The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205271

F²MC - 8FX Family, MB95200H/210H Series, Basic FW Setup

There are three methods to design a keyboard function using the MB95200 series MCU: external interrupt, AD and This application note describes how to start up the basic FW setting of the Cypress F2MC-8FX Family MB95200H/210H Series. This application note describes the initializing routine, the stack pointer (SP) and direct bank pointer (DP) setting, the clock control and the standby control.

Contents

1 Introduction....1
2 Initializing Routine....2
2.1 Initialize the Stack....2
2.2 SP and DP Setting....2
3 Interrupt Vectors....3
3.1 Interrupt Levels....3
3.2 Interrupt Handler Function Prototypes....3
3.3 Vector Definition....3
3.4 Default Interrupt Handler....4
3.5 Examples....4
4 Clock Controller....5
4.1 Overview of the Clock Controller....5
4.2 Block Diagram of the Clock Controller....5
4.3 Register....6
5 Clock Modes Selection....7
5.1 Introduction....7
5.2 Key Features....7
5.3 Main Clock....7
5.4 Sub-clock....8
5.5 Main CR Clock....9
5.6 Sub-CR Clock....10
5.7 Clock Divide Ratio....10
6 Standby Modes Selection....11
6.1 Key Features....11
6.2 Standby Modes and Clock Supply States....11
6.3 Combinations of Clock Mode and Standby Mode....12
6.4 Sleep Mode....13
6.5 Stop Mode....13
6.6 Time base Timer Mode....14
6.7 Watch Mode....15
6.8 Standby Mode Transition Code....16
7 Appendix....17
7.1 Sample Code....17
8 Additional Information....30
Document History....31

1 Introduction

There are three methods to design a keyboard function using the MB95200 series MCU: external interrupt, AD and This application note describes how to start up the basic FW setting of the Cypress F^{2}MC-8FX Family MB95200H/210H Series.

This application note describes the initializing routine, the stack pointer (SP) and direct bank pointer (DP) setting, the clock control and the standby control.

2 Initializing Routine

The start up file startup.asm is the entry point just after a reset is performed. It contains several settings for the compiler and the device operation. It initializes the stack and variable initial values.

2.1 Initialize the Stack

When the MCU is started up, initialize the stack is pre-requisite, and set the stack size and the definition to the stack area.

Sample code to initialize:

```
//Sample code for initialization
//definition to stack area
     .SECTION       STACK,    STACK,    ALIGN=1
     .RES.B  128-2
STACK_TOP:
     .RES.B  2
```

Note: The above-mentioned code is intercepted from the start.asm. Please refer to Appendix.

2.2 SP and DP Setting

SP and DP are dedicated registers.

The stack pointer (SP) is a 16-bit register which holds the address referenced when an interrupt or a sub-routine call occurs by the stack push and pop instructions. The initial value after a reset is "0000H".

DP represents the direct bank pointer. DP in bits 10 to 8 of the program status (PS) register specifies the area to be accessed by a direct addressing.

Sample code to initialize:

```
      .SECTION CODE,CODE,ALIGN=1
;-------------------------------------------
;set stack pointer
;-------------------------------------------
     MOVW          A , #STACK_TOP
     MOVW    SP, A
// Set Register bank Pointer 0 / set Direct bank Pointer 0 (0x80...0xFF) //
The PS register consists of the register bank pointer (RP), direct      //
pointer (DP), and condition code register (CCR).
      MOVW         A, PS
      MOVW         A, #0x07FF    // RP=0, DP=0, I=0
      ANDW         A
      MOVW         PS, A
```

Note: The above-mentioned code is intercepted from the start.asm.

3 Interrupt Vectors

When the basic FW setting is started, initialization of the interrupt vectors is necessary.

C file vector.c is available for users. In this C file, the interrupt levels and numbers are declared and it contains the default interrupt handler. Corresponding interrupt control registers may be initialized as well.

3.1 Interrupt Levels

In the file vector.c, the interrupt level function void InitIrqLevels (void) is defined firstly:

```
void InitIrqLevels (void)
{
     ILR0 = 0Xff    // IRQ0:      external interrupt ch.4          --> 01
                     // IRQ1:     external interrupt ch.5          --> 01
                     // IRQ2:     external interrupt ch.2/ch.6     --> 01
                     // IRQ3:     external interrupt ch.3/ch.7     --> 01
    ......
}
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

This function initializes the interrupt control register, which defines the interrupt levels for each shared interrupt channel. FF indicates the lowest priority.

3.2 Interrupt Handler Function Prototypes

The function prototype of the interrupt handler shall be declared after the level initialization:

```
__interrupt void DefaultIRQHandler (void)
// Add your own prototypes here like above.
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

3.3 Vector Definition

The vector numbers are linked with the interrupt handler functions. For unused interrupts, please use the default interrupt handler:

```
#pragma   intvect  DefaultIRQHandler 0   //IRQ0: external interrupt ch.4
#pragma   intvect  DefaultIRQHandler 1   //IRQ1: external interrupt ch.5
#pragma   intvect  DefaultIRQHandler 2
                                        //IRQ2: external interrupt ch.2/ch.6
#pragma   intvect  DefaultIRQHandler 3
                                        //IRQ3: external interrupt ch.3/ch.7

... ...
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

3.4 Default Interrupt Handler

The default interrupt handler function is listed next to the vector.c file. This function performs a system halt with an endless loop. For debugging purpose, it is quite useful to set a breakpoint here to detect uninitialized interrupts.

Sample code for default Interrupt Handler:

```
//default interrupt handler
__interrupt void DefaultIRQHandler (void)
{
     __DI ();              // disable interrupts
     While (1)
     __wait_nop ();        // halt system
}
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

3.5 Examples

Where modification of the interrupt vectors is necessary in the use of the interrupt, for example, set the external interrupt ch4 as key_in, please modify the code in vector.c.

```
#pragma intvect Key_in      1    // IRQ0:  external interrupt ch4
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

Definition of the interrupt levels:

```
// defines the interrupt levels
void InitIrqLevels (void)
{
     ILR0=0xFC      // IRQ0:  external interrupt   ch.4            --> Level 00
                    // IRQ1:  external interrupt   ch.5            --> 01
                    // IRQ2:  external interrupt ch.2 | ch.6    --> 01
               // IRQ3:  external interrupt ch.3 | ch.7       --> 01
    ......    // elide the following code
}
```

Note: The above-mentioned code is intercepted from the vector.c. Please refer to Appendix.

In the file main.c, enter your interrupt handler function as __interrupt void Key_Int (void).

```
//interrupt response function
_interrupt void Key_int (void)   // key_int external interrupt ch.4
{
          // enter your interrupt handler function(s) here
}
```

In

Note: The above-mentioned code is intercepted from the main.c. Please refer to Appendix.

4 Clock Controller

Chapter 4 describes the functions and operations of the clock controller.

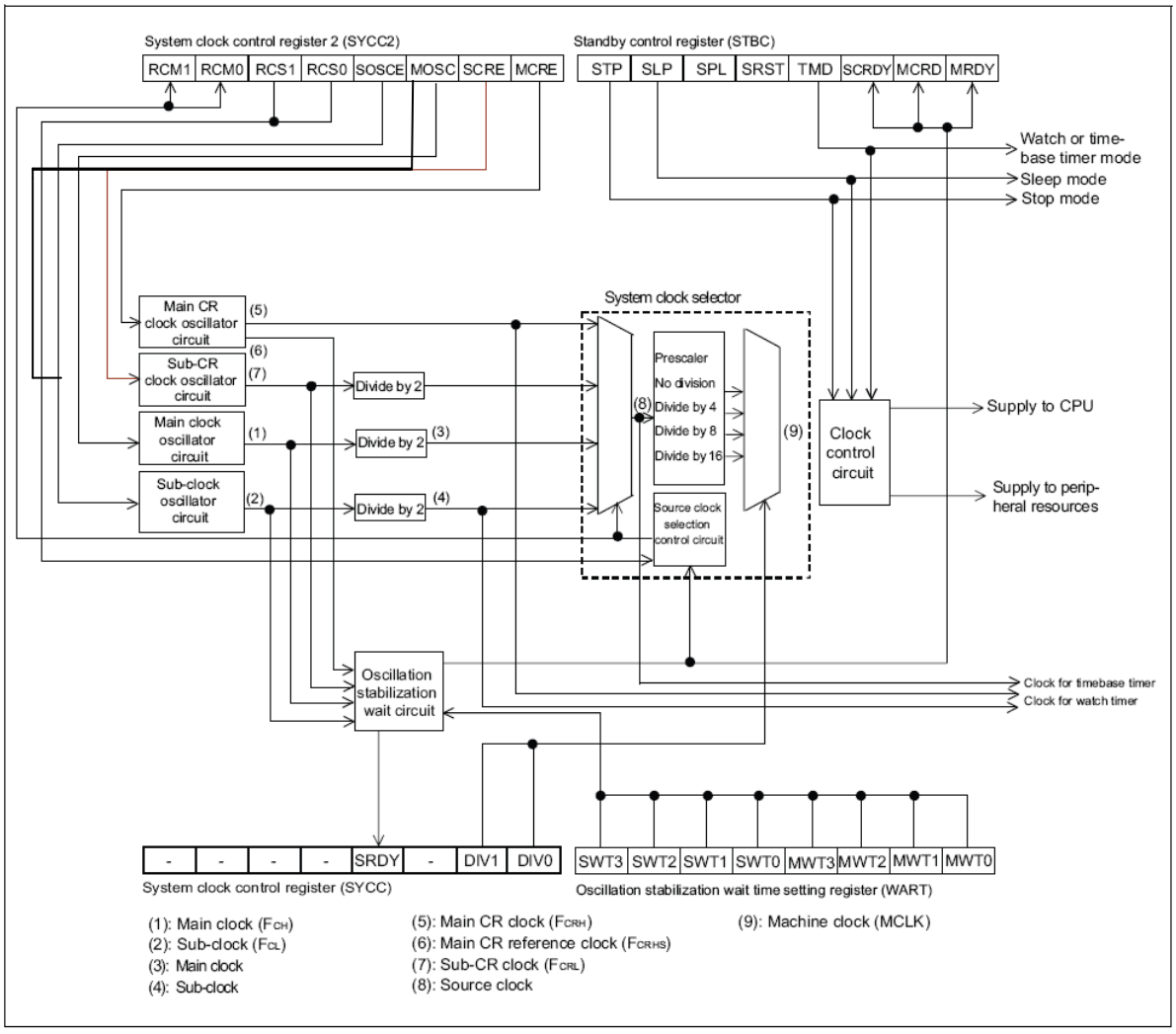
4.1 Overview of the Clock Controller

The F²MC-8FX family has a built-in clock controller that optimizes its power consumption. It includes a dual external clock product supporting both of the external main clock and the external sub-clock, and a single external clock product only supporting the external main clock.

The clock controller enables/disables the clock oscillation, enables/disables the supply of clock signals to the internal circuitry, selects the clock source, and controls the internal CR oscillator and frequency divider circuits.

4.2 Block Diagram of the Clock Controller

Figure 1. Clock Controller Block Diagram



4.3 Register

The clock controller consists of four registers: System Clock Control Register (SYCC), Standby Control Register (STBC), System clock Control Register 2 (SYCC2) and Oscillation Stabilization Wait Time Setting Register (WATR).

Please refer to MCU MB95200H/210H Series HWM Chapter 6 for the details.

Figure 2 shows the configuration of System Clock Control Register (SYCC). This register is used to control machine clock divide ratio selection.

Figure 2. Configuration of System Clock Control Register (SYCC)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0007_H	-	-	-	-	SRDY	-	DIV1	DIV0	$0000X011_B$
	R0/WX	R0/WX	R0/WX	R0/WX	R/WX	R0/WX	R/W	R/W	

Figure 3 shows the Configuration of Oscillation Stabilization Wait Time Setting Register (WATR).This register is used to set the oscillation stabilization wait time.

Figure 3. Configuration of Oscillation Stabilization Wait Time Setting Register (WATR)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0005_H	SWT3	SWT2	SWT1	SWT0	MWT3	MWT2	MWT1	MWT0	11111111_B
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Figure 4 shows the Standby Control Register (STBC). This register is used to control transition from the RUN state to a sleep mode, a stop mode, a time base timer mode, or a watch mode, set the pin state in a stop mode, a time base timer mode, and a watch mode, and to control the generation of a software reset.

Figure 4. Standby Control Register (STBC

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0008_H	STP	SLP	SPL	SRST	TMD	SCRDY	MCRDY	MRDY	$00000XXX_B$
	R0/W	R0/W	R/W	R0/W	R0/W	R/WX	R/WX	R/WX	

Figure 5 shows the Configuration of System Clock Control Register 2 (SYCC2).This register is used to indicate and switch current clock mode, and control sub-clock, main clock, sub-CR clock, main CR clock, and main clock oscillation.

Figure 5. Configuration of System Clock Control Register 2 (SYCC2)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
$000D_H$	RCM1	RCM0	RCS1	RCS0	SOSCE	MOSCE	SCRE	MCRE	10100011_B
	R/WX	R/WX	R/W	R/W	R/W	R/W	R/W	R/W	

R/W: Readable/writeable (Read value is the same as write value)

R/WX: Read only (Readable, writing has no effect on operation)

R0/WX: Undefined bit (Read value is “0”, writing has no effect on operation)

R0/W: Write only (Writable, “0” is read)

5 Clock Modes Selection

5.1 Introduction

Four clocks (main clock, sub-clock, main CR clock and sub-CR clock) are available here bearing various division selections. (Single external clock products only have 3 clocks)

5.2 Key Features

- Selectable Main Clock Source
  - ☐ External main clock (The frequency is up to 32.5 MHz and maximum machine clock frequency is 16.25 MHz)
  - ☐ Main-internal CR clock (1/8/10 MHz)
- Selectable Sub-clock Source
  - ☐ External sub-clock (32.768 kHz)
  - ☐ Sub-OSC clock (32.768 kHz)
  - ☐ Sub-internal CR clock (Typ. 100 kHz, Min. 50 kHz, Max. 200 kHz)
- Initial clock is Main CR clock (8 MHz).

5.3 Main Clock

Main clock is the external main clock. The maximum frequency of the external clock is up to 32.5 MHz, while the maximum machine clock frequency is 16.25MHz. Before using the main clock, make sure the SYSC_PFSEL bit is “0”. This bit is used to select the function of the PF1/PF2 pins. If this bit is set to “0”, the PF1/PF2 pins are selected as the main clock oscillator pins.

Note: The above-mentioned PFSEL setting refer to the Chapter 23 in Hardware Manual.

The SYCC2_MOSCE can be updated to enable the main clock, update the SYCC2_RCS to select the main clock mode, update the WATR to change the Oscillation Stabilization Wait Time and check the STBC_MRDY to indicate main clock oscillation being stable.

If the WATR is set to 0x0F, the Oscillation Stabilization Wait Time is $(2^{14}-2)/F_{CH}$, about 4.10ms.

The following code shows how to operate the main clock, it includes enable the main clock, change to main clock mode, update the Oscillation Stabilization Wait Time and wait for the oscillation stabilization.

```
// enable the Main clock oscillation
SYCC2_MOSCE = 1;
// Clock Mode Selection, select the main clock mode
SYCC2_RCS0 = 0x01;
SYCC2_RCS1 = 0x01;
//update Oscillation Stabilization Wait Time
//when Main Oscillation Clock F_CH=4MHz
WATR = 0X0F;        // Oscillation Stabilization Wait Time mean
                     // (2^14-2)/F_CH About 4.10 ms
While (!STBC_MRDY); //Indicates main clock oscillation being stable
```

Note: The above-mentioned code is intercepted from the project of Clock_Change. Please refer to Appendix.

5.4 Sub-clock

Sub-clock is the external sub-clock. The maximum frequency of the external sub-clock is 32.768 kHz, correspondingly, the sub-clock frequency is 32.768 kHz. Before using the sub clock, make sure the SYSC_PGSEL bit is “0”. This bit is used to select the function of the PG1/PG2 pins. If this bit is set to “0”, the PG1/PG2 pins are selected as the sub clock oscillator pins.

Note: The above-mentioned PGSEL setting refer to the Chapter 23 in Hardware Manual.

The SYCC2_SOSCE can be updated to enable the sub-clock, update the SYCC2_RCS to select the sub-clock mode, update the WATR to change the Oscillation Stabilization Wait Time and check the SYCC_SRDY to indicate main clock oscillation being stable.

If the WATR is set to 0xF0, the Oscillation Stabilization Wait Time is $(2^{15}-2)/F_{CL}$, about 1.00 s.

The following code shows how to operate the sub-clock, it includes how to enable the sub-clock, change to a sub-clock mode, update the Oscillation Stabilization Wait Time and wait for the oscillation stabilization.

```
// enable the sub-clock oscillation
SYCC2_ SOSCE = 1;
// Clock Mode Selection, select the sub-clock mode
SYCC2_RCS0 = 0x01;
SYCC2_RCS1 = 0x00;
//Update the Oscillation Stabilization Wait Time
// when sub Oscillation Clock FCL=32.768 kHz
WATR = 0xF0;        //(2^15-2)/FCL About 1.00s
while (!SYCC_SRDY); //Indicates sub-clock oscillation being stable
```

Note: The above-mentioned code is intercepted from the project of Clock_Change. Please refer to Appendix.

5.5 Main CR Clock

The main CR clock is the main-internal CR clock bearing three frequency clocks: 1 MHz, 8 MHz, 10 MHz. The update of the NVR can change the main CR clock.

The SYCC2_MCRE can be updated to enable the Main CR Clock, update the SYCC2_RCS to select the Main CR Clock mode and check the STBC_MCRDY to indicate the main CR clock oscillation being stable.

The following code shows how to operate the main CR clock, it includes how to change the main CR clock frequency, enable the main CR clock, change to the main CR clock mode, update the Oscillation Stabilization Wait Time and wait for the oscillation stabilization.

```
;change the main CR clock frequency
MOV    A ,    0x0FE4        ;READ CRTH for NVR trimming value protect
AND    A ,    #0x9F         ;Just CLEAR CRTH [6:5]
OR     A ,  #0x60           ; Main CR clock update to 8MHz
;Can update the Main CR clock as below setting
;0x00: 1MHz;  0x40: 10MHz;   0x60: 8MHz
MOV    0x0FE4 , A           ;WRITE CRTH to enable the update
```

Note: The above-mentioned code is intercepted from the project1 start.asm. Please refer to Appendix.

```
// enable the main CR clock oscillation
SYCC2_ MCRE = 1;
// Clock Mode Selection, select the main CR clock mode
SYCC2_RCS0 = 0x00;
SYCC2_RCS1 = 0x01;
//update the Oscillation Stabilization Wait Time
//F_CRHS represents the main clock frequency
//2^8/F_CRHS = Main CR oscillation Stabilization Wait Time
while(!STBC_MCRDY);//Indicates main CR clock oscillation being stable
```

Note: The above-mentioned code is intercepted from the project of Clock_Change. The oscillation stabilization wait time changes according to the CR start up timing.

The Main CR oscillation Stabilization Wait Time = $2^8/F_{CRHS}$. F_{CRHS} represents the main CR clock frequency.

5.6 Sub-CR Clock

The Sub-CR clock is a single internal clock with the frequency of its typical clock is 100 kHz, the minimum frequency is 50 kHz and the maximum frequency is 200 kHz.

The SYCC2_SCRE can be updated to enable the sub-CR clock, update the SYCC2_RCS to select the sub-CR clock mode and check the STBC_SCRDY to indicate sub-CR clock oscillation being stable.

The following code shows how to operate the sub-CR clock, it includes how to enable the sub-CR clock, change to sub-CR clock mode, update the Oscillation Stabilization Wait Time and wait for the oscillation stabilization.

```
// enable the Sub-CR Clock oscillation
SYCC2_ SCRE = 1;
// Clock Mode Selection, select the Sub-CR Clock mode
SYCC2_RCS = 0x00;
//update the Oscillation Stabilization Wait Time
//FCRHS represents the main clock frequency
//2^4/FCRL = Sub-CR oscillation Stabilization Wait Time
while (!STBC_SCRDY); //Indicates sub-CR clock oscillation being stable
```

Note: The above-mentioned code is intercepted from the project of Clock_Change. The oscillation stabilization wait time changes according to the timing of the CR start up. Main CR oscillation Stabilization Wait Time = $2^4/F_{CRL}$. F_{CRL} represents the sub-CR clock frequency.

5.7 Clock Divide Ratio

The machine clock is generated from the source clock according to the divide ratio. The SYCC_DIV can be set to divide the source clock. If SYCC_DIV is 0x11 is set, the source clock will be divided by 16.

The following code shows how to set the clock divide ratio:

```
//source clock can be main clock divided by 2, sub-clock divided by 2
//sub-CR clock divided by 2 or main CR clock no div
SYCC_DIV = 0x00;    // Source clock (No division)
                     // SYCC_DIV = 0x01 mean Source clock/4
                     // SYCC_DIV = 0x02 mean Source clock/8
                     // SYCC_DIV = 0x03 mean Source clock/16
```

Note: The above-mentioned code is intercepted from the project of Clock_Change.

6 Standby Modes Selection

The standby modes consist of four modes: sleep mode, time base timer mode, watch mode, stop mode. The clock controller selects whether to enable or disable the clock oscillation and clock supply to internal circuitry depending on each standby mode.

With the exception of the time base timer mode and the watch mode, the standby mode can be set independent of the clock mode.

6.1 Key Features

- Low-power Consumption Modes (standby modes)
  - ☐ Stop mode
  - ☐ Sleep mode
  - ☐ Watch mode
  - ☐ Time base timer mode
- Initial mode
  - ☐ Normal operating status

6.2 Standby Modes and Clock Supply States

Table 1. Standby Modes and Clock Supply States

Standby Modes	Clock Supply States
Sleep mode	Stops the clock supply to the CPU. As a result, the CPU stops operating, but other peripheral resources continue operating.
Time base timer mode	Supplies clock signals only to the time base timer, the watch prescalar while the clock supply stops supplying to other circuits. As a result, all the functions other than the time base timer, the watch prescalar, the external interrupt, and the low-voltage detection reset (optional) are stopped. The Time base timer mode is the only standby mode for main clock mode or main CR clock mode.
Watch mode	Stops the main clock oscillation, but supplies clock signals only to the watch prescaler while the clock supply stops supplying to other circuits. As a result, all the functions other than the watch prescaler, the external interrupt, and the low-voltage detection reset (optional) are stopped. The watch mode is the only standby mode for a sub-clock mode or a sub-CR clock mode.
Stop mode	Stops main clock oscillation. sub-clock oscillation and the supply of all clock signals. As a result, all the functions other than external interrupt and the low-voltage detection reset (option) are stopped.

6.3 Combinations of Clock Mode and Standby Mode

Table 2 lists the combinations of clock mode and standby mode and their respective operating states of internal circuits.

Please refer to MCU MB95200H/210H Series HWM Chapter 6 for details.

Table 2. Combinations of Standby Mode and Clock Mode and Internal Operating States

Function	RUN				Sleep				Timebase Timer		Watch prescaler		Stop	
	Main clock mode	Main CR clock mode	Sub-clock mode (Dual-system clock product)	Sub-CR clock mode	Main clock mode	Main CR clock mode	Sub-clock mode (Dual-system clock product)	Sub-CR clock mode	Main clock mode	Main CR clock mode	Sub-clock mode (Dual-system clock product)	Sub-CR clock mode	Main CR clock mode	Sub-CR clock mode
Main clock	Operating	Stopped*1	Stopped		Operating	Stopped*1	Stopped		Operating	Stopped*1	Stopped		Stopped	
Main CR clock	Stopped*2	Operating	Stopped		Stopped*2	Operating	Stopped		Stopped	Operating	Stopped		Stopped	
Sub-clock	Operating*3		Operating		Operating*3		Operating		Operating*3		Operating		Operating*3	Stopped
Sub-CR clock	Stopped*4		Stopped*4	Operating	Stopped*4		Stopped*4	Operating	Stopped*4		Stopped*4	Operating	Stopped*4	Stopped
CPU	Operating		Operating		Stopped		Stopped		Stopped		Stopped		Stopped	
ROM	Operating		Operating		Value held		Value held		Value held		Value held		Value held	
RAM														
I/O ports	Operating		Operating		Output held		Output held		Output held		Output held		Output held	
Timebase timer	Operating		Stopped		Operating		Stopped		Operating		Stopped		Stopped	
Watch prescaler	Operating*3		Operating		Operating*3		Operating		Operating*3		Operating		Operating*3	Stopped
External interrupt	Operating		Operating		Operating		Operating		Operating		Operating		Operating	
Hardware watchdog timer	Operating		Operating		Operating		Operating		Operating		Operating		Operating	
Software watchdog timer	Operating		Operating		Stopped		Stopped		Stopped		Stopped		Stopped	
Low-voltage detection reset	Operating		Operating		Operating		Operating		Operating		Operating		Operating	
Other peripheral resources	Operating		Operating		Operating		Operating		Stopped		Stopped		Stopped	

Note: Set the standby mode after the transition to the clock mode has completed.

6.4 Sleep Mode

6.4.1 Introduction

The sleep mode stops the operations of the CPU and the software watchdog timer.

6.4.2 Operations in the Sleep Mode

The sleep mode stops the operating clock for the CPU and the software watchdog timer. In this mode, the CPU stops while retaining the contents of registers and RAM that exist immediately before the transition to the sleep mode, but the peripheral resources except for the watchdog timer continue operating.

6.4.3 Transition and Cancellation

Writing "1" to the sleep bit in the standby control register (STBC: SLP) causes the device to enter the sleep mode.

The following code is the setting of the STBC from the RUN state to the sleep mode:

```
STBC_SLP = 1;              // Causes transition to sleep mode
```

Note: The above-mentioned code is intercepted from the project of Mode_Change.

A reset or an interrupt from a peripheral resource releases the device from the sleep mode.

6.5 Stop Mode

6.5.1 Introduction

The stop mode stops the main clock.

6.5.2 Operations in the Stop Mode

The stop mode stops the main clock, main CR clock, and sub-clock. In this mode, the device stops all the functions except for the external interrupt and the low-voltage detection reset while retaining the contents of registers and RAM that exist immediately before the transition to the stop mode.

6.5.3 Transition and Cancellation

The following code is the setting of the STBC from the RUN state to the sleep mode:

Writing "1" to the stop bit in the standby control register (STBC: STP) causes the device to enter the stop mode. At this time, the states of external pins are retained when the pin state setting bit in the standby control register (STBC: SPL) is "0", and the states of external pins become high impedance when that bit is "1".

The following code shows how to transit to a stop mode:

```
STBC_STP = 1;              //Causes transition to stop mode
```

Note: The above-mentioned code is intercepted from the project of Mode_Change.

The device is released from the stop mode in response to a reset or an external interrupt. After the recovery from the stop mode, initialize each peripheral resource as necessary.

6.6 Time base Timer Mode

6.6.1 Introduction

The time base timer mode only allows the main clock oscillation, the sub-clock oscillation, the time base timer, and the watch prescalar to work. The operating clock for the CPU and peripheral resources are stopped in this mode.

6.6.2 Operations in the Time base Timer Mode

In time base timer mode, the main clock supply is stopped except for the time base timer. The device stops all the functions except the time base timer, the external interrupt and the low-voltage detection reset while retaining the contents of registers and RAM that exist immediately before the transition to the time base timer mode.

6.6.3 Transition and Cancellation

Transition to the time base timer mode: the SYCC2_RCS is set to 0x02 or 0x03 and "1" is written to the watch bit in the standby control register (STBC: TMD).

The following code shows how to transit to a time base timer mode:

```
SYCC2_RCS0 = 0x00;
SYCC2_RCS1 = 0x01;
STBC_TMD = 1;              // Causes the device to enter time base timer mode
```

Note: The above-mentioned code is intercepted from the project of Mode_Change.

The cancellation of the time base timer mode: the device is released from the time base timer mode in response to a reset, time base timer interrupt, or an external interrupt. After the recovery from the time base timer mode, initialize each peripheral resource as necessary.

6.7 Watch Mode

6.7.1 Introduction

The watch mode only allows the sub-clock, the sub-CR clock and the watch prescaler to work. The operating clock for the CPU and peripheral resources is stopped in this mode.

6.7.2 Operations in the Watch Mode

In the watch mode, the operating clock for the CPU and peripheral resources is stopped. The device stops all the functions except for the watch prescaler, the external interrupt, and the low-voltage detection reset while retaining the contents of registers and RAM that exist immediately before the transition to the watch mode.

6.7.3 Transition and Cancellation

Transition to the watch mode: set SYCC2_RCS to 0x00 or 0x01 and write "1" to the watch bit in the standby control register (STBC: TMD).

The device can enter a watch mode only when the clock mode is a sub-clock mode or a sub-CR clock mode.

```
SYCC2_RCS0 = Ox00;
SYCC2_RCS1 = 0x00;
STBC_TMD = 1;              // Causes the device to enter watch mode
```

Note: The above-mentioned code is intercepted from the project of Mode_Change.

Cancellation of the watch mode: the device is released from the watch mode in response to a reset, a watch interrupt, or an external interrupt. After the recovery from the watch mode, initialize each peripheral resource as necessary.

6.8 Standby Mode Transition Code

The following code can realize the transition between standby modes, such as a stop mode, a watch mode. The switch mode can be updated to the mode what you want.

```
//The following code can realize the transition of the standby mode from //one
to another mode, such as a stop mode, a watch mode
//The following software is for demonstration purpose only
#define switchmode  normal //set transition to some standby modes
// Can define the switchmode to Stop, Sleep, Watch or Timebase
void main (void)
{
      MCU_initialization();
      while(1)
        {      // choice standby mode what you want transition to
            switch (switchmode)
               {
                      case normal:  break;
                      case stop:    STBC_STP = 1;
                                    break;                //normal to stop mode
                      case sleep:   STBC_SLP = 1;
                                    break;                //normal to sleep mode
                      case watch:   SYCC2_SOSCE = 1;      //Enable Sub Clock
                                    SYCC2_RCS0 = 0x01;    //Select Sub Clock
                                    SYCC2_RCS1 = 0x00;    //Select Sub Clock
                                    WATR = 0xF0;          //About 1.00s Wait Time
               while (!SYCC_SRDY); //Wait Clock stable
                                    STBC_TMD = 1;
                                    break;                //normal to watch mode
                      case Timebase: SYCC2_MOSCE = 1;     //Enable Main Clock
                                       SYCC2_RCS0 = 0x01;       //Select Main Clock
                                       SYCC2_RCS1 = 0x01;
                                    WATR = 0X0F;          // About 4.10ms Wait Time
               while (!STBC_MRDY); //Wait Clock stable
                                    STBC_TMD = 1;
                                    break;         //normal to timebase time mode
                         default: break;
               }
      vDelay(2000);                                       //delay the time
      led_display();                                      //lighten three LED
      }
}
```

Note: The above-mentioned code is intercepted from the project of Mode_Change. Please refer to Appendix.

7 Appendix

7.1 Sample Code

7.1.1 Project 1 Name: Basic_Initialization

NAME: Start.asm
FUNCTION: Initialization MCU

```
;===============================================================================
; F2MC-8FX Family SOFTUNE C Compiler sample startup routine,
;===============================================================================
; Sample code for initialization
;-------------------------------------------------------------------------------
.PROGRAM      start
           .TITLE    start
;-------------------------------------------------------------------------------
; variable define declaration
; #define HWD_DISABLE
; if define this, Hard Watchdog will disable.
; external declaration of symbols
;-------------------------------------------------------------------------------
           .EXPORT   __start
              .IMPORT_main
              .IMPORTLMEMTOMEM
              .IMPORTLMEMCLEAR
              .IMPORT_RAM_INIT
              .IMPORT_ROM_INIT
              .IMPORT_RAM_DIRINIT
              .IMPORT_ROM_DIRINIT
;-------------------------------------------------------------------------------
; definition to stack area
;-------------------------------------------------------------------------------
              .SECTION      STACK, STACK, ALIGN=1
              .RES.B 128-2
STACK_TOP:
           .RES.B    2
;-------------------------------------------------------------------------------
; definition to start address of data, const and code section
;-------------------------------------------------------------------------------
           .SECTION  DIRDATA,      DIR,          ALIGN=1
           .SECTION  DIRINIT,      DIR,          ALIGN=1
           .SECTION  DATA,  DATA,  ALIGN=1
           .SECTION  INIT,  DATA,  ALIGN=1
;-------------------------------------------------------------------------------
; The Mode Byte is defined at the beginning of the start.asm
;-------------------------------------------------------------------------------
              .SECTION      RESVECT,      CONST, LOCATE=H'FFFD
       .DATA.B0
       .DATA.W__start
;-------------------------------------------------------------------------------
//code area
;-------------------------------------------------------------------------------
              .SECTION      CODE,  CODE,  ALIGN=1
__start:
;-------------------------------------------------------------------------------
; set stack pointer
;-------------------------------------------------------------------------------
           MOVW             A,     #STACK_TOP
           MOVW      SP,    A
```

```
;------------------------------------------------------------------------
; Set Register bank Pointer 0
;------------------------------------------------------------------------
              MOVW          A,      PS
              MOVW          A,      #0x07FF       // RP=0, DP=0, I=0
              ANDW          A
              MOVW          PS,     A
;------------------------------------------------------------------------
; Set ILM to the lowest level (3)
;------------------------------------------------------------------------
              MOVW          A,      PS
              MOVW          A,      #0x0030
              ORW           A
              MOVW          PS,     A
;change the main CR clock frequency
              MOV     A ,     0x0FE4        ;READ CRTH for NVR trimming value protect
              AND     A ,     #0x9F                 ;Just CLEAR CRTH [6:5]
              OR      A ,      #0x60                ; Main CR clock update to 8MHz
;Can update the Main CR clock as below setting
;0x00: 1MHz;   0x40: 10MHz;    0x60: 8MHz
              MOV     0x0FE4 , A                    ;WRITE CRTH to enable the update
;------------------------------------------------------------------------
; copy initial value *CONST (ROM) section to *INIT(RAM) section
;------------------------------------------------------------------------
#macro ICOPY  src_addr,     dest_addr,    src_section
              MOVW          EP,     #\src_addr
              MOVW          A,      #\dest_addr
              MOVW          A,      #SIZEOF(\src_section)
           CALL             LMEMTOMEM
#endm
           ICOPY     _ROM_INIT,     _RAM_INIT,     INIT
              ICOPY  _ROM_DIRINIT, _RAM_DIRINIT, DIRINIT
;------------------------------------------------------------------------
; zero clear of *VAR section
;------------------------------------------------------------------------
#macro FILL0  src_section
              MOVW          A,      #\src_section
              MOVW          A,      #SIZEOF (\src_section)
              CALL          LMEMCLEAR
#endm
              FILL0  DIRDATA
              FILL0  DATA
;------------------------------------------------------------------------
; call main routine
;------------------------------------------------------------------------
              CALL          _main
End:          JMP           end
;------------------------------------------------------------------------
; Hard Watchdog
;------------------------------------------------------------------------
#ifdef HWD_DISABLE
           .SECTION  WDT,    CONST, LOCATE=H'FFBE
           .DATA.W   0xA596
#endif
;
;------------------------------------------------------------------------
; reset vector
;------------------------------------------------------------------------
           .SECTION  RESET, CONST, LOCATE=0xFFFC
              .DATA.B0xFF
              .DATA.B0
              .DATA.H__start
              .END   __start
```

NAME: vector.c
FUNCTION: Interrupt level (priority) setting and Interrupt vector definition

```
NAME:  vector.c
FUNCTION:     Interrupt level (priority) setting
           Interrupt vector definition
*****************************************************************************/
#include "mb95200.h"
;---------------------------------------------------------------------
//This function pre-sets all interrupt control registers. It can be used
//to set all interrupt priorities in static applications. If this file
// contains assignments to dedicated resources, verify that the appropriate
//controller is used.
//---------------------------------------------------------------------
void InitIrqLevels (void)
{
      ILR0 = 0xFC;          // IRQ0:  external interrupt ch.4         --> Level 00
                            // IRQ1:  external interrupt ch.5         --> 01
                            // IRQ2:  external interrupt ch.2/ch.6    --> 01
                            // IRQ3:  external interrupt ch.3/ ch.7--> 01
      ILR1 = 0xFF;          // IRQ4:  UART/SIO ch.0
                            // IRQ5:  8/16-bit timer ch.0 (lower)
                            // IRQ6:  8/16-bit timer ch.0 (upper)
                            // IRQ7:  LIN-UART (reception)
      ILR2 = 0xFF;          // IRQ8:  LIN-UART (transmission)
                            // IRQ9:  8/16-bit PPG ch.1 (lower) / UART/SIO ch.1
                            // IRQ10: 8/16-bit PPG ch.1 (upper) / I2C ch.1
                            // IRQ11: 16-bit reload timer ch.0
      ILR3 = 0xFF;          // IRQ12: 8/16-bit PPG ch.0 (upper)
                            // IRQ13: 8/16-bit PPG ch.0 (lower)
                            // IRQ14: 8/16-bit timer ch1 (upper)
                            // IRQ15: 16-bit PPG ch.0 + ch.2
      ILR4 = 0xFF;          // IRQ16: 16-bit reload timer ch.1 / I2C ch.0
                            // IRQ17: 16-bit PPG ch.1
                            // IRQ18: 10-biat A/D-converter
                            // IRQ19: Timebase timer
      ILR5 = 0xFF;          // IRQ20: Watch timer / counter
                            // IRQ21: external interrupt ch 8-11
                            // IRQ22: 8/16-bit timer ch1 (lower) / external //interrupt ch.12-15
                            // IRQ23: Flash | Custom ch.1
}
//---------------------------------------------------------------------
//   Prototypes
// Add your own prototypes here. Each vector definition needed is a //prototype. Either do it here
or include a header file containing them.
//---------------------------------------------------------------------
//extern unsigned int delay_timer;
__interrupt void DefaultIRQHandler (void);
__interrupt void Key_int (void);
//---------------------------------------------------------------------
//Vector definition
//Use the following statements to define vectors.
//All resources related to the vectors are predefined.
//Remaining software interrupts can be added here as well.
//---------------------------------------------------------------------
#pragma intvect Key_int 0                 //IRQ0: external interrupt ch.4
#pragma intvect DefaultIRQHandler 1       //IRQ1: external interrupt ch.5
#pragma intvect DefaultIRQHandler 2//IRQ2: external interrupt ch.2|ch.6
#pragma intvect DefaultIRQHandler 3//IRQ3: external interrupt ch.3|ch.7
#pragma intvect DefaultIRQHandler 5//IRQ5: 8/16-bit timer ch.0 (lower)
#pragma intvect DefaultIRQHandler 6//IRQ6: 8/16-bit timer ch.0 (upper)
#pragma intvect DefaultIRQHandler 7//IRQ7: LIN-UART (reception)
```

```
#pragma intvect DefaultIRQHandler 8       //IRQ8: LIN-UART (transmission)
#pragma intvect DefaultIRQHandler 14      //IRQ14: 8/16-bit timer ch1 (upper)
#pragma intvect DefaultIRQHandler 18      //IRQ18: 10-biat A/D-converter
#pragma intvect DefaultIRQHandler 19      //IRQ19: Timebase timer
#pragma intvect DefaultIRQHandler 20      //IRQ20:Watchtimer/ counter
#pragma intvect DefaultIRQHandler 22      //IRQ22: 8/16-bit timer ch.1 //(lower)/interrupt ch.12-
                                             15
#pragma intvect DefaultIRQHandler 23      //IRQ23: Flash / Custom ch.1
__interrupt void DefaultIRQHandler (void)
{
     __DI ();                             // disable interrupts
     While (1)
     __wait_nop ();                       // halt the system
}
```

NAME: Main.c

Function: External transition interrupt as key input

```
/*****************************************************************************
NAME: MAIN.C
FUNCTION: The following code can realize initialization the MCU
The following software is for demonstration purpose only
*****************************************************************************/
#include "mb95200.h"
#define switchmode stop     //set transition to some standby modes

void vSysInit (void)
{
      //elide code
      InitIrqLevels ();
      __EI ();
}
__interrupt void Key_int (void)
{
      //elide key functions
}
      //include vSysInit(),__interrupt and other functions
void main (void)
{
      vSysInit ();
      while (1)
      {
                           //enter other test codes
      }
}
```

7.1.2 Project 2 Name: Clock_Change

NAME: Main.c

Function: choose the clock, clock mode, clock divide ratio

```
/*****************************************************************************
//The following code can realize choose the clock and clock mode
//The following software is for demonstration purposes only
/*****************************************************************************
NAME: MAIN.C
FUNCTION: Change the system clock and lighten three LED
        LED flicker frequency different in different clock
*****************************************************************************/
#include "mb95200.h"
#define MAIN         0x00
#define SUB          0x01
#define MAIN_CR      0x02
#define SUB_CR       0x03
unsigned char switchclock = MAIN;         //select the start clock
unsigned char toggle_status = 0;          //LED change bit
unsigned char i;
/*****************************************************************************
NAME: MCU initialization
FUNCTION: Initialization the IO port, system clock, interrupt level
*****************************************************************************/
void MCU_initialization()
{
      __DI();
      SYSC = 0x03;
        /*IO port*/
      PDR0_P05=1;
      DDR0_P05=1;                         //Enable output
        PDR6_P63=1;
        PDR6_P64=1;
        DDR6_P63=1;                       //Enable output
        DDR6_P64=1;                       //Enable output

        /*external interrupt*/
      EIC30=0x55;                         //INT06 INT07 enable falling edge
      //source clock can be main clock divided by 2, sub-clock divided by 2
        //sub-CR clock divided by 2 or main CR clock no div
        SYCC_DIV = 0x00;                  // Source clock (No division)
                                          // SYCC_DIV = 0x01 mean Source clock/4
                                          // SYCC_DIV = 0x02 mean Source clock/8
                                          // SYCC_DIV = 0x03 mean Source clock/16
      /* initialise Interrupt level register and IRQ vector table*/
      InitIrqLevels();
        __EI();
}
/*****************************************************************************
NAME: led_display()
FUNCTION: Set three LED light cycle one by one
*****************************************************************************/
void led_display()
{
      switch(toggle_status)
        {
            case 0:                       //lighten the PDR0_P05 (LED2)
               {
                  PDR0_P05=0;
                      PDR6_P64=1;
                      PDR6_P63=1;
                      toggle_status=1;
```

```
                break;}
        case 1:                         //lighten the PDR6_P64 (LED3)
        {
                PDR0_P05=1;
                PDR6_P64=0;
                PDR6_P63=1;
                toggle_status=2;
                break;}
        case 2:                         //lighten the PDR6_P63 (LED4)
        {
                PDR0_P05=1;
                PDR6_P64=1;
                PDR6_P63=0;
                toggle_status=0;
                break;}
    }
}
/*****************************************************************************
NAME: vDelay
FUNCTION: Delay the fix time
*****************************************************************************/
void vDelay (unsigned int uiCount)
{
  while(uiCount--)
  {
    asm ("\tNOP");
  }
}
/*****************************************************************************
NAME: _interrupt void external_int06(void)
FUNCTION: Change the clock
*****************************************************************************/
__interrupt void external_int06 (void)
{
    EIC30_EIR0=0;
     switch (++switchclock)
    {
     case MAIN:
                        // enable the Main clock oscillation
                        SYCC2_MOSCE = 1;
                        // Clock Mode Selection, select the main clock mode
                        SYCC2_RCS0 = 0x01;
                        SYCC2_RCS1 = 0x01;
                        //update Oscillation Stabilization Wait Time
                        //when Main Oscillation Clock FCH=4MHz
                        WATR = 0X0F;
                        // Oscillation Stabilization Wait Time mean
                        // (2^14-2)/FCH About 4.10 ms
                        while (!STBC_MRDY);
                        break;
        case SUB:
                        // enable the sub-clock oscillation
                        SYCC2_ SOSCE = 1;
                        // Clock Mode Selection, select the sub-clock mode
                        SYCC2_RCS0 = 0x01;
                        SYCC2_RCS1 = 0x00;
                        //Update the Oscillation Stabilization Wait Time
                        // when sub Oscillation Clock FCL=32.768 kHz
                        WATR = 0xF0;  //(215-2)/FCL About 1.00s
                        //Indicates sub-clock oscillation being stable
                        while (!SYCC_SRDY);
                        break;
        case MAIN_CR:
```

```
                        // enable the main CR clock oscillation
                        SYCC2_ MCRE = 1;
                        // Clock Mode Selection, select the main CR clock mode
                        SYCC2_RCS0 = 0x00;
                        SYCC2_RCS1 = 0x01;
                        //update the Oscillation Stabilization Wait Time
                        //FCRHS represents the main clock frequency
                        //2^8/FCRHS = Main CR oscillation Stabilization Wait Time
                              //Indicates main CR clock oscillation being stable
                              while (!STBC_MCRDY);
                              break;
              case SUB_CR:
                              // enable the Sub-CR Clock oscillation
                              SYCC2_ SCRE = 1;
                              // Clock Mode Selection, select the Sub-CR Clock mode
                              SYCC2_RCS = 0x00;
                              //update the Oscillation Stabilization Wait Time
                              //FCRHS represents the main clock frequency
                              //2^4/FCRL = Sub-CR oscillation Stabilization Wait Time
                              //Indicates sub-CR clock oscillation being stable
                              while (!STBC_SCRDY);
                              break;
              default:
                     switchclock = MAIN;
                       SYCC2=0x34;
                              //update Oscillation Stabilization Wait Time
                              WATR = 0X03;
                       //Indicates main-clock oscillation being stable
                              while (!STBC_MRDY);
                              break;
       }
}
/*************************************************************************
NAME:         main ()
FUNCTION:     lighten three LED
*************************************************************************/
void main()
{
      MCU_initialization();
      while(1)
        {
            vDelay(10);             //delay the time
               led_display();       //lighten three LED
        }
}
```

NAME: vector.c

FUNCTION: Interrupt level setting and interrupt vector definition

```
/*****************************************************************************
NAME: vector.c
FUNCTION: Interrupt level (priority) setting
          Interrupt vector definition
*****************************************************************************/
#include "mb95200.h"
//---------------------------------------------------------------------------
//This function pre-sets all interrupt control registers. It can be used
//to set all interrupt priorities in static applications. If this file
//contains assignments to dedicated resources, verify that the appropriate //controller is used.
//---------------------------------------------------------------------------
void InitIrqLevels (void)
{
     ILR0 = 0x1F;          // IRQ0:  external interrupt ch.4
                           // IRQ1:  external interrupt ch.5
                           // IRQ2:  external interrupt ch.2/ch.6
                           // IRQ3:  external interrupt ch.3/ch.7
    //... ...
}
//---------------------------------------------------------------------------//   Prototypes
//Add your own prototypes here. Each vector definition needed is a //prototype. Either do it here
or include a header file containing them.
//---------------------------------------------------------------------------__interrupt void
DefaultIRQHandler (void);
__interrupt void external_int06 (void);
//---------------------------------------------------------------------------//Vector definition
//Use the following statements to define vectors.
//All resources related to the vectors are predefined.
//Remaining software interrupts can be added here as well.
//---------------------------------------------------------------------------#pragma intvect
DefaultIRQHandler 0  //IRQ0: external interrupt ch.4
#pragma intvect DefaultIRQHandler 1//IRQ1: external interrupt ch.5
#pragma intvect external_int06 2          //IRQ2: external interrupt ch.2|ch.6
//... ...
__interrupt void DefaultIRQHandler (void)
{
     __DI ();                                          // disable interrupts
     while (1)
     __wait_nop ();                              // halt the system
}
```

7.1.3 Project 3 Name: Mode_Change

NAME: Main.c

Function: standby mode change

```
/*****************************************************************************
//The following code can realization transition from normal mode to other
//mode, such as stop mode, watch mode
//The following software is for demonstration purposes only
//You can update the switchmode to the mode what you want
/*****************************************************************************
NAME: MAIN.C
FUNCTION: Change the mode and lighten three LED to one fix light
          Change the mode and LED light fixed
*****************************************************************************/
#include "mb95200.h"
#define normal       0x00
#define stop         0x01
#define sleep        0x02
#define Timebase     0x03
#define watch        0x04
unsigned char switchmode = normal;        //Select the start mode
unsigned char toggle_status = 0;          //LED change bit
unsigned char i;
/*****************************************************************************
NAME: MCU initialization
FUNCTION: Initialization the IO port, system clock, interrupt level
*****************************************************************************/
void MCU_initialization()
{
      __DI();
       /*system clock*/
       SYSC = 0x03;
       SYCC2 = 0x34;
       /*IO port*/
       PDR0_P05=1;
       DDR0_P05=1;                        //Enable output
       PDR6_P63=1;
       PDR6_P64=1;
       DDR6_P63=1;                        //Enable output
       DDR6_P64=1;                        //Enable output
       /*external interrupt*/
       EIC30=0x55;                        //INT06 enable falling edge
       /*initialise Interrupt level register and IRQ vector table*/
       InitIrqLevels();
       __EI();
}
/*****************************************************************************
NAME: led_display()
FUNCTION: Set three LED light cycle one by one
*****************************************************************************/
void led_display()
{
      switch(toggle_status)
       {
              case 0:                     //Lighten the PDR0_P05 (LED2)
              {
                     PDR0_P05=0;
                     PDR6_P64=1;
                     PDR6_P63=1;
                     toggle_status=1;
                     break;
```

```
            }
            case 1:                     //Lighten the PDR6_P64 (LED3)
            {
                  PDR0_P05=1;
                  PDR6_P64=0;
                  PDR6_P63=1;
                  toggle_status=2;
                  break;
            }
            case 2:                     //Lighten the PDR6_P63 (LED4)
            {
                  PDR0_P05=1;
                  PDR6_P64=1;
                  PDR6_P63=0;
                  toggle_status=0;
                  break;
            }
      }
}

/*****************************************************************************
NAME: vDelay
FUNCTION: Delay the fix time
*****************************************************************************/
void vDelay (unsigned int uiCount)
{
     while(uiCount--)
     {
          asm ("\tNOP");
     }
}
/*****************************************************************************
NAME: __interrupt void external_int06(void)
FUNCTION: Change the mode
*****************************************************************************/
__interrupt void external_int06(void)
{
     EIC30_EIR0=0;
     if(switchmode <= 3)
          switchmode++;
      if(switchmode >= 4)
          switchmode = 1;
}
/*****************************************************************************
NAME: __interrupt void external_int07(void)
FUNCTION: Reset the mode to normal
*****************************************************************************/
__interrupt void external_int07(void)
{
     EIC30_EIR1 = 0;
     switchmode = normal;
}
```

```
/*****************************************************************************
NAME: main ()
FUNCTION: Lighten three LED, when change the mode,
          only one LED light
*****************************************************************************/
void main()
{
      MCU_initialization();
       while(1)
       {
            switch (switchmode)
               {
                  case normal:      break;
                       case stop:    STBC_STP = 1;//Causes transition to stop mode
                                    break;                //Normal to stop mode
                       case sleep:   STBC_SLP = 1;//Causes transition to sleep mode
                                    break;                //Normal to sleep mode
                       case watch:   SYCC2_SOSCE = 1;      //Enable Sub Clock
                                     SYCC2_RCS0 = 0x01;    //Select Sub Clock
                                            SYCC2_RCS1 = 0x00;
                                    WATR = 0xF0;           //About 1.00s Wait Time
                                            while (!SYCC_SRDY);  //Wait Clock stable
                                            STBC_TMD = 1;
                                            //Causes the device to enter watch mode
                                     break;                //normal to watch mode
                       case Timebase:SYCC2_MOSCE = 1;      //Enable Main Clock
                                            SYCC2_RCS0 = 0x01;   //Select Main Clock
                                            SYCC2_RCS1 = 0x01;
                                    WATR = 0X0F;           // About 4.10 ms Wait Time
                                            while (!STBC_MRDY);  //Wait Clock stable
                                            STBC_TMD = 1;
                                     //Causes the device to enter timebase timer mode
                                    break;          //normal to timebase time mode
                       default:
                                            break;
               }
               vDelay(2000);                        //delay the time
               led_display();                       //lighten three LED
       }
}
```

NAME: vector.c

FUNCTION: Interrupt level setting and interrupt vector definition

```
/*****************************************************************************
NAME: vector.c
FUNCTION: Interrupt level (priority) setting
          Interrupt vector definition
*****************************************************************************/
#include "mb95200.h"
//---------------------------------------------------------------------
//This function pre-sets all interrupt control registers. It can be used
//to set all interrupt priorities in static applications. If this file
//contains assignments to dedicated resources, verify that the appropriate //controller is used.
//---------------------------------------------------------------------
void InitIrqLevels (void)
{
     ILR0 = 0x1F;          // IRQ0:  external interrupt ch.4
                           // IRQ1:  external interrupt ch.5
                           // IRQ2:  external interrupt ch.2/ch.6
                           // IRQ3:  external interrupt ch.3/ch.7
    //... ...
}
//---------------------------------------------------------------------
//Prototypes
//Add your own prototypes here. Each vector definition needed is a //prototype. Either do it here
or include a header file containing them.
//---------------------------------------------------------------------
__interrupt void DefaultIRQHandler (void);
__interrupt void external_int06 (void);
__interrupt void external_int07 (void);
//---------------------------------------------------------------------
//Vector definition
//Use the following statements to define vectors.
//All resources related to the vectors are predefined.
//Remaining software interrupts can be added here as well.
//---------------------------------------------------------------------
#pragma intvect DefaultIRQHandler 0//IRQ0: external interrupt ch.4
#pragma intvect DefaultIRQHandler 1//IRQ1: external interrupt ch.5
#pragma intvect external_int06 2          //IRQ2: external interrupt ch.2/ch.6
#pragma intvect external_int07 3          //IRQ3: external interrupt ch.3/ch.7
//... ...
__interrupt void DefaultIRQHandler (void)
{
     __DI ();                             // disable interrupts
     while (1)
     __wait_nop ();                       // halt the system
}
-- END --
```

8 Additional Information

For more information on Cypress Microcontrollers Products, please visit the following websites:

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

Document History

Document Title: AN205271 – F²MC - 8FX Family, MB95200H/210H Series, Basic FW Setup

Document Number: 002-05271

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	Lori.Li	26/03/2008	V1.0, First Draft
			21/07/2008	V1.1, Modify the chapter 5 clock modes selection code and comment.
			26/02/2010	V1.2, Remove 12.5M CR description in page 11, 13, 24
*A	5260320	HUAL	05/06/2016	Migrated Spansion Application Note “MCU-AN-500038-E-10” to Cypress format.

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
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. 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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation,2008-2016. 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. 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, 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.