1 Introduction

This application note describes how to implement a simple gang programming technique for Freescale's MC56F8013 devices using the JTAG interface. The proposed programming method does not verify programming success in all devices, consequently it is necessary to include a mechanism for verification. For this purpose a flash checksum routine was developed in the application’s start up code. It verifies whether the flash content is equal to the expected code. The main objective of gang programming is to reduce the programming time.
2 Production Flash Programming

There are four ways to program the flash blocks in a factory environment. This is described in AN3118 Production Flash Programming for the 56F8000 Family:

- Using the serial bootloader present in the internal flash
- Using a commercially available device programmer
- Using the JTAG/OnCE™ port
- Using GPIO pins with a custom bootloader

The programming data speeds are estimated below:

- SCI serial bootloader = 2.5 k words per second
- In-circuit JTAG/OnCE port = 3 k words per second

These two programming methods are mainly suitable for low/mid volume production lines, and generally are too slow for high volume lines.

Listed at the Freescale website (www.freescale.com) under the Freescale Alliance Member Portal are several third parties that offer a variety of production line flash programming solutions.

The solution described in this application note is for a flash gang programming that is easily implemented.

3 JTAG/EOnCE

The flash gang programming method uses a command converter interface. This interface is described in Suite56™ Parallel Port Command Converter User’s Manual.

This interface allows the connection of digital signal controllers (DSC) with a PC host to debug and program Freescale DSCs via JTAG/EOnce (Figure 1 and Figure 2).
Another possible hardware for the JTAG/EOnce interface is the CodeWarrior USB TAP. This tool is connected to the USB port.

The gang programming method works with any of these tools. They connect with Freescale DSCs via the JTAG/EOnce interface.

The JTAG/EOnce interface connector is illustrated in Figure 3 and described in Table 1. This table briefly describes each pin of the JTAG/EOnce interface. Figure 4 describes the JTAG connections in the 56F8013.
The command converter and the CodeWarrior USB TAP are not stand-alone tools. They need to be controlled via a PC to program the DSC. In a development environment they can both work with CodeWarrior IDE.

Table 1. JTAG Pins Description

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCK</td>
<td>Test Clock Input — This input pin provides the clock to synchronize the test logic. It shifts serial data to and from all TAP controllers and the TLM. If the EOncE module is not accessed using the master or 56800E core TAP controllers, the maximum TCK frequency is 1/4 the maximum frequency for the 56800E core. When accessing the EOncE module through the 56800E core TAP controller, the maximum frequency for TCK is 1/8 the maximum frequency for the 56800E core. The TCK pin has a pull down non-disabled resistor.</td>
</tr>
<tr>
<td>TDI</td>
<td>Test Data Input — This input pin provides a serial input data stream to the TAP and the TLM. It is sampled on the rising edge of the TCK. The TDI has an on-chip pullup resistor that can be disabled through PUPEN register in the GPIO module.</td>
</tr>
<tr>
<td>TMS</td>
<td>Test Mode Select Input — This input pin is used to sequence the TAP controller’s TLM state machine. It is sampled on the rising edge of the TCK. The TMS has an on-chip pull-up resistor that can be disabled through PUPEN register in the GPIO module.</td>
</tr>
<tr>
<td>TDO</td>
<td>Test Data Output — This three state output pin provides a serial output data stream from the Master TAP, or the 56800E core TAP controller. It operates in the Shift-IR and Shift-DR controller states of the TAP controller state machines. Output data changes on the falling edge of TCK.</td>
</tr>
</tbody>
</table>

Figure 4. JTAG Pins Connections to the 56F8013
Freescale also provides a PC application called *56800E Flash Programmer*. It is suitable for a low volume production environment as well as in-field firmware upgrades.

The 56800E Flash Programmer is provided in two forms. The first is command-line oriented. This allows easy integration with scripts. The second is a graphical user interface (GUI). This runs in windows to allow easy rapid flash loading onto the target device. The command line version is backwards compatible with the old flash over the JTAG tool.

The 56800E Flash Programmer can be downloaded at www.freescale.com.

Flash programming tool for 56F83xx/1xx and 56F801x DSCs. Requires installation of CodeWarrior 7.2 or higher.

4 Multiple Target Connection in Series

Multiple target devices may be connected in series. This allows a single command converter, JTAG/OnCE connector to control multiple devices, as in Figure 5. Data flows from the JTAG host to each JTAG implementation through TDI, out through TDO and back into TDI in the next chip, eventually returning to the JTAG host.

![Figure 5. Multiple Target Connection in Series](image)

In a series programming method, the devices are programmed sequentially; this means, one-by-one. Consequently, this solution reduces the connecting time. Multiple devices can be connected at the same time in the programmer.

This programming topology also requires the development of a JTAG controller capable of addressing multiple devices.

5 Multiple Target Connection in Parallel

A system is developed to make the gang programming simple. This system connects four MC56F8013 with the same JTAG connector according with the schematics in Figure 6. Connected in parallel are the pins RESET, TCK, TMS, VDD and GND.
The TDI is also connected parallel with the four DSCs, supplying them with the programming data/command stream.

The TDO is connected to a DSC to avoid a short-circuit with multiple devices. The DSC must always be connected to the programming circuits. It acts as the feedback supplier to the JTAG programmer.

This makes it possible to gang program four DSCs with the same JTAG interface. The programming time is around 4.5 s.

The main drawback of this implementation is that only one DSC has its programming success guaranteed: the DSC with the TDO signal connected to the JTAG interface. All the other DSCs go through a blind programming process, and the correct programming can not be verified via the JTAG host.

The proposed solution for this drawback is to include in the DSC a flash firmware integrity check at the beginning of the application. If the check is successful, the firmware signals it to the external world via an Led or any other user interface. This signal must be verified in the production line together with other tests.

![Multiple Target Connection in Parallel](image)

### 6 Flash Firmware Integrity Check

The firmware integrity check routine calculates a 32-bit checksum over the whole flash program area. The firmware verifies if it is equal with the predefined expected value. If the verification succeeds the code executes normally the applications code. If the check fails, the application execution is blocked.

The code lines below illustrate a simple use of the flash firmware integrity check.

```c
void main(void)
{
    if (VerifyFlashIntegrity())
        for(;;) // Processor Expert internal initialization. DON'T REMOVE THIS CODE!!!
    // End of Processor Expert internal initialization.
    for(;;) // Write your code here *
    Table[0]:
        for(;;) //
}
```

The verify flash integrity routine verifies the flash content after executing the start up code. If it succeeds, the application initialization code executes normally, as well as the application code. If it fails, the
application code is blocked and the program execution is trapped in an infinite loop. If the COP watchdog is enabled the DSC resets.

The verify flash integrity routine code is shown below. A pointer (ptr) appears and sweeps all flash addresses excluding the addresses 0x1FFE and 0x1FFF. These last two specific addresses store the expected checksum, and are not included in the checksum calculations.

```c
flashchecksec:
    "WRTENV(0x1Ffdd1oo): >.flashcheck"
```

The read program memory is an optimized routine for accessing data in the program flash. It has an input pointer for specifying the reading address, and as output it returns a word with the data.

```asm
asm uWord16 ReadProgramMemory(uWord16 Address)
{
    move.w #0, R0
    move.w P (R0)+, Y0
    ret
}
```

Finally, the flash signature routine reads the expected checksum already programmed in flash and returns it as a double word. Below is the routine code.

```asm
asm uWord32 FlashSignature(void)
{
    move. w #0x1FfF, Y0
    move. w Y0,R0
    move. v P (R0)+, Y0
    move. v Y0, A0
    move. v P (R0)+, A1
    ret
}
```

The last important point is how to program the expected checksum in specified addresses. This is done via a linker file edition.

For using processor EXPERT an extra memory section is created as shown in the Figure 7. The address is where the expected checksum is programmed.

After creating this memory section, the generate linker file option in EXPERT processor is turned off. This procedure prevents overwriting the changes that are done in the linker file.

The following code is added to the linker file. This code forces the programming of the expected checksum in the specified section address.

```c
SECTIONS {
    flashchecksec:
        "WRTENV(0x1Ffdd1oo): >.flashcheck"
}
```
7 Conclusion

This application note proposes a very simple and low cost solution for the MC56F8013 gang programming via JTAG interface and reduces the production line flash programming time.

This method requires a flash firmware integrity check, that verifies correct programming of all parts.

As an important improvement, the flash expected checksum is located in another flash address (other then 0x1FFE) and does not overlap with the flash configuration field. This is located by the DSC hardware in the address range from 0x1FF7 to 0x1FFF. For more information regarding the flash configuration field, please refer to the 56F8000 Peripheral Manual, in the flash memory section.