C Code for Interfacing the FPSLIC AVR Core to AT17 Series Configuration Memories

Features:

- **C Routines for 2-wire Serial Interface with AT17 Series Device**
- **Example Circuit for AVR® Programming of Configuration Memories**
- **Supports AT17 and AT24 Families of EEPROMs**

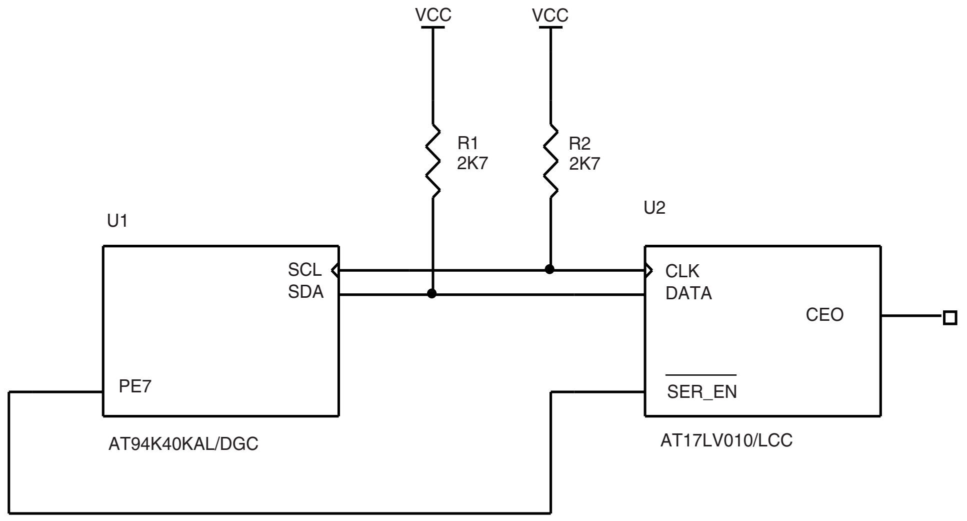
Introduction:

This application note describes how to in-system program (ISP) an Atmel configuration memory using the embedded AVR core of the FPSLIC device. The AT17 Family of configuration memories, ranging from 64 Kb to 1 Mb, uses the 2-wire serial protocol for in-system programming.

This application note assumes that the user is familiar with the AT94K and AT17 series datasheets and the application note entitled "*Programming Specification for Atmel's AT17 and AT17A Series FPGA Configuration EEPROMs*".

This application note is written specifically for the 1 Mb device. C routines to read and write data are included. The code can easily be recompiled for all AT94K series devices and other AT17 Series EEPROMs.

Figure 1. Circuit Diagram



FPSLIC™

AT94K Series Field Programmable System Level Integrated Circuit and AT17 Series Configuration EEPROM Memory

Application Note

Rev. 2289A–04/01

Theory of Operation:

The 2-wire serial interface is a synchronous bus consisting of one data (SDA) and one clock (SCL) line. By using open drain/collector outputs, the 2-wire serial bus supports any fabrication process (CMOS, bipolar and more). The 2-wire serial bus is a multi-master bus where one or more devices, capable of taking control of the bus can be connected. Only master devices can drive both the SCL and SDA lines, but a slave device is only allowed to issue data on the SDA line.

Software Description:

These semi-generic 2-wire serial macros listed below were compiled with the IAR Systems Embedded Workbench v1.50B and ImageCraft ICCAVR C compiler v6.19B. The macros assist in implementing the software for the 2-wire serial interface, utilizing the built in 2-wire serial interface on the AT94K device. The AT94K used to perform the master function is driven by an external 4 MHz crystal.

As previously stated, this code utilizes the built-in 2-wire serial interface along with PORTE (Pin 7) to control the serial enable input pin on the AT17 series device. Table 1 lists the semi-generic 2-wire serial interface routines and the amount of code space used by each routine. The very simple routines could be incorporated into the more complex routines for further code size optimization.

Table 1 lists the number of clock cycles and program space used while implementing the macros. Compiler options were set to generate minimum code (if possible).

Table 1. Size and Execution Time for 2-wire Serial Routines

Macro	Cycles	Bytes
CLEAR_TWINT	2	4
GET_BYTE	1	2
REP_START_CONDITION	2	4
SEND_ACK	2	4
SEND_BYTE	1	2
START_CONDITION	2	4
STOP_CONDITION	2	4
TWS_MASTER_INIT	6	12
TWS_SLAVE_INIT	11	22
WAIT	3+	6

The AT17LV010 device is programmed/verified on a 128-byte page boundary. During normal FPSLIC configuration operations the read of the device starts at address 0x000000 and continues until the FPSLIC™ has completed its configuration. The routine writePage and readPage write and read 128-byte pages from the configuration memory and use the semi-generic 2-wire serial interface routines to perform this function. writePage and readPage are both called with the page address to write/read to, and a pointer to a 128-byte page buffer. At the end of a page write a 20 ms timeout method is used to determine the end of the internal page programming cycle, alternatively a polling method may be implemented - consult the "*Configuration Memory Programming Specification"* for more information. programResetPolarity and verifyResetPolarity write and read data from memory locations 0x020000-0x020003 in effect setting and verifying the reset polarity.

The routine initTWS is called with the 2-wire serial bit rate generator division factor, initializes the 2-wire serial interface and configures PE7 (PORTE, Pin 7). The main program is used to call writePage, readPage, programResetPolarity, and verifyResetPolarity and serves to illustrate proper calling conventions for those routines.

Modifications and Optimizations:

If the user decides to change the oscillator frequencies the programResetPolarity, writePage, and initTWS routines would need to be modified. In the programResetPolarity and writePage routines Timer/Counter1 is used to generate a timeout after 20 ms, so the programming operation should have completed by then. Also a new division factor for the serial bit rate generator needs to be calculated and loaded into the TWBR register from within the initTWS routine.

The configuration memory used with the AT94KAL device may also be used as external data storage. The space available depends upon the size of the configuration memory and the size of the current design. Table 2 lists the various AT94KAL devices along with the suggested configuration memory and assumes the maximum design size.

Table 2. Unused Configurator Space on AT94KAL Designs

AT94K Device	Configuration Bits	Configurator	Unused Space	Pages Available
AT94K05AL	228K	AT17LV256	28K	56
AT94K10AL	423K	AT17LV512	89K	89
AT94K40AL	809K	AT17LV010	215K	215

References:

"The I²C Bus and How to use it", April 1995 Update - Philips Semiconductors

"*Programming Specification for Atmel's AT17 and AT17A Series FPGA Configuration EEPROMs*" - Atmel Corporation

"*AT17 Series Configuration Memory*" datasheet - Atmel Corporation

"*AT24 Series 2-wire Serial EEPROM*" datasheet - Atmel Corporation

```
/***************************************************************************
 *  File Name:          TWS.H
 *  Title:              AT94K Two-wire Serial Header
 *  Last Updated:       January 1, 2000
 *  Target Device:      AT94K05/10/40
 *
 *  Support Email:      fpslic@atmel.com
 *  Support Hotline:    +1 (408) 436-4119 (USA)
 ***************************************************************************/

#ifndef __TWS_H
#define __TWS_H

#include <ioat94k.h>

/* General Master Status Codes */
#define START                   0x08
#define REP_START               0x10

/* Master Transmitter Status Codes */
#define MT_SLA_ACK              0x18
#define MT_SLA_NACK             0x20
#define MT_DATA_ACK             0x28
#define MT_DATA_NACK            0x30
#define MT_ARB_LOST             0x38

/* Master Receiver Status Codes */
#define MR_ARB_LOST             0x38
#define MR_SLA_ACK              0x40
#define MR_SLA_NACK             0x48
#define MR_DATA_ACK             0x50
#define MR_DATA_NACK            0x58

/* Slave Receiver Status Codes */
#define SR_SLA_ACK              0x60
#define SR_ARB_LOST_SLA_ACK     0x68
#define SR_GCALL_ACK            0x70
#define SR_ARB_LOST_GCALL_ACK   0x78
#define SR_DATA_ACK             0x80
#define SR_DATA_NACK            0x88
#define SR_GCALL_DATA_ACK       0x90
#define SR_GCALL_DATA_NACK      0x98
#define SR_STOP                 0xA0

/* Slave Transmitter Status Codes */
#define ST_SLA_ACK              0xA8
#define ST_ARB_LOST_SLA_ACK     0xB0
#define ST_DATA_ACK             0xB8
#define ST_DATA_NACK            0xC0
```

```
#define ST_LAST_DATA            0xC8

/* Miscellaneous States */
#define NO_INFO                 0xF8
#define BUS_ERROR               0x00


/* TWS Macro Definitions */
#define CLEAR_TWINT ()               (    TCR      =  ( (1 << TWINT) | (1 << TWEN)                ) )
#define GET_BYTE(DATA)               (    DATA ) =  TWDR                                          )
#define REP_START_CONDITION()        (    TWCR   =  ( (1 << TWINT) | (1 << TWSTA) | (1 << TWEN) ) )
#define SEND_ACK()                   (    TWCR   =  (1 << TWINT)   | (1 << TWEA ) | (1 << TWEN) ) )
#define SEND_BYTE(DATA)                   TWDR   =  ( DATA                                      ) )
#define START_CONDITION()                 TWCR   =  ( (1 << TWSTA) | (1 << TWEN )               ) )
#define STOP_CONDITION()                  TWCR   =  ( (1 << TWINT) | (1 << TWSTO) | (1 << TWEN) ) )

#define TWS_MASTER_INIT()                 TWCR  &= ~( (1 << TWSTA)   |  (1 << TWSTO)              ) ); \
                                          TWCR   =  ( (1 << TWEN)                               ) )

#define TWS_SLAVE_INIT(ADDRESS)           TWAR  &= ~( 0xFE                                      ) ); \
                                     (    TWAR  |=  ( ADDRESS                                   ) ); \
                                          TWCR  &= ~( (1 << TWSTA) | (1 << TWSTO)               ) ); \
                                          TWCR   =  ( (1 << TWEA)  | (1 << TWEN)                ) )

#define WAIT()                  while( !( TWCR & (1 << TWINT) ) )


#endif /* __TWS_H */
```

```
/****************************************************************************
 *  File Name:          AT17.H
 *  Title:              AT17 Two-wire Serial Header File
 *  Last Updated:       January 1, 2000
 *  Target Device:      AT94K05/10/40
 *
 *  Support Email:      fpslic@atmel.com
 *  Support Hotline:    +1 (408) 436-4119 (USA)
 ***************************************************************************/

#ifndef __AT17_H
#define __AT17_H

#include "tws.h"
#include <ioat94k.h>


/* Device Address Constants */
#define AT17LV010                       0xA6            /* Device Address Byte */
#define WRITE                           0
#define READ                            1

/* AT17LV010 Device Constants */
#define A_BYTE                          3               /* Bytes per Address */
#define R_BYTE                          4               /* Reset Polarity Locations */
#define T_BYTE                          128             /* Bytes per Page */
#define T_PAGE                          1024            /* Pages per Device */
#define TT_BYTE                         131072          /* Bytes per Device */

#define LOW                             0x00            /* Reset Polarity */
#define HIGH                            0xFF            /* Reset Polarity */


/* AT17 TWS Function Prototypes */
void programResetPolarity(unsigned char newResetPolarityLevel, unsigned long resetAddress);
void writePage(unsigned char * data, unsigned long pageAddress);
void readPage(unsigned char * data, unsigned long pageAddress);
unsigned char verifyResetPolarity(unsigned long resetAddress);


#endif /* __AT17_H */
```

```
/****************************************************************************
 *  File Name:          AT17.C
 *  Title:              AT17 Two-wire Serial Implementation File
 *  Last Updated:       January 1, 2000
 *  Target Device:      AT94K05/10/40
 *
 *  Support Email:      fpslic@atmel.com
 *  Support Hotline:    +1 (408) 436-4119 (USA)
 ***************************************************************************/

#include "at17.h"

/* programResetPolarity
 *
 * Locations 0x20000 through 0x20003 are used to store the Reset/OuputEnable
 * polarity.
 *
 * 0x00 => Active HIGH Reset and Active LOW Output Enable
 * 0xFF => Active LOW Reset and Active HIGH Output Enable
 *
 * So, the memory location values determine the Reset Polarity.  After
 * programming the 20 ms timeout method is used to determine the end of
 * the internal programming cycle.
 */

void programResetPolarity(unsigned char newResetPolarityLevel, unsigned long resetAddress)
{
        unsigned char byteCount;
        unsigned char addressCount;

        /* Send START Condition */
        START_CONDITION();
        WAIT();

        if (TWSR == START)
        {
                /* Send Device Address */
                SEND_BYTE(AT17LV010 & ~(WRITE));
                CLEAR_TWINT();
                WAIT();

                /* Send Memory Address */
                for(addressCount = 0; addressCount < A_BYTE; addressCount++)
                {
                        if( (TWSR == MT_SLA_ACK) || (TWSR == MT_DATA_ACK) )
                        {
                              SEND_BYTE( (resetAddress >> ( ((A_BYTE - 1) - addressCount) * 8 )) & 0x0000FF);
                               CLEAR_TWINT();
                               WAIT();
                        }
                }

                /* Program Reset Polarity */
                for(byteCount = 0; byteCount < R_BYTE; byteCount++)
                {
                        if(TWSR == MT_DATA_ACK)
                        {
                                if(newResetPolarityLevel == LOW)
                                        SEND_BYTE(0xFF);
                                else SEND_BYTE(0x00);

                                CLEAR_TWINT();
                                WAIT();
                        }
                }

                /* Send STOP Condition */
                if(TWSR == MT_DATA_ACK)
                {
                        STOP_CONDITION();

                        /* Configurator Timeout -> 20 ms
                         * 4 MHz / 1024 = 3906.25 Hz
                         * 3906.25 Hz = 256 us
                         * 20 ms / 256 us = 78.125
                         * 65536 - 78.125 = 65457.875 = 0xFFB1
                         * Interrupt on 0xFFFF to 0x0000 Transition */

                        TCNT1H = 0xFF;
                        TCNT1L = 0xB1;
                        TCCR1B = 0x05;
                        while(!(TIFR & (1 << TOV1)));
                }
        }
}




/* readPage
 * * Read 128 bytes at address into bufptr.  Starts reading at address 0
 * within the page.  Please refer to the Application Note entitled,
 * "Programming Specification for Atmel's AT17 and AT17A Series FPGA
 * Configuration EEPROMs" found at http://www.atmel.com for detailed
 * device address decoding and page address formatting. */

void readPage(unsigned char * bufptr, unsigned long pageAddress)
{
        unsigned char byteCount;
        unsigned char addressCount;

        /* Send START Condition */
        START_CONDITION();
        WAIT();

        if (TWSR == START)
        {
                /* Send Device Address */
                SEND_BYTE(AT17LV010 & ~(WRITE));
                CLEAR_TWINT();
                WAIT();

                /* Send Memory Address */
                for(addressCount = 0; addressCount < A_BYTE; addressCount++)
                {
                        if( (TWSR == MT_SLA_ACK) || (TWSR == MT_DATA_ACK) )
                        {
                               SEND_BYTE( (pageAddress >> ( ((A_BYTE - 1) - addressCount) * 8 )) & 0x0000FF);
                                CLEAR_TWINT();
                                WAIT();
                        }
                }

                if(TWSR == MT_DATA_ACK)
                {
                        /* Send START Condition */
                        REP_START_CONDITION();
                        WAIT();

                        if (TWSR == REP_START)
                        {
                                /* Send Device Address */
                                SEND_BYTE(AT17LV010 | READ);
                                CLEAR_TWINT();
                                WAIT();

                                if(TWSR == MR_SLA_ACK)
                                {
                                        SEND_ACK();
                                        WAIT();

                                        /* Receive Data */
                                        for(byteCount = 0; byteCount < (T_BYTE - 2); byteCount++)
                                        {
                                                if(TWSR == MR_DATA_ACK)
                                                {
                                                        GET_BYTE(bufptr[byteCount]);
                                                        SEND_ACK();
                                                        WAIT();
                                                }
                                        }

                                        /* Receive Last 2 Bytes and Send STOP Condition */
                                        if(TWSR == MR_DATA_ACK)
                                        {
                                                GET_BYTE(bufptr[T_BYTE - 2]);
                                                CLEAR_TWINT();
                                                WAIT();

                                                if(TWSR == MR_DATA_NACK)
                                                {
                                                        GET_BYTE(bufptr[T_BYTE - 1]);

                                                        STOP_CONDITION();
                                                }
                                        }
                                }
                        }
                }
        }
}



/* writePage
 *
 * Writes 128 bytes at address from bufptr.  Starts writing at address 0
 * within the page.  Please refer to the Application Note entitled,
 * "Programming Specification for Atmel's AT17 and AT17A Series FPGA
 * Configuration EEPROMs" found at http://www.atmel.com for detailed
 * device address decoding and page address formatting.
 *
 * After programming the 20 ms timeout method is used to determine
 * the end of the internal programming cycle.
 */

void writePage(unsigned char * bufptr, unsigned long pageAddress)
{
        unsigned char byteCount;
        unsigned char addressCount;

        /* Send START Condition */
        START_CONDITION();
        WAIT();

        if (TWSR == START)
        {
                /* Send Device Address */
                SEND_BYTE(AT17LV010 & ~(WRITE));
                CLEAR_TWINT();
                WAIT();

                /* Send Memory Address */
                for(addressCount = 0; addressCount < A_BYTE; addressCount++)
                {
                        if( (TWSR == MT_SLA_ACK) || (TWSR == MT_DATA_ACK) )
                        {
                               SEND_BYTE( (pageAddress >> ( ((A_BYTE - 1) - addressCount) * 8 )) & 0x0000FF);
                                CLEAR_TWINT();
                                WAIT();
                        }
                }

                /* Send Data */
                for(byteCount = 0; byteCount < T_BYTE; byteCount++)
                {
                        if(TWSR == MT_DATA_ACK)
                        {
                                SEND_BYTE(bufptr[byteCount]);
                                CLEAR_TWINT();
                                WAIT();
                        }
                }

                /* Send STOP Condition */
                if(TWSR == MT_DATA_ACK)
                {
                        STOP_CONDITION();


                        /* Configurator Timeout -> 20 ms
                         * 4 MHz / 1024 = 3906.25 Hz
                         * 3906.25 Hz = 256 us
                         * 20 ms / 256 us = 78.125
                         * 65536 - 78.125 = 65457.875 = 0xFFB1
                         * Interrupt on 0xFFFF to 0x0000 Transition */

                        TCNT1H = 0xFF;
                        TCNT1L = 0xB1;
                        TCCR1B = 0x05;
                        while(!(TIFR & (1 << TOV1)));
                }
        }
}


/* verifyResetPolarity
 *
 * 4 bytes are read from locations 0x20000 through 0x20003.  The
 * bytes are verified to be of all the same value.  If the are then
 * the value is returned.
 *
 * Returned Value 0x00 => Active HIGH Reset and Active LOW Output Enable
 * Returned Value 0xFF => Active LOW Reset and Active HIGH Output Enable
 *
 * If they are not the same then 0xAA is returned to signal an
 * ERROR condition.
 */

unsigned char verifyResetPolarity(unsigned long resetAddress)
{
        unsigned char byteCount;
        unsigned char addressCount;
        unsigned char resetPolarity[R_BYTE];
        unsigned char polarity = 0xAA;

        /* Send START Condition */
        START_CONDITION();
        WAIT();

        if (TWSR == START)
        {
                /* Send Device Address */
                SEND_BYTE(AT17LV010 & ~(WRITE));
                CLEAR_TWINT();
                WAIT();

                /* Send Memory Address */
                for(addressCount = 0; addressCount < A_BYTE; addressCount++)
                {
                        if( (TWSR == MT_SLA_ACK) || (TWSR == MT_DATA_ACK) )
                        {
                               SEND_BYTE( (resetAddress >> ( ((A_BYTE - 1) - addressCount) * 8 )) & 0x0000FF);
                                CLEAR_TWINT();
                                WAIT();
                        }
                }

                if(TWSR == MT_DATA_ACK)
                {
                        /* Send Repeat START Condition */
                        REP_START_CONDITION();
                        WAIT();

                        if (TWSR == REP_START)
                        {
                                /* Send Device Address */
                                SEND_BYTE(AT17LV010 | READ);
                                CLEAR_TWINT();
                                WAIT();

                                if(TWSR == MR_SLA_ACK)
                                {
                                        SEND_ACK();
                                        WAIT();

                                        /* Receive Data */
                                        for(byteCount = 0; byteCount < (R_BYTE - 2); byteCount++)
                                        {
                                                if(TWSR == MR_DATA_ACK)
                                                {
                                                        GET_BYTE(resetPolarity[byteCount]);
                                                        SEND_ACK();
                                                        WAIT();
                                                }
                                        }

                                        /* Receive Last 2 Bytes and Send STOP Condition */
                                        if(TWSR == MR_DATA_ACK)
                                        {
                                                GET_BYTE(resetPolarity[R_BYTE - 2]);
                                                CLEAR_TWINT();
                                                WAIT();

                                                if(TWSR == MR_DATA_NACK)
                                                {
                                                        GET_BYTE(resetPolarity[R_BYTE - 1]);
                                                        STOP_CONDITION();
                                                }
                                        }
                                }
                        }
                }
        }

        /* Compare Received Reset Polarity */
        for(byteCount = 0; byteCount < (R_BYTE - 1); byteCount++)
        {
                if(resetPolarity[byteCount] == resetPolarity[byteCount + 1])
                        polarity = (resetPolarity[byteCount]);
                else polarity = 0xAA;
        }

        return (polarity);
}
```

```
/****************************************************************************
 *  File Name:          TWS-AT17.C
 *  Title:              AT17 Two-wire Serial Driver File
 *  Last Updated:       January 1, 2000
 *  Target Device:      AT94K05/10/40
 *
 *  Support Email:      fpslic@atmel.com
 *  Support Hotline:    +1 (408) 436-4119 (USA)
 ****************************************************************************/

#include "at17.h"
#include <ioat94k.h>


/* 2-wire Serial Interface Bit Rate Table for 4 MHz
 *
 *       +-------+-------------+---------------------+
 *       |       |    TWBR     |      Bit Rate       |
 *       |       +------+------+----------+----------+
 *       | Clock | Hex  | Dec  |    Hz    |    kHz   |
 *       +-------+------+------+----------+----------+
 *       | 4 MHz | 0x00 |    0 | 250000   | 250      |
 *       |       | 0x02 |    2 | 200000   | 200      |
 *       |       | 0x08 |    8 | 125000   | 125      |
 *       |       | 0x0C |   12 | 100000   | 100      |
 *       |       | 0x11 |   17 |  80000   |  80      |
 *       |       | 0x18 |   24 |  62500   |  62.5    |
 *       |       | 0x20 |   32 |  50000   |  50      |
 *       |       | 0x2A |   42 |  40000   |  40      |
 *       |       | 0x38 |   56 |  31250   |  31.25   |
 *       |       | 0x48 |   72 |  25000   |  25      |
 *       |       | 0x5C |   92 |  20000   |  20      |
 *       |       | 0x78 |  120 |  15625   |  15.625  |
 *       |       | 0x98 |  152 |  12500   |  12.5    |
 *       |       | 0xC0 |  192 |  10000   |  10      |
 *       |       | 0xF8 |  248 |   7812.5 |   7.8125 |
 *       +-------+------+------+----------+----------+
 */

/* initTWS */
void initTWS(unsigned char bitRate)
{
        TWBR  = bitRate;                        /* Setting TWC Bit Rate */
        TWS_MASTER_INIT();                      /* Included in TWS.H */

        PORTE |= (1 << PE7);                    /* PE7 High, SER_EN Disabled */
        DDRE  |= (1 << PE7);                    /* PE7 Output */
}
```

```
/* initAT94K */
void initAT94K(void)
{
        PORTD = 0x00;                          /* PORTD0..7 Low */
        DDRD = 0xFF;                           /* PORTD0..7 Output */
}

/* Main Program */
void main(void)
{
        unsigned long pageAddress = 0x0001FF80;
        unsigned long resetAddress = 0x0020000;
        unsigned char byteCount = 0;
        unsigned char rTemp = 0;
        unsigned char readBuffer[T_BYTE];
        unsigned char writeBuffer[T_BYTE];

        /* Initialize AT94K & TWS @ 200 kHz */
        initAT94K();
        initTWS(2);

        /* Initialize Read & Write Buffers */
        for(byteCount = 0; byteCount < T_BYTE; byteCount++)
        {
                writeBuffer[byteCount] = byteCount;
                readBuffer[byteCount] = 0;
        }

        /* SER_EN = LOW => Programming Mode Enabled */
        PORTE &= ~(1 << PE7);

        writePage(&writeBuffer[0], pageAddress);
        readPage(&readBuffer[0], pageAddress);

        /* Verify Programmed Page */
        for (rTemp = 0; rTemp < T_BYTE; rTemp++)
        {
                if (writeBuffer[rTemp] != readBuffer[rTemp])
                {
                        PORTD = 0x55;
                }
        }

        /* Clear Verify Buffer */
        for (rTemp = 0; rTemp < T_BYTE; rTemp++)
        {
                readBuffer[rTemp] = 0;
        }

        programResetPolarity(LOW, resetAddress);
        if (verifyResetPolarity(resetAddress))
        {
                PORTD = 0xAA;
        }

        programResetPolarity(HIGH, resetAddress);
        if (!(verifyResetPolarity(resetAddress)))
        {
                PORTD = 0x55;
        }

        /* SER_EN = HIGH => Programming Mode Disabled */
        PORTE |= 0x80;
}
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Atmel FPSLIC Hotline
1-(408) 436-4119

Atmel FPSLIC e-mail
fpslic@atmel.com

FAQ
Available on web site

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309