Embedding it better...

µTasker Document

µTasker – EzPort Cloner

Table of Contents

1. Introduction

Various Freescale Coldfire and Kinetis devices include an EzPort, which allows reading and writing the processor's internal Flash memory in a similar fashion to reading and writing SPI Flash. This interface is of interest when new parts are to be programmed as an alternative to using a JTAG, SWD or other debug interface, as well as for performing a mass erase to un-protect parts that have been previously secured or whose debug interfaces have been disabled.

In some circumstances the use of EzPort could be in an emergency, for example to recover a board that no longer operates together with its normal debugging interfaces, but it also offers a solution for automating the software programming in volume production.

This document details the EzPort control interface that is integrated into the ***µTasker*** project for quick and easy experimental use or for efficient cloning of the software from one board to another. The cloning application can also be used to highly automate programming in a production environment.

Although there are many processors and boards that support the EzPort and could be used for such work and for cloning, the popular FRDM-K64F is used as test vehicle since it is both a popular board and has large internal memory space. It will be shown how this board can be used to control EzPort operation when connected to another board as well as how it can clone its own software to a second board.

2. Connecting the EzPort of the FRDM-K64F

Unfortunately the EzPort on the FRDM-K64F is not fully accessible on its board connectors and so a little modification is required to allow an EzPort master processor to be connected to it. This section shows how to prepare the EzPort slave for later connection to an EzPort mater. Since the EzPort master is flexible as to which signals it uses for the operation nothing special is required for its own connection, which is simply shown at the end of the section as a part of the complete assembly.

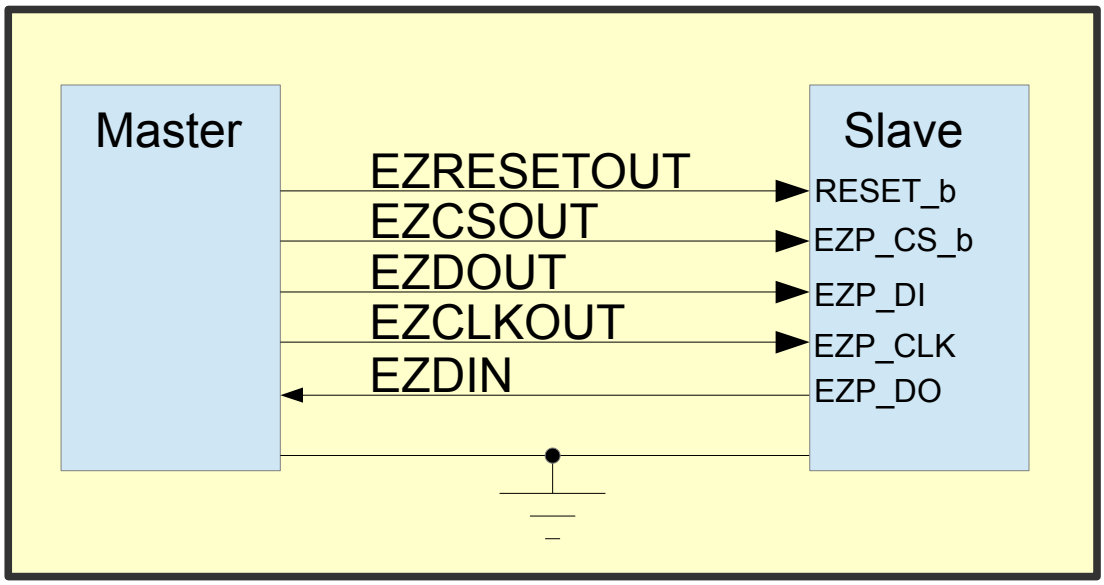

Figure 1: EzPort Signals

Figure 1 shows the signals that need to be connected between the EzPort master and EzPort slave and their drive directions. In some situations the master may also supply power to the slave but how this is performed will depend on voltages present and required and so it is assumed that each is independently powered and only a common ground signal is necessary.

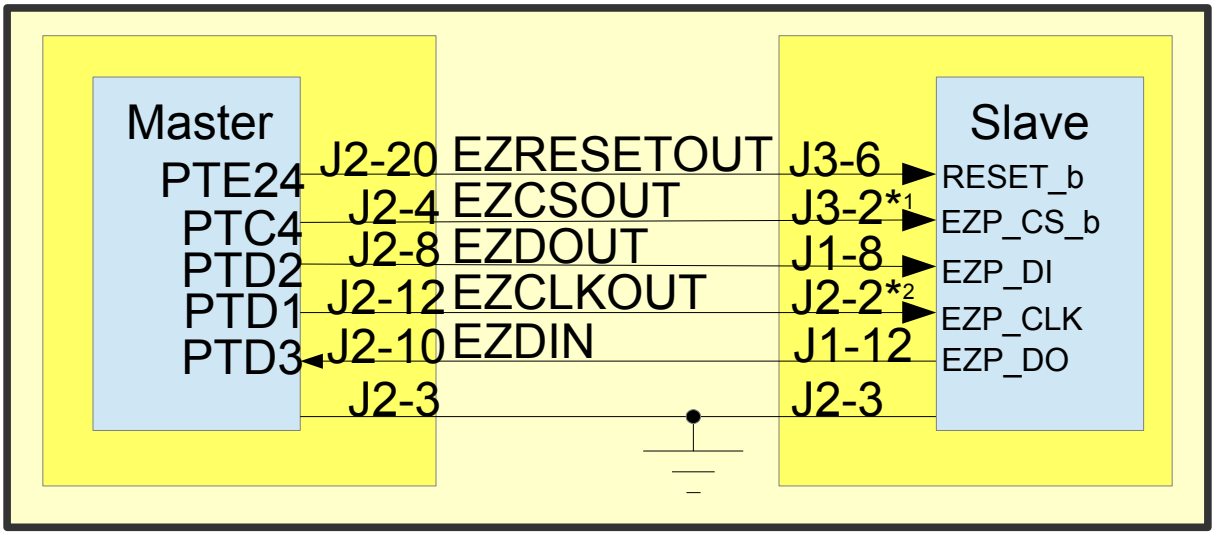

Figure 2: FRDM-K64F EzPort Connections

Figure 2 shows a practical connection between two FRDM-K64F boards, whereby the following notes are relevant and important:

***1** The FRDM-K64F doesn't have the `EZP_CS_b` (PTA4) signal normally accessible on a connector but instead it is connected to switch SW3. **A wire connection can thus be made to terminal 1 of this switch by connecting it to the free pin on connector J3 (J3-2).**

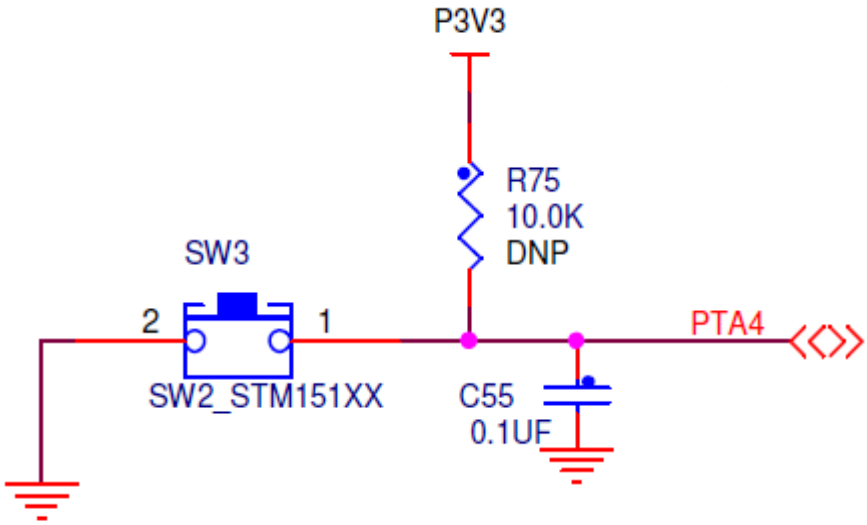

Figure 3: FRDM-K64F Circuit for PTA4 (`EZP_CS_b`)

Note further that the capacitor C55 is mounted on some boards (depending in exact board revision) and needs to be removed since it otherwise distorts the chip select signal.

***2** J11 needs to be removed in order for the OpenSDA to `SWD_CLK` not to disturb the operation. [*J11 is not mounted on a new board and consists of a track connection on the bottom side of the board- this should be cut and the a normal jumper connector can be soldered in instead*.]

WARNING: There are two common FRDM-K64F revisions: D and E. `EZP_CLK` is only connected to J2-2 on the Revision D version and this pin is instead connected to PTC12 on the Revision E. This means that owners of revision E boards will need to **temporarily connect J11-1 to J2-2** at the slave so that the EzPort's clock signal reaches its processor.

2.1 Verifying that the EzPort is not Disabled

Some Kinetis parts allow the EzPort to be disabled in their flash configuration. In order to be able to control an EzPort slave the slave must not have enabled this protection, otherwise it will not be possible to work with its EzPort.

Devices that have been fully erased will not have the EzPort disabled.

Devices that have software running on them *may* have their EzPorts disabled.

If a ***µTasker*** software controlling the flash configuration is loaded (either a stand-alone application or a boot loader) it will control this with the following setting (in `app_hw_kinetis.h`):

```
#define KINETIS_FLASH_CONFIGURATION_NONVOL_OPTION  (FTFL_FOPT_EZPORT_DISABLED |
FTFL_FOPT_LPBOOT_NORMAL | FTFL_FOPT_NMI_DISABLED)
```

Note that this software would disable EzPort access (this is sometimes performed to stop the EzPort mode being entered due to the 0.1uF capacitor C55 connected to the `EZP_CS_b` input (see figure 3)).

```
#define KINETIS_FLASH_CONFIGURATION_NONVOL_OPTION  (FTFL_FOPT_EZPORT_ENABLED |
FTFL_FOPT_LPBOOT_NORMAL | FTFL_FOPT_NMI_DISABLED)
```

By using `FTFL_FOPT_EZPORT_ENABLED` rather than `FTFL_FOPT_EZPORT_DISABLED` the operation can be allowed.

3. EzPort Command Interface

The EzPort interface is an option in its command line index and can be enabled using the define EZPORT_CLONER in the file debug.c in the ***µTasker*** project. When enabled, the following commands are added to the I/O menu:

```
ez_config      Prepare EzPort interface
ez_status      Read status
ez_reset       Reset target and enter EzPort mode
ez_start       Allow slave to start
ez_cmd         Send command [val]
ez_rd          Write [add] <len>
ez_prepare     Prepare for programming <speed> - only Coldfire
ez_unsec       Unsecure                        - only Kinetis
ez_sect        Erase sector [add]
ez_bulk        Perform Bulk erase
ez_prg         Program value [add] [val]
ez_verif       Verify [<b(lank)>|<c(loned)>]
ez_clone       Clone software
ez_s_clone     Securely Clone software
```

The initial commands in the list allow basic operations to be performed, which, when combined in the corresponding sequence, allow various functions to be executed as well as individual commands to be tested for test or experimental purposes.

The final two commands allow the complete software to be cloned as a single operation from the host board to the connected slave. The second of the two also sets the security of the target processor so that it is not possible to read out the programmed software, neither using its debug interface nor using the EzPort. *A secured target device can however always be unsecured by commanding the bulk erase command*.

3.1 Command “ez_config”

This command is used to configure the ports at the EzPort master ready for operation. Before using this command the pins will not have been configured and so could have a different function or be undefined.

The command configures the outputs and drives them to default states, whereby the **EZRESETOUT** is low (asserted reset) to hold the slave in its reset state.

EZCSOUT is logic '0', which will be used to latch the EzPort mode when the slave exits reset.

EZDOUT and **EZCLKOUT** are driven to '1' and EZDIN is configured as an input so that it can later read serial data from the EzPort slave.

3.2 Command “ez_reset”

This command is used to take the EzPort slave out of reset and latch it into the EzPort mode of operation. This means that the **RESETOUT** signal is taken high while the **EZCSOUT** remains low, which causes the EzPort to exit reset to the EzPort mode rather than starting normal operation.

The **EZCSOUT** signal is then set to '1' (after adequate delay to ensure that the slave has correctly entered the EzPort mode) in preparation for subsequent communication.

3.3 Command “ez_status”

This command is used to read the status register of the EzPort slave. This allows checking that the communication is operating and whether the EzPort slave is in a secured state or not.

This is a very fundamental command that operates by sending the EzPort command RDSR (value 0x05) to the slave than then reading back a single byte of data that it returns in response.

	7	6	5	4	3	2	1	0
R	FS	WEF			FLEXRAM	BEDIS	WEN	WIP
W								
Reset:	0/1	0	0	0	0/1	0/1	0	1

Reading the status register is a simple way to ensure that the slave is connected and is in the EzPort mode. If 0xff is read it indicates that the slave is not communicating an is not in the EzPort mode.

Typical values returned are 0x00 or 0x80 after the reset into the EzPort mode.

- 0x80 is the secure bit, indicating that the device is in the secured mode and can presently no data can be read from it.
- The Write Enable bit (0x02) and Write In Progress (0x01) are never set out of reset.

3.4 Command “ez_cmd”

This command is used to write a byte of data to the EzPort slave's control register. This enables changing its status, writing commands and also writing data. A single byte of data is specified (eg. `ez_cmd 6`) and this command is useful to test the EzPort command set and experiment with the effect on the EzPort status.

For example “`ez_cmd 6`” will cause the Write Enable bit (0x02) in the status register to be set and “`ez_cmd 4`” will clear the same [this is a further simple verification of the correct EzPort operation]. When this command is executed, the status register value is always displayed as feedback.

3.5 Command “ez_rd”

The read command allows data to be read from the slave's flash and displayed. For example “`ez_rd 1000 32`” will read and display 32 bytes form the address 0x1000. Reads should always be from long-word aligned addresses (divisible by 4) and will be aligned in case incorrect values are entered.

In case no value for the quantity of bytes to be displayed is given it defaults to 16, or else the previously used value

Repeating “`ez_rd`” will repeat the read from the latest address, meaning that the command can be repeated to automatically progress through the memory range.

A secured device will always return 0xff for each data value read.

3.6 Command “ez_unsec”

The un-secure command can be used to unsecure a device that can not be read due to its security setting not allowing this (the status register shows 0x80 set).

It is assumed that the user has already used commands to set the slave to the EzPort mode and has seen that it is secured. When commanded a mass erase is performed after first verifying that the device really is in the secured state. This command will not perform the bulk erase if the security flash is not set ,in order to unintentionally erasing a device that isn't secured.

3.7 Command “ez_prepare”

This command is available/required only by Coldfire devices and not Kinetis parts.

This command is a short sequence of commands that prepare the EzPort slave for subsequent programming operations. It includes configuring a speed so that the internal timing of such operations is correct for the part, based on the clock speed that it is presently using. If the speed is set incorrectly the programming operations could either be too short (unreliable programming/erasing) or too long (unnecessary programming stress and possible damage in the worst case).

The default speed that is set is suitable for programming the same part as used on the EzPort master. Other frequencies can be configured by commanding “`ez_prepare 16000000`” where the clock speed 16MHz is used (this parameter is optional and only used to override the default speed).

3.8 Command “ez_sect”

The section erase command is used to delete a single sector in the Flash in the slave. For example “`ez_sect 2000`” will erase the sector in which the address 0x2000 is in. The address should be aligned to a 128 bit address boundary (divisible by 16) and incorrect addresses are automatically aligned accordingly.

The command automatically sets the Write enable bit before commanding the erase.

3.9 Command “ez_bulk”

This command is used to perform a bulk erase of the Flash in the slave. It is useful for unlocking secured processors so that the can be accessed via their debug interfaces and programmed normally. In the case of the Kinetis the bulk erase is followed by a write of the value 0xfe to the location 0x40c so that the device remains unsecured after subsequent resets, which represents the state that new chips are delivered in.

This command incorporates the complete configuration and programming preparation.

The command automatically sets the Write enable bit before commanding the erase.

3.10 Command “ez_prg”

This program command is used to test writing to the slave's Flash memory.

erase a single sector in the Flash in the slave. For example “ez_sect 2000” will erase the sector in which the address 0x2000 is in. The address should be aligned to a 128 bit address boundary (divisible by 16) and incorrect addresses are automatically aligned accordingly.

"ez_prg 2000 20" writes 16 bytes with the values 0x20, 0x21, ..0x2f to the address starting at 0x2000. If no pattern is defined it will be set to 16 x 0x55.

The special command "ez_prg U" can be used to program unsecured flash configuration state after a sector erase of the first Kinetis Flash memory sector in order to avoid leaving the device in a secured mode.

The command automatically sets the Write enable bit before commanding the programming operation.

3.11 Command "ez_verif"

This verify command can be used to either verify that the device is erased or that its Flash content is an exact match of the cloning master.

"ez_verif b" causes a blank check to be executed and "ez_verif c" a cloning match verification. *The default is a cloning match in case no parameter is entered*.

Note that the blank check at the address 0x40c at a Kinetis is considered to be blank since it is the state in an unsecured device configuration.

The verification command can be executed without previously commanding the EzPort mode using the low-level command sequences.

3.12 Command "ez_clone"

This command is used to erase the Flash in the EzPort slave and clone the program in the Flash memory of the EzPort master. It also reads back the programmed memory to verify its accuracy.

This command incorporates the complete configuration and programming preparation, erase, programming and verification. Since it is intended for cloning to the same processor as the EzPort master is running on it doesn't need any (optional) frequency setting.

A verify sequence is always executed after the cloning has been executed and the target is reset normally so that its new code executes as long as the code could be successfully verified.

The verification command can be executed without previously commanding the EzPort mode using the low-level command sequences.

3.13 Command "ez_s_clone"

This command is the "*secure*" clone command. It is the same as the "ez_clone" command but completes the programming by securing the target so that its content can no longer be read using the debug or EzPort. A secured target can be unsecured again by executing the "ez_bulk" command.

The target is secured even if the software loaded to the master is not secured, which means that the user doesn't need to prepare a special software version that has the security pre-set.

A verify sequence is always executed after the cloning has been executed and the target is reset normally so that its new code executes as long as the code could be successfully verified. It is to be noted that, although the device has been prepared for secure mode of operation, this only takes effect after the reset has been performed.

The verification command can be executed without previously commanding the EzPort mode using the low-level command sequences.

4. Defining Skip Regions for the Cloner

When cloning software from a master to a slave it may not always be desirable to copy the entire Flash content from the master. For example, there may be areas of Flash memory that contain parameters that should initially be empty.

When the define EZPORT_CLONER_SKIP_REGIONS is enabled skip regions can be defined which the cloner will jump over when cloning, or when verifying cloned code. *They will not be skipped when verifying that the Flash content is erased*.

The following example shows a skip region defined to avoid cloning to two areas.

```
#define PARAMETER_BLOCK1_START         (128 * 1024)
#define PARAMETER_BLOCK1_END           (PARAMETER_BLOCK1_START + (4 * 1024))
#define FILE_SYSTEM_SPACE_START        (512 * 1024)

static const CLONER_SKIP_REGION ulSkipRegion[] = {
    {PARAMETER_BLOCK1_START, PARAMETER_BLOCK1_END},
    {FILE_SYSTEM_SPACE_START, (FLASH_START_ADDRESS + SIZE_OF_FLASH)},
    {0}                                                    // end of list
};
```

This results in the first area of 4k at 0x20000 to be avoided and also all Flash space after 0x8000.

The final zero entry signifies the end of the list.

5. Cloning Software in a Production Environent

Software cloning is possible with a single command “ez_clone” or “ez_s_clone” as appropriate, which makes its use fairly efficient – the result of the cloning process is also clearly displayed and the target starts execution of the code immediately after being successfully programmed.

The command can also be repeated by repeating the last command by using the up-arrow key to reselect it.

The following shows an example of a cloning operation sequence in action:

```
#ez_clone

Cloner V1.0 cloning
Bulk erase issued

#Sector [0x00000000] erase issued
.Unsecured
.....................................................................................
................................................
Region: 0x00022000 - 0x000fffff skipped

Completed - verifying
.
Target Not Secured
.....................................................................................
................................................
Region: 0x00022000 - 0x000fffff skipped

Completed - target starting
```

In case further simplification is required the user can simply modify the code to command the process by another means, such as by pressing a button/input, with the result displayed on an output – eg. Lighting a green LED for success or a red LED for failure. This means that it can be customised to require only minimum control but clear indication of the result, which is often a priority in such environments.

Since the commands are also possible via USB, Telnet, as well as the serial interface it is easy to integrate the step into a more automated production environment too.

6. Software Realisation and Technical Details

The EzPort interface requires the master to control 5 signals. The serial protocol that the EzPort uses is SPI compatible and so an SPI interface can be used to control the bi-directional SPI lines (4 wires) or these can be bit-banged when either no SPI interface is available at the master or else in case these lines can not be connected due to some particular hardware restriction.

The SPI interface can be used in either CPOL = 0, CPHA = 0 or CPOL = 1, CPHA = 1 modes.

The maximum clock speed that the EzPort can use is half of the target's system clock frequency, with exception of reading data, where it is one eighth of the system clock speed. Assuming that the target's system clock is 20.971 MHz (as is the case for the K64 in the reference), this gives maximum 2.62MHz data read and maximum 10.48MHz command speeds.

6.1 Clock Setting

This paragraph is only valid for Coldfire EzPort – *Kinetis doesn't require a setting*.

It is important that the clock setting matches the clock used by the slave's processor when writing to its Flash in its EzPort mode. The clock setting is used by the EzPort's internal programming algorithm to time the flash operations. Should the clock be set too high it is possible that erase and programming operations are unreliable due to the fact that the programming operations will be performed for too short a time. In case of a clock value set too low it will cause the programming operations to be applied longer than normal, which stresses the device and can, in the worst case, lead to damage or early failures.

7. Conclusion

This document has introduced the EzPort which is available in a number of Freescale Coldfire and Kinetis processors as well as the set of commands that are optionally available in the ***µTasker*** project to allow a board to act as an EzPort master.

As well as the ability to execute basic commands to experiment with the interface it is possible to erase the Flash at the EzPort slave and clone the master's software to it. The cloning can optionally secure the target device so that the cloned software can no longer be read via the EzPort or debug adapter.

The cloning operation enables efficient and simple target in a production environment.

Modifications:

- V1.00 7.12.2015: - First draft
- V1.03 18.12.2015: - First release