STMicrolectronics

ROM Patch Library

UM0179
User Manual

CD00074887 Rev 1

December 2005

www.st.com

BLANK

UM0179
USER MANUAL

ROM Patch Library

Introduction

This document describes the ROM Patch Library for ST30R7xx and how to use it to update ROM contents.

Contents

List of tables

List of figures

1 Introduction

This document describes the ST30R7xx's ROM Patch module and the procedure to update ROM contents.

1.1 Referenced documents

The following documents have been used as reference while writing this document:

- ST30 ROM Patching Application Note
- ST30R77x Design Specifications (Zephyrus_ROM_02.pdf)
- ST30 Software Package v.0.9.4

1.2 Summary

This document is divided in four chapters. The first chapter is this introduction, chapter two introduces the ROM memory, the protection and the patching mechanism, chapter three gives general information about this library, chapter four describes how to use the library and chapter five contains a reference to the functions of the library.

2 The ST30 ROM memory and the ROM Patching Mechanism

This chapter describes the organization of the ST30R7xx ROM memory.

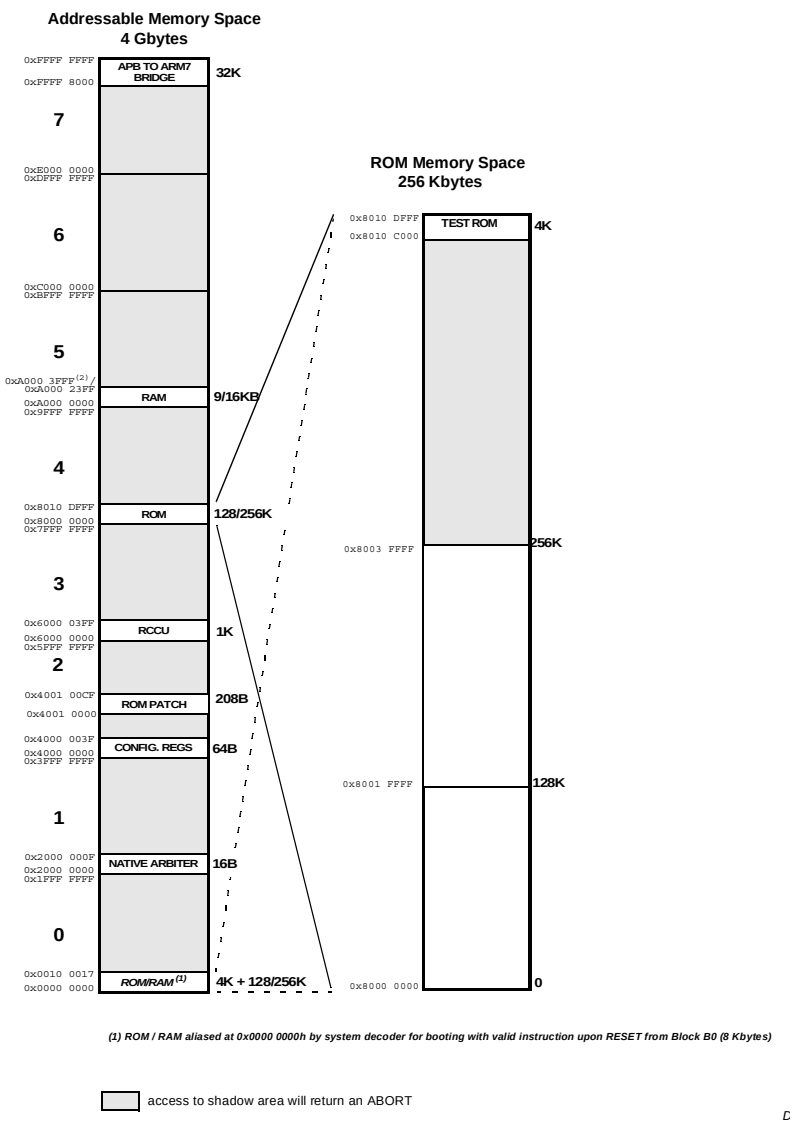
For further information, please refer to the ROM Patch Application Note and to the Zephyrus ROM datasheet.

2.1 ROM organization

In ST30 ROM devices can have up to 256KB of ROM. A Test ROM of 4KB is available in read mode for configuration and initial startup code when boot mode is used.

The following figure details the memory organization of the ST30R7xx.

Figure 1. ST30R7xx memory organization



2.2 ROM protection

The ST30R7xx implements ROM protection with the following capabilities:

- JTAG debug protection.
- ROM access protection.

- Temporary access unprotection.

2.2.1 ROM protection mechanism

The ST30 ROM family implements a mechanism for protecting ROM access from RAM and JTAG Debug. Both protections have to be set to assure maximum protection against piracy.

Debug protection

If JTAG Debug protection is enabled then the core cannot be accessed through JTAG. Protection is enabled through the metal option JTAG ACCESS PROT option and its status can be determined by reading JTAG ACCESS PROT bit of the Device Identification Register 2 (DIDR2).

Access protection

If ROM data access protection is enabled then any attempt to read the ROM from internal RAM, through DMA or through ARM debugger, will result in dummy data (0x00000000) and the ROM Patch will be not accessible. Protection is enabled through the metal option ROM ACCESS PROT and its status can be determined by reading ROM ACCESS PROT bit of the Device Identification Register 2 (DIDR2).

Temporarily ROM access protection disabling

It is possible to temporarily disable the ROM access protection by setting to ‘1’ the TAUB bit of the Rom Control and Status Register (ROM CSR). This bit can only be written from code executing from ROM. To restore the access protection, it is necessary to program the TAUB bit to ‘0’ or reset the device. Any other means to set/reset this bit (such as scanning any instruction through JTAG using ARM debugger) will have no effect.

Table 1. Levels of ROM protection

ROM ACCESS PROT	TAUB	ROM Status
0	x	unprotected
1	0	protected
1	1	unprotected (this must be done with ROM code)

2.3 The ROM Patch module

This paragraph describes the ROM Patch module and its functionalities.

In ST30R7xx, the ROM Patch module is available with 16 independent entries. By programming the ROM Patch module’s address comparators and data patch registers with suitable configuration, data or opcode in ROM can be changed. In particular, the following functionalities are available:

- Change the instruction at a particular ROM address.
- Change the value of any data residing in ROM.
- Generate an abort while instruction / data fetch occurs in ARM mode or in Thumb mode,
- Password access protection and supervisor mode only access for initializing the ROM Patch registers.

2.3.1 Functional description

The ROM Patch module provides 16 address comparators. Each comparator contains a Comparator Address Register (CAR), Patch Data Register (PDR) and a Comparator Control Register (CCR). Each comparator can be used to patch a single location or sequence in ROM whenever the ARM7TDMI CPU address matches with the address stored in the address register of the comparator. A comparator is able to patch the code only when the corresponding Patch Enable bit of the CCR register is set. Execution from code-patch has same instruction timing to the execution from ROM and has same behavior.

2.3.2 Address comparison

The ARM address bus A<17:1> is compared with comparator address register CAR<17:1>. In case of match, the Patch data register PDR<31:0> will be output on ARM input data bus DIN<31:0>. Chip select of ROM is used to enable this comparison. This also removes the absolute address comparison problem when ROM is remapped to 00000000h.

2.3.3 Abort generation

In case the needed patch is more than data or opcode substitution, each comparator is capable of generating an abort in the following cases:

- An instruction is fetched from the patched address and both Patch Enable (PE), Opcode Abort Enable (OAE) bits are set.
- A data is fetched from the patched address and both Patch Enable (PE), Data Abort Enable (DAE) bits are set.

In this case, the Patch Data Register can be used to store information that will be used by the abort handler (ex: location where to branch) and the Comparator Hit Register used to identify the comparator that generated the Abort.

2.3.4 ROM Patch access protection

Two 16-bit password registers, Reference Password register and Password register are provided to protect patch configuration. To avoid leakage in ROM protection, if ROM protection is active both password registers must be written only by supervisor code executed out of the ROM. All of the other registers (except password registers) can be written only if the following conditions are satisfied:

- Password register matches with the inverted value of reference password
- Code is executing from ROM(1)
- Device is in Supervisor mode.

1. If ROM is unprotected, ROM Patch registers can also be accessed from code executed in RAM.

3 About the ROM Patch Library

This chapter gives an overview of the ROM Patch Library.

3.1 General information

The ROM Patch library allows to easily configure the ROM Patch module.

The library provides basic functionality to add and configure code and data patch, both as single replacement or using the abort generation. Functions to lock and unlock protection are also provided.

3.2 Using the library

All the library functions directly access the ROM Patch module, so they have to respect the access rules defined by the active protection level (ROM access protection, core mode). Neither ROM protection level nor CPU mode is changed from within the library so it is up to the user to remove/disable all active protections. This is detailed into the paragraph Preliminary Operations on page 9.

In order to use the library, the following files should be included into your project's sources:

- RPL.h
- ST30R7xx.h

You can then add the source `rpl.c` to your project or link your sources with the precompiled binary, `rpl10d.a` for debug code and `rpl10.a` for release code.

The library supports patching by Abort generation by providing a sample code for the abort handling routine. This handler is contained into the file `RPAbort.s`; you should include this file if you want to use this functionality and not write your own handler. Refer to Patching using Prefetch and Data Aborts on page 1for more information on this.

The library is built over the ST30 Startup kit register's description. For this reason either the user includes the `ST30R7xx.h` file from the Software Package or redefines the missing data types. Please refer to the Software Package documentation for more information.

3.3 Further considerations

While choosing your patch method, please take into account the following points.

The ROM Patch mechanism is totally transparent to the core, so no delays are introduced during address comparison. Of course, the abort generation introduces some latency due to the exception management by the core (maximum delay can be easily determined, please refer to the ARM7TDMI documentation).

Debugging functionalities are not weakened by patching operations. Just take into account that the original content of the ROM will not be any longer visible after patching.

Exception's capture functionality of your debugger can be used to check the patching code executed during abort.

4 Using the ROM Patch Library

This chapter describes the operations needed to program the ROM Patch module using the ROM Patch Library.

4.1 Preliminary operations

In order to proceed with ROM patching all the protection features must the temporarily disabled. These operations are summarized by the following list:

1. If device is not in a privileged mode, change the device mode to supervisor.
2. Temporarily remove the ROM protection by setting the TAUB bit of ROM_CSR register to '1'. Refer to Temporarily ROM access protection on page 6.

Note: If ROM is access protected, patch code in RAM can not access static data in ROM or call the ROM Patch library. If this is needed, temporarily remove the ROM protection (see above). Another method is to copy the needed parameters/data from ROM to RAM during startup and before executing the patch code.

The following code shows how to temporarily disable the ROM protection. This code must be executed from ROM in order to access the ROM_CSR register.

```
#define   ROM_CSR (*(volatile unsigned int*)(0x40000040))


// Set TAUB bit in order to temporarily disable ROM protection.
ROM_CSR = 0x01;
```

4.2 Static Patching at startup

ROM Patch programming is usually performed at boot during micro initialization but before the execution of the application code. This operation is performed by reading addresses of code and data to patch from an external serial memory and programming the appropriate comparators registers. Performing patching during the boot phase assures that all security conditions are satisfied: the micro is in Supervisor mode and that the ROM Patch password in not yet set (see ROM Protection on page 6 and ROM Patch Access Protection on page 7).

- Disable ROM protection
- Read data from serial memory
- Program the Patch Module
- Protect the Patch Module
- Re-enable ROM protection (if DMA is not used)
- Start normal execution, possibly switching to USER mode to further protect the ROM patch.

4.3 Sample code

All the examples in this section refer to the sample code below. This simple code configures the PLL of a micro running with a 4MHz crystal to generate a 12MHz clock (DIV16, DX=1, MX=1).

```
// Registers definition
#define   CLKCONTROL_CLK_FLAG \
            (*(volatile unsigned short *)(0x60000008))
#define   CLKCONTROL_PLLCONF \
            (*(volatile unsigned short *)(0x60000018))


CLKCONTROL_PLLCONF    = 0x11;             // Set the PLL

while (!(CLKCONTROL_CLK_FLAG & 0x02));    // Wait for synch

CLKCONTROL_CLK_FLAG = 0x8009;             // Select PLL Clock
```

After C compilation, the above code is translated in assembly to:

```
80000358  [0xe3a00011]   MOV      r0,#0x11
8000035c  [0xe3a01460]   MOV      r1,#0x60000000
80000360  [0xe1c101b8]   STRH     r0,[r1,#0x18]

80000364  [0xe1a00000]   NOP
80000368  [0xe3a00460]   MOV      r0,#0x60000000
8000036c  [0xe1d000b8]   LDRH     r0,[r0,#8]
80000370  [0xe3100002]   TST      r0,#2
80000374  [0x0afffffb]   BEQ      0x80000368

80000378  [0xe59f0008]   LDR      r0,0x80000388
8000037c  [0xe3a01460]   MOV      r1,#0x60000000
80000380  [0xe1c100b8]   STRH     r0,[r1,#8]

80000388  [0x00008009]   DCD      0x00008009
```

4.4 Patching a single instruction or data

This example shows how to patch data and instructions in ROM.

To patch a single instruction, the following function can be used:

```
void SetCodePatch( u8ID, u32Address, u32Opcode );
```

To patch a single data, the following function can be used:

```
void SetDataPatch( u8ID, u32Address, u32Data );
```

Let's change the sample code by inserting mistakes on line number two and number three respectively. In particular we assume that the AND operator has been replaced by mistake with an OR (wrong instruction) and that the value 0x00008009 has been replaced by mistake with 0x00008019 (wrong data).

```
// Registers definition
#define   CLKCONTROL_CLK_FLAG \
            (*(volatile unsigned short *)(0x60000008))
#define   CLKCONTROL_PLLCONF \
            (*(volatile unsigned short *)(0x60000018))


CLKCONTROL_PLLCONF = 0x11;                // Set the PLL

while (!(CLKCONTROL_CLK_FLAG | 0x02));    // Wait for synch

CLKCONTROL_CLK_FLAG = 0x8019;             // Select PLL clock
```

The corresponding assembly code will be:

```
; CLKCONTROL_PLLCONF = 0x11;
80000358  [0xe3a00011]   MOV      r0,#0x11
8000035c  [0xe3a01460]   MOV      r1,#0x60000000
80000360  [0xe1c101b8]   STRH     r0,[r1,#0x18]

; while (!(CLKCONTROL_CLK_FLAG | 0x02));
80000364  [0xE1A00000]   NOP
80000368  [0xE3A00460]   MOV      r0,#0x60000000
8000036C  [0xE1D000B8]   LDRH     r0,[r0,#8]
80000370  [0xE3900002]   ORRS     r0,r0,#2
80000374  [0x0AFFFFFB]   BEQ      0x80000368

; CLKCONTROL_CLK_FLAG = 0x8019;
80000378  [0xE59f0008]   LDR      r0,0x80000388
8000037C  [0xE3A01460]   MOV      r1,#0x60000000
80000380  [0xE1C100B8]   STRH     r0,[r1,#8]

80000388  [0x00008019]   DCD      0x00008019
```

In order to repair mistakes in this code, two patches must be applied: one to correct the wrong instruction (`ORRS r0, r0, #2` instead of `TST r0, #2`, respectively E3900002h and 3100002h) and one to correct the wrong data (8019h instead of 8009h).

In order to correct the wrong instruction, call the function `SetCodePatch` with the comparator number (1), the address of the wrong instruction (80000370h) and the new opcode (E3100002h, corresponding to `TST r0, #2`) that will replace the wrong one.

```
// Configures first comparator to patch opcode at 0x80000370
SetCodePatch( 0x01, 0x00000370, 0xE3100002 );
```

In order to correct the wrong data, call the function `SetDataPatch` with the comparator number (2), the address holding the wrong value (800000388h) and the new, correct, value (00008009h).

```
// Configures second comparator to patch data at 0x80000388
SetDataPatch( 0x02, 0x00000388, 0x00008009 );
```

4.5 Patching using Prefetch and Data Aborts

When more complex patches are needed, the ROM Patch should be configured to generate an Abort in order to reduce the number of comparators used. This example shows how to insert a Prefetch Abort at a given address and execute patched code using the following functions:

```
void SetCodeAbortPatch( u8ID, u32Address, pfnCallback );
```

Let's assume for example to change the sample code to insert some mistakes:

```
// Registers definition
#define   CLKCONTROL_CLK_FLAG \
            (*(volatile unsigned short *)(0x200000F8))
#define   CLKCONTROL_PLLCONF \
            (*(volatile unsigned short *)(0x60000018))


CLKCONTROL_PLLCONF = 0x11;                // Set the PLL

while (!(CLKCONTROL_CLK_FLAG & 0x04));    // Wait for synch

CLKCONTROL_CLK_FLAG = 0x8009;             //
```

The generated assembly code will be:

```
; CLKCONTROL_PLLCONF = 0x11;
80000358  [0xE3A00011]   MOV      r0, #0x11
8000035C  [0xE3A01460]   MOV      r1, #0x60000000
80000360  [0xE1C101B8]   STRH     r0, [r1,#0x18]

; while (!(CLKCONTROL_CLK_FLAG & 0x04));
80000364  [0xE1A00000]   NOP
80000368  [0xE3A00460]   MOV      r0, #0x20000000
8000036C  [0xE1D00FB8]   LDRH     r0, [r0,#0xF8]
80000370  [0xE3100004]   tst      r0, #4
80000374  [0x0AFFFFFB]   BEQ      0x80000368

; CLKCONTROL_CLK_FLAG = 0x8009;
80000378  [0xE59F0008]   LDR      r0, 0x80000388
8000037C  [0xE3A01460]   MOV      r1, #0x60000000
80000380  [0xE1C100B8]   STRH     r0, [r1,#8]

80000388  [0x00008019]   DCD      0x00008009
```

To correct the introduced mistakes, it could be possible to use a number of comparators as described in the above example, which inserts patches at three different addresses. However, as the instructions to patch are consecutive in memory, it is possible to reduce the number of comparators used to one. This can be done by configuring the Patch module to generate a Prefetch Abort at the first address (80000368h) and to implement a prefetch handler procedure that will replace all the wrong instructions (at addresses 80000368h, 8000036Ch and 80000370h) with the correct ones.

In order to do this, the following step must be followed:

- Configure the Prefetch abort vector to point to a handler routine that can manage exceptions generated from the ROM Patch.
- Copy the patched code from the external EPROM into the RAM.
- Configure the ROM Patch module to generate a prefetch exception.

The exception handler must check if the exception is a real exception or has been generated from the ROM Patch module. If the exception has been generated from the ROM Patch, the handler should run the patch code related to the comparator hit. The following code can be used for this:

```
;-------------------------------------------------------------
;
ROMPATCH_PDR0     EQU 0x40          ; PDR[0] register offset
ROMPATCH_CIR      EQU 0xC4          ; CIR register offset
ROMPATCH_CIR_CHF  EQU 0x10          ; CIR THF bit mask

;-------------------------------------------------------------;
ROMPATCH_BASE     DCD 0x40010000    ; ROM Patch Base address
```

```
;-------------------------------------------------------------;
      AREA Patch, CODE, READONLY

;-------------------------------------------------------------;
;  Prefetch_Handler
;
;  Description:
;    Prefetch Abort handler.
;
;-------------------------------------------------------------
Prefetch_Handler PROC

    ;// Update LR to point to the next instruction
    SUB     lr, lr, #4

    ;// Backup registers
    STMFD   sp!, {r0-r4, r12}
    ;// Must store LR if using BL instructions

    ;// Check if the ROM Patch generated this Abort
    LDR       r0, ROMPATCH_BASE        ; ROMPatch base address
      LDR     r1, [r0, #ROMPATCH_CIR]    ; Load CIR register
    TST       r1, #ROMPATCH_CIR_CHF      ; Test the THF bit
    BEQ     PH_Exit

    ; Retrieves address of patch code from PDR of this
    ; comparator
    BIC     r1, #ROMPATCH_CIR_CHF     ; Obtain CID (clear CHF)
    ADD     r0, r0, #ROMPATCH_PDR0    ; Point to PDR[0]
    LDR     r1, [r0, r1, LSL#2 ]      ; Load PDR[ CID ]

    ; Jump to patch
    MOV     pc, r1                  ; Jump to patch code


PH_Exit
    ;// Restore PA registers
    LDMFD     sp!, {r0-r4, r12}

    ;// Exit from the Prefetch Abort mode
    MOVS    pc, lr

  ENDP
```

Note: The ROM Patch library stores the address of the patch code into the PDR register. The address of this handler must be stored into the exception vector table at address 00000000h.

Below is the code that will substitute the erroneous code in ROM. It should be copied from the external EPROM into the RAM. This code should be written in assembly as it must update the `lr` register in order to return from the Abort mode by skipping all the patched code.

```
;---------------------------------------------------------------
;
; Patch01
;
; Description:
:   Sample patch code.
; Remarks:
;     lr points to the patched address and must be adjusted
;     before leaving.
;     Code is executed in Abort mode.
;     This function must return restoring r0-r4, r12 and CPSR
;   from stack.
;     ROM is not accessible if ROM access protection is enabled.
;     ROM Patch is not accessible if ROM access protection is
;   enabled.
;---------------------------------------------------------------

Patch01 PROC

    MOV      r0, #0x60000000
    LDRH     r0, [r0,#8]
    TST      r0, #2
    BEQ      Patch01

    ; Adjusts lr to skip patched code (5 instr.)
    ADD       lr, lr, #0x10

    ;// Restore PA registers
    LDMFD     sp!, {r0-r4, r12}

    ;// Exit from the Prefetch Abort mode
    MOVS    pc, lr


    ; ... or jump to the Prefetch Handler epilogue
    ;B        PH_Exit

  ENDP
```

Note: Please note that patch code executed in RAM cannot access data in protected ROM.

To configure the Patch module to generate an Abort, call the function `SetCodeAbortPatch` with the comparator number (1), the address of the first wrong instruction (80000368h) and a pointer to the routing that will replace the wrong code (Patch01).

The code below configures the comparator (1) and configures it to generate a Prefetch Abort.

```
// Configures the first comparator to generate a prefetch abort
// if code at 0x80000368 is executed.
SetCodeAbortPatch( 0x01, 0x00000368, Patch01 );
```

4.5.1 Identifying access to patched memory

In case of code patching, functionality provided by the `GetComparatorHitFlag` function becomes useful to identify the comparator that has generated the exception.

When an address match occurs, the ROM Patch module can keep track of the match in the Comparator Identification Register so that the exception handler can detect which patch to run. To enable tracking of comparator hits, please use the `RPF_COMPARE_HIT_MASK` flag when configuring the comparator. The function `GetComparatorHitFlag()` can be used to retrieve the value of the Comparator Hit register.

Note: This function will clear the contents of the Comparator Hit register.

5 Library reference

This chapter describes functions and structures provided by the ROM Patch Library.

5.1 Functions

The ROM Patch Library provides the following functions:

SetPatch	Configures a patch.
SetCodePatch	Configures a patch on code.
SetLongJumpPatch	Configures a patch to generate a Long Jump.
SetDataPatch	Configures a patch on data.
SetDataAbortPatch	Configures a patch to generate a Data Abort.
SetCodeAbortPatch	Configures a patch to generate a Prefetch Abort.
GetComparatorHitFlag	Retrieves the contents of the Comparator Hit register.
SetPassword	Set the password for access protection.
LockPatch	Enable access protection to the ROM Patch.
UnlockPatch	Disable access protection to the ROM Patch.

The following paragraphs describe these functions more in detail.

5.1.1 SetPatch

```
void SetPatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 u32Data,
  UINT8 u8Flags
);
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] address to patch.

u32Data

[in] data/opcode to return on address match.

u8Flags

[in] configuration flag for the comparator.

Return values

This function has no return value.

Remarks

This function configures the ROM Patch comparator. Content of flags can be a combination of: `RPF_COMPARE_HIT_MASK`, `RPF_WORD_COMPARISON_ENABLE`, `RPF_DATA_ABORT_ENABLE`, `RPF_CODE_ABORT_ENABLE`, `RPF_LONGJUMP_ENABLE`, `RPF_PATCH_ENABLE`.

5.1.2 SetCodePatch

```
void SetCodePatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 u32Opcode
);
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] ROM Address to patch.

u32Opcode

[in] opcode to substitute.

Return values

This function has no return value.

Remarks

This function configures the patch comparator for opcode substitution. It is equivalent to call `SetPatch( u8Id, u32Address, u32Opcode, 0 );`

5.1.3 SetLongJumpPatch

```
void SetLongJumpPatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 u32NewAddress
) ;
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] ROM Address to patch.

u32NewAddress

[in] Address of the new code to execute.

Return values

This function has no return value.

Remarks

This function configures the patch comparator for opcode substitution. It is equivalent to call `SetPatch( u8Id, u32Address, u32NewAddress, RPF_LONGJUMP_ENABLE )`

5.1.4 SetDataPatch

```
void SetDataPatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 u32Data
);
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] ROM Address to patch.

u32Data

[in] New data value for ROM.

Return values

This function has no return value.

Remarks

This function configures the patch comparator for opcode substitution. It is equivalent to call `SetPatch( u8Id, u32Address, u32Data, 0 ). SetCodeAbortPatch()` stores the address of the callback routine into the Patch Data Register (PDR) of the corresponding comparator.

5.1.5 SetDataAbortPatch

```
void SetDataAbortPatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 pfnCallback
);
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] ROM Address to patch.

pfnCallback

[in] Pointer to the callback function to call on DATA ABORT.

Return values

This function has no return value.

Remarks

This function configures the patch comparator for opcode substitution. It is equivalent to call `SetPatch( u8Id, u32address, pfnCallback, RPF_COMPARE_HIT_MASK | RPF_DATA_ABORT_ENABLE )`. `SetDataAbortPatch()` stores the address of the callback routine into the Patch Data Register (PDR) of the corresponding comparator.

5.1.6 SetCodeAbortPatch

```
void SetCodeAbortPatch(
  UINT8 u8ID,
  UINT32 u32Address,
  UINT32 pfnCallback
);
```

Parameters

u8Id

[in] ID of the comparator to set.

u32Address

[in] ROM Address to patch.

pfnCallback

[in] Pointer to the callback function to call on DATA ABORT.

Return values

This function has no return value.

Remarks

This function configures the patch comparator to generate a Prefetch Abort on comparator's hit. It is equivalent to call `SetPatch( u8Id, u32Address, pfnCallback, RPF_COMPARE_HIT_MASK | RPF_CODE_ABORT_ENABLE )`.

5.1.7 GetComparatorHitFlag

```
UINT8 GetComparatorHitFlag( void );
```

Parameters

This function takes no parameters.

Return values

Returns a mask of hit comparators.

4	3	2	1	0
Hit	Comparator ID			

Remarks

This function returns the comparator hits, if any. In order to be notified of a comparator match, the `RPF_COMPARE_HIT_MASK` flag must be set.

5.1.8 SetPassword

```
void SetPassword(
  UINT16 u16Password
);
```

Parameters

u16Password

[in] password to unlock the ROM Patch.

Return values

This function has no return value.

Remarks

This function sets the password for the ROM Patch module in order to avoid accidental writing to its registers.

5.1.9 LockPatch

```
void LockPatch( void );
```

Parameters:

This function has no parameters.

Return values:

This function has no return value.

Remarks:

This function enables access protection to the ROM Patch module. A password must have been set before by SetPassword().

5.1.10 UnlockPatch

```
void UnlockPatch(
  UINT16 u16Password
);
```

Parameters

u16Password, [in] password to unlock the ROM Patch.

Return values

This function has no return value.

Remarks

This function removes access protection to the ROM Patch module.

5.2 Structures and data types

This paragraph describes structures and data types defined by the ROM Patch Library.

5.2.1 ROM Patch flags

Flags listed in *Table 2.* can be used to configure ROM Patch comparators.

Take into account that if you use the helper macros, the only flags you would specify are `COMPARE_HIT_MASK` and `WORD_COMPARISON_ENABLE`. Helper macros correctly set the needed flags.

Table 2. ROM Patch flags

Flag	Value	Description
RPF_COMPARE_HIT_MASK	0x20	Enables the Hit Mask, usually used in conjunction with Abort generation.
RPF_WORD_COMPARISON_ENABLE	0x10	Enables the Word Comparison mode for Thumb code.
RPF_DATA_ABORT_ENABLE	0x08	Enable Data Abort generation.
RPF_CODE_ABORT_ENABLE	0x04	Enable Prefetch Abort generation.
RPF_LONGJUMP_ENABLE	0x02	Enable Long Jump on comparator match.
RPF_PATCH_ENABLE	0x01	Enable Patch comparator.

6 Software end-user agreement

END-USER LICENSE AGREEMENT

YOU ("You") SHOULD CAREFULY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE (the "Software") WHICH IS LICENSED BY STMICROELECTRONICS NV ("ST") TO ITS CUSTOMERS FOR THEIR USE ONLY AS SET FORTH BELOW.

Acceptance

If You agree with and accept the terms and conditions of this Agreement it shall become a legally binding agreement between You and ST and You may proceed to install, copy and use the Software in accordance with the terms and conditions of the Agreement.

Rejection and Right to a Refund

If You do not agree with or do not wish to be bound by the terms and conditions of this Agreement You may NOT install, copy or use the Software.

License

ST hereby grants to You, subject to the terms and conditions of this Agreement, a non-exclusive, non-transferable, worldwide license, without the right to sub-license, to use the Software to develop software applications for ST microcontroller product only. You are not permitted to lease, rent, distribute or sublicense the Software or to use the Software in a time-sharing arrangement or in any unauthorized manner.

Restrictions on Use of the Software

The Software is delivered free of charge by electronically means. You are allowed to copy the Software in its initial form to be installed on as many computers as needed.

You shall only use the Software on a single computer connected to a single monitor at any one time.

Limited Warranty

ST warrants to You that the Software will perform substantially in accordance with the accompanying documentation.

ST's total liability and your exclusive remedy for breach of the limited warranty given above shall be limited to ST, it is ST's sole option, using reasonable efforts to correct material, documents, reproduce defects in the Software.

EXCEPT AS PROVIDED ABOVE ST EXPRESSLY DISCLAIMS ALL OTHER REPRESENTATIONS, WARRANTIES, CONDITIONS OR OTHER TERMS, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Limitation of Liability

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ST BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING LOSS OF PROFITS) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE WHETHER BASED ON A CLAIM UNDER CONTRACT, TORT OR OTHER LEGAL THEORY, EVEN IF ST WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. ST does not seek to limit or exclude liability for death or personal injury arising from ST's negligence and because some jurisdictions do not permit the exclusion or limitation of liability for consequential or incidental damages the above limitation relating to liability for consequential damages may not apply to You.

Third Party Rights

Software provided under this Agreement may contain or be derived from portions of materials provided by a third party under license to ST. THE THIRD PARTY DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED WITH RESPECT TO THE USE OF SUCH MATERIALS IN CONNECTION WITH THE SOFTWARE, INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF SATISFACTORY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE.

Software provided under this Agreement may contain or be derived from portions of materials provided by a third party under license to ST. The third party may enforce any of the provisions of this Agreement to the extent such third party materials are effected. Additionally, any limitation of liabilities described in this Agreement also applies to any third-party supplier of materials supplied to You. ST and its third party supplier limitations of liabilities are not cumulative. Such third party is an intended beneficiary of this section.

Term and Termination

This Agreement shall remain in force until terminated by You or by ST.

Without prejudice to any of its other rights if You are in breach of any of the terms and conditions of this Agreement then this Agreement will terminate immediately. Upon such termination You agree to destroy the Software and documentation together will all copies in any form.

You may terminate this Agreement at any time by destroying the Software and documentation together will all copies in any form.

General

This Agreement is governed by the law of France.

This is the only agreement between You and ST relating to the Software and it may only be modified by written agreement between You and ST.

This Agreement may not be modified by purchase orders, advertising or other representation by any person.

If any clause in this Agreement is held by a court of law to be illegal or unenforceable the remaining provisions of the Agreement shall not be affected thereby.

The failure by ST to enforce any of the provisions of this Agreement, unless waived in writing, shall not constitute a waiver of ST's rights to enforce such provision or any other provision of the Agreement in the future.

Use, copying or disclosure by the US Government is subject to the restrictions set out in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 C.F.R. 52.227-19, as applicable.

You agree that You will not export or re-export the Software to any country, person or entity or end user subject to U.S.A. export restrictions, or other countries export restrictions.

7 Revision history

Date	Revision	Changes
02-May-2005	1	Initial release.

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 connection with therir products.