AN204770

F²MC-16FX Family, Memory Patch Function

This application note describes the functionality of the Memory Patch Function and gives some examples. The Memory Patch Function is mostly used for debugging. It allows monitor of internal bus transfers or CPU execution.

Contents

1 Introduction....1
1.1 Key Features....1
2 The Memory Patch Function....1
2.1 Introduction....1
2.2 Block Diagram....2
2.3 Registers....2
3 Memory Patch Function Examples....11
3.1 Data Match on (Write) Access by CPU (Data Value Break)....11
3.2 Patch on Instruction Access (Instruction Break)....12
3.3 Communication Interface Interrupt configuration....15
3.4 Data Match on (Write) Access by DMA (Data Value Break)....17
4 Additional Information....19
5 Document History....20

1 Introduction

This application note describes the functionality of the Memory Patch Function and gives some examples. The Memory Patch Function is mostly used for debugging. It allows monitor of internal bus transfers or CPU execution.

1.1 Key Features

- Detection of Access of Memory Address with Address Mask or Access of Address Area
- Detection of Access of Data with Data Mask
- Detection of Access by Byte and/or Word, Read and/or Write, Data or Code, DMA and/or CPU
- Instruction / Data Replacement or INT9 Generation at Detection
- Memory Write Protection

2 The Memory Patch Function

The Basic Functionality of Memory Patch Function

2.1 Introduction

The Memory Patch Function allows the user to watch data exchange on the internal bus and detect address hits. It can be used for supervising the memory in operating systems or for debugging user software on-chip.

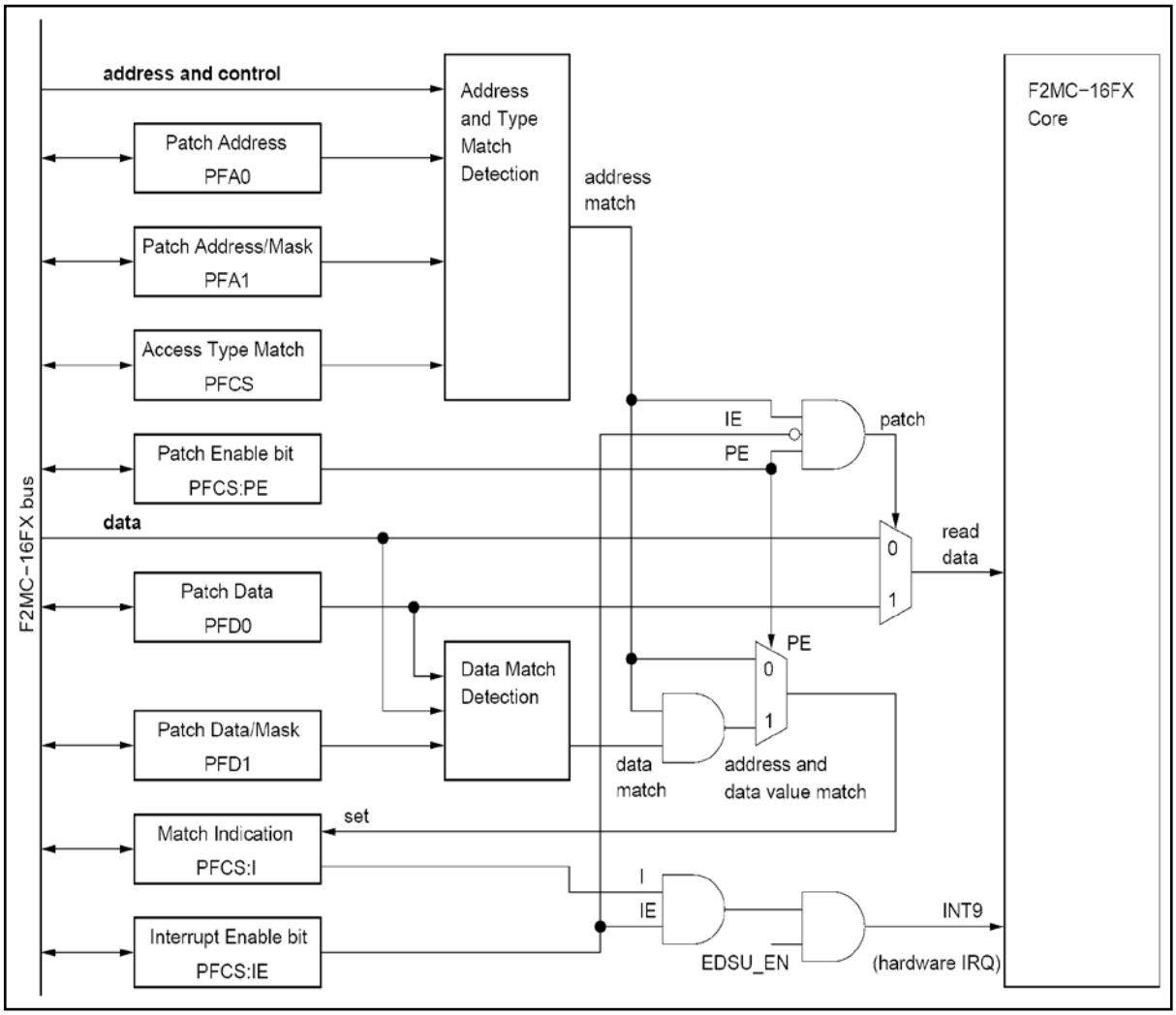
It contains 8 channels, which can be grouped by 2 neighbored channels.

The Memory Patch Function is also used by the Euroscope Debugger.

2.2 Block Diagram

Figure 1 shows the Internal Block Diagram of 2 Channels of the Memory Patch Function

Figure 1. Memory Patch Function Block Diagram



2.3 Registers

2.3.1 Patch Function Address Registers (PFA[0-7])

These eight registers contain 3 bytes each for 24-bit addresses. The lower bytes are named PFALn, the middle PFAMn, and the upper byte PFAHn.

Two neighboured registers are forming a group, which can represent two separate addresses, an address range or an address and a mask.

2.3.2 Patch Function Data Registers (PFD[0-7])

These eight 2 byte registers are used for data replacement for Memory Patch or data match and data mask for Data Value Break.

2.3.3 Patch Function Control/Status Register (PFCS0-3)

Each PFCSn controls two of the PFAx and PFDx register, where PFCS0 correspond to PFA0, PFA1 and PFD0, PFD1, PFCS1 to PFA2, PFA3, PFD2, PFD3, and so on …

Table 1. PFCS

Bit No.	Name	Explanation	Value	Operation
15	READ	Read Access Detection enable	0	Read Accesses do not match
			1	Read Accesses match (also at RMW instructions)
14	Write	Write Access Detection enable	0	Write Accesses do not match
			1	Write Accesses match (also at RMW instructions)
13	BYTE	8-Bit Access Detection enable	0	8-Bit Accesses do not match
			1	8-Bit Accesses match
12	WORD	16-Bit Access Detection enable	0	16-Bit Accesses do not match
			1	16-Bit Accesses match
11	CODE	Code Access Detection enable	0	Code Accesses do not match
			1	Code Accesses match
10	DATA	Data Access Detection enable	0	Data Accesses do not match
			1	Data Accesses match
9	CPU	CPU Access Detection enable	0	CPU Accesses do not match
			1	CPU Accesses match
8	DMA	DMA Access Detection enable	0	DMA Accesses do not match
			1	DMA Accesses match
7	AM	Address Mask Bit	0	Disable Mask Option for PFA*2n*. PFA*2n+1*define Address for channel n+1.
			1	PFA*2n+1* is used as OR-Mask of PFA*2n* and Address
6	AR	Address Range Enable	0	Disable Address Range for PFA*2n* and PFA*2n+1*. PFA*2n* and PFA*2n+1* are used for Addresses of Channel *2n* and *2n+1*, *if PFA2n+1*is not used as mask.
			1	Enable Address Range. PFA*2n* as start address and PFA*2n+1* as end address.
5	PE1	Patch Enable Channel *2n+1*	0	Please refer Table 3
			1	
4	PE0	Patch Enable Channel *2n*	0	
			1	
3	IE1	Interrupt Enable Channel *2n+1*	0	
			1	
2	IE0	Interrupt Enable Channel *2n*	0	
			1	
1	I1	Interrupt Flag Channel *2n+1*	0	For Channel *2n+1*, Read: No Interrupt occurred Write: Clear Interrupt
			1	Interrupt occurred on Channel *2n+1*

Bit No.	Name	Explanation	Value	Operation
0	I0	Interrupt Flag Channel *2n*	0	For Channel *2n,* Read: No Interrupt occurred Write: Clear Interrupt
			1	Interrupt occurred on Channel *2n*
The initial values of all bits are “0”.				

The initial values of all bits are “0”.

Correspondence between PFAx, AM and AR

Table 2. PFAx, AM and AR

PFCSn_AR	PFCSn_AM	PFA2n	PFA2n+1
0	0	point	point
0	1	point	mask
1	x	start point	end point

- PFCS*n*_AR = 0 &PFCS*n*_AM = 0: PFA*2n* and PFA*2n+1* are configured to set two different addresses for the two separate match detections.
- PFCS*n*_AR = 0 & PFCS*n*_AM = 1: PFA*2n* specifies the address and PFA*2n+1* specifies the mask for single match detection. If the bit 2 of the mask (PFA*2n+1*) is 0, then bit 2 of the address (PFA*2n*) is compared with bit 2 on the address on the bus for a match detection. If the bit 2 of the mask (PFA*2n+1*) is 1, then bit 2 of the address (PFA*2n*) is not compared with bit 2 on the address on the bus.

  E.g. If PFA*0* = 0xDF0000 and PFA1 = 0x000100, then the match detection would happen if the addresses 0xDF0000 and 0xDF0100 are accessed.
- PFCS*n*_AR = 1: PFA*2n* specifies the start address and PFA*2n+1* specifies the end address for the match detection.

Channel Operation corresponding to IE and PE

Table 3. Channel Operation

IE	PE	Operation
0	0	Interrupts and Data Patch are disabled. Address Match can be detected by polling I flag.
0	1	Memory Patch is enabled. In this case the following are the possible operations: ROM Correction: If the data (constant) in Flash Memory is read-accessed by CPU/DMA, then the data in the PFDx register is read instead. RAM Data Replacement: If the data in RAM is read-accessed by CPU/DMA, then the data in the PFDx register is read instead. RAM Write Protection: If the data at particular RAM location is attempted to be written by CPU/DMA, then the write is ignored. Instruction Break: If the instruction (code) in Flash Memory/RAM is accessed by CPU and if the PFDx register is written with 0x01 (INT Instruction), then SW-INT9 ISR would be executed.
1	0	Operand Address Break is enabled and HW-INT9 is requested upon match detection. This means that if the configured memory address or range of or multiple memory addresses is / are accessed (read / written) by CPU or DMA then the HW-INT9 ISR would be executed after such access.[1]
1	1	Data Value Break enabled (available for PFD0, PFD2, PFD4, and PFD6, the other neighbored are used as Data Mask) and HW-INT9 is requested upon match detection. This means that if the configured memory or range of or multiple memory address / addresses is / are read / written by CPU or DMA with a specific / multiple / range of data then the HW-INT9 ISR would be executed after such access.

[1]It should be noted that this operation mode should not be used for instruction breaks. This is because pre-fetch mechanism of the CPU (8-Byte Instruction Queue – It fetches 8 instructions ahead of their execution). That means if the instruction break is placed at location 0xDF2000, then the break would occur as early as after execution of instruction at 0xDF1FF8 (which is placed 8 bytes before instruction at address 0xDF2000) and this is not intended.

Operation Types with CODE, DATA, CPU, and DMA

Please note that CODE/DATA and CPU/DMA are two pairs of bits. At least 1 bit has to be set to “1” for each pair for function.

Table 4. Types of Operation

CODE	DATA	CPU	DMA	Detection
0	0	0	0	No Detection
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	DMA Data Access is detected
0	1	1	0	CPU Data Access is detected
0	1	1	1	DMA and CPU Data Access is detected
1	0	0	0	No Detection
1	0	0	1	No Detection
1	0	1	0	CPU instruction code (pre-)fetch is detected
1	0	1	1	CPU instruction code (pre-)fetch is detected
1	1	0	0	No Detection
1	1	0	1	DMA Data Access is detected
1	1	1	0	CPU Data Access and CPU instruction code (pre-)fetch are detected
1	1	1	1	CPU Data Access and CPU instruction code (pre-)fetch are detected - DMA Data Access is detected
Grey entries are non-recommended settings				

Different Match Detection Conditions

The below table explains different scenarios of Match Detection configuration for channel 0 and 1:

Table 5. Match Detection Conditions – I

Sr. No.	Control (PFCS0)						Patch Address		Patch Data	
	IE0	PE0	IE1	PE1	AR	AM	PFA0	PFA1	PFD0	PFD1
1	0	1	---	---	0	0	0xDF401E	---	0x1234	---
	The above configuration achieves ROM Correction on channel 0. If the ROM address 0xDF401E is read-accessed then the CPU/DMA will read 0x1234instead of the original data. Channel 1 can be used for another Match Detection.									
2	0	1	---	---	0	0	0x006C00	---	0x5500	---
	The above configuration achieves RAM Word Data Replacement on channel 0, if READ = 1, WRITE = 0, BYTE = 0 and WORD = 1. If the RAM address 0x006C00 is read-accessed then the CPU/DMA will read 0x5500instead of the original data. Channel 1 can be used for another Match Detection.									
3	0	1	---	---	0	0	0x006C28	---	0x55XX	---
	The above configuration achieves RAM Byte Data Replacement on channel 0, READ = 1, WRITE = 0, BYTE = 1 and WORD = 0. If the RAM address 0x006C29 is read-accessed then the CPU/DMA will read 0x55instead of the original data. Channel 1 can be used for another Match Detection.									
4	0	1	---	---	0	0	0x006C00	---	---	---
	The above configuration achieves RAM Write Protection on channel 0, if READ bit is cleared to 0 and WRITE bit is set to 1. If the RAM address 0x006C00 is attempted to be written then the access would be ignored and the write would not happen. Channel 1 can be used for another Match Detection.									
5	---	---	0	1	0	0	---	0xDF40CA	---	0xXX01
	The above configuration achieves Instruction Break on channel 1. If the code at address 0xDF40CA is accessed then the SW-INT9 ISR would be executed. Channel 0 can be used for another Match Detection.									
6	0	1	0	1	0	0	0xDF4028	0xDF40F0	0x01XX	0x1234
	The above configuration achieves Instruction Break on channel 0 and Memory Patch Function on channel 1. If the code at address 0xDF4029 is accessed then the SW-INT9 ISR would be executed. If the address 0xDF40F0 is accessed then the CPU/DMA will read 0x1234.									
7	0	1	---	---	0	1	0xDF4000	0x000002	0xXX01	---
	The above configuration achieves Instruction Break with Address Mask on channel 0. If the code at address 0xDF4000 or 0xDF4002 is accessed then the SW-INT9 ISR would be executed. Channel 1 can't be used for another Match Detection.									
8	0	1	---	---	1	0	0xDF4000	0xDF407E	0xXX01	---
	The above configuration achieves Instruction Break with Address Range on channel 0. If the code at any address from 0xDF4000 to 0xDF407E is accessed then the SW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can't be used for another Match Detection.									

Sr. No.	Control (PFCS0)						Patch Address		Patch Data	
	IE0	PE0	IE1	PE1	AR	AM	PFA0	PFA1	PFD0	PFD1
9	1	0	---	---	0	0	0xDF4000	---	---	---
	The above configuration achieves Operand Address Break on channel 0. If the constant at address 0xDF4000 is read then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can be used for another Match Detection.									
	XXdenotes the original data/instruction at the adjoining memory location.									

Table 6. Match Detection Conditions - II

Sr. No.	Control (PFCS0)						Patch Address		Patch Data	
	IE0	PE0	IE1	PE1	AR	AM	PFA0	PFA1	PFD0	PFD1
10	---	---	1	0	0	0	---	0x006E00	---	---
	The above configuration achieves Operand Address Break on channel 1. If the data at address 0x006E00 is accessed then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 0 can be used for another Match Detection.									
11	1	0	1	0	0	0	0x006C56	0x006CF2	---	---
	The above configuration achieves Operand Address Break on channel 0 and 1. If the data at address 0x006C56 or 0x006CF2 is accessed then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1.									
12	1	0	---	---	0	1	0xDF4000	0x0000FF	---	---
	The above configuration achieves Operand Address Break with Address Mask on channel 0. If constant at address from 0xDF4000 to 0xDF40FE(if BYTE = 0 and WORD = 1) isread then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can't be used for another Match Detection.									
13	1	0	---	---	1	0	0x006C00	0x006CFE	---	---
	The above configuration achieves Operand Address Break with Address Range on channel 0. If data at any address from 0x006C00 to 0x006CFE is accessed then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can't be used for another Match Detection.									
14	1	1	---	---	0	0	0xDF4000	---	0x1234	0x0000
	The above configuration achieves Data Value Break on channel 0. If the constant at address 0xDF4000 is read with 0x1234 then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can be used for Operand Address Break.									
15	1	1	---	---	0	0	0x006C20	---	0x1200	0x00FF
	The above configuration achieves Data Value Break on channel 0. If the data at address 0x006C20 is read or written with any value from 0x1200 to 0x12FF, then the INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can be used for Operand Address Break.									

Sr. No.	Control (PFCS0)						Patch Address		Patch Data	
	IE0	PE0	IE1	PE1	AR	AM	PFA0	PFA1	PFD0	PFD1
16	1	1	---	---	0	0	0x006C20	---	0xFEDC	0xFFFF
	The above configuration achieves Data Value Break on channel 0. If the data at address 0x006C20 is read or written with any value then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can be used for Operand Address Break.									
17	1	1	---	---	0	1	0x006C20	0x00000F	0x1234	0x0000
	The above configuration achieves Data Value Break with Address Mask on channel 0. If data at any address between 0x006C20 to 0x006C2E is read or written with 0x1234 then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can't be used for another Match Detection.									
18	1	1	---	---	1	0	0xDF4000	0xDF407E	0xXXXX	0xFFFF
	The above configuration achieves Data Value Break with Address Range on channel 0. If constant at any address between 0xDF4000 to 0xDF407E is read with any value then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. Channel 1 can't be used for another Match Detection.									
19	1	1	1	0	0	0	0x006C20	0x006D20	0X1234	0x0000
	The above configuration achieves Data Value Break on channel 0 and Operand Address Break on channel 1. If the address 0x006C20 is read or written with 0x1234 then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1. If the address 0x006D20 is accessed then the HW-INT9 ISR would be executed if EN bit of EDSU register is set to 1.									
	XXXX denotes any 16-bit value									

2.3.4 EDSU Extension Register (EDSU)

This register controls the embedded debug support unit.

Table 7. EDSU

Bit No.	Name	Explanation	nitial Value	Value	Operation	
15	EN	INT9 Enable Bit	0	0	Disable INT9 Generation	
				1	Enable INT9 Generation	
14	-	*Unused Bit*	X	-	*No Operation*	
13	TIE	INT9 Enable for UART Transmission Interrupt	0	0	Disable HW-INT9 generation on UART Transmit Interrupt	
				1	Enable HW-INT9 generation on UART Transmit Interrupt	
12	TINT	Transmit Interrupt Flag	X	0	No Interrupt occurred	
				1	Interrupt occurred; Clearing is done via corresponding UART Interrupt Transmit flag	
11, 10	SEL1, SEL0	UART Selection for Monitor Debugger	0,0	0, 0	UART0	Used as monitor debugger communication device
				0, 1	UART1	
				1, 0	UART2	
				1, 1	UART3	
9	RIE	INT9 Enable for UART Reception Interrupt	0	0	Disable HW-INT9 generation on UART Receive Interrupt	
				1	Enable HW-INT9 generation on UART Receive Interrupt	
8	RINT	Receive Interrupt Flag	0	0	No Interrupt occurred	
				1	Interrupt occurred; Clearing is done via corresponding UART Interrupt Receive flag	

2.3.5 EDSU Extension Register 2 (EDSU2)

The EDSU extension register 2 (EDSU2) controls the extended break interrupt selection to trigger an HW-INT9 exception. For both transmit and receive interrupt channels an interrupt number can be chosen out of the complete interrupts available in the system.

Table 8. EDSU2

Bit No.	Name	Explanation	nitial Value	Value	Operation
15 - 8	TSEL	UART Transmit Channel Selection for Monitor Debugger	0	0x00	UART Transmit channel is selected by SEL1 & SEL0 of EDSU register
				0x01 – 0xFF	Reflects the UART Transmit channel number for HW-INT9 interrupt generation
7 – 0	RSEL	UART Receive Channel Selection for Monitor Debugger	0	0x00	UART Receive channel is selected by SEL1 & SEL0 of EDSU register
				0x01 – 0xFF	Reflects the UART Receive channel number for HW-INT9 interrupt generation

3 Memory Patch Function Examples

Examples for the Memory Patch Function

3.1 Data Match on (Write) Access by CPU (Data Value Break)

In this example a global variable is watched by the Memory Patch Function via the address of the variable. If a certain value is written to it, then INT9 interrupt is generated.

Main.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/
__far unsigned int data_hit;   // Declared as __far for 24 bit addressing.
                               // If declared as __near, PFAHn must be set
                               // to 0x00.
void InitDataMatch (void)
{
   // Set address
   PFAL2 = (__far unsigned long) &data_hit & 0xFF;
   PFAM2 = (__far unsigned long) &data_hit >> 8;
   PFAH2 = (__far unsigned long) &data_hit >> 16;

   // Set data to 0xAAAA
   PFDL2 = 0xAA;
   PFDH2 = 0xAA;

   // Set mask to 0x0000
   PFDL3 = 0x00;
   PFDH3 = 0x00;

   PFCS1 = 0x5614;         // Write, Word, Data, CPU, Interrupt
}

void main (void)
{
   InitIrqLevels();
   __set_il(7);            // allow all levels
   __EI();                 // globally enable interrupts

   InitDataMatch();

   EDSU = 0x80;     // enable EDSU for INT9

   data_hit = 0x5555;      // no hit here
   data_hit = 0xAAAA;      // hit here!

   while(1);
}

__interrupt void INT9_ISR (void)
{
   if (PFCS1 & 0x0001)     // Data match?
   {
      PFCS1 = 0x5614;  // clear IRQ

      // Do something ...
   }
}
```

vectors.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.     */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                         */
/*-----------------------------------------------------------------------*/
      . . .

/* ISR prototype */
__interrupt void INT9_ISR (void);

      . . .

#pragma intvect INT9_ISR 9             /* HW-INT9 of 16-FX Series */

      . . .
```

3.2 Patch on Instruction Access (Instruction Break)

In this example the instruction code at a certain address is patched to INT9 (0x01), so that this interrupt is generated. This mechanism can be used for break points during debug.

It should be noted that the address registers (PFAx) of the memory patch unit should always be configured with an appropriate even address (i.e. word-aligned) even if the actual address where the break needs to be set is an odd address. Also INT9 instruction code (0x01) isprogrammed on theaccording byte position in the data register (PFDx) and the other byte of the data registers needs to be configured with the original code at the corresponding memory location.

This means if the instruction break needs to be set at a memory location 0xFE0183, then the PFAx register needs to be configured with a value equal to 0xFE0182 and the PFDx register higher byte (PFDHx) needs to be written with a value equal to 0x01 (op-code for INT9 instruction) the PFDx register lower byte (PFDLx) needs to be written with original code byte at memory location 0xFE0182.

Main.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.       */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                           */
/*-------------------------------------------------------------------------*/
extern __far unsigned long MATCH;

void InitAddressMatch (void)
{
unsigned long matchAddress;
__far unsigned int *matchPtr;

// if the address where break needs to be set is an odd address?
if (((__far unsigned long) &MATCH) % 2 == 1)
{
    // get the word-aligned address
        matchAddress =(__far unsigned long) &MATCH - 1;
        // get the code word from word-aligned address
        matchPtr = (__far unsigned int *) matchAddress;
        // copy the lower byte of original code to PFDL register
        PFDL0 = (unsigned char)(*matchPtr);
        // 0x01 = INT9-Instruction
        PFDH0 = 0x01;
}
    else   // the address where break needs to be set is an even address
    {
        // get the word-aligned address
        matchAddress = (__far unsigned long) &MATCH;
        // get the code word from word-aligned address
        matchPtr = (__far unsigned int *) matchAddress;
        // 0x01 = INT9-Instruction
        PFDL0 = 0x01;
    // copy the higher byte of original code to PFDH register
        PFDH0 = (unsigned char)(*matchPtr >> 8);
    }
    // Set appropriate address
    PFAL0 = matchAddress & 0xFF;
        PFAM0 = matchAddress >> 8;
        PFAH0 = matchAddress >> 16;
        PFCS0 = 0xBA10;              // Read, Byte, Word, Code, CPU, Patch
}
void main (void)
{
    InitIrqLevels();
    __set_il(7);            // allow all levels
    __EI();                 // globally enable interrupts
    InitDataMatch();

        #pragma asm
                 NOP
                 NOP
          _MATCH: NOP          ; Match address here -> patched to INT9
                 NOP
NOP
        #pragma endasm

    while(1);
}
__interrupt void INT9_ISR (void)
{
        if (PFCS0 & 0x0001)     // Code address match?
        {
        PFCS0 = 0xBA10;  // clear IRQ
        // Do something ...
    }
}
```

vectors.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.      */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                          */
/*------------------------------------------------------------------------*/
   . . .

/* ISR prototype */
__interrupt void INT9_ISR (void);

   . . .

#pragma intvect INT9_ISR 9             /* INT9 of 16-FX Series */

   . . .
```

3.3 Communication Interface Interrupt configuration

In memory patch unit, USART communication interface's receive and/or transmit interrupt is able to trigger an INT9 exceptionto realize a break on request of this interface. The communication device can be selected using SEL bits of EDSU register and also using the EDSU2.

In this example, the INT9 generation on receive interrupt of USART channel 0 is enabled. For the same the RIE bit of the SSR0 register also needs to be set. The interrupt vector definition remains the same, but in the ISR the receive data register RDR0 needs to be read to clear the RDRF flag of SSR0 register and also to clear RINT flag of EDSU register.

Main.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.     */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                          */
/*------------------------------------------------------------------------*/
void InitMemPatch (void)
{
   . . .

   EDSU = 0x82;     // enable INT9, communication interface – USART
      // channel 0, INT9 generation on receive IRQ enabled
      . . .
}

voidInitUart0(void)
{
   PIER08_IE2 = 1;  // Enable UART0 RX
   DDR08_D2 = 0;    // Enable UART0 RX

   BGR0 = 1666;     // 9600 baud at 16MHz CLKP1
   SCR0 = 0x17;     // 8 bit, clear reception errors, Tx & Rx enabled

   SSR0_RIE = 1;    // Enable receive interrupt

   SMR0 = 0x0D;     // Mode 0, Reset Counter, Reset UART, SOT0 enabled
}

__interrupt void INT9_ISR (void)
{
      unsigned char rx_data;

      . . .

// read the received data to clear SSR0_RDRF &
// EDSU_RINT

      . . .
}
```

vectors.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.      */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                          */
/*------------------------------------------------------------------------*/

      . . .

/* ISR prototype */
__interrupt void INT9_ISR (void);

   . . .

#pragma intvect INT9_ISR 9             /* INT9 of 16-FX Series */

   . . .
```

3.4 Data Match on (Write) Access by DMA (Data Value Break)

In this example an element global array is watched by the Memory Patch Function via the address of the variable. If it is written with any value, then HW-INT9 interrupt is generated.

Main.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.     */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                         */
/*-----------------------------------------------------------------------*/

__far unsigned char data_hit[50];   // Declared as __far for 24 bit addressing.
                                    // If declared as __near, PFAHn must be set
                                    //   to 0x00.
void InitDataMatch (void)
{
    // Set address
    PFAL2 = (__far unsigned long) &data_hit[10] & 0xFF;
    PFAM2 = (__far unsigned long) &data_hit[10] >> 8;
    PFAH2 = (__far unsigned long) &data_hit[10] >> 16;

    // Set mask to 0xFFFF, this ensures the break would happen once any value
       // is written to the required memory location
    PFDL3 = 0xFF;
    PFDH3 = 0xFF;

    PFCS1 = 0x6514;         // Write, Byte, Data, DMA, Interrupt
}

voidInitUart0(void)
{
    PIER08_IE2 = 1;  // Enable UART0 RX
    DDR08_D2 = 0;    // Enable UART0 RX

    BGR0 = 1666;     // 9600 baud at 16MHz CLKP1
    SCR0 = 0x17;     // 8 bit, clear reception errors, Tx & Rx enabled

    ESIR0 = 0x01;    // Disable USART automatic interrupt clear,
              // Clear TDRE and RDRF flags

    SSR0_RIE = 1;    // Enable receive interrupt

    SMR0 = 0x0D;     // Mode 0, Reset Counter, Reset UART, SOT0 enabled
}

void InitDma(void)
{
    unsigned long temp;
    DISEL0 = 79;// UART0 receive interrupt number for MB9634x Series

    DSR = 0x0000;// Clear transfer end interrupt, if any
    DER = 0x0001;// DMA 0 enable
```

Here the DMA is configured to transfer the data from receive buffer RDR0 of UART0 to the array data_hit[] after every receive interrupt. Once the 10th byte is written to 10th element data_hit[] array, the memory patch function identifies this write and then the INT9 interrupt is generated.

```
    DCT0 = 0x0032;                  // 50 Bytes
    IOA0 = (unsigned int)&RDR0;   //Source
    temp = (unsigned long)&data_hit;     //Destination
    BAPL0 = (unsigned char) temp;
    temp >>= 8;
    BAPM0 = (unsigned char) temp;
    temp >>= 8;
    BAPH0 = (unsigned char) temp;

    DMACS0 = 0x10; // no IOA update, BAP update-Increment, byte transfer,
    // IOA -> BAP
}

void main (void)
{
    InitIrqLevels();
    __set_il(7);            // allow all levels
    __EI();                 // globally enable interrupts

    InitUart0();
    InitDma();
    InitDataMatch();

    EDSU = 0x80;     // enable EDSU for INT9

    while(1);
}

__interrupt void INT9_ISR (void)
{
    if (PFCS1 & 0x0001)     // Data match?
    {
        PFCS1 = 0x5614;  // clear IRQ

        // Do something ...
    }
}

__interruptvoid UART0_RXISR (void)
{
    DSR = 0x0000;    // Clear DMA end request
    ESIR0_RDRF = 0;  // Clear RDRF flags
    SCR0_CRE = 1;    // Clear all error flags

    // Do something ...
}
```

vectors.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.      */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                          */
/*------------------------------------------------------------------------*/
void InitIrqLevels (void)
{
       . . .

   ICR = (79 << 8) | 2;    /* Priority Level 2 forUART0-Rx of MB9634x
      Series */
   . . .
}

/* ISR prototype */
__interrupt void INT9_ISR (void);
__interrupt voidUART0_RXISR (void);

   . . .

#pragma intvect INT9_ISR         9      /* INT9 of MB96340 Series*/
#pragma intvect UART0_RXISR      79     /* UART0-Rx of MB96340 Series */
   . . .
```

4 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

96340_mempatch

96340_mempatch_dma

It can be found on the following Internet page:

http://www.cypress.com/products/16FX

5 Document History

Document Title: AN204770 - F²MC-16FX Family, Memory Patch Function

Document Number:002-04770

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	08/16/2006	Initial Release
			07/13/2007	Reviewed the document and updated with review findings
			08/20/2007	Modified channel operation, match detection conditions and instruction break example
*A	5056555	MKEA	12/22/2015	Migrated Spansion Application Note from MCU-AN-300221-E-V12 to Cypress format
*B	5836277	AESATMP8	07/28/2017	Updated logo and Copyright.
*C	6036665	NOFL	01/18/2018	Updated logo. Updated links. Updated Sales page and Copyright year.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2006-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.