APPLICATION NOTE

ST9 FLASH PROGRAMMING

by 8-bit Micro Application

INTRODUCTION

The ST92F120, ST92F124, ST92F150 microcontrollers provide embedded Flash memory and emulated E3PROM. This document gives guidelines to program the ST9 Flash devices depending on the following three main programming situations:

1) In the manufacturing line: In-System Programming.

2) In the development laboratory: with the EPB.

3) In the field: In-Application Programming.

TABLE OF CONTENT

1 ST9 FLASH DEVICE FEATURES AND CONSTRAINTS 3

1.1 ST9 FLASH DEVICE FEATURES 3

1.2 ST9 FLASH CHARACTERISTICS 3

1.3 APPLICATION CONSTRAINTS 3

1.3.1 In the Manufacturing Line 3
1.3.2 In the Development Laboratory 4
1.3.3 In the Field: In-Application Programming 6

2 COMMUNICATION WITH A VIRGIN FLASH PART 6

2.1 HOW TO COMMUNICATE WITH A VIRGIN PART (ISP) 7

2.1.1 Communication Description 7
2.1.2 Example of Complete Communication Sequence. 8

2.2 SCI0 TESTFLASH CODE 10

3 FLASH PROGRAMMING IN THE MANUFACTURING LINE 12

3.1 PARALLEL MODE 12

3.1.1 RAM code 12
3.1.2 External Memory Code 14

3.2 HANDSHAKE MODE 18

3.3 SERIAL MODE THROUGH SCI0 18

3.3.1 RAM Code 19

1 ST9 FLASH DEVICE FEATURES AND CONSTRAINTS

1.1 ST9 FLASH DEVICE FEATURES

ST9 Flash Device features are all described in the ST92F120 and ST92F150 datasheets. This list shows the main parts of the chip used for this application note:

– Serial Communication Interface SCI, J1850, CAN (ST92F150 only)

– External Memory Interface EMI

– Memory Management Unit MMU and Memory Organization

– Single Voltage Flash and E3PROM: up to 128Kbytes of User Flash, up to 6Kbytes of RAM, 1Kbytes of E3PROM, 8Kbytes of TestFlash (Bootcode reserved)

1.2 ST9 FLASH CHARACTERISTICS

When programming or erasing the user Flash, the instruction fetch can only be performed from the ST9 internal RAM or from external memory by using the external address and data bus (testFlash is not available). Thus, the interrupt vectors, code and interrupt routines that act on the Flash itself require to be loaded into the internal RAM before being executed, or to be located in an external memory.

The Flash operations (write byte, write page, Flash erase...) are controlled through some registers. Only the internal ST9 CPU is able to initiate those operations. The Flash memory cannot be considered as a stand alone Flash able to be programmed by an external tools. In all cases the CPU has to execute a code to program the internal Flash.

The Flash byte programming time is typically 10µs. Depending on the method used to program the Flash, the whole Flash programming time varies from 1 to 5 seconds for a 60K Flash part.

1.3 APPLICATION CONSTRAINTS

There are three different Flash programming situations:

– in the manufacturing line: In-System Programming,

– in the development laboratory,

– in the field, In-Application Flash programming.

Depending on those situations the constraints and the way to program the Flash differ.

1.3.1 In the Manufacturing Line

Usually, the part is virgin and the programming time is an important factor. The faster ways to program the Flash are:

– in parallel mode using the External Memory Interface (address and data bus). Firstly the CPU executes a user code located in RAM to configure the external memory interface then

it executes a code located in an external memory. This code located in RAM has to be loaded at the reset time.

– in parallel mode using the Input/Output ports and a handshake protocol. The CPU executes a code located in RAM. This code has to be loaded at the reset time.

– in serial mode through the Serial Communication Interface SCI. The CPU executes a user code located in RAM. This code has to be loaded at the reset time.

In all cases the CPU executes a code not located in internal Flash. The major constraint consisting in programming the Flash not from the Flash itself is solved.

By default in the manufacturing line the Flash is virgin. To be able to program the Flash, at the reset time, depending on an external condition, the CPU loads a user code in RAM through the Serial Communication Interface (SCI0) and executes it automatically when the last byte is received.

This code configures properly the microcontroller to initiate the Flash programming. It is up to the user to define which programming mode is the most convenient, either through external memory interface, either through an handshake protocol involving a 8-bit data port, either through a serial interface.

The mechanism consisting in loading a user code in RAM through the SCI is fully defined by ST microelectronics. The CPU at that time, executes a code located in testFlash (bootloader code).

All the I/O ports involved in the Flash programming have to be accessible and have to be able to toggle at the proper speed. As it is mandatory to download a code in RAM through the SCI0 interface, at least the I/O corresponding to the SCI0 must have a pull-up or pull-down resistor and no capacitor perturbing the communication.

In case of external memory interface usage, port0 and port1 (address and data bus) must be available. It means that the I/O can be used as standard I/O in the application but must not be connected directly to the ground or the power supply.

All methods are quite similar in term of speed. Actually, the time needed by the SCI to receive a byte in synchronous mode is equivalent to the Flash byte programming time.

Note: ST has a broad range of Third Party programmers. These tools are production oriented and are an excellent complement to ST range of development tools. Contact sales offices to get full details on contacts.

1.3.2 In the Development Laboratory

In the development laboratory the EPROM Programming Board is used. STMicroelectronics had developed a programming tool (ST92F120-EPB and ST92F150-EPB) able to program the ST9 Flash Device either in-circuit through a little connector or directly a part on the EPB (device dependent). Please refer to your nearest ST sales office to order an EPB.

If the part is not soldered on the application board, it can be programmed directly on the EPB board (only for the qfp100 pin package, this option is not available for the qfp64 pin package). If the part is soldered on the application board, the part can be programmed through a little connector link to the SCI0 interface. The user application must implement this connector on his board.

1.3.2.1 In System Programming Mechanism with the EPB

The EPB exchanges some information via the SCI0. This mechanism is fully transparent for the user, the only constraints for the application point of view are:

– The EPB must be able to control the reset pin of the application.

– As SIN0, SOUT0, RxCLK0 and TxCLK0 are used to communicate with the EPB, these lines must be able to transmit or receive some data. Therefore they must be connected to either a pull-up or pull-down resistor.

– A 12 pin connector must be implemented on the user application.

At the reset time if the SOUT0 pin is pulled down, the CPU executes a code located in test-Flash. This code configures the SCI0 interface and allows to initiate a Flash operation.

On the user application board, it is recommended to set a pull-up resistor on SOUT0 to avoid to initialize the SCI0 on each application reset. Actually when SOUT0 is pulled low at the reset time, the ST9 Flash Device considers that a Flash update has to be initiated. Then in the test-Flash, the SCI0 is configured in synchronous mode and is waiting for some data. After a 10ms time out delay (4Mhz crystal) if no datum is received, the part tests the reset vector.

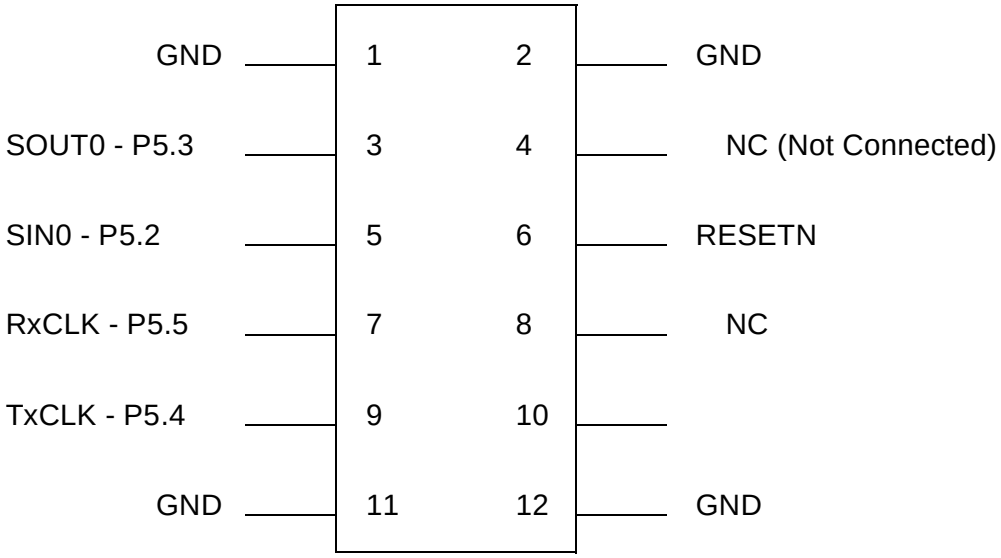
If the reset vector is virgin, meaning that the Flash is virgin, the part executes a halt instruction waiting for a reset and a next attempt to program the Flash.

If the reset vector is programmed, the part executes the user code.

By this way even if SOUT0 is accidentally pulled down at the reset time, the user code can still be executed.

1.3.2.2 User Application Board Connector Specifications

It is a 12-pin connector, 2 rows header (6 pins in a row), 2.54 mm (.100 inch) centers. The port number and the name are indicated, please refer to the datasheet for each pin number as it is related to the ST9 Flash Device package.



1.3.3 In the Field: In-Application Programming

Upon external request, the Flash content can be updated in the field. It is up to the user application to manage all the Flash update and programming. This update can be done through all the serial interface (SCI, I2C, J1850, SPI, CAN...).

The Application Note AN1275, In-Application Programming for the ST92F120, presents how to develop a boot loader always present in the Flash and used to update the application.

2 COMMUNICATION WITH A VIRGIN FLASH PART

When the Flash is virgin, the user has only one way to communicate with the microcontroller. At the reset time the SOUT0 pin (SCI0 peripheral data output pin) must be pulled low, a specific code located in the testFlash sector (bootloader code) is then executed. This code configures the SCI0 peripheral and allows to copy in RAM the data received through the SCI0. Then the microcontroller executes the code loaded in RAM.

According to the user application, this code executed from RAM initiates a Flash programming operation either in parallel mode through the external address and data bus, either in parallel mode through a handshake protocol involving some I/O pins either in serial mode through a serial interface (SCI, SPI, I2C, J1850, CAN).

The serial interface commonly used for this operation is the SCI0. The hardware interface is already done, the RAM code is downloaded through the SCI0. The SCI in synchronous mode is the fastest serial interface.

In the manufacturing line, the Flash is virgin, so it will be mandatory to use this method to start programming the Flash.

2.1 HOW TO COMMUNICATE WITH A VIRGIN PART (ISP)

2.1.1 Communication Description

The programming tool holds the ST9 in Reset and ties low the pin SOUT0. To ensure a proper Reset, the Reset pin must be kept low for at least 20µs.

The programming tool releases the Reset while keeping SOUT0 in a low state.

It waits for CLKOUT0 to go high. Then the programming tool releases SOUT0 and properly configures its serial interface to receive data from the ST9.

The ST9 sends 0x25 (‘%’) as a synchronisation byte

The programming tool has to reply by sending the datum 0x23 (‘#’) informing that a Flash programming operation is initiated. If, after a 10ms delay (4Mhz oscillator), the datum 0x23 is not received, the microcontroller tests the user reset vector value.
If the reset vector is not programmed meaning that the Flash is virgin, the microcontroller executes a HALT instruction waiting for the next reset.
If the Reset vector is programmed meaning that the Flash is already programmed, the microcontroller executes the user code. In this way if a parasitic reset with the SOUT0 pin pulled low occurs, the application is still able to execute the user code.

1. After receiving 0x23, the ST9 answers by sending a ready to receive byte 0x21 (‘!’).
2. The programming tool sends the MSB of the RAMCODE size. This is the RAMCODE size and not the number of bytes transmitted.
3. The ST9 sends a ready to receive byte 0x21 (‘!’).
4. The programming tool sends the LSB of the RAMCODE size.
5. The ST9 sends a ready to receive byte 0x21 (‘!’).
6. Then the programming tool sends the entire RAMCODE, waiting between each byte for a ready to receive byte 0x21 from the ST9. The RAMCODE is stored in RAM starting at 0x200010. The execution of the RAMCODE will start in 0x200018.
7. After having sent the last byte of the RAMCODE the programming tool waits for a ready to receive byte 0x21 (‘!’). Then the programming tool sends a last byte the end the communication, this byte can be anything except 0x25.
8. After receiving the end of communication byte the ST9 jumps to 0x200018 and starts to execute the RAMCODE.

Note: after receiving the last byte of the RAMCODE (not the end of communication byte), the ST9 fills all the unused RAM with 0xFF. This is done to improve the memory protection strategy. In consequence the last ready to receive byte is not issued as fast as the previous one.

2.1.2 Example of Complete Communication Sequence.

Code: 0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0xBF 0x01

size: 0x000A

V_{CC} is provided to the ST9.

The programming tool ties Reset low for at least 20µs.

The SOUT0 pin is held low by the pull down resistor.

The programming tool releases the Reset and waits to receive 0x25 through its serial interface.

Table 1. Communication sequence example

Byte sent by the ST9 Flash Device	Byte sent by the Programming tool	
0x25		Synchronisation byte
	0x23	ISP enable (must be sent within 10ms after receiving the synchronization byte)
0x21		Ready To Receive next byte
	0x00	MSB RAMCODE size
0x21		Ready To Receive next byte
	0X0A	LSB RAMCODE size
0x21		Ready To Receive next byte
	0x00	RAMCODE
0x21		Ready To Receive next byte
	0x11	RAMCODE
0x21		Ready To Receive next byte
	0x22	RAMCODE
0x21		Ready To Receive next byte
	0x33	RAMCODE
0x21		Ready To Receive next byte
	0x44	RAMCODE
0x21		Ready To Receive next byte

Table 1. Communication sequence example

Byte sent by the ST9 Flash Device	Byte sent by the Programming tool	
	0x55	RAMCODE
0x21		Ready To Receive next byte
	0x66	RAMCODE
0x21		Ready To Receive next byte
	0x77	RAMCODE
0x21		Ready To Receive next byte
	0xBF	RAMCODE
0x21		Ready To Receive next byte
	0x01	RAMCODE
0x21		Ready To Receive next byte
	0x00	End of communication (could be anything except 0x25)

The RAMCODE is now loaded in RAM and the ST9 executes the instruction at the address 0x200018h.

The RAM content will be:

0x200010 0x00
0x200011 0x11
0x200012 0x22
0x200013 0x33
0x200014 0x44
0x200015 0x55
0x200016 0x66
0x200017 0x77
0x200018 0xBF
0x200019 0x01

So the ST9 executes the instruction 0xBF 0x01 which is HALT

Starting from that time, the microcontroller executes only the user code.

In case of the Flash is programmed through the SCI0, the first eight locations in RAM, starting at the address 0x200010 should be reserved for the four SCI0 interrupt vectors, then the next byte at the address 0x200018 must be the user code.

The SCI0 is configured in synchronous mode described below, the baud rate depends on the external crystal or resonator and his equal to (Oscillator clock/(2*4)). For a 4Mhz resonator, the baud rate is equal to 500Khz. In this synchronous mode the transmitter provides the clock. When the ST9 sends a datum, it provides the transmitter clock, when the external device sends a datum it provides the transmitter clock. In that case the transmitter clock is linked to the ST9 Flash Device receiver clock. To avoid any overrun error it is recommended to send the data at a 250Khz baud rate.

SOUT0 and CLKOUT0 are used to transmit data, SIN0 and RxCLK are used to receive data.

In this mode when a byte is sent, the least significant bit is sent first.

2.2 SCI0 TESTFLASH CODE

Here is how the TestFlash code configures the SCI0.

```
spp     #21                 ;select MMU paged registers
ld      ISR,#0x23           ;select the segment 0x23 for interrupts
ld      DMASR,#0x20         ;select segment 0x20 for DMA
ld      DPR3,#0x80h         ;DPR3 points on page 0x80 (RAM)
ld      DPR2,#0x02h         ;DPR2 points on page 0x02
ld      DPR1,#0x01h         ;DPR1 points on page 0x01
ld      DPR0,#0x00h         ;DPR0 points on page 0x00
sdm                         ;set data memory

spp     #3                  ;select port5 paged control registers
ld      P5C2R,#0x00         ;SCI0 pin configuration
ld      P5C1R,#0x18         ;SIN0 and RXCLK0 in input
ld      P5C0R,#0xFF         ;SOUT0 and CLKOUT0 in AF

spp     #24                 ;select SCI0 registers
ld      BRGHR,#0            ;reset SCI0
ld      IVR,#0x10           ;set SCI0 interrupt vector table at 0x230010
ld      IDPR,#0             ;SCI0 interrupt request level = 0
```

```
ld      CHCR,#0x83        ;Address mode, 8bit data
ld      CCR,#0xE8         ;CLKOUT0 synchronous Internal baud rate
ld      SOCR,#0x01        ;Output polarity and phase
ld      SICR,#0x83        ;Synchronous mode, Input polarity
ld      BRGLR,#0x04       ;Baud rate INTCLK/4
                          ;4Mhz oscillator, Div2 enable
                          ;baud rate 2Mhz/4 = 500Khz
```

Before executing the user in RAM, some registers are re-initialized:

```
di                        ;disable interrupts
spp     #0                ;select page0
clr     FLAGR             ;reset flag register
calls   0x20,0xC018       ;call 0x200018 through DPR3
```

Then it is up to the user code to properly configure the microcontroller allowing the Flash programming either in parallel mode through the address and data bus either parallel mode through a handshake protocol either in serial mode through a serial interface.

3 FLASH PROGRAMMING IN THE MANUFACTURING LINE

In the manufacturing line the Flash is usually virgin. The only way to initiate a Flash programming operation is to communicate through the ST9 Flash Device SCI0 interface at the reset time as explained in the previous chapter. As soon as the user code is loaded in RAM at the reset time, the CPU starts to execute this code automatically.

STmicroelectronics recommends three main ways to receive the data to be programmed in Flash or E3PROM.

– in parallel mode by using the external memory interface,
– in handshake mode by using some ports to pass the address and data and some I/O to control the different operations,
– in serial mode through the SCI0 in synchronous mode.

3.1 PARALLEL MODE

In this mode, the code loaded in RAM configures mainly the external memory interface, then jumps into external memory and executes a main code located in the external memory. Usually in the manufacturing line, a tester with a bed of nails is going down to the application board and is in contact with all the necessary pins. Those pins are mainly the SCI0 interface, the port 0, port 1, DSN, ASN and R/WN corresponding to the external memory interface. In that case, the tester should be able to emulate a memory. The code located in this memory is constituted by a main loop programming the Flash and a constant table corresponding to the code to be programmed in the ST9 Flash. The application note AN1076 ST9 external memory interface configuration presents how to interface the ST9 with an external memory. It is available either on the CDROM or either on STMicroelectronics internet server. The example below is for guidance only.

3.1.1 RAM code

```
#include <sys/st92f120/st92f120.spp>


   .section .text
   .word                ;no interrupt vector
   .word
   .word
   .word


                        ;no PLL initialization
```

```
spp   #MMU_PG
ld    EMR1, #082h      ;EMR1 register: Normal mode & high-speed buffers
 ld    EMR2, #050h     ;EMR2 register: zero wait states & (Bit 4)=1

spp   #0
ld    WCR, #40h        ; WCR: zero wait states
ld    MODER, #20h      ; MODER: No prescaler division & No High
                       Impedance

spp   #P0C_PG          ;select Port0 port1 control register
ld    P0C2R,#000h      ;Port 0 in Alternate function
ld    P0C1R,#0FFh      ; Push-Pull
ld    P0C0R,#0FFh      ; (Address LSB/Data multiplexed)
ld    P1C2R,#000h      ;Port 1 in Alternate function
ld    P1C1R,#0FFh      ; Push-Pull
ld    P1C0R,#0FFh      ;(Address MSB)

spp   #P6C_PG          ;select Port6 control register
and    P6C2R,#0FDh     ;Port6.1 in Alternate function
or    P6C1R,#002h      ; Push-Pull
ld    P6C0R,#002h      ;R/WN signal
                       ; ASN, DSN are some dedicated pins

jps   0x02,0x0000      ;jump in external memory
                       ;segment 0x02 offset 0x0000
```

3.1.2 External Memory Code

The following example is given for a ST92F120 60K Flash device. Be careful that ST92F120 128K flash, ST92F150 60K and 128K flash have different memory mapping.

; this code is executed from the external memory

; it has to be located in the segment 0x02 starting at the offset 0x0000

; it erases first the flash when the Flash is not virgin

; then program the flash

; in this example it will program 60K flash area (the reset vector is at 0x000000)

; starting from 0x000000 to 0x00EFFF

; the code going to be programmed in the flash is located in external memory

; starting from 0x021000 to 0x02ffff (60K)

```
#include <sys/st92f120/st92f120.spp>


  ld   CICR, #0x87


  spp   #MMU_PG
  ld   DPR0_P, #0x00   ; point on page 0 segment 0
  ld   DPR1_P, #0x89   ; point on FCR and ECR register
  ld   DPR2_P, #0x88   ; point on E3PROM
  ld   DPR3_P, #0x80   ; point on the RAM


  sdm ; set data memory
  ldw   SSPR,#_stack_end; setup stack


  srp   #0 ; select group0
  ldw   rr0,0x0000     ; read user reset vector
  cpw   rr0,#0xFFFF     ; test if flash virgin
  jrz   flash_prog     ; jump to flash programming
```

```
erase_device:          ; erase F0, F1, F2, F3

lp0:
  ld   r10,FCR
  btjxt  r10.0,lp0     ; wait until flash not busy

  or   FCR,#0x20        ; flash chip erase
  or   FCR,#0x80       ; flash write mode start

flash_prog:            ; flash programming
  ldw   rr0,#0x0000    ; initialize rr0 to point on the first address
                       ; going to be programmed
  ldw   rr2,#0x1000    ; initialize rr2 to point on the first address in
                       ; external memory where is located the code to
                       ; program
  call  prog_16K

  spp   #MMU_PG
  ld   DPR0_P,#0x01    ; point on page 1 segment 0
  ldw   rr0,#0x0000    ; initialize rr0 to point on the first address
                       ; page 1
  call  prog_16K

  spp   #MMU_PG
  ld   DPR0_P,#0x02    ; point on page 2 segment 0
  ldw   rr0,#0x0000    ; initialize rr0 to point on the first address
                       ; page 1
  call  prog_16K
```

```
   spp   #MMU_PG
   ld    DPR0_P,#0x03    ; point on page 3 segment 0
   ldw   rr0,#0x0000     ; initialize rr0 to point on the first address
                         ; page 1
   ldw   rr6,#0x333; last 12K
   call   lp2

; the 60K flash located from 0x000000 to 0x00F000 are now programmed
; the user should add a checksum test

   spp   #RCCU_PG        ; software reset to execute the user code in flash
   ld    CLKCTL,#0x20    ; enable software reset
   halt                  ; software Reset
   jp    .               ; infinit loop

prog_16K:

   ldw rr6,#0x444        ; counter each 16K boundary to update DPR0
lp2:
   call prog_16byte
   decw rr2
   jrnz lp2
   ret

prog_16byte:
lp1:
   ld r10,FCR
```

```
btjxt r10.0,lp1       ; wait until flash not busy

or FCR,#0x40          ; flash page program
lddp (rr0)+,(rr2)+    ; load 16 byte from a memory pointed by CSR
lddp (rr0)+,(rr2)+    ; to a memory pointed by a DPR
lddp (rr0)+,(rr2)+    ; in this case DPR0 will be used to point
                      ; on the flash
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
lddp (rr0)+,(rr2)+    ;
or FCR,#0x80          ; flash write mode start
ret
```

3.2 HANDSHAKE MODE

The concept of this mode is to define a handshake protocol between the ST9 and the tool used in the manufacturing line to program the Flash. Some ST9 I/O port pins are configured as standard I/O. A full 8-bit port is used to pass the data some I/O pins are used for status lines or control operation lines. Depending on the application, two 8-bit ports can be assigned to pass the address. The basic operations are:

– Flash erase, erase all the sectors

– Flash byte program

– Flash byte read

– E3PROM erase

– E3PROM page program

– E3PROM read

– Flash and E3PROM programming verification

There is no standard protocol defined. It is up to the user to define his own protocol. As soon as the ST9 is ready to program the next byte, the external tool should put the next datum on the 8-bit data port.

For a write operation the port is configured as an input port to be able to read the next datum to be programmed.

For a read operation the data port must be configured as an output.

The I/O control lines inform the ST9 on the next operation to be performed, they are typically configured as inputs. The control line and operation numbers is defined by the user.

This mode is quite simple to implement, it does not need the usage of the external memory interface or a serial interface. Only the handshake protocol must be well defined. The I/O pin involved in the handshake protocol must be able to be driven by the external programming tool.

The code located in RAM must be loaded at the reset time through the SCI interface as described in Section 2.

3.3 SERIAL MODE THROUGH SCI0

In this mode, the code loaded in RAM configures mainly the PLL and the SCI0. To decrease the Flash programming time through the SCI0 interface, the PLL must be turned on to allow a faster data transfer. The Flash programming is done in two steps:

– At first, a code is downloaded in RAM through the testFlash SCI0 protocol.

– Then this code in RAM is executed. It will configure the PLL and SCI0 interface in a proper mode. This mode is fully defined by the user. Basically, the main loop program the data re-

ceived through the SCI0 in the ST9 Flash. While the data going to be programmed are received through the SCI, the Flash is programmed with the previous data received.

The example below is for guidance only.

3.3.1 RAM Code

The data are received byte per byte. Inside the receive interrupt routine a Flash byte program operation is initiated with the datum just received. Then an output pin is toggled telling that the microcontroller is ready to receive the next datum. This code is located in RAM and must be executed from RAM.

In this way, with a 250Kbaud rate, a 36K Flash area is programmed in 2.5 seconds.

```
#define Disable_Reception (P5DR &= ~0x08) /* P53 is used for the hand-
shake */
#define Enable_Reception (P5DR |= 0x08)



void Init_Handshake(void)
{
   spp(P5C_PG);
   P5C2R &= ~0x08;          /* P53 output for handshake */
   P5C1R |= 0x08;
   P5C0R &= ~0x08;
   Disable_Reception;
}



void Init2_PLL(void)
{
```

```
   unsigned char i;
   spp(RCCU_PG);          /* Select RCCU register page */
   PLLCONF = 0x30;        /* Clock2 x 8 = 16Mhz */
   for (i=0; i<0x3f; i++); /* Delay, wait for PLL lock (500micro_s) */
   CLK_FLAG |= 0x09;      /* Select the PLL clock, all other bit are in
reset conf */
}

void Init2_SCI0(void)
{
/* Init SCI0 Rx_DMA mode Tx */

   spp(P5C_PG);
   P5C2R &= ~0x3c;        /* RxCLK0 (P55) & SIN0 (P52) Input */
   P5C1R |= 0x18;
   P5C1R &= ~0x24;
   P5C0R |= 0x3c;

   spp(SCI0_PG);
   S_BRGHR = 0x00;        /* reset SCI cell */
   S_IVR &= 0x07;
   S_IVR |= 0x10;         /* Set IT vector base to 0x20 */
   S_ISR = 0x00;          /* clear all status flags */
   S_IDPR = 0x01;         /* set PL SCI IT request to 1*/
   S_CHCR = Sm_wl8;       /* 8 bits data length */
   S_CCR = 0xE8;          /* TxCLK, 1X  */
   S_SICR = 0x83;         /* Synchronous mode enabled, no DCD */
   S_SOCR = 0x01;         /* no RTS */
```

```
    S_IMR = 0x02;           /* Receive IT mask */
    S_BRGLR = 32;           /* baud rate generator initialization */
}

void main2(void)
{
    P5DR |= 0x01;
    Init2_PLL();
    Init2_SCI0();
    ei();
    Pointer = 0;
    Init_Handshake();
    Enable_Reception;
    while (1);
}

#pragma interrupt(SCI0_Rx)
void SCI0_Rx(void)
{
    SAVE_PAGE;
    Disable_Reception;
     if(Pointer == 0x4000)
                 {
                 Pointer =0;
                 spp(MMU_PG);
                 DPR0_P++;
    }
```

```
   spp(SCI0_PG);
   asm("
               srp #10
lp2:
               ld r10,FCR
               btjxt r10.0,lp2  ; wait until flash not busy

               or FCR,#0x10     ; flash byte program
               ld (rr0)+,R248   ; load Rx data register in flash
               or FCR,#0x80     ; flash write mode start
               srp #0
   ");
   spp(SCI0_PG);
   S_ISR &= ~Sm_rxdp ;           /* clear the pending bit */
   Enable_Reception;
   RESTORE_PAGE;
}
```

THE SOFTWARE INCLUDED IN THIS NOTE IS FOR GUIDANCE ONLY. STMicroelectronics SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM USE OF THE SOFTWARE.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without the express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics

©2001 STMicroelectronics - All Rights Reserved.

Purchase of I^2C Components by STMicroelectronics conveys a license under the Philips I^2C Patent. Rights to use these components in an I^2C system is granted provided that the system conforms to the I^2C Standard Specification as defined by Philips.

STMicroelectronics Group of Companies
Australia - Brazil - China - Finland - France - Germany - Hong Kong - India - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain Sweden - Switzerland - United Kingdom - U.S.A.

http://www.st.com