AN1474
APPLICATION NOTE

ST92F150 AND ST92F120 PROBLEM RESOLUTION GUIDELINES

by Microcontroller Division Applications

INTRODUCTION

The ST92F120 and ST92F150 are part of the same family of powerful microcontrollers from STMicroelectronics, featuring single voltage flash memory and an innovative E3prom concept.

This document gives guidelines on resolving several common application problems that have been reported by ST Application Engineers.

Testflash features

The testflash features are described in the ST92F120 and ST92F150 datasheets.

Located in a reserved flash area, the TestFlash contains the bootcode which performs the E3prom initialization, provides the possibility to program the flash by In System Programming (refer to the application note *AN1450 ST9 Flash Programming* for a detailed description) and then jumps to the start address application code via the reset vector.

1 EXTERNAL WATCHDOG ROUTINE

During the bootcode execution (from 12.5 ms up to 75ms if an aborted E3prom write operation is detected), a code routine provided by the user may be executed to refresh an external watchdog.

The routine address offset is located at addresses 0x000006h and 0x000007h while the segment where the routine is located has to be written in 0x000009h.

You must fill the word at the address 0x000006h with 0xffffh if your application does not use an external watchdog.

Startup file example (usually called crtbegin.spp) :

```
.org 0x06
.word 0xffff
```

Refer to the application examples provided with the ST9+ User Guide.

1.1 PROBLEM DESCRIPTION

The application is not executed at all and it is not possible to communicate by ISP with the programming board.

1.2 ANALYSIS

When the chip is read directly with the programming board, the contents of addresses 0x000006h and 0x000007h is not equal to 0xFFh.

These addresses in the application code are not equal to 0xFF whereas an external watchdog routine is not provided.

1.3 RESOLUTION

Modify the application code as mentioned at the beginning of this section and erase and reprogram the device.

There is no way to modify this chip by ISP, it has to be unsoldered.

2 ST92F120 RESET VECTOR

One of the powerful features provided by the ST9 Flash device family is the possibility to upgrade the application code by downloading and programming through an I/O interface on the MCU (refer to the application note *AN1275 In Application Programming for the ST92F120* for a detailled description).

In case of a parasitic reset during the flash upgrade operation, the application must be able to recover safely by executing the bootloader code.

To perform this, the bootloader is present in the flash sector where the application starts, the one pointed to by the reset vector. This sector must never be erased.

This is why the reset vector is always located in the smallest sector.

ST92F120 60Kbyte Flash device map

Sector	Size
F0	4 Kbyte
F1	48 Kbyte
F2	8 Kbyte

ST92F120 128Kbyte Flash device map

Sector	Size
F0	64 Kbyte
F1	48 Kbyte
F2	8 Kbyte
F3	8 Kbyte

ST92F150 60 Kbyte Flash device map

Sector	Size
F0	8 Kbyte
F1	8 Kbyte
F2	32 Kbyte
F3	12 Kbyte

ST92F150 128Kbyte Flash device map

Sector	Size
F0	8 Kbyte
F1	8 Kbyte
F2	48 Kbyte
F3	64 Kbyte

2.1 SPECIAL RESET VECTOR MAPPING IN ST92F120 128KBYTE FLASH DEVICE

The reset vector is located at the beginning of Sector F0, at address 0x000000, in all ST92F150 and ST92F120 microcontrollers but one : the ST92F120 128Kbyte Flash device. In this device, the reset vector is located in 0x01E000h, at the begining of the sector F3.

The ST92F120 emulator has the same behaviour as the ST92F120 60Kbyte Flash device : its reset vector is located in 0x000000h, at the beginning of Sector F0.

So, how to emulate an ST92F120 128Kbyte Flash device ?

There are two possibilities :

1. The application startup code is located in Sector F3 and an instruction is added to jump from Sector F0 to Sector F3. This jump is executed by the emulator but not by the final application.
2. The application startup code is located in F3 and an instruction is added to jump from Sector F3 to Sector F0. This jump is executed by the final application but not by the emulator.

2.1.1 Jump from Sector F0 to F3

If the application starts in Sector F3 (for example if you are developing an IAP bootloader), a line of code has to be added in Sector F0 to jump to the reset vector. This way the emulator will jump to the beginning of the application, executing just one more instruction than the chip.

Example, resetf0.spp :

```
;Use this file only when emulating the code
#include "config.spp"
#include "define.h"
PROGRAMMING_MODEL
.section .init

;emulator reset vector redirection

.word __ResetF0
.org    0x06     ;external watchdog routine vector
.word   0xffff
.org    0x0a
.proc __ResetF0
__ResetF0:
    jps seg(__Reset), sof(__Reset)
.endproc __ResetF0
```

This file must be defined in the makefile.mak as a spp source file.

Example :

```
# Give here the name of the C/SPP/ASM source files used in the appli-
cation
C_SRC = main.c
SPP_SRC = resetf0.spp
ASM_SRC =
# Give here the name of the stratup files used in the application
STARTUP_SRC = crtbegin.spp crtend.spp
```

The “init” section of the resetf0.spp file must be loaded in Sector F0 in the scriptfile.ld.

Example :

```
MEMORY
{
  FLASH0      : ORIGIN = 0x000000, LENGTH = 64K, MMU = IDPR0
  FLASH1      : ORIGIN = 0x010000, LENGTH = 48K, MMU = IDPR1
  FLASH2      : ORIGIN = 0x01c000, LENGTH =  8K
  FLASH3      : ORIGIN = 0x01e000, LENGTH =  8K, MMU = IDPR2
  RAM         : ORIGIN = 0x200000, LENGTH =  4K, MMU = IDPR3
  EEPROM      : ORIGIN = 0x220000, LENGTH =  1K,
  REGFILE (t) : ORIGIN =    0x00, LENGTH = 0xb0 /* Groups 0 to 0x0B */
}

SECTIONS
{
.initF0 :
  { resetf0.spp(.init) }            > FLASH0
.init :
  { *(.init) }                      > FLASH3
.text :
  { *(.text) }                      > FLASH3
.fini :
  { *(.fini) }                      > FLASH3
```

Remark :

If the compact memory model is selected, the compiler will generate the following warning message :

appli.u(.init): warning: code section not in the same segment as section `.initF0'
appli.u(.text): warning: code section not in the same segment as section `.initF0'

It is normal, because in the compact memory model, all the application code must be located in the same 64Kbyte segment. But, since the added jump is the first instruction executed and the only one in Segment F0, there is no impact on the execution of the application.

2.1.2 Jump from Sector F3 to F0

If the application does not start in Sector F3 (8Kbyte) but in F0 (64Kbyte) you must add a jump from Sector F3 to Sector F0. This jump will not be executed by the emulator.

Example, resetf3.spp :

```
#include "config.spp"
PROGRAMMING_MODEL
.section .init
.word ResetF3
ResetF3:
jps seg(__Reset), sof(__Reset)
```

This file must be defined in the makefile.mak as a spp source file.

Example :

```
# Give here the name of the C/SPP/ASM source files used in the appli-
cation
C_SRC = main.c
SPP_SRC = resetf3.spp
ASM_SRC =
# Give here the name of the stratup files used in the application
STARTUP_SRC = crtbegin.spp crtend.spp
```

The “init” section of the resetf3.spp file must be loaded in Sector F3 in the scriptfile.ld.

Example :

```
MEMORY
{
  FLASH0      : ORIGIN = 0x000000, LENGTH = 64K, MMU = IDPR0
  FLASH1      : ORIGIN = 0x010000, LENGTH = 48K, MMU = IDPR1
  FLASH2      : ORIGIN = 0x01c000, LENGTH =  8K
  FLASH3      : ORIGIN = 0x01e000, LENGTH =  8K, MMU = IDPR2
  RAM         : ORIGIN = 0x200000, LENGTH =  4K, MMU = IDPR3
  EEPROM      : ORIGIN = 0x220000, LENGTH =  1K,
  REGFILE (t) : ORIGIN =    0x00, LENGTH = 0xb0 /* Groups 0 to 0x0B */
}

SECTIONS
{
.initF3 :
  { resetf3.spp(.init) }            > FLASH3
.init :
  { *(.init) }                      > FLASH0
.text :
  { *(.text) }                      > FLASH0
.fini :
  { *(.fini) }                      > FLASH0
```

Remark :

If the compact memory model is selected, the compiler will generate the following warning message :

appli.u(.init): warning: code section not in the same segment as section `.initF3'
appli.u(.text): warning: code section not in the same segment as section `.initF3'

It is normal, because in the compact memory model, all the application code must be located in the same 64Kbyte segment. But, since the added jump is the first instruction executed and the only one in Segment F3, there is no impact on the execution of the application.

2.2 PROBLEM DESCRIPTION

We are using ST92F120 128Kbyte flash devices from several lots.

The application works fine with the oldest ones but is not executed at all with the new ones. The microcontroller seems to be halted.

2.3 ANALYSIS

Before tracecode V99080027 (mid-2000), the reset vector of the ST92F120 128Kbyte flash device was located in 0x000000h, which was not suitable for In Application Programming needs. It has been changed and the reset vector is now located in 0x01E000h.

2.4 RESOLUTION

Add a jump from Sector F3 to Sector F0.

"THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS."