  //offset - address where FLASH starts – must be set according to the target application
  addr=0x1800000;
  program_addr= addr+0x20000;

  APIPrintString("**** Program word");
  APIWriteMemory(0, addr+0xAAAA, 0xAAAA, 2);
  APIWriteMemory(0, addr+0x5554, 0x5555, 2);
  APIWriteMemory(0, addr+0xAAAA, 0xA0A0, 2);

  // program 0x1234 to flash address 0x20000
  APIWriteMemory(0, program_addr, 0x1234, 2);

  APISleep(1000); //delay

  //check if location was programmed
  w=APIReadMemory (0, program_addr, 2);
}
```

Disclaimer: iSYSTEM assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information herein.

© iSYSTEM. All rights reserved.