Freescale Semiconductor
Application Note

Document Number: AN4406
Rev. 0, 12/2011

KwikStik-based EzPort programmer

by: Witek Ewert, EMEA, FAE Team

1 Introduction

This document explains the use of the Kinetis KwikStik development tool as a memory programmer (EzProg) for Freescale MCUs which support memory access through the EzPort interface.

Contents

1.1 EzPort

EzPort is a serial memory interface present in Kinetis, ColdFire+ and ColdFire v2 devices. It allows the entire contents of the target's memory to be erased, programmed and read in-system through a simple SPI-compatible interface. An external controller can hold the target microcontroller in reset, and by asserting low a mode-select pin (~EZP_CS) and releasing the reset, the microcontroller enters a special programming mode called EZP. When in the mode, a range of commands may be sent to the microcontroller to control a full suite of programming tasks.

1.2 KwikStik

KwikStik is a low-cost, self-contained development tool for Freescale's Kinetis brand of Cortex-M4 32-bit MCUs. It features a Kinetis MK40DX256 MCU, which is in-system programmable via an on-board USB J-Link programmer, a MicroSD card slot, dot-matrix LCD and a secondary USB connector capable of OTG operation. The functionality of KwikStik can be extended by either its edge connector to a TOWER system, or by using the two, 20-pin TWRPI ("Tower peripheral interface" or "twippy") headers on the back side of the PCB.

1.3 MQX

Freescale MQX is a real-time operating system for Kinetis, ColdFire, ColdFire+ and Power Architecture. Among its numerous features, MQX provides task management, I/O device file abstraction and file management compatible with the FAT filesystem. The use of MQX greatly accelerated the development of this project, with its readily available drivers for USB Device, MicroSD and SPI peripherals.

MQX is available from Freescale as a complimentary download from freescale.com/mqx.

2 EzProg features

- Runs on an unmodified KwikStik (hardware revision 5 or greater[1])
- Supports Kinetis, ColdFire+[2], ColdFire v2
- Connects to the target system via standard 20-pin CF+ EZP/Cortex Debug header (TWRPI-EzPORT adapter board required)
- Stand-alone operation – MicroSD card used for programming script and memory image storage
- Accepts standard Motorola S-Record, Intel HEX and raw binary memory image files
- Configuration via human-readable programming script
- Windows utility to create programming scripts
- Memory commands: mass erase, blank check, program, verify

1.The hardware revision number is printed on the PCB on the top (LCD) side of the board, in the lower right corner.
2.See ColdFire+ chip errata.

3 Hardware description

3.1 Target device signal connections

Table 1. Target device signal connections

K40 pin	TWRPI pin	Target signal	0.050 EZP HDR pin
PTA6	L20 RESET_TWRPI	RESET	10
PTA14/SPO0_PCS0	R11 SPI: SS	EZP_CS	9
PTA15/SPI_SCK	R12 SPI: CLK	EZP_CK	4
PTA16/SPI0_SOUT	R10 SPI: MOSI	EZP_DI	8
PTA17/SPI0_SIN	R9 SPI: MISO	EZP_DO	6

EzPort programming signal assignment. L and R are the left and right TWRPI connectors, respectively.

- RESET – active low, resets the target
- EZP_CS – Chip select for EzPort SPI. If held low during reset, the target will enter EZP mode.
- EZP_CK – SPI clock
- EZP_DI – Serial data sent from the programmer to the target.
- EZP_DO – Serial data received from the target by the programmer.

3.2 Control signals

Table 2. Control signal assignment

Signal Name	K40 pin	TWRPI pin
READY	PTA10	R15 GPIO0
FAIL	PTA11	R16 GPIO1
GO	PTA12	R17 GPIO2

- READY – output, logic high indicates that EzProg is ready to run the programming script.
- FAIL – output, logic high indicates an error during the last run of the programming script.
- GO – input, if READY is high, a positive (rising) edge runs the programming script and reverts FAIL to low.

3.3 TWRPI-EzPORT user header

The 100 mil 2x7 pin header on the TWRPI-EzPORT PCB can be used to connect the EzPort signals to the target device, to supply either 3.3V or 5V to the target from KwikStik, and for ATE to control the programming process in an industrial environment.

Table 3.

Pin	Signal
1	TWRPI_5V
2	TWRPI_3V3
3	TGT_5V
4	TGT_3V3
5	EZP_RST
6	EZP_CS
7	EZP_DI
8	EZP_CLK
9	EZP_DO
10	GND
11	READY
12	FAIL
13	GO
14	GND

3.4 Modes of operation

EzProg can operate in two modes – memory programmer and MicroSD card reader. In memory programmer mode, the device is connected to a 5V voltage source through USB_JLINK Micro-USB connector (right-hand side of the KwikStik when facing the LCD). The device can also be connected to a PC through USB_K40 (left-hand side), in which case it functions as a USB mass storage device card reader.

4 Software description

The firmware for EzProg was written to run under MQX 3.7, in C, using IAR Embedded Workbench 6.21. Project files for CodeWarrior for MCU version 10.1 are also available.

The KwikStik BSP has been modified for this project, as the BSP available at the time of writing supported KwikStik Revision 4. A clone of the modified BSP is provided with the project files.

4.1 Task list

- Startup_Task – Initializes the MicroSD card (which includes waiting for a card to be inserted) and the LCD controller. Detects which USB port is connected to a voltage source and runs either MSD_Task or EZP_Task
- MSD_Task - acts as a Mass Storage USB Device card reader
- EZP_Task – runs the programming script

4.2 File list

- tasks.c – includes the MQX task list

  Dependencies: startup.h, msd.h, ezp.h

  Portability: This file contains no hardware-dependent code.

  startup.c / startup.h – The start-up function. Detects the operating mode based on USB voltage sensing and runs either the EZP task or the MSD task.

  Dependencies: ezp.h, msd.h

  Portability: USB voltage detection is KwikStik-specific. The word 'PORTABILITY' is used in the comments to indicate KwikStik-specific code.
- msd.c / msd.h – USB mass storage device task.

  Dependencies: usb_descriptor.h, microsd.h, debug.h

  Portability: MQX with USB device support required.
- microsd.c / microsd.h – MicroSD card support

  Dependencies: debug.h

  Portability: MQX and MCU with ESDHC controller required.
- cfg.c / cfg.h – Config file support

  Dependencies: None

  Portability: These files contain no hardware-dependent code.
- srec.c / srec.h – SREC file parser

  Dependencies: None

  Portability: These files contain no hardware-dependent code.
- ezp.c / ezp.h – EzPort programmer task

  Dependencies: misc.h, debug.h, srec.h, microsd.h, cfg.h

  Portability: These files contain hardware-dependent code. The word 'PORTABILITY' is used in the comments to indicate KwikStik-specific code.
- debug.c / debug.h – These files specify the global error reporting function.

  Dependencies: misc.h

  Portability: This file contains no hardware-dependent code.
- misc.c / misc.h – implement the user interface.

  Dependencies: debug.h, LCD, buzzer and touch sense drivers

  Portability: All the Misc_* functions are KwikStik-specific and need to be re-written for use on a different board. The word 'PORTABILITY' is used in the comments to indicate KwikStik-specific bare-metal I/O code.
- usb_descriptor.c / usb_descriptor.h – USB descriptors and functions for the USB mass storage device class. These files were copied from the <MQX root>\usb\device\examples\msd\disk project.

  Dependencies: None

  Portability: Portable to all systems with a USB device controller supported by MQX.

4.3 Programming scripts

A programming script is a text file describing all user-configurable programming settings and the sequence to be followed when the script is run for every target device. The syntax of EzProg programming scripts is compatible with the one used for Microsoft Windows .INI files.

4.3.1 Syntax

4.3.1.1 General

- Whitespace characters are ‘ ‘ (0x20, space), CR (0x0D, carriage return) and tab. They are ignored at the beginning and end of a line, before and after variable names, section names and values.
- New lines are signified by the LF (0x0A, line feed) character. As CR is ignored, files following both CRLF and LF newline conventions are accepted.
- Special characters are ‘”’, ‘[‘, ‘]’, ‘=’, ‘;’.

4.3.1.2 Sections

A programming script section begins with a line containing the section name in square brackets (‘[‘, ‘]’) and ends when the next section begins or EOF (end of file) is encountered.

Example:

[section]

Specifying a section of unknown type will cause an error.The first statement not being a comment, must specify the name of the first section in the file. Section names are case-sensitive.

4.3.1.3 Variable assignments

Sections contain a list of variables with values assigned to them. Assignment is represented by the equality sign ‘=’. A value can be either an integer or a string. An integer can be decimal or hexadecimal, with hexadecimal integers beginning with ‘0x’. The maximum size of integers is 32 bits. Strings must be delimited by ‘”’ (double quotation marks).

Examples:

```
foo = 12
hexfoo = 0x0C
bar = “kats”
```

Variable names and string values are case-sensitive.

4.3.1.4 Comments

A comment begins with a semicolon and takes up the whole line. Comments appended at the end of other statements are illegal.

4.3.2 Functional description

4.3.2.1 [target]

The [target] section describes the device to be programmed and general programming settings. This must be the first section of the programming script. There may be only a single [target] section.

Variables

Table 4. Variables

Name	Type	Description
image	string	File name of the SREC memory image to be programmed to the device
architecture	string	Target architecture, valid options are “Kinetis”, “ColdFire”, “ColdFire+”.
flash size	integer	The size of the target device’s Flash memory
sector size	integer	The Flash sector size for the device, 0 to disable sector boundary check when writing. Default: 0
flash program block	integer	Length of individual EzPort transfers. Default: 256
baud rate	integer	SPI clock rate. Default: 10000
reset delay A	integer	Delay, in milliseconds, before asserting RESET after setting EZP_CS. Default: 100
reset delay B	integer	Duration, in milliseconds, of the low RESET pulse. Default: 100
reset delay C	integer	Delay, in milliseconds, after RESET. Default: 100
touch button	integer	The touch-sense button to use to run the programming script (1-6) or zero to disable touch sensing. Default: 0
buzzer	integer	Non-zero to enable the on-board buzzer. Default: 0
coldfire div	integer	ColdFire v2 clock divider value. Default: 62
coldfire prdiv8	integer	Non-zero to enable ColdFire v2 8:1 clock pre-divider. Default: 0

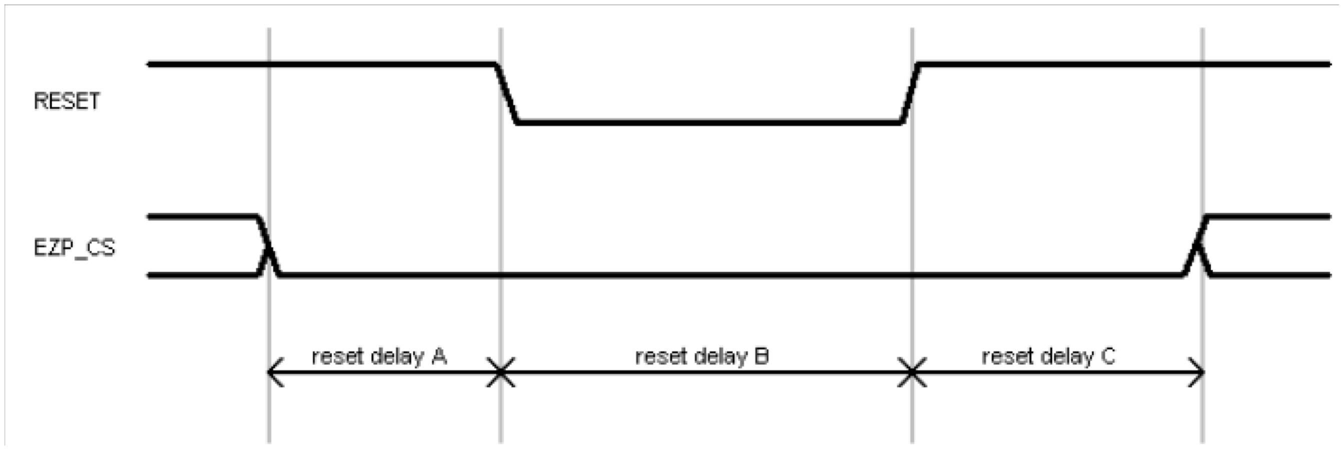


Figure 1. EZP Reset timing. For a normal reset, EZP_CS is driven high.

4.3.2.2 [reset]

When a [reset] section is encountered, EzProg performs a RESET on the device, after which either the user code is run on the target, or the target enters EZP mode and can be programmed.

Table 5. Variables

Name	Type	Description
ezp	integer	If non-zero, the target enters EzPort mode after reset (EZP_CS is held low).

4.3.2.3 [mass erase]

Erases the contents of the Flash memory on the device.

There are no variables for this section.

4.3.2.4 [blank check]

Reads the Flash memory and checks if it has been erased. Some locations are skipped depending on the selected architecture:

Table 6. Skipped flash locations

Architecture	Location(s)
Kinetis	0x400 – 0x40C
ColdFire v2	0x400 – 0x417
ColdFire+	0x400 – 0x40F

If a value different than 0xFF is read, a programming failure is indicated. There are no variables for this section.

4.3.2.5 [program]

Programs the SREC image, specified in the [target] section, to the device. No verification is performed during programming.There are no variables for this section.

4.3.2.6 [verify]

Reads the contents of the Flash memory and compares it to the contents of the SREC image file specified in the [target] section. If the image file and Flash contents do not match, a programming failure is indicated.

There are no variables for this section.

5 Typical use example

5.1 Preparations

You will need:

- PC running Windows and with IAR Embedded Workbench installed
- KwikStik Rev. 5 or later
- MicroSD card – a 2GB SanDisk card was used.
- USB A to micro-B cable
- TWRPI-EzPORT board (or a custom-made programming cable)
- A target board with a Kinetis, ColdFire V2 or ColdFire+ MCU.
- 20-way ribbon cable with 50mil IDC connectors at both ends – for programming Kinetis and ColdFire+ TWR boards
- Jumper wires for other systems
- A jumper

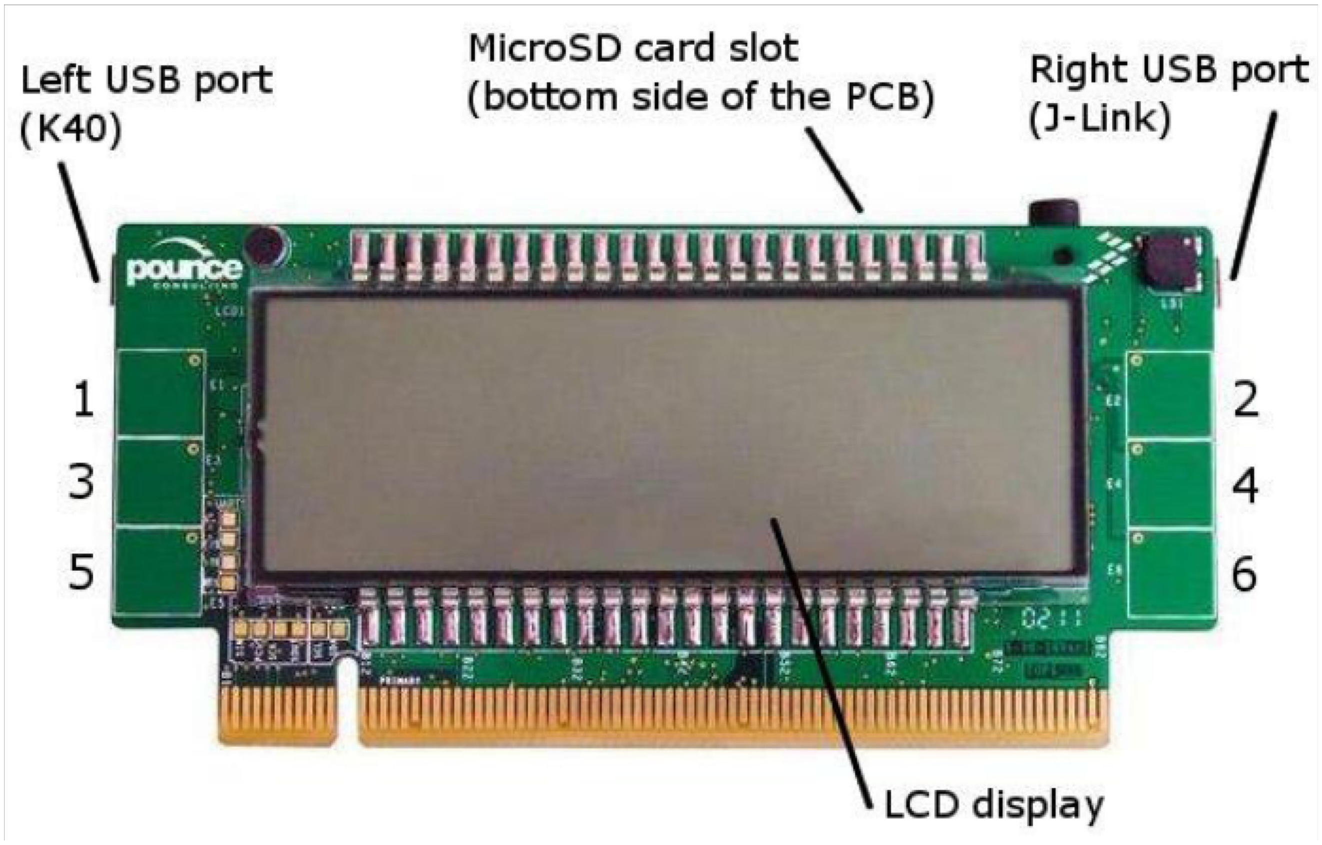


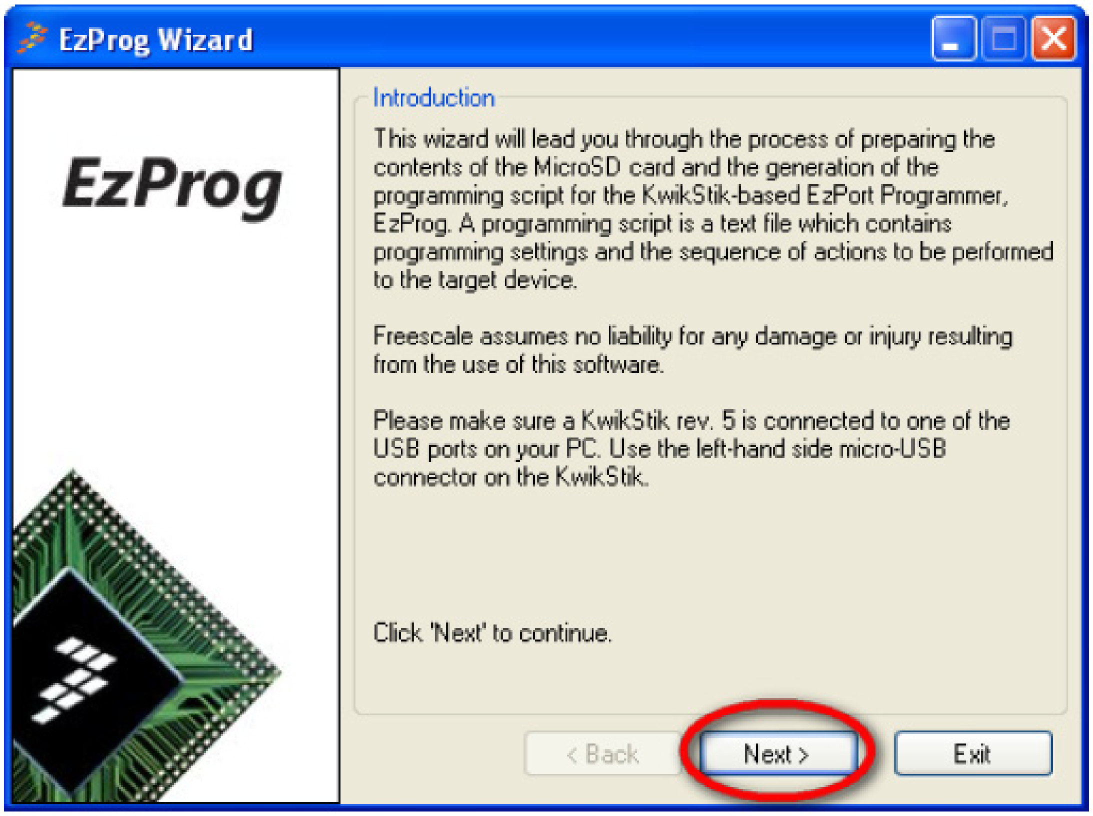
Figure 2. KwikStik LCD view

5.2 KwikStik firmware upload

1. Extract the EzProg binary package to a convenient location. The following two folders will be visible:
- EzProg_fw - contains the KwikStik firmware
- EzProg Wizard – contains a Windows tool for preparing the programming files
2. Connect the right-hand side KwikStik USB port (J-Link) to your PC.
3. Open the IAR project for flashing the KwikStik, Ezprog_fw\ezprog.eww. Click the 'Download and Debug button on the toolbar. Press Ctrl+Shift+D to end the debugging session and close IAR.

5.3 Preparing the programming files

4. Remove the USB plug from the J-Link USB port on the KwikStik and connect it to the port on the left (K40).
5. KwikStik should display "CARD" on the LCD screen. Insert the memory card to the MicroSD slot. Windows will detect a new USB mass storage device and the LCD screen goes blank. When Windows has installed the new USB mass storage device, the USB icon is displayed in the top left-hand corner of the LCD screen.
6. Run the EzProg Wizard - ezprog_gui.exe. Click [Next>].



This wizard will lead you through the process of preparing the contents of the MicroSD card and the generation of the programming script for the KwikStik-based EzPort Programmer, EzProg. A programming script is a text file which contains programming settings and the sequence of actions to be performed to the target device.

Freescale assumes no liability for any damage or injury resulting from the use of this software.

Please make sure a KwikStik rev. 5 is connected to one of the USB ports on your PC. Use the left-hand side micro-USB connector on the KwikStik.

Click 'Next' to continue.

Figure 3. First step running the ExProg Wizard

7. The wizard will show a list of all removable drives on the system. Choose the logical drive which corresponds to KwikStik. You can also specify a custom path to copy the generated files to instead remember to copy the files to the root folder of the memory card afterwards. You may decide to store a selection of firmware images to the memory card, and promote one to the root folder when you are ready to program that image. Click [Next>].

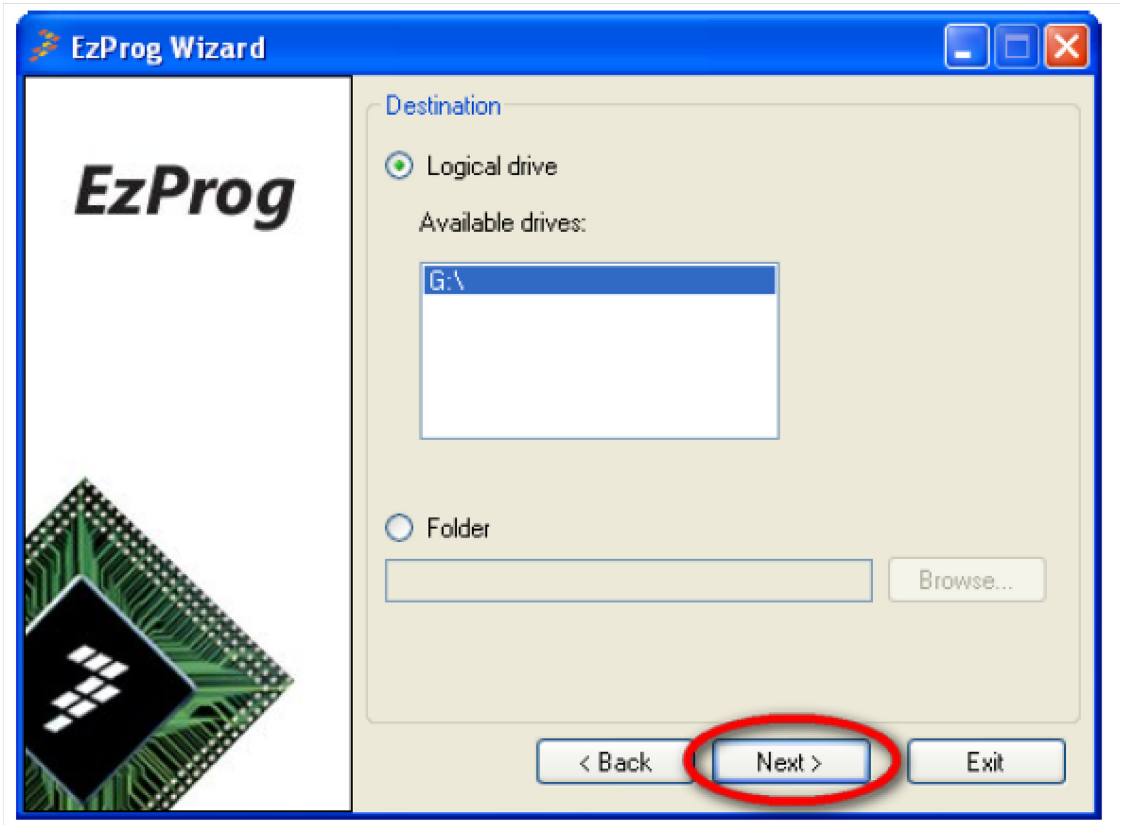
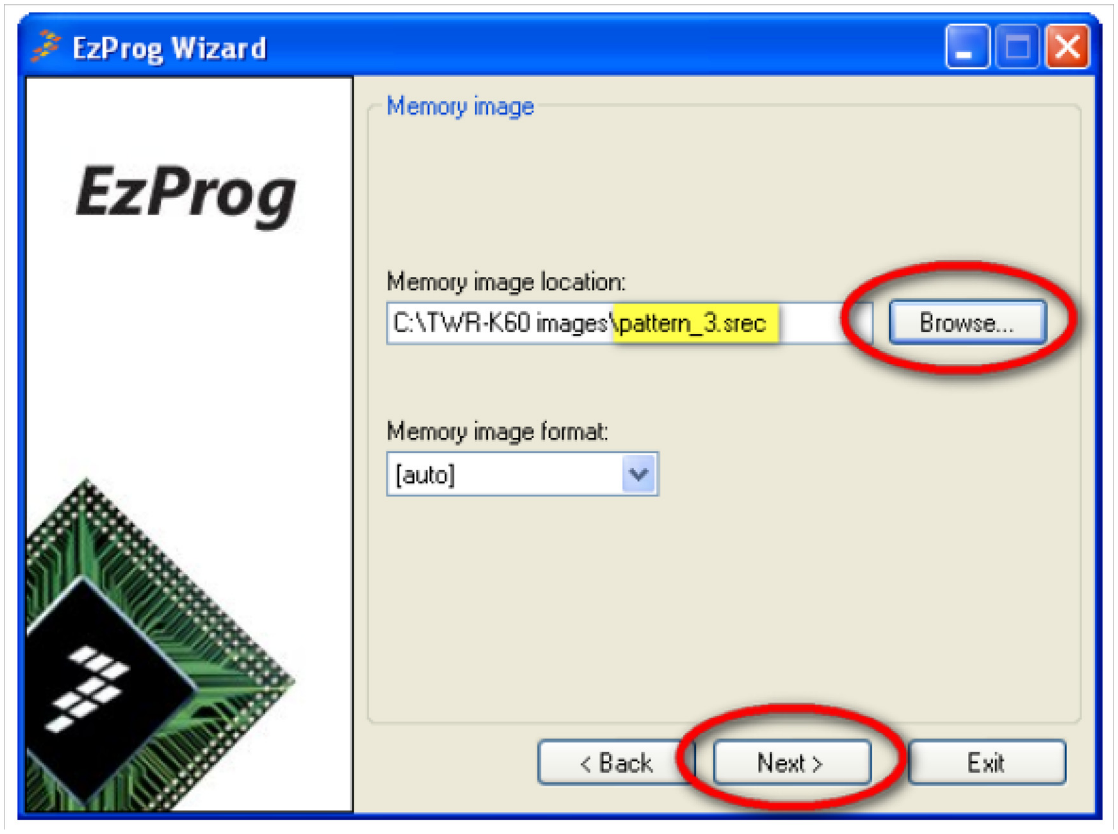


Figure 4. Destination file in EzProg Wizard

8. Click [Browse] and select the Flash image file to program to the device. If the file extension is not on the list, you need to pick the proper memory image format from the drop-down list. If you select a format that is not SREC format (for example BIN or HEX), EzProg will convert the file to SREC format using objcopy.exe. Click [Next>].

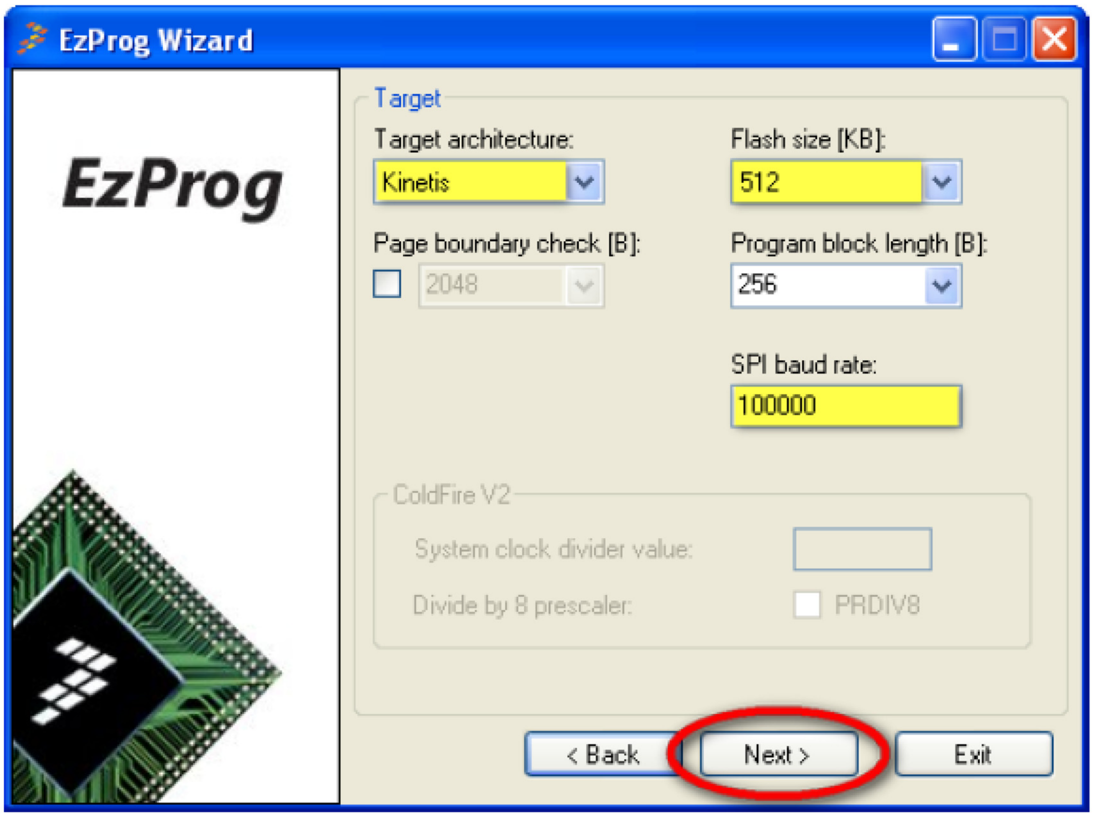


Memory image location: C:\TWR-K60 images\pattern_3.srec

Memory image format: [auto]

Figure 5. Memory image location in EzProg

9. Choose the target architecture and Flash memory size. You may also want to increase the SPI baud rate to 100000. In this example, EzProg is being used to program a Kinetis MK60DN512.



Target architecture: Kinetis
Flash size [KB]: 512
Page boundary check [B]: 2048
Program block length [B]: 256
SPI baud rate: 100000

System clock divider value:
Divide by 8 prescaler: PRDIV8

Figure 6. Target in EzProg

If the target is a ColdFire V2 MCU, you will need to enter the clock divider settings. The method to calculate these values is described in ColdFire V2 reference manuals, refer to ‘Writing the CFMCLKD Register’, ‘ColdFire Flash Module (CFM)’ chapter available at www.freescale.com.

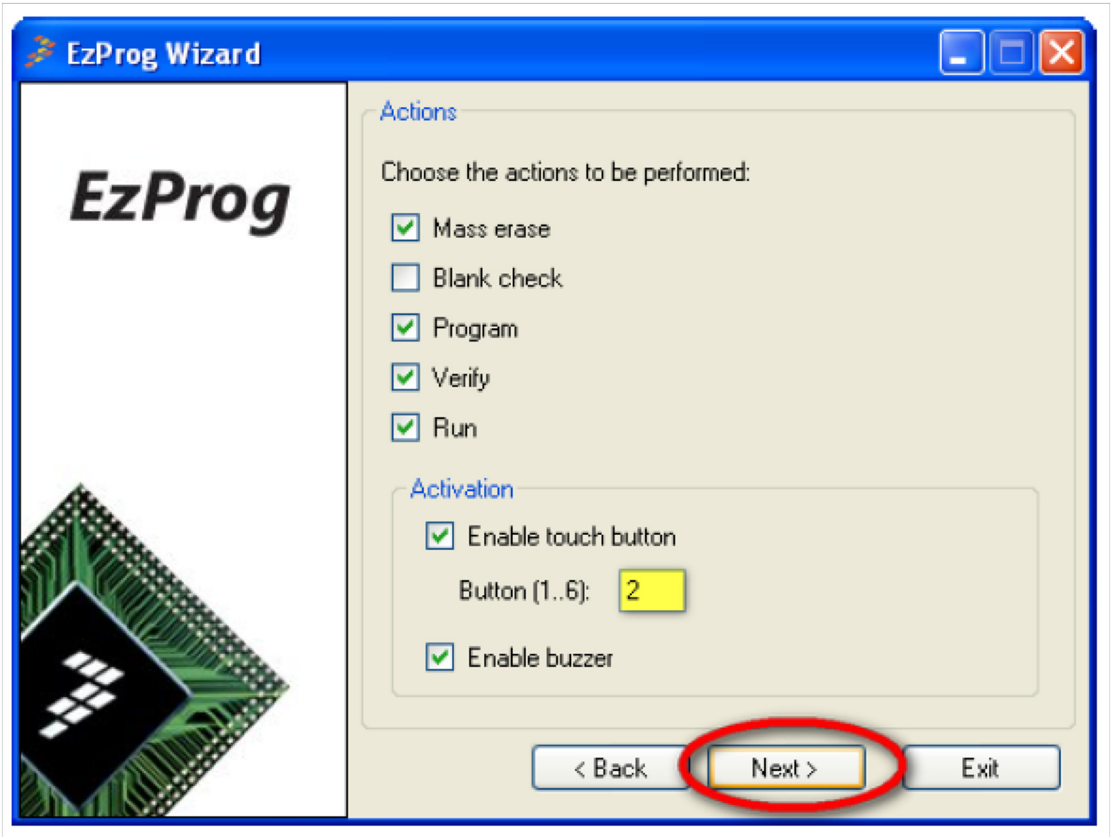
If ‘Page boundary check’ is enabled, an attempt to write across a page boundary would result in an error. This option is only usable with Kinetis and ColdFire+ targets.

The ‘Program block length’ is the maximum number of bytes that can be read from or written to the memory in a single EZP transfer. It must be a multiple of the size of the smallest write-addressable memory unit (write address alignment) and must divide the page size without a remainder if write requests must fit within one page.

Click [Next>].

10. Choose the actions to be performed each time a target device is about to be programmed. The device will reset into EzPort mode before any of these actions are performed.

    You can specify if programming is to be started when a touch-button is activated, in addition to when there is a rising edge on the gpio GO input. If you turn on touch sensing, you may also want to enable the on-board buzzer for a better user experience.



Actions

Choose the actions to be performed:

- [✓] Mass erase
- [ ] Blank check
- [✓] Program
- [✓] Verify
- [✓] Run

Activation

- [✓] Enable touch button

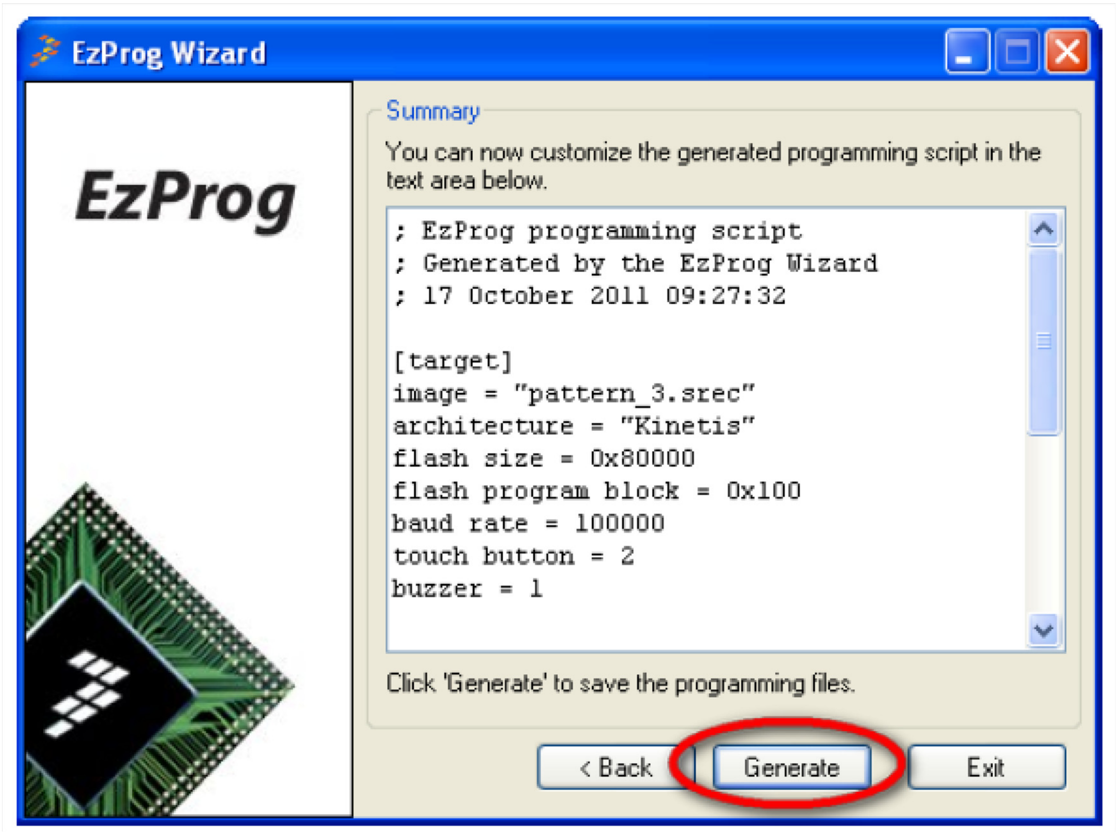
Button (1..6): 2

- [✓] Enable buzzer

Figure 7. Actions and activation in EzProg

Click ‘Next’.

11. This page lets you review the programming script and modify it if necessary.



You can now customize the generated programming script in the text area below.

```
; EzProg programming script
; Generated by the EzProg Wizard
; 17 October 2011 09:27:32

[target]
image = "pattern_3.srec"
architecture = "Kinetis"
flash size = 0x80000
flash program block = 0x100
baud rate = 100000
touch button = 2
buzzer = 1
```

Click 'Generate' to save the programming files.

Figure 8. Summary and generate window

After you click ‘Generate’, the programming script and memory image file will be written to the KwikStik (or to the path you specified in step 7). The EzProg firmware on the KwikStik will only process S-Record files, so `objcopy.exe` will be run to convert binary and HEX files to SREC.

12. Click ‘Safely remove hardware…’ in the wizard. Stop the mass-storage device mounted in Step 7 and disconnect it from the PC.

5.4 Connecting to the target system

13. Connect KwikStik to the target system. You can either use the EZP 0.050in. header and an IDC ribbon cable, or the 100mil header and connect each signal individually. See the table in Section 3.3, “TWRPI-EzPORT user header for the header pinout. If using the EZP header, install a jumper on the 100mil header:

Table 7.

Pins	Target voltage source
1-3	KwikStik, 5V
2-4	KwikStik, 3.3V
None	Target is self powered

You can also apply 5V to pin 3 or 3.3V to pin 4 to power the target separately.

5.5 Programming

14. Connect the right-hand side KwikStik USB port (J-Link) to a PC or a 5V USB mains adapter. KwikStik should display ‘READY’ on the LCD. Press the touch-button you had selected in step 10 or momentarily short the pins 13 and 14 on the user header. The green LED will come off and the programmer will perform the actions selected in step 10.

    If the device has been programmed successfully, the green LED will come back on.

    If programming fails, the red LED will come on along with the green RDY led, and the word ‘FAIL’ will be displayed on the LCD.

    If an error occurs, only the red LED will come on and you will see ‘ERROR’ on the LCD.

6 Modifying the code

A source package containing all project files for the programmer firmware and the Wizard are available for download from the Freescale website.

The KwikStik firmware was developed with IAR Embedded Workbench. It runs under MQX 3.7 operating system with the KwikStik patch applied to it. The original BSP for KwikStik needed to be modified, so the project uses a cloned BSP under the name ‘kwikstikk40x256_ezprog’.

EzProg Wizard was written in Microsoft Visual C++ .NET 2008. It is a Windows Forms application.

6.1 Portability guide

6.1.1 Porting to another MQX system

The word ‘PORTABILITY’ is used in the comments to indicate parts of the code which may need to be modified when porting to a different board/platform. Refer to Section 4.2, “File list in this document for portability details.

6.1.2 Re-using the code on a non-MQX system

Files which can be re-used:

- ezp.c / ezp.h – remove MicroSD_Mount, implement GPIO support and SPI as file I/O. Add templates for Misc_* functions.
- cfg.c / cfg.h, srec.c / srec.h – change MQX_FILE_PTR to FILE*.

6.2 Potential improvements

- Add support for industry-standard I2C and SPI memories. 24Cxx EEPROM code for MQX can be found in <MQX root>\mqx\examples\i2c\. SPI memories are very similar to EZPort devices and few changes need to be made to program them.
- Add USB mass storage host controller support. This may require either replacing the diode blocking KwikStik to USB-K40 current with a shunt, or building a USB connector adapter to

bypass the diode. USB class code for MSD host is available in <MQX root>\mfs\examples\mfs_usb.
- Security features, e.g. programming a serial number to the target’s write-once memory, then calculating a CRC and programming it to the Flash.
- Blank check can be significantly accelerated by using an FCCOB command instead of reading the entire memory over SPI.

7 Bibliography

- MCF51F128 RM
- TWR-MCF51JF128 UG
- MCF52259 RM
- TWR-MCF5225x UG
- K60N512 RM
- TWR-K6N512 UG
- TWR-K6N512 schematic
- KwikStik UG
- KwikStik schematic

8 Conclusion

The EzPort programmer described in this Application Note is a simple, yet powerful way to device program supported Kinetis, ColdFire and ColdFire+ microcontrollers. By developing the firmware to run on a KWIKSTIK-K40 development board, Freescale has provided route to a very low-cost programmer. The firmware supports many of the common tasks required for production programming, but the firmware is provided in source form so that the user can make further enhancements. It is necessary to use the MK40X256 SPI peripheral to implement the EzPort communication. This peripheral terminates at one of the TWRPI connectors on the rear of the KwikStik, and so a simple adaptor PCB has been developed. The PCB simply translates the TWRPI footprint to a standard 20-way Samtec FTSH header for a Cortex Debug header cable. The schematic and PCB design files are published to allow the user to manufacture a similar PCB, or to modify the PCB for a custom application. Freescale MQX has been used to develop the firmware, the MFS file system and USB Device stack are a key part of the EzProg solution. Using a real-time operating system, and a task-based approach to the firmware greatly reduced the software development time.