Freescale Semiconductor
Application Note

Document Number: AN4981
Rev 0, 08/2014

On-Chip Flash Programming of Freescale's MC56F827xx via JTAG Interface

by: Ethan Cheng

Contents

1 Overview

This application note describes the Erase, Program, and Verify functions for the on-chip flash memory of Freescale's MC56F827xx, using the JTAG interface. Sample code is also provided as a reference for understanding the concepts presented in this document, and to aid development of MC56F827xx programmer solutions.

1.1 Architecture description

For on-chip flash memory programming of the MC56F827xx via the Joint Test Access Group (JTAG) interface, one useful method is to control the FTFA flash module using the Enhanced On-Chip Emulation (EOnCE) module. The EOnCE module was designed by Freescale for use in DSP chips to debug application software used by the chip. Figure 1 shows a system-level view of the Test Access Port (TAP) and EOnCE module of the Freescale DSC family.

At any one point, only one of either the core TAP, the TAP linking module, or the chip TAP, will own the JTAG port. Both the core TAP and the chip TAP are compatible with IEEE Standard 1149.1.

In order to access the on-chip data bus through the 56800EX core, it is necessary to control the EOnCE by the core TAP with the JTAG signal. This document will illustrate how to control the FTFA module to program flash memory via the JTAG port to access the EOnCE. The methodology in this material can be implemented on microprocessors and programmable logic devices. Please refer to Reference[1] for further JTAG application considerations.

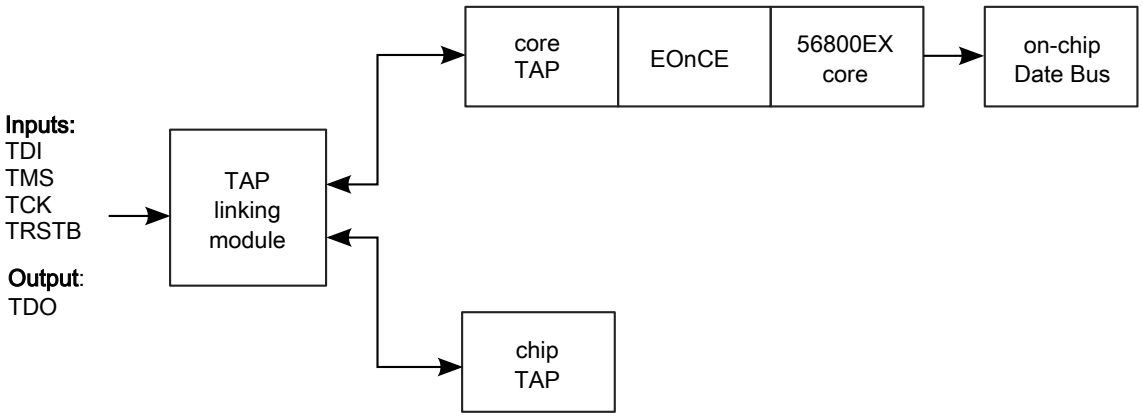

Figure 1. Serial communications via the Test Access Port of Freescale DSC family

1.2 Introduction to FTFA

The flash memory of MC56F827xx is an FTFA module which includes a memory controller providing various commands for flash operations. The basic programming functions of Erase, Blank Check, Program, and Verify will be illustrated step-by-step in the next sections. Further information about the FTFA module can be found in Reference [2].

The procedure for issuing an FTFA flash command includes several steps. The flow for a generic command write sequence is illustrated in Figure 2. The basic functions of the FTFA flash commands are listed in Table 2.

The first step to start an FTFA command is READ FSTAT register, which indicates the FTFA module status. The Command Complete Interrupt Flag (CCIF) in the FSTAT register indicates whether the command has completed or is in progress. If the previous command has completed (CCIF equals 1), go to the second step to check the error flags and confirm that the command was executed successfully. When the command is in progress (CCIF equals 0), keep updating the FSTAT register and checking the CCIF bit periodically until the command completes (CCIF equals 1).

The second step is to check the Flash Access Error (ACCERR) flag and the Flash Protection Violation (FPVIOL) flag in the FSTAT register. The ACCERR flag indicates an illegal access has occurred while the FPVIOL flag indicates an attempt to program or erase a protected flash area. If one of these two flags has been set, it means the previous command has aborted by error. All the error flags must be cleared by writing one to the ACCERR and FPVIOL bits, or by writing 0x03 to the FSTAT register, before the next command is executed.

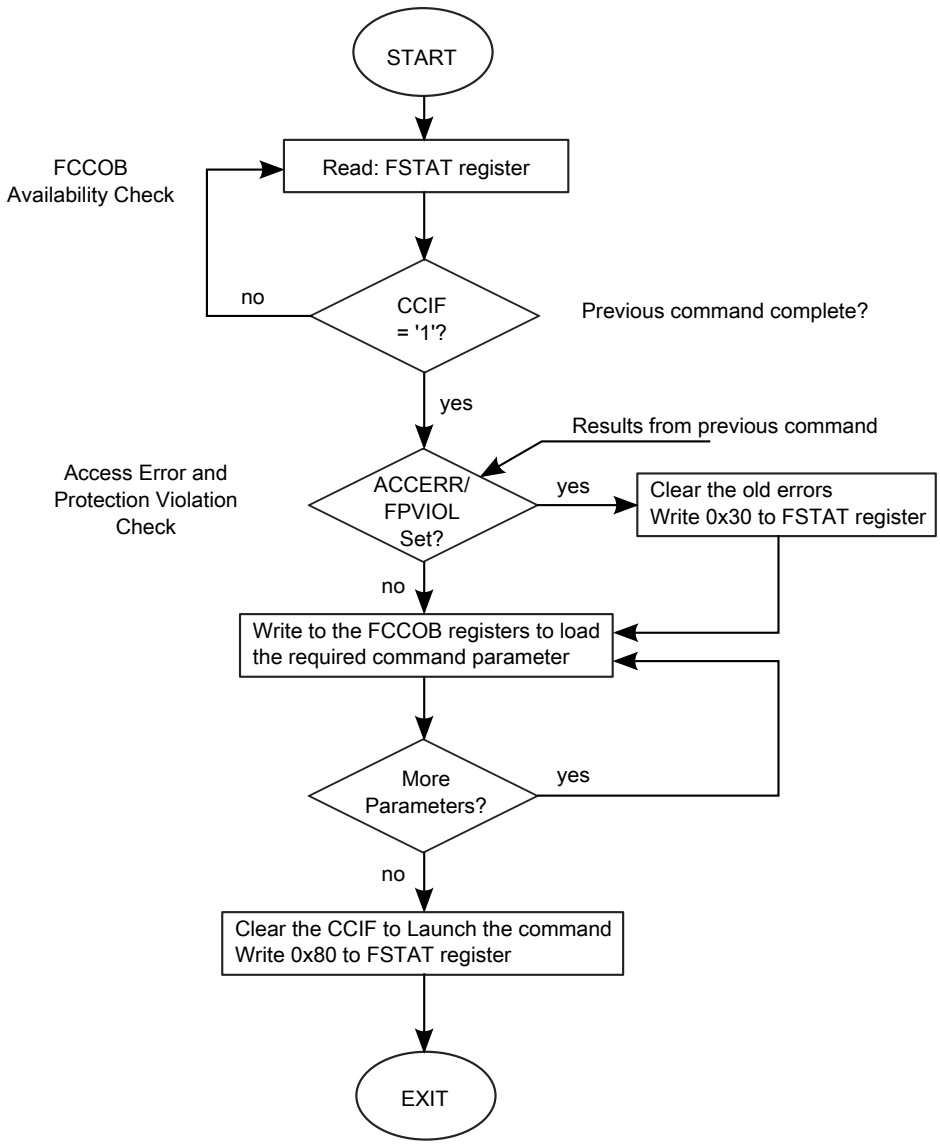

Figure 2. Generic flash memory module command write sequence flowchart

The third step is to write the Flash Common Command Object (FCCOB) register of the FTFA module. Table 1 lists the command code and relevant parameters in the FCCOB register for the Program Longword command. The Program Longword command (0x06) is written in the FCCOB0 register, the flash memory address is written in FCCOB1-3, and the program value is written in FCCOB4-7. Then, write '1' to the FSTAT[CCIF] flag to launch the Program Longword command. The flash memory module will then program the four-byte program value into the flash memory.

Table 1. Program Longword command FCCOB requirements

FCCOB number	FCCOB contents [7:0]
0	0x06 (Program 4 bytes Command)
1	Flash Address [23:16]
2	Flash Address [15:8]
3	Flash Address [7:0]
4	Byte 0 program Value

Table continues on the next page...

Table 1. Program Longword command FCCOB requirements (continued)

FCCOB number	FCCOB contents [7:0]
5	Byte 1 program Value
6	Byte 2 program Value
7	Byte 3 program Value

Program Longword Function demonstrates how to execute a Program Longword command in the FTFA module. The eonce_ftfa_execute_command function will send each FCCOB object to the FTFA module and launch the command by writing '1' to the FSTAT[CCIF] flag.

Example: 1.2.1 Program Longword Function

```
unsigned int FlashProgram_4bytes_LongWord(unsigned int stAddr,unsigned char *data)
{
    unsigned char chVar;
    unsigned char command[8];

    command[0] = FTFx_PROGRAM_LONGWORD; // FTFx_PROGRAM_LONGWORD = 0x06
    command[1] = (UINT8)( stAddr>>16);
    command[2] = (UINT8)( (stAddr>>8) & 0xff );
    command[3] = (UINT8)( stAddr & 0xff );
    for (chVar = 7; chVar>3; chVar--)
    {
        command[chVar] = data[7-chVar]; // data[0] means "byte 3 program value".
    }
    return( eonce_ftfa_execute_command(command, 7) );
}
```

Table 2. Basic function table of FTFA flash commands

Flash command	Command
Erase flash sector	0x09
Read ones all blocks	0x40
Program longword (4 bytes programming)	0x06
Program check	0x02

1.3 Introduction to JTAG

The JTAG port is a dedicated user-accessible TAP, compatible with the IEEE 1149.1a-1993 Standard Test Access Port and Boundary Scan Architecture. Problems associated with testing high-density circuit boards have led to the development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and JTAG. The 56800EX series of components supports circuit board test strategies based on this standard. As described in the IEEE 1149.1a-1993 specification, the JTAG port requires a minimum of four pins for support. They are these signals:

- TDI
- TDO
- TCK
- TMS

The MC56F827xx also uses the optional test reset (TRST) input signal and a DE pin for debug event monitoring. The descriptions of the JTAG pins are listed in Table 3.

Table 3. Description of JTAG pins

Pin name	Description
TCK	**Test Clock Input** — This input proves a gated clock to synchronize the test logic and shift serial data through the JTAG/EOnCE port. The maximum frequency for TCK is 1/8 the maximum frequency of the MC56F827xx (for example, 5 MHz if the IP bus clock is 40 MHz). The TCK pin has an on-chip pull-down resistor.
TMS	**Test Mode Select Input** — This input sequences the TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDI	**Test Data Input** — This input provides a serial data stream to the JTAG and the EOnCE module. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDO	**Test Data Output** — This tri-state output provides a serial data stream from the JTAG and the EOnCE module. It is driven in the Shift-IR and Shift-DR controller states of the JTAG state machine and changes on the falling edge of TCK.
TRST(optional)	**Test Reset** — This input provides a reset signal to the TAP controller. This pin has an on-chip pull-up resistor.

The TAP controller is a synchronous finite state machine that contains sixteen states, as illustrated in Figure 3. The manipulation of the TAP state machine depends on the TMS pin. For initialization of the JTAG state machine into the Test-Logic-Reset state, pull the TMS pin high and toggle the TCK pin more than five times with a positive edge clock signal. In the Test-Logic-Reset state, pull the TMS pin low and toggle the TCK pin to force the JTAG state machine to go into the Run-Test/Idle state.

Example: 1.3.1 Initialization of JTAG State Machine

```
JTAG_TMS_SET;
for (i=0;i<10;i++) {
  JTAG_TCK_RESET; /* TMS must be  sampled as '1' at least 5 times after power-up */
  JTAG_TCK_SET;
}
JTAG_TMS_RESET;
JTAG_TCK_RESET; /* Go to Run-Test-Idle */
JTAG_TCK_SET;
```

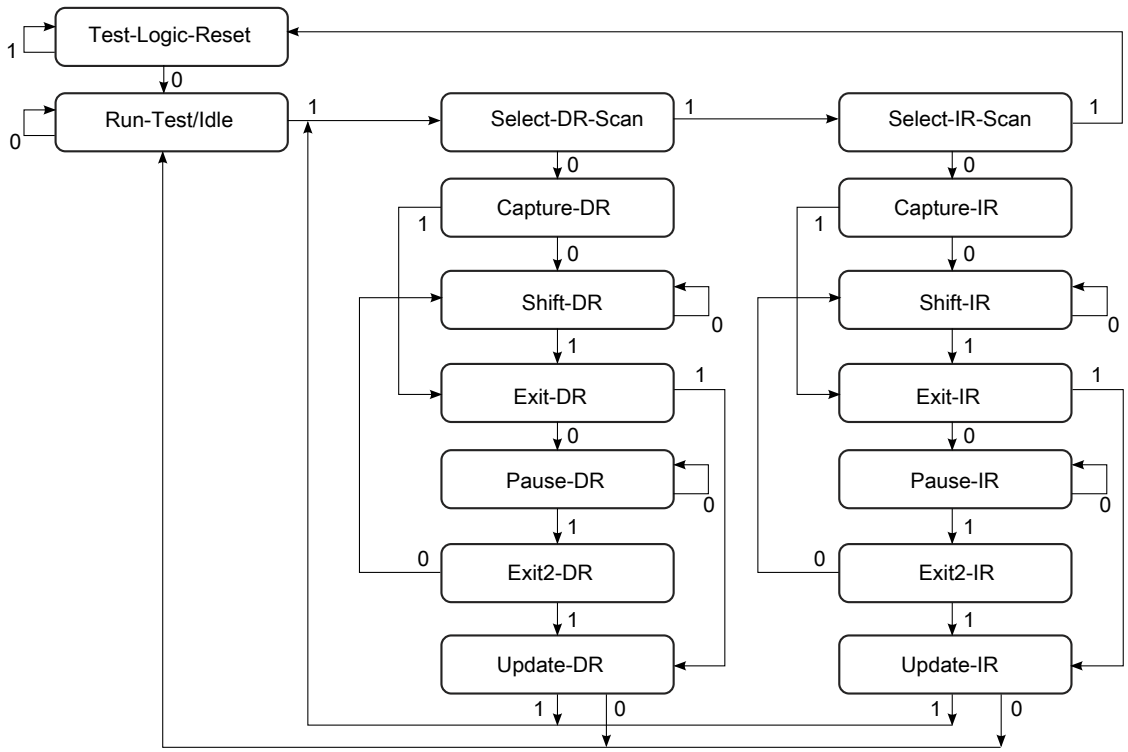

Figure 3. Description of the TAP state machine

To execute the JTAG instruction:

1. Bring the TAP state machine to the Shift-IR phase.
2. Shift in the new instruction.
3. Bring the TAP state machine to the Update-IR state to decode the new instruction.

Code example for execution of JTAG instruction shows sample code in which the state machine starts from Run-Test/Idle state, then goes to Select-DR-Scan、Select-IR-Scan, and shifts in the instruction with the LSB first sequence.

Example: 1.3.2 Code example for execution of JTAG instruction

```
JTAG_TMS_SET;   /* Go to Select-DR-Scan */
JTAG_TCK_RESET;
JTAG_TCK_SET;

JTAG_TMS_SET;   /* Go to Select-IR-Scan */
JTAG_TCK_RESET;
JTAG_TCK_SET;

JTAG_TMS_RESET; /* Go to Capture-IR */
JTAG_TCK_RESET;
JTAG_TCK_SET;

JTAG_TCK_RESET;
JTAG_TCK_SET;   /* Go to Shift-IR */ /* Now the Jtag is in the Shift-IR state */
for (i=0;i<instr_len;i++) {
JTAG_TDI_ASSIGN(instruction);instruction>>=1;
if (i==(instr_len-1))
    JTAG_TMS_SET;    /* Go to Exit1-IR */
JTAG_TCK_RESET;JTAG_TCK_SET;
status>>=1;  status|=JTAG_TDO_VALUE<<3;
}
```

```
JTAG_TCK_RESET; /* Go to Update-IR */
JTAG_TCK_SET;

JTAG_TMS_RESET;
JTAG_TCK_RESET; /* Go to RUN-TEST-IDLE */
JTAG_TCK_SET;
```

1.4 Introduction to EOnCE

The DSP56800E family provides board and chip-level testing capabilities through two on-chip modules that are both accessed through the JTAG/OnCE port. These two modules are Enhanced On-Chip Emulation module (Enhanced OnCE) and Test Access Port with 16-state controller (TAP, commonly called the JTAG port). The presence of the JTAG/OnCE interface allows the user to insert the DSP chip into a target system while retaining debug control. This capability is especially important for devices without an external bus, because it eliminates the need for a costly cable to bring out the footprint of the chip, as required by a traditional emulator system.

The DSP56800E Enhanced OnCE (EOnCE) module is a Freescale-designed module that is used to develop and debug application software used with the chip. The EOnCE port is a separate on-chip block that allows non-intrusive interaction with the DSP and is accessible through the pins of the JTAG interface. The EOnCE port makes it possible to examine registers, memory, or on-chip peripherals' contents in a special debug environment. In order to access the EOnCE port via the JTAG interface, it is necessary to switch the JTAG port to the CORE TAP. Switching into CORE TAP shows source code in which jtag_instruction_exec() is the JTAG IR command and jtag_data_shift() is the JTAG DR command to switch the JTAG port to CORE TAP.

Example: 1.4.1 Switching into CORE TAP

```
/* selects the CORE TAP Controller by modifying the TLM register */
/* to be used when Master TAP is selected */
void select_core_tap(void) {
  jtag_instruction_exec(0x05,8);  /* select TLM */
  jtag_data_shift(0x02,4);        /* select core TAP */
  jtag_measure_paths();           /* rescan jtag chain (Core TAP and Master TAP have
different instruction lengths) */
}
```

2 Debug mode

2.1 Communication with the EOnCE Module

The debug mode is a special processing state in which the core, when responding to a debug request:

1. Stalls the pipeline
2. Halts instruction execution
3. Holds the PC at its current position
4. Waits for user commands from the JTAG/EOnCE port

Send Debug Request to CORE and check mode shows how to request the processor to enter debug mode and check the processor status.

Example: 2.1.1 Send Debug Request to CORE and check mode

```
status=jtag_instruction_exec(0x7,4);        /*Debug Request*/
for(i=0;i<350;i++)
    WAIT_100_NS;
status=jtag_instruction_exec(0x7,4);        /*Debug Request #2 */
i=10;
do {
    status=jtag_instruction_exec(0x6,4);    /*Enable OnCE*/
if (!(i--))
    return(1);
} while (status!=0xd);
```

2.2 R/W Register via EOnCE

After the processor enters debug mode as Send Debug Request to CORE and check mode illustrated, instructions can be executed via the JTAG/EOnCE port. Data bus read through JTAG port illustrates the data read function in 16-bit mode from the JTAG port through the EOnCE. This example puts the source address into the r2 register and the OTx/ORx register address of EOnCE into the r0 register, then executes nop to update these two parameters into the 56800Ex core. Then the data of the source address stored in the r2 register will be read into the y0 register and written back to the EOnCE module via indirect addressing by r0. Finally, software can fetch the desired data from the OTx register of the EOnCE module.

Example: 2.2.1 Data bus read through JTAG port

```
unsigned int eonce_reg_read(unsigned int address) {
    eonce_move_long_to_r2(address);     /* move.l #value,r2 */
    eonce_move_long_to_r0(((unsigned long)(eonce_base)<<16)+0xffff);  /* load OTx/ORx reg
address */
    eonce_nop();                /* nop */
    eonce_move_at_r2_to_y0();       /* move.w x:(r2),Y0 */
    eonce_move_y0_at_r0();      /* move y0,x:(r0) */
    return eonce_rx_upper_data();    /* read data from upper OTx register of the target */
}
```

Data bus write through JTAG port illustrates the instructions used to write data to a specific address via the EOnCE. Software will load the target address into the r2 register and write data via indirect addressing by r2.

Example: 2.2.2 Data bus write through JTAG port

```
void eonce_reg_write(unsigned int address, unsigned int data) {
    eonce_move_long_to_r2(address);  /* move.l #value,r2 */
    eonce_move_value_at_r2(data);    /* move.w #<value>,x:(R2) */
    return ;
}
```

3 Flash memory programming via JTAG

3.1 Mass erase

The Erase All Blocks command erases all flash memory. It is a one-parameter FTFA command (0x44) as shown in Table 4. After clearing the FSTAT[CCIF] to launch the Erase All Blocks command, the flash memory module erases all program flash memory. Sample code of Erase All Blocks command illustrates some sample code to execute the Erase All Block operation command. In this example, eonce_ftfa_execute_command is a function which executes the FTFA commands using EOnCE API to operate the JTAG signal.

Table 4. Erase All Blocks command FCCOB requirements

FCCOB number	FCCOB content
0	0x44 (Erase All Command)

Example: 3.1.1 Sample code of Erase All Blocks command

```
UINT32 FlashEraseAllBlock(void){
    unsigned char commandArray[1];    /* command sequence array */
    unsigned int returnCode;          /* return code variable */

    commandArray[0] = FTFx_ERASE_ALL_BLOCK;  //#define FTFx_ERASE_ALL_BLOCK  0x44
    returnCode = eonce_ftfa_execute_command(commandArray, 0);

    return(returnCode);
}
```

3.2 Blank Check

The Blank Check function is a check procedure to ensure that the erase function has been successfully executed. This operation is a two-parameter FTFA command. Blank Check implementation of FTFA module sets up commandArray with two parameters and calls eonce_ftfa_execute_command() to execute the command.

Table 5. Blank Check command FCCOB requirements

FCCOB number	FCCOB content
0	0x40 (Erase All Command)
1	Read-1 Margin Choice

Example: 3.2.1 Blank Check implementation of FTFA module

```
UINT32 FlashVerifyAllBlock(unsigned char marginLevel){ //Margin Level: normal=0x0, user=0x01
    unsigned char commandArray[2];     /* command sequence array */
    unsigned int returnCode;           /* return code variable */

    commandArray[0] = FTFx_VERIFY_ALL_BLOCK;  //#define FTFx_VERIFY_ALL_BLOCK  0x40
    commandArray[1] = marginLevel;

    returnCode = eonce_ftfa_execute_command(commandArray, 1);
    return(returnCode); }
```

3.3 Programming

We use the Program Longword command to program four previously erased bytes in the program flash memory using an embedded algorithm in the FTFA module. Note that a flash memory location must be in the erased state before being programmed.

Table 6. Program Longword command FCCOB requirements

FCCOB number	FCCOB content
0	0x06 (Program Longword Command)
1	Flash address [23:16]
2	Flash address [15:8]
3	Flash address [7:0]
4	Byte 0 program value
5	Byte 0 program value
6	Byte 0 program value
7	Byte 0 program value

Example: 3.3.1 Write four bytes of data into flash memory

```
unsigned int FlashProgram_4bytes_LongWord(unsigned int stAddr,unsigned char *data)
{
    unsigned char chVar;
    unsigned char command[8];

    command[0] = FTFx_PROGRAM_LONGWORD; //#define FTFx_PROGRAM_LONGWORD  0x06
    command[1] = (UINT8)(  stAddr>>16);
    command[2] = (UINT8)( (stAddr>>8) & 0xff );
    command[3] = (UINT8)(  stAddr     & 0xff );
    for (chVar = 7; chVar>3; chVar--) {
      command[chVar] = data[7-chVar];
    }
    return( eonce_ftfa_execute_command(command, 7) );
}
```

3.4 Verify

The Program Check command verifies a previously programmed Program Flash longword to see if the flash memory has been read correctly at the specified margin level. The sample code of Flash Content Verification demonstrates how to translate a 32-bit address into separate parameters. If the command is executed and no error occurs, then eonce_ftfa_execute_command() will return 0.

Table 7. Program Check command requirements

FCCOB number	FCCOB content
0	0x02 (Program Check Command)
1	Flash address[23:16]
2	Flash address[15:8]
3	Flash address[7:0]
4	Margin Choice
8	Byte 3

Table continues on the next page...

Table 7. Program Check command requirements (continued)

FCCOB number	FCCOB content
9	Byte 2
0xa	Byte 1
0xb	Byte 0

Example: 3.4.1 Flash Content Verification

```
unsigned int FlashCheck_4bytes_LongWord(unsigned int stAddr,unsigned char *data)
{
    unsigned char command[0xc];
    unsigned int flashSTATE;

    command[0] = FTFx_PROGRAM_CHECK;
    command[1] = (UINT8)(  stAddr>>16);
    command[2] = (UINT8)( (stAddr>>8) & 0xff );
    command[3] = (UINT8)(  stAddr     & 0xff );
    command[4] = (UINT8) 2; //  factory margin

    command[8] = (UINT8) ( data[3]);  //write to stAddr
    command[9] = (UINT8) ( data[2]);  //write to stAddr+1
    command[0xa] = (UINT8) ( data[1]);//write to stAddr+2
    command[0xb] = (UINT8) ( data[0]);//write to stAddr+3
    flashSTATE = eonce_ftfa_execute_command(command, 11);

    return flashSTATE;
}
```

3.5 Read flash memory content

The program flash memory of the MC56F827xx has been remapped into data memory address from 0x4000 to 0xBFFF, as described in Reference [2]. Reading data at the memory address of 0x4000 is effectively equal to reading data at the flash address of 0x0000; reading the data memory address of 0x4001 is effectively equal to reading the flash address of 0x0001 and so on. Sample code of memory reading is illustrated in Read data by EOnCE API. The target address is placed into register r2, and sets the EOnCE RX register as the destination address. Then the processor executes a move instruction to fetch data from the JTAG controlled EOnCE module.

Example: 3.5.1 Read data by EOnCE API

```
unsigned short REG_READ(unsigned short reg_addr)
{
    unsigned short reg_val;

    eonce_move_long_to_r2(reg_addr);  /* load reg_addr to r2 */
    eonce_move_long_to_r0(((unsigned long)(eonce_base)<<16)+0xffff);
                        /* load OTx /ORx reg address */
    eonce_nop();

    eonce_move_at_r2_to_y0();
    eonce_move_y0_at_r0();
    reg_val=eonce_rx_upper_data();

    return reg_val;
}
```

The Freescale DSC microcontroller is designed for 16-bit data manipulation. The most efficiency way is to read flash content in a 16-bit length, as shown in Read flash memory data.

Example: 3.5.2 Read flash memory data

```
UINT16 Flash_Read_2Bytes(UINT16 addr)
{
    UINT16 content;
    content = REG_READ(addr+0x4000);

    return content;
}
```

4 Porting guide

The source code introduced in this document is written in ANSI C format, compiled by the LCC compiler, and executed on 32-bit Windows 7 with a parallel port interface that is able to simulate the JTAG interface. It is also possible to port this source code into any microprocessor. There are only two parts of the instructions in Source code architecture of Main function that must be modified for different platform environments: the “Environment Initialization” and the “JTAG Operation.” In the Windows 7 operating system, Environment Initialization will register the zlportio library to Windows and assign initial pin status.

Example: 4.1 Source code architecture of Main function

```
int main (int argc,char *argv[]) {
//----------Variable Declaration
//----------Environment Initialization
//----------JTAG Operation
  return(SUCCESS);
}
```

When porting this source code to other microprocessors, the Environment Initialization will be replaced by the IO pin configuration. JTAG pin definition in hw_access.h shows the JTAG pin definition in the header file. When porting to different platforms, the macro listed in Example 15 must be modified to fit specific pin control functionality.

Example: 4.2 JTAG pin definition in hw_access.h

```
//*****************************************************
  #if env_type == xxxx_Programmer
//*****************************************************
    #define WAIT_100_NS      XXXX         //100 nanosecond delay
    #define JTAG_TCK_SET     XXXX_SET     //Pull high TCK pin
    #define JTAG_TCK_RESET   XXXX_RESET   // Pull down TCK pin

    #define JTAG_TMS_SET     XXXX_SET     //Pull high TMS pin
    #define JTAG_TMS_RESET   XXXX_RESET   //Pull down TMS pin

    #define JTAG_TDI_SET     XXXX_SET     //Pull high TDI pin
    #define JTAG_TDI_RESET   XXXX_RESET   //Pull down TDI pin

    #define JTAG_TDO_VALUE   XXX_TDO_VALUE   //Read TDO pin status
    #define JTAG_TDO_VALUE_SHIFTED_15     XXXX_TDO_VALUE_SHIFTED //Shift TDO value
```

5 Conclusion

This material introduces the JTAG capabilities and Enhanced On-Chip Emulation (EOnCE) which enable programmers to examine registers and chip peripherals. With these features, we are able to read and write the on-chip flash controller registers to program the flash memory. This application note also provides source code examples of on-chip flash programming via JTAG. The sample code can be ported to platforms based on any processor.

6 References

1. "JTAG 101: IEE 1149.x and Software Debug," January 2009. http://www.intel.com/content/www/us/en/intelligent-systems/jtag-101-ieee-1149x-paper.html. Accessed 08/06/2014.
2. Freescale document MC56F827XXRM, *MC56F827xx Reference Manual* Rev. 3, 10/2013.
3. Freescale document AN1935, "Programming On-Chip Flash Memories of 56F80x Devices Using the JTAG/OnCE Interface", Rev. 1, 11/2005.
4. Freescale document AN4507, "Using the Kinetis Security and Flash Protection Features," Rev. 1, 6/2012.
5. "Parallel port": Wikipedia, http://en.wikipedia.org/wiki/Parallel_port. Accessed 08/06/2014.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4981
Revision 0, 08/2014