FlashFlex51 Microcontroller
Single Master, Multi-Slave Serial Peripheral Interface

Application Note
June 2003

1.0 INTRODUCTION

The SPI protocol is a widely accepted and easily used serial transfer protocol. It is fast and efficient, allowing for simultaneous bi-directional data transfer. The protocol involves a master-slave configuration which includes a master device and one or more slave devices. However, the protocol does allow for multiple slave devices to simultaneously communicate with a single master device. This application note provides information on both the hardware and software aspects of a single master, multi-slave SPI setup under two operating conditions: (1) when both master and slave are operating from the same source voltage and (2) when master and slave are operating from different source voltages.

This application note shows the hardware and software for an SST89E/V564RD microcontroller utilizing its 8-bit hardware SPI to communicate with several SST25VFxxx Serial Flash memory devices. The software routines, written in C, contain extensive comments to describe the function of each routine. Port 1 of the MCU is used to interface 25VFxxx devices, and Port 2 of the MCU can contain LEDs for debugging.

Companion product data sheets for the SST89E/V564RD MCU and 25VFxxx Serial Flash should be reviewed in conjunction with this application note for a complete understanding of the hardware and software examples provided here.

2.0 HARDWARE

2.1 Master-Slave Running from the Same Voltage Source

Figure 2-1 shows the hardware setup of master and slaves operating at a single supply voltage. P1.2 - P1.4 control the Chip-Enable inputs for the three 25 series memory chips. P1.5 - P1.7 provide the SI, SO, and SCK interfaces respectively to each chip. This means that all three memory chips are clocked off the same clock and communicate on the same serial bus. Also note that the SO outputs on the memory are in a Hi-Z state when CE is HIGH. For this reason, no buffers are required between the memory and the MCU. For the same reason, only one memory chip may be enabled at one time. Enabling more than 1 memory chip at one time risks data corruption and possible hardware damage. Although there are extra port pins available on the MCU to implement the HOLD and Write-Protect features, such implementation is up to customer preference and beyond the scope of this application note.

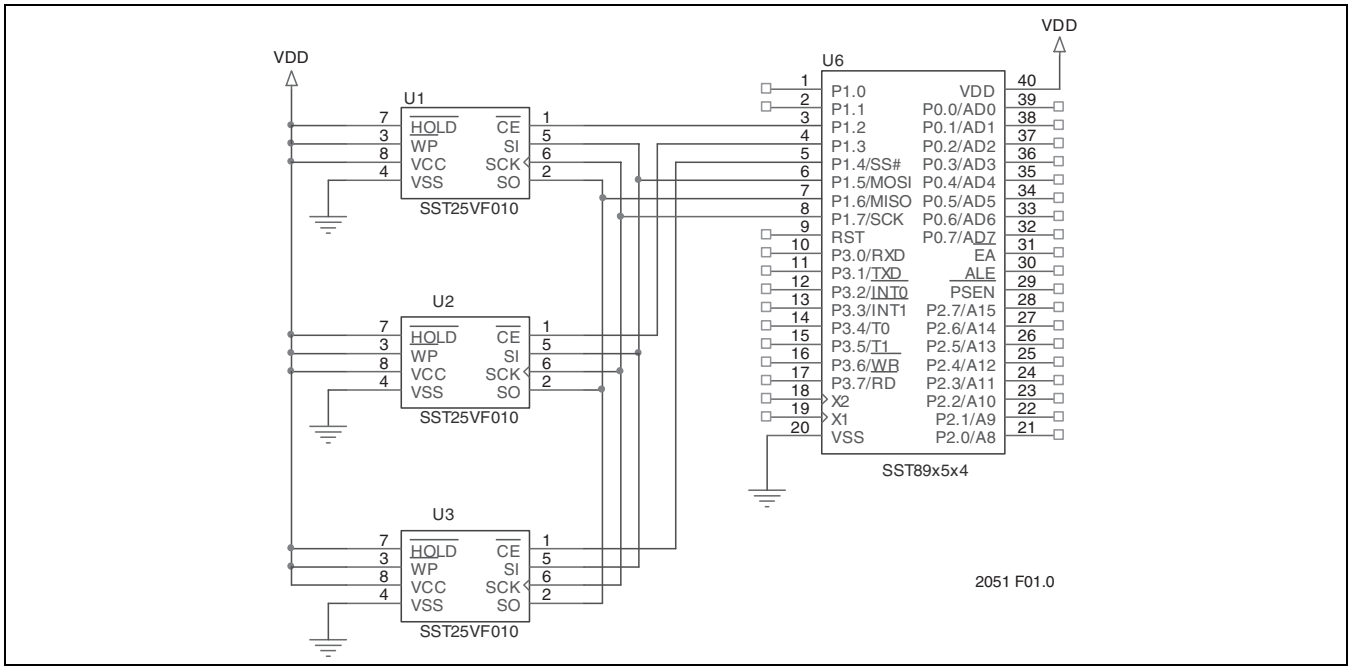

FIGURE 2-1: MASTER-SLAVE SPI SETUP WITH SINGLE VOLTAGE

Application Note

2.2 Master-Slave Running from Different Source Voltages

The schematic in Figure 2-2 shows how the same master-slave setup can be achieved when the master is running at a different voltage than the slaves. As you can see, the first modification is the addition of a voltage regulator. Any suitable voltage regulator will be adequate. The one used with the above system is an STMicroelectronics LF30CV. Next, the addition of the high-speed Texas Instruments SN74LVC4245A level shifters is necessary due to the voltage differences between the master and slave devices. P1.2 - P1.4 outputs are still used to control the Chip-Enables of the three memory chips. The only difference now is that they are routed through the level shifters first so that the proper voltage levels can be conveyed to the memory devices. Level shifters are also used between the MCU and the SI, SO and SCK signals of the serial flash. Even with the addition of the level shifters, there is no software driver change between the formal single voltage system and the latter dual voltage system. One possible design concern is making sure the maximum data transmission speed for the level shifters matches or (preferably) exceeds the maximum desired transmission speed of the SPI bus. Since the TI level shifters shown above can transmit data much faster than the SPI protocol would allow, this is not an issue in our design. However, if your design uses different level shifters, make sure the new level shifters can keep up with the SPI bus speed being used.

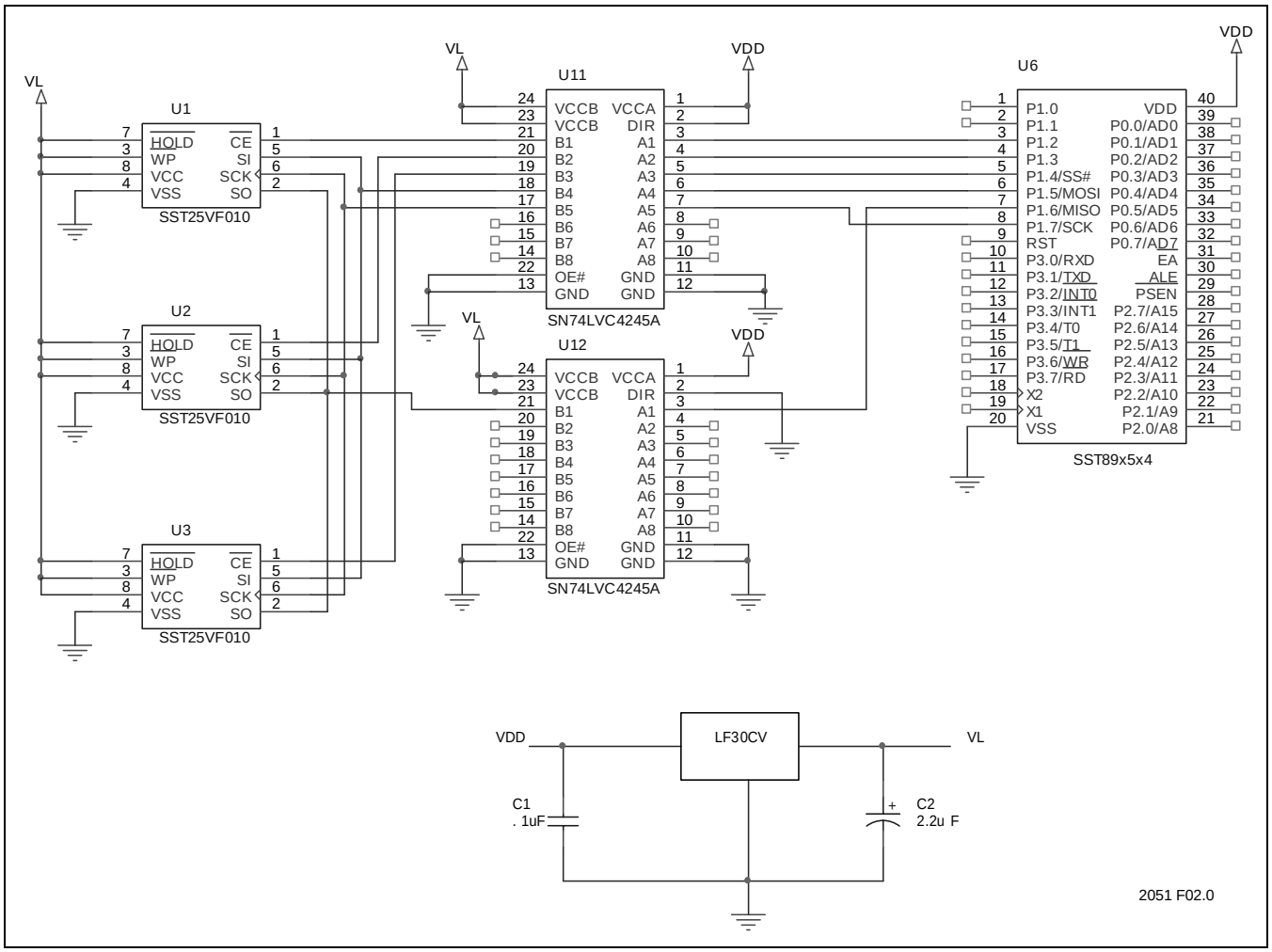

FIGURE 2-2: MASTER-SLAVE WITH DIFFERENT VOLTAGES

3.0 SOFTWARE

3.1 Driver Description

Custom code is required for proper communication between a single master and multi slave devices. The following code is written for this purpose. It is also highly modular so it can be flexible while efficient. The code can be used in its entirety or modified to suit different application needs.

The code contained within this application note is designed as a driver set for communication between a single master device and multiple slave devices via the SPI bus as illustrated in the preceding hardware section. As such, the functions shown below are meant to be called from the customer's main program.

A sample MAIN program is provided to illustrate proper use of the driver functions. The sample MAIN will initialize all three memory chips and perform a chip erase to all three chips. Then it will write the byte values 0-19 to the first memory chip, 20-39 to the second chip, and 40-59 to the third chip. All chips are being written to starting at memory address 0000H.

Name	Function
HW_SPI_Init	Initializes SPI.
SST_MasterIO	Handles byte transfer to and from slave device.
CE_High	Sets Chip Enable of the serial flash high
CE_Low	Clears Chip Enable of the serial flash low
Read_Status_Register	Reads the status register of the serial flash
EWSR	Enables the Write Status Register
WRSR	Performs a write to the status register
WREN	Write enables the serial flash
WRDI	Write disables the serial flash
Read_ID	Reads the manufacturer ID and device ID
Read	Reads one byte from the serial flash and returns byte
Read_Cont	Reads multiple bytes
Byte_Program	Program one byte to the serial flash
Auto_Add_IncA	Initial Auto Address Increment process
Auto_Add_IncB	Successive Auto_Address_Increment process after AAI initiation
Chip_Erase	Erases entire serial flash
Sector_Erase	Erases one sector (4 KB) of the serial flash
Block_Erase	Erases one block (32 KB) of the serial flash
Wait_Busy	Polls status register until busy bit is low

3.2 Driver Functions

```
/*  ----------- Serial Flash Memory Driver Functions ----------  */

/*****************************************************************************************************************
Single-master, multi-slave C code for interfacing SST25VFxxx serial flash devices to the 8051.
*****************************************************************************************************************/

/* Applicable SST Serial Flash Memory:
SST25VF512 512 Kbit(64K x 8) Serial Flash Memory
                        SST25VF010 1 Mbit(128K x 8) Serial Flash Memory
                        SST25VF020 2 Mbit(256K x 8) Serial Flash Memory
                        SST25VF040 4 Mbit(512K x 8) Serial Flash Memory
*/


/*              Description Port Pin Layout                                     */
/*******************************************************************************/
/* P1.7    P1.6    P1.5    P1.4    P1.3    P1.2    P1.1    P1.0    */
/* SCK     MISO    MOSI    SS3#    SS2#    SS1#    NC      NC      */
/*******************************************************************************/


#include <stdio.h>
#include <stdlib.h>

sbit SS1                = P1^2;                         /* Port bit assignments */
sbit SS2                = P1^3;
sbit SS3                = P1^4;
sfr P1                  = 0x90;                         /* SFR addresses */
sfr SPDR                = 0x86;
sfr SPSR                = 0xAA;
sfr SPCR                = 0xD5;

unsigned char SlaveID = 0;                              /* Global Slave ID for slave device selection */

/* Function Prototypes */
void HW_SPI_Init();
void CE_High();
void CE_Low();
unsigned char Read_Status_Register();
void EWSR();
void WRSR(unsigned char byte);
void WREN();
void WRDI();
unsigned char Read_ID(unsigned char ID_addr);
unsigned char Read(unsigned long Dst);
void Read_Cont(unsigned long Dst, unsigned long no_bytes, unsigned char DataArray[]);
void Byte_Program(unsigned long Dst, unsigned char byte);
void Auto_Add_IncA(unsigned long Dst, unsigned char byte);
void Auto_Add_IncB(unsigned char byte);
void Chip_Erase();
void Sector_Erase(unsigned long Dst);
void Block_Erase(unsigned long Dst);
void Wait_Busy();
unsigned char SST_MasterIO(unsigned char HW_SPI_out);
```

```
/*****************************************************************************/
/* PROCEDURE:          CE_High                                               */
/*                                                                           */
/* This procedure drives CE# of the device to logic high.                    */
/*                                                                           */
/*                                                                           */
/* Input:      SlaveID:      Determines which slave                          */
/*                           device is disabled                              */
/*                                                                           */
/* Output:     None                                                          */
/*                                                                           */
/*****************************************************************************/
void CE_High()
{
        if (SlaveID == 1)
        {
                SS1 = 1;                        /* set CE high */
        }
        else if (SlaveID == 2)
        {
                SS2 = 1;
        }
        else if (SlaveID == 3)
        {
                SS3 = 1;
        }
}

/*****************************************************************************/
/* PROCEDURE:          CE_Low                                                */
/*                                                                           */
/* This procedure drives CE# of the device to logic low.                     */
/*                                                                           */
/*                                                                           */
/* Input:      SlaveID:      Determines which slave                          */
/*                           device is enabled                               */
/*                                                                           */
/* Output:     None                                                          */
/*                                                                           */
/*****************************************************************************/
void CE_Low()
{
        if (SlaveID == 1)
        {
                SS1 = 0;                        /* clear CE */
        }
        else if (SlaveID == 2)
        {
                SS2 = 0;
        }
        else if (SlaveID == 3)
        {
                SS3 = 0;
        }
}
```

```
/*****************************************************************************/
/* PROCEDURE:        Read_Status_Register                                    */
/*                                                                           */
/* This procedure reads from Read_Status_Register.                           */
/*                                                                           */
/* Input:     None                                                           */
/*                                                                           */
/* Returns:  status byte                                                     */
/*                                                                           */
/*****************************************************************************/
unsigned char Read_Status_Register()
{
        unsigned char byte = 0;
        CE_Low();                           /* enable device */
        SST_MasterIO(0x05);                 /* send RDSR command */
        byte = SST_MasterIO(0x00);          /* receive byte */
        CE_High();                          /* disable device */
        return byte;
}

/*****************************************************************************/
/* PROCEDURE:        EWSR                                                    */
/*                                                                           */
/* This procedure enables the Write Status Register.                         */
/*                                                                           */
/* Input:     None                                                           */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void EWSR()
{
        CE_Low();                           /* enable device */
        SST_MasterIO(0x50);                 /* enable writing to the status register */
        CE_High();                          /* disable device */
}

/*****************************************************************************/
/* PROCEDURE:        WRSR                                                    */
/*                                                                           */
/* This procedure writes a byte to the Status Register.                      */
/*                                                                           */
/* Input:     data byte                                                      */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void WRSR(unsigned char byte)
{
        CE_Low();                           /* enable device */
        SST_MasterIO(0x01);                 /* select write to status register */
        SST_MasterIO(byte);                 /* data that will change the status of BPx or BPL
                                               (only bits 2,3,7 can be written) */
        CE_High();                          /* disable the device */
        Wait_Busy();
}
```

```
/*****************************************************************************/
/* PROCEDURE:        WREN                                                    */
/*                                                                           */
/* This procedure enables the Write Enable Latch.                            */
/*                                                                           */
/* Input:     None                                                           */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void WREN()
{
        CE_Low();                               /* enable device */
        SST_MasterIO(0x06);                     /* send WREN command */
        CE_High();                              /* disable device */
}

/*****************************************************************************/
/* PROCEDURE:        WRDI                                                    */
/*                                                                           */
/* This procedure disables the Write Enable Latch.                           */
/*                                                                           */
/* Input:     None                                                           */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void WRDI()
{
        CE_Low();                               /* enable device */
        SST_MasterIO(0x04);                     /* send WRDI command */
        CE_High();                              /* disable device */
}

/*****************************************************************************/
/* PROCEDURE:        Read_ID                                                 */
/*                                                                           */
/* This procedure reads the manufacturer's ID and device ID.                 */
/* It will use 90h as the command to read the ID.  It is up to               */
/* the user to give the last byte ID_addr to determine whether               */
/* the device outputs manufacturer's ID first, or device ID first.           */
/* Review the data sheets for details.                                       */
/* Returns ID in variable byte.                                              */
/*                                                                           */
/* Input:     ID_addr                                                        */
/*                                                                           */
/* Returns:  byte:           ID1                                             */
/*                                                                           */
/*****************************************************************************/
unsigned char Read_ID(unsigned char ID_addr)
{
        unsigned char byte;
        CE_Low();                               /* enable device */
        SST_MasterIO(0x90);                     /* send read ID command */
        SST_MasterIO(0x00);                     /* send address */
        SST_MasterIO(0x00);                     /* send address */
        SST_MasterIO(ID_addr);                  /* send address - either 00H or 01H */
        byte = SST_MasterIO(0x00);              /* receive byte */
        CE_High();                              /* disable device */
        return byte;
}
```

```
/*****************************************************************************/
/* PROCEDURE:        Read                                                    */
/*                                                                           */
/* This procedure reads one address of the device.                           */
/* It will return the byte read in variable byte.                            */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*                                                                           */
/* Returns:  byte                                                            */
/*                                                                           */
/*****************************************************************************/
unsigned char Read(unsigned long Dst)
{
        unsigned char byte = 0;

        CE_Low();                                   /* enable device */
        SST_MasterIO(0x03);                         /* read command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16));     /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        byte = SST_MasterIO(0x00);
        CE_High();                                  /* disable device */
        return byte;                                /* return one byte read */
}

/*****************************************************************************/
/* PROCEDURE:        Read_Cont                                               */
/*                                                                           */
/* This procedure reads multiple consecutive addresses of                    */
/* the device and stores the data into DataArray.                            */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*           no_bytes  Number of bytes to read                               */
/*           DataArray Array storing read data                               */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Read_Cont(unsigned long Dst, unsigned long no_bytes, unsigned char DataArray[])
{
        unsigned long i = 0;
        CE_Low();                                   /* enable device */
        SST_MasterIO(0x03);                         /* read command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16));     /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        for (i = 0; i < no_bytes; i++)              /* read until no_bytes is reached */
        {
                DataArray[i] = SST_MasterIO(0x00);  /* receive byte and store in DataArray */
        }
        CE_High();                                  /* disable device */
}
```

```
/*****************************************************************************/
/* PROCEDURE:        Byte_Program                                            */
/*                                                                           */
/* This procedure programs one address of the device.                        */
/* Assumption:  Address being programmed is already                          */
/* erased and is NOT block protected.                                        */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*           byte:   byte to be programmed                                   */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Byte_Program(unsigned long Dst, unsigned char byte)
{
        WREN();
        CE_Low();                                   /* enable device */
        SST_MasterIO(0x02);                         /* send Byte Program command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16));     /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        SST_MasterIO(byte);                         /* send byte to be programmed */
        CE_High();                                  /* disable device */
        Wait_Busy();
}

/*****************************************************************************/
/* PROCEDURE:        Auto_Add_IncA                                           */
/*                                                                           */
/* This procedure programs consecutive addresses of                          */
/* the device. This is used to start the AAI process.                        */
/* It should be followed by Auto_Add_IncB.                                   */
/* Assumption:  Address being programmed is already                          */
/* erased and is NOT block protected.                                        */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*           byte:   byte to be programmed                                   */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Auto_Add_IncA(unsigned long Dst, unsigned char byte)
{
        WREN();
        CE_Low();                                   /* enable device */
        SST_MasterIO(0xAF);                         /* send AAI command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16));     /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        SST_MasterIO(byte);                         /* send byte to be programmed */
        CE_High();                                  /* disable device */
        Wait_Busy();
}
```

```
/*****************************************************************************/
/* PROCEDURE:        Auto_Add_IncB                                           */
/*                                                                           */
/* This procedure programs consecutive addresses of                          */
/* the device. This is used after Auto_Address_IncA.                         */
/* Assumption:  Address being programmed is already                          */
/* erased and is NOT block protected.                                        */
/*                                                                           */
/* Input:     byte:        byte to be programmed                             */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Auto_Add_IncB(unsigned char byte)
{
        CE_Low();                               /* enable device */
        SST_MasterIO(0xAF);                     /* send AAI command */
        SST_MasterIO(byte);                     /* send byte to be programmed */
        CE_High();                              /* disable device */
        Wait_Busy();
}

/*****************************************************************************/
/* PROCEDURE: Chip_Erase                                                     */
/*                                                                           */
/* This procedure erases the entire Chip.                                    */
/*                                                                           */
/* Input:     None                                                           */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Chip_Erase()
{
        WREN();
        CE_Low();                               /* enable device */
        SST_MasterIO(0x60);                     /* send Chip Erase command */
        CE_High();                              /* disable device */
        Wait_Busy();
}
```

```
/*****************************************************************************/
/* PROCEDURE:        Sector_Erase                                            */
/*                                                                           */
/* This procedure Sector Erases the Chip.                                    */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Sector_Erase(unsigned long Dst)
{
        WREN();
        CE_Low();                               /* enable device */
        SST_MasterIO(0x20);                     /* send Sector Erase command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16)); /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        CE_High();                              /* disable device */
        Wait_Busy();
}

/*****************************************************************************/
/* PROCEDURE: Block_Erase                                                    */
/*                                                                           */
/* This procedure Block Erases the Chip.                                     */
/*                                                                           */
/* Input:    Dst:    Destination Address 000000H - 07FFFFH                   */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Block_Erase(unsigned long Dst)
{
        WREN();
        CE_Low();                               /* enable device */
        SST_MasterIO(0x52);                     /* send Block Erase command */
        SST_MasterIO(((Dst & 0xFFFFFF) >> 16)); /* send 3 address bytes */
        SST_MasterIO(((Dst & 0xFFFF) >> 8));
        SST_MasterIO(Dst & 0xFF);
        CE_High();                              /* disable device */
        Wait_Busy();
}

/*****************************************************************************/
/* PROCEDURE:        Wait_Busy                                               */
/*                                                                           */
/* This procedure waits until device is no longer busy.                      */
/*                                                                           */
/* Input:    None                                                            */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void Wait_Busy()
{
        while ((Read_Status_Register() & 0x03) == 0x03)
                Read_Status_Register();         /* waste time until not busy */
}
```

Application Note

```
/*****************************************************************************/
/* PROCEDURE:          HW_SPI_Init                                           */
/*                                                                           */
/* This procedure initializes the hardware SPI on the MCU.                   */
/*                                                                           */
/* Input:      None                                                          */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
void HW_SPI_Init()
{
        P1 = 0xFF;
        SPCR = 0x50;
}

/*****************************************************************************/
/* PROCEDURE:          SST_MasterIO                                          */
/*                                                                           */
/* This procedure handles byte transfer to and from                          */
/* the slave device.                                                         */
/*                                                                           */
/* Input:      None                                                          */
/*                                                                           */
/* Returns:  Nothing                                                         */
/*                                                                           */
/*****************************************************************************/
unsigned char SST_MasterIO(unsigned char HW_SPI_out)
{
        unsigned char temp;
        SPDR = HW_SPI_out;
        do
        {
                temp = SPSR & 0x80;
        } while (temp != 0x80);
        SPSR = SPSR & 0x7F;
        return SPDR;
}
```

3.3 Sample MAIN Program

```
/* ------------------------------------MAIN ROUTINE  ---------------------------------- */

int main()
{
                    unsigned char i = 0;

                    HW_SPI_Init();                  // Initialize the SPI hardware

                    SlaveID = 1;                    // Set SlaveID 1 to access the first memory chip
                    EWSR();                         // Enable writing to the Status Register
                    WRSR(0x00);                     // Enable writing to the memory chip.
                    Chip_Erase();                   // Chip Erase the memory chip

                    Auto_Add_IncA(0, i);            // Start Auto Address Increment Programming from
                    i++;                            // address 0000h
                    for (; i < 20; i++)
                    {
                              Auto_Add_IncB(i);     // Write 20 bytes
                    }
                    WRDI();                         // Stop programming operation

                    SlaveID = 2;                    // Repeat process for the second memory chip
                    EWSR();
                    WRSR(0x00);
                    Chip_Erase();

                    Auto_Add_IncA(0, i);
                    i++;
                    for (; i < 40; i++)
                    {
                              Auto_Add_IncB(i);
                    }
                    WRDI();

                    SlaveID = 3;                    // Repeat process for the third memory chip
                    EWSR();
                    WRSR(0x00);
                    Chip_Erase();

                    Auto_Add_IncA(0, i);
                    i++;
                    for (; i < 60; i++)
                    {
                              Auto_Add_IncB(i);
                    }
                    WRDI();

                    return 0;
}
```