



本ドキュメントはCypress (サイプレス) 製品に関する情報が記載されております。本ドキュメントには、仕様の開発元企業として「スパンション」, 「Spansion」, 「富士通」または「Fujitsu」の名が記載されておりますが、これらの製品は Cypress が新規および既存のお客様に引き続き提供してまいります。

商品仕様の継続性について

Cypress 製品として提供することに伴う商品仕様としての変更はなく、ドキュメントとしての変更もありません。また本ページのお知らせは、変更情報として追記いたしません。本ドキュメントに変更情報が記載されている場合、それは本お知らせを除いた前版からの変更点です。なお、今後改訂は必要に応じて行われますが、その際の変更内容は改訂後のドキュメントに記載いたします。

オーダ型格および品名について

Cypress は既存のオーダ型格および品名を引き続きサポートいたします。これらの製品をご注文の際は、このドキュメントに記載されているオーダ型格および品名をご使用ください。

詳しいお問い合わせ先

Cypress 製品およびそのソリューションの詳細につきましては、お近くの営業所へお問い合わせください。

サイプレスについて

サイプレス (銘柄コード: CY) は、車載や産業機器、ネットワーキング プラットフォームから高機能民生機器およびモバイル機器まで、今日の最先端組み込みシステム向けに高性能で高品質のソリューションを提供します。NOR フラッシュ メモリや F-RAMTM、SRAM、TraveoTM マイクロコントローラー、業界唯一の PSoC[®] プログラマブル システムオンチップ ソリューション、アナログおよび PMIC Power Management IC、CapSense[®] 静電容量タッチセンシング コントローラー、Wireless BLE Bluetooth[®] Low-Energy、USB コネクティビティ ソリューションなど、幅広い差別化製品ポートフォリオを、一貫した革新性と業界最高クラスの技術サポート、比類のないシステム バリューとともにグローバルに提供します。

F²MC 8FX Family
8-BIT MICROCONTROLLER
MB95200H/210H series

ベーシックファームウェアセットアップ

注意事項

- 本資料の記載内容は、予告なしに変更することがありますので、ご用命の際は営業部門にご確認ください。
- 本資料に記載された動作概要や応用回路例は、半導体デバイスの標準的な動作や使い方を示したもので、実際に使用する機器での動作を保証するものではありません。したがって、これらを使用するにあたってはお客様の責任において機器の設計を行ってください。これらの使用に起因する損害などについては、当社はその責任を負いません。
- 本資料に記載された動作概要・回路図を含む技術情報は、当社もしくは第三者の特許権、著作権等の知的財産権やその他の権利の使用権または実施権の許諾を意味するものではありません。また、これらの使用について、第三者の知的財産権やその他の権利の実施ができることの保証を行うものではありません。したがって、これらの使用に起因する第三者の知的財産権やその他の権利の侵害について、当社はその責任を負いません。
- 本資料に記載された製品は、通常の産業用、一般事務用、パーソナル用、家庭用などの一般的用途に使用されることを意図して設計・製造されています。極めて高度な安全性が要求され、仮に当該安全性が確保されない場合、社会的に重大な影響を与えかつ直接生命・身体に対する重大な危険性を伴う用途（原子力施設における核反応制御、航空機自動飛行制御、航空交通管制、大量輸送システムにおける運行制御、生命維持のための医療機器、兵器システムにおけるミサイル発射制御をいう）、ならびに極めて高い信頼性が要求される用途（海底中継器、宇宙衛星をいう）に使用されるよう設計・製造されたものではありません。したがって、これらの用途にご使用をお考えのお客様は、必ず事前に営業部門までご相談ください。ご相談なく使用されたことにより発生した損害などについては、責任を負いかねますのでご了承ください。
- 半導体デバイスはある確率で故障が発生します。当社半導体デバイスが故障しても、結果的に人身事故、火災事故、社会的な損害を生じさせないように、お客様は、装置の冗長設計、延焼対策設計、過電流防止対策設計、誤動作防止設計などの安全設計をお願いします。
- 本資料に記載された製品を輸出または提供する場合は、外国為替及び外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとりください。
- 本書に記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

Copyright© 2008 FUJITSU MICROELECTRONICS LIMITED all rights reserved

改版履歴

版数	日付	内容
1.0 版	2008.10.7	新規作成

目次	
注意事項	1
改版履歴	2
目次	3
1 はじめに	5
2 初期化ルーチン	6
2.2 SP と DP の設定	6
3 割込みベクタ	8
3.1 割込みレベル	8
3.2 割込みハンドラ関数のプロトタイプ	8
3.3 ベクタの定義	9
3.4 デフォルトの割込みハンドラ	10
3.5 例	10
4 クロック制御	12
4.1 クロック制御部の概要	12
4.2 クロック制御部のブロックダイアグラム	12
4.3 レジスタ	13
5 クロックモードの選択	15
5.1 はじめに(クロックモード)	15
5.2 クロックモードの主な特長	15
5.3 外部クロック	15
5.4 メイン発振クロック	15
5.6 内部メイン CR クロック	18
5.7 内部サブ CR クロック	20
5.8 クロック分周比	21
6 スタンバイモードの選択	22
6.1 主な特長	22
6.2 スタンバイモードとクロック供給状態の概要	22
6.3 クロックモードとスタンバイモードの組み合わせ	23
6.4 スリープモード	26
6.4.1 はじめに	26
6.4.2 スリープモードにおける動作	26
6.4.3 移行と解除	26
6.5 ストップモード	26
6.5.1 はじめに	26
6.5.2 ストップモードにおける動作	26

6.5.3	移行と解除	26
6.6	タイムベースタイマモード	27
6.6.1	はじめに	27
6.6.2	タイムベースタイマモードにおける動作	27
6.6.3	移行と解除	27
6.7	時計モード	28
6.7.1	はじめに	28
6.7.2	時計モードにおける動作	28
6.7.3	移行と解除	28
6.8	スタンバイモードへの遷移サンプルコード	30
7	その他の情報	32
8	付録	33
8.1	図の索引	33
8.2	サンプルコード	34
8.2.1	Project 1 Name: Basic_Initialization	34
8.2.2	Project 2 Name: Clock_Change	42
8.2.3	Project 3 Name: Mode_Change	53

1 はじめに

このアプリケーションノートでは、富士通 F²MC-8FX ファミリ MB95200H/210H シリーズの基本的なファームウェアの設定方法について説明します。

初期化ルーチン、スタックポインタ (SP) とダイレクトバンクポインタ (DP) の設定、クロック制御、およびスタンバイ制御について。

2 初期化ルーチン

スタートアップファイル **start.asm** は、リセット実行直後のエントリポイントです。このファイルには、コンパイラおよびデバイス動作の設定値が複数含まれています。これによって、スタックと変数の初期値が初期化されます。

2.1 スタックの初期化

MCU の起動時には、スタックの初期化が必要になります。スタックサイズの設定やスタック領域の定義を始めに行います。

スタックの初期化サンプルコード

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

注：上記のコードは **start.asm** から抜粋したものです。付録を参照してください。

2.2 SP と DP の設定

SP と DP は専用レジスタです。

スタックポインタ (SP) は、スタックの PUSH/POP 命令によって割込みやサブルーチンコールが発生したときに参照されるアドレスを保持する 16 ビット・レジスタです。リセット後の初期値は「0000H」です。

DP はダイレクトバンクポインタです。プログラムステータス (PS) レジスタのビット 10 ～ 8 の DP は、直接アドレス指定によりアクセスされる領域を指定します。

SP, DP 初期化サンプルコード

```
.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
```

注：上記のコードは **start.asm** から抜粋したものです。付録を参照してください。

3 割込みベクタ

ファームウェアの基本的な設定には割込みベクタの初期化処理が必要です。ベクタ設定ファイル `vector.c` は、ベクタ割込みのサンプルとしてユーザ用に提供されています。この `C` ファイルには、割り込みレベルと割込み番号が宣言されており、デフォルトの割込みハンドラが含まれています。

3.1 割込みレベル

ファイル `vector.c` では、割込みレベル関数 `void InitIrqLevels (void)` が最初に定義されています。

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

    //.....
}
```

注：上記のコードは参考例です。

この関数は、共有割込みチャネルごとに割込みレベルを定義する割込み制御レジスタを初期化します。0xff は最も低い優先度を示します。

3.2 割込みハンドラ関数のプロトタイプ

割込みハンドラの関数プロトタイプは、割り込みレベル初期化後に宣言します。

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

注：上記のコードは参考例です。

3.3 ベクタの定義

ベクタナンバーは、割込みハンドラ関数とリンクされます。未使用の割込みについては、デフォルトの割込みハンドラを使用してください。

```
#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
```

注：上記のコードは参考例です。

3.4 デフォルトの割込みハンドラ

デフォルトの割込みハンドラ関数は、**vector.c** ファイルの次に記載されます。この関数は、無限ループによってシステム停止を実行します。デバッグでは、ここにブレークポイントを設定し、初期化されていない割込みを検出するのに有効です。
次に、デフォルトの割込みハンドラのサンプルコードを示します。

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

注：上記のコードは **vector.c** から抜粋したものです。付録を参照してください。

3.5 例

割込みの使用において割込みベクタの修正が必要な例、たとえば、外部割込み **ch4** を **key_int** としてラベルの定義を行なった場合、**vector.c** のコードも修正してください。

```
#pragma intvect Key_int 0           // IRQ0: external interrupt ch4
```

注：上記のコードは **vector.c** から抜粋したものです。付録を参照してください。

次に、割込みレベルの定義を示します。

```
// defines the interrupt levels
void InitIrqLevels (void)
{
    IIR0=0xFC // IRQ0: external interrupt ch.4      --> Level 00
               // IRQ1: external interrupt ch.5
               // IRQ2: external interrupt ch.2 | ch.6
               // IRQ3: external interrupt ch.3 | ch.7
    //.....
}
```

注：上記のコードは、**vector.c** から抜粋したものです。付録を参照してください。

ファイル `main.c` で、割込みハンドラ関数を `_interrupt void Key_Int (void)` として入力します。

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

注：上記のコードは `main.c` から抜粋したものです。付録を参照してください。

4 クロック制御

第 4 章では、クロック制御部の機能と動作を説明します。

4.1 クロック制御部の概要

MB95200H/210H シリーズには、電力消費を最適化するクロック制御回路が内蔵されています。外部メイン発振クロック、外部サブ発振クロック、外部クロック入力をサポートする製品(MB95200H)、および外部クロック入力のみをサポートする製品(MB95210H)があります。クロック制御部は、クロック発振の有効/無効、内部回路へのクロック信号供給の有効/無効、クロックソースの選択、および内部 CR 発振回路と分周回路の制御を行います。

4.2 クロック制御部のブロックダイアグラム

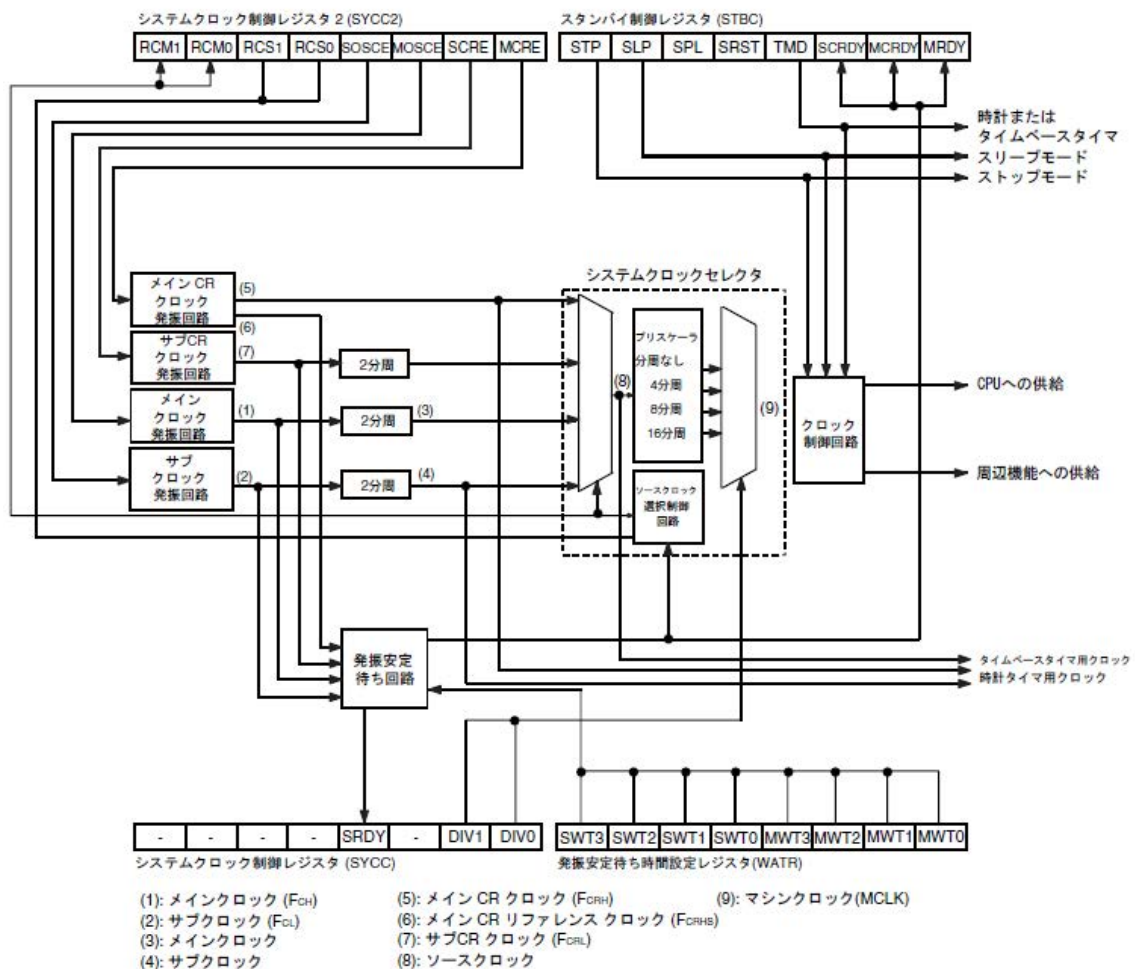


図 4.2-1 クロック制御部のブロックダイアグラム

4.3 レジスタ

クロック制御部は、システムクロック制御レジスタ (SYCC), スタンバイ制御レジスタ (STBC), システムクロック制御レジスタ 2 (SYCC2), および発振安定待ち時間設定レジスタ (WATR) の 4 つのレジスタで構成されています。

詳細については, MCU MB95200H/210H シリーズ ハードウェアマニュアル第 6 章 クロック制御部を参照してください。

図 4.3-1 は, システムクロック制御レジスタ (SYCC) の構成を示しています。このレジスタは, マシンクロック分周比の選択を制御するために使用します。

アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	初期値
0007 _H	-	-	-	-	SRDY	-	DIV1	DIV0	0000X011 _B
	R0/WX	R0/WX	R0/WX	R0/WX	R/WX	R0/WX	R/W	R/W	

図 4.3-1 システムクロック制御レジスタ (SYCC) の構成

図 4.3-2 は, 発振安定待ち時間設定レジスタ (WATR) の構成を示しています。このレジスタは, 発振安定待ち時間を設定するために使用します。

アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	初期値
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	

図 4.3-2 発振安定待ち時間設定レジスタ (WATR) の構成

図 4.3-3 は, スタンバイ制御レジスタ (STBC) を示しています。このレジスタは, RUN 状態からスリープモード, ストップモード, タイムベースタイマモード, ウォッチモードへの遷移, ストップモード, タイムベースタイマモード, 時計モードの端子状態設定, およびソフトウェアリセットの発生制御を行ないます。

アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	初期値
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	

図 4.3-3 スタンバイ制御レジスタ (STBC)

図 4.3-4 は, システムクロック制御レジスタ 2 (SYCC2) の構成を示しています。このレジスタは, 現在のクロックモードの状態表示と切り替を行い, サブクロック, メインクロック,

サブ CR クロック, メイン CR クロックの発振制御を行ないます。

アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	初期値
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	

図 4.3-4 システムクロック制御レジスタ 2 (SYCC2) の構成

R/W : 読出し可能/書き込み可能 (読み取り値は書き込み値と同じ)

R/WX : 読出し専用 (書き込みは動作に影響を与えません)

R0/WX : 未定義ビット (読出し値は「0」, 書き込みは動作に影響を与えません)

R0/W : 書き込み専用 (書き込み可能, 読出し時の値は「0」となります。)

5 クロックモードの選択

5.1 はじめに(クロックモード)

MB95200H シリーズは、5 つのクロック (外部クロック、メイン発振クロック、サブ発振クロック、メイン CR クロック、サブ CR クロック) がサポートされ、さまざまなクロック選択が可能です。MB95210H シリーズではサポートされるクロックが 3 つになります。

5.2 クロックモードの主な特長

- 選択可能なメインクロックソース
 - 外部クロック (最大 32.5 MHz, 最大マシクロック周波数 : 16.25 MHz)
 - メイン発振クロック (最大 16.25 MHz, 最大マシクロック周波数 : 8.125 MHz)
 - 内部メイン CR クロック (1 / 8 / 10 MHz)
- 選択可能なサブクロックソース
 - 外部サブクロック (32.768 kHz)
 - サブ発振クロック (32.768 kHz)
 - 内部サブ CR クロック (標準 100 KHz, 最小 50 kHz, 最大 200 kHz)
- 初期クロック設定は内蔵メイン CR クロック : (8 MHz)

5.3 外部クロック

外部クロックについて、外部クロックの最大周波数は最大 32.5 MHz, 最大マシクロック周波数は 16.25 MHz です。

5.4 メイン発振クロック

メイン発振クロックを利用する場合には SYSC レジスタの PFSEL ビットを"0"に設定します。このビットは PF1/PF0 の機能選択ビットです。"0"に設定することで、PF1/PF0 を外部のメイン発振クロック端子に設定します。

```
IO_SYSC.bit.PFSEL = 0;           //Enable the external Main clock input
```

注: PFSEL の設定に関する記述は MB95200H/210H シリーズハードウェアマニュアルの第 23 章を参照ください。

SYCC2 レジスタの MOSCE ビットにより、メイン発振クロックの許可/禁止設定を行ないます。"1"を設定することで、メイン発振クロックの発振が許可となります。SYCC2 の RCS1/RCS0 ビットはクロックモードの選択を行ないます。"11b"の設定にてメイン発振ク

ロックモードに切り替わります。

WATR レジスタの下位 4 ビットはメイン発振クロック発振安定待ちの時間設定を行います。また、STBC レジスタの MRDY ビット はメイン発振クロックの発振安定状態を示します。このビットを確認することにより、メイン発振クロックの発振が安定していることを確認することができます。

WATR が 0x0F に設定されている場合、発振安定待ち時間は $(2^{14}-2)/F_{CH}$, 約 4.10 ms となります。

次のサンプルコードにて、メイン発振クロックの設定変更処理の方法例を示します。これには、メイン発振クロックの有効化、モード変更、発振安定待ち時間の設定、および発振安定待ちの処理が含まれています。

```
// enable the Main clock oscillation
SYCC2_MOSCE = 1;
// Clock Mode Selection, select the main clock mode
IO_SYCC2.bit.RCS1 = 1;
IO_SYCC2.bit.RCS0 = 1;
//update Oscillation Stabilization Wait Time
//when Main Oscillation Clock FCH=4MHz
WATR = 0x0F; // Oscillation Stabilization Wait Time mean
// (214-2)/FCH About 4.10 ms
While (!STBC_MRDY); //Indicates main clock oscillation being stable
```

注：上記のコードは、Clock_Change プロジェクトから抜粋したものです。付録を参照してください。

5.5 サブ発振クロック

サブ発振クロックを利用する場合には SYSC レジスタの PGSEL ビットを"0"に設定します。このビットは PG1/PG0 の機能選択ビットです。"0"に設定することで、PG1/PG0 をサブ発振クロック端子に設定します。

```
IO_SYSC.bit.PGSEL = 0; //Enable the external Sub clock input
```

注: PGSEL の設定に関する記述は MB95200H/210H シリーズハードウェアマニュアルの第 23 章を参照ください。

SYCC2 レジスタの SOSCE ビットにより、サブ発振クロックの許可/禁止設定を行ないます。"1"を設定することで、サブ発振クロックの発振が許可となります。SYCC2 の

RCS1/RCS0 ビットはクロックモードの選択を行ないます。"01 b"の設定にてサブ発振クロックモードに切り替わります。WATR レジスタの上位 4 ビットはサブ発振クロックの発振安定待ちの時間設定を行います。また、SYCC レジスタの SRDY ビットはサブ発振クロックの発振安定状態を示します。このビットを確認することにより、サブ発振クロックの発振が安定していることを確認することができます。

WATR が 0xF0 に設定されている場合、発振安定待ち時間は $(2^{15}-2)/F_{CL}$, 約 1.00 s です。次のサンプルコードにて、サブ発振クロックの設定変更処理の方法例を示します。これには、サブ発振クロックの有効化、サブ発振クロックモードへの変更、発振安定待ち時間の設定、および発振安定待ちの処理が含まれます。

```
// enable the sub-clock oscillation
    SYCC2_SOSCE = 1;
// Clock Mode Selection, select the sub-clock mode
    IO_SYCC2.bit.RCS1 = 0;
    IO_SYCC2.bit.RCS0 = 1;
//Update the Oscillation Stabilization Wait Time
// when sub Oscillation Clock FCL=32.768 kHz
WATR = 0xF0;          //(215-2)/FCL About 1.00s
while (!SYCC_SRDY);  //Indicates sub-clock oscillation being stable
```

注：上記のコードは、Clock_Change プロジェクトから抜粋したものです。付録を参照してください。

5.6 内部メイン CR クロック

内部メイン CR クロックは、1 MHz, 8 MHz, 10 MHz の 3 つの周波数クロックを備えています。IO レジスタ領域の NVR インタフェースレジスタにより、メイン CR クロックを変更できます。SYCC2 レジスタの MCRE ビットにより、メイン CR クロックの許可/禁止設定を行ないます。"1"を設定することで、メイン CR クロックの発振が許可となります。SYCC2 の RCS1/RCS0 ビットはクロックモードの選択を行ないます。"10 b"の設定にてメイン CR クロックモードに切り替わります。

また、STBC レジスタの MCRDY ビット はメイン CR クロックの発振安定状態を示します。このビットを確認することにより、メイン CR クロックの発振が安定していることを確認することができます。

次のサンプルコードは、メイン CR クロックの設定変更処理の方法例を示します。これには、メイン CR クロックの有効化、メイン CR クロックモードへの変更、発振安定待ち時間の設定、および発振安定待ち処理が含まれます。

```
; 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
```

注：上記のコードは参考例です。

```
// enable the main CR clock oscillation
SYCC2_MCRE = 1;
// Clock Mode Selection, select the main CR clock mode
IO_SYCC2.bit.RCS1 = 1;
IO_SYCC2.bit.RCS0 = 0;
// 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
```

注：上記のコードは、Clock_Change プロジェクトから抜粋したものです。付録を参照してください。

発振安定待ち時間は, CR スタートアップのタイミングに応じて変化します。

メイン CR 発振安定待ち時間は $2^8/F_{CRHS}^{*1}$ です。

(*1 F_{CRHS} : メインCR 基準クロック周波数 1MHz)

5.7 内部サブ CR クロック

内部サブ CR クロックは、標準クロック周波数 100 kHz, 最小周波数 50 kHz, 最大周波数 200 kHz のサブクロックです。

SYCC2 レジスタの SCRE ビットにより、サブ CR クロックの許可/禁止設定を行ないます。"1"を設定することで、サブ CR クロックの発振が許可となります。SYCC2 の RCS1/RCS0 ビットはクロックモードの選択を行ないます。"00 b"の設定にてサブ CR クロックモードに切り替わります。

また、STBC レジスタの SCDY ビット はサブ CR クロックの発振安定状態を示します。このビットを確認することにより、サブ CR クロックの発振が安定していることを確認することができます。

次のサンプルコードは、サブ CR クロックの設定変更処理の方法例を示します。これには、サブ CR クロックの有効化、サブ CR クロックモードへの変更、発振安定待ち時間の設定、および発振安定待ち処理が含まれます。

```
// enable the Sub-CR Clock oscillation
SYCC2_SCRE = 1;

// Clock Mode Selection, select the Sub-CR Clock mode
IO_SYCC2.bit.RCS1 = 0;
IO_SYCC2.bit.RCS0 = 0;

//update the Oscillation Stabilization Wait Time
//FCRHS represents the main clock frequency
//25/FCRL = Sub-CR oscillation Stabilization Wait Time
while (!STBC_SCDY); //Indicates sub-CR clock oscillation being stable
```

注：上記のコードは、Clock_Change プロジェクトから抜粋したものです。付録を参照してください。発振安定待ち時間は、CR スタートアップのタイミングに応じて変化します。サブ CR 発振安定待ち時間は $2^5/F_{CRL}$ です。 (F_{CRL} はサブ CR クロック周波数)

5.8 クロック分周比

マシナクロックは、分周比に基づいてソースクロックから生成されます。ソースクロックの分周設定は SYCC レジスタの DIV1 / DIV0 にて設定することができます。SYCC の DIV1 / DIV0 が “11b” に設定されている場合、ソースクロックは 16 分周されます。

次のコードは、クロック分周比の設定方法を示しています。

```
//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
IO_SYCC.byte = 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
```

注：上記のコードは、Clock_Change プロジェクトから抜粋したものです。付録を参照してください。

6 スタンバイモードの選択

スタンバイモードは、スリープモード、タイムベースタイマモード、時計モード、ストップモードの 4 つのモードで構成されます。クロックコントローラは、各スタンバイモードに応じて、クロック発振、および内部回路へのクロック供給の有効化 / 無効化を選択します。タイムベースタイマモードはメインクロックモードまたは、メイン **CR** クロックモードにて利用することができます。時計モードはサブクロックモードまたは、サブ **CR** クロックモードにて利用することができます。

6.1 主な特長

- 低電力消費モード (スタンバイモード)
 - ストップモード
 - スリープモード
 - 時計モード
 - タイムベースタイマモード

6.2 スタンバイモードとクロック供給状態の概要

スタンバイモード	クロック供給状態
スリープモード	スリープモードでは CPU とソフトウェアウォッチドッグタイマの動作を停止します。 CPU はデバイスがスリープモードへ遷移する直前に存在しているレジスタと RAM の内容を保持して停止しますが、ウォッチドッグタイマを除く周辺機能は動作を続けます。
タイムベースタイマモード	タイムベースタイマモードではメインクロック発振、サブクロック発振、タイムベースタイマ、および時計プリスケアラのみ動作します。このモードでは CPU と周辺機能の動作クロックは停止となります。 このモードではデバイスはタイムベースタイマモードへ遷移する直前に存在しているレジスタと RAM の内容を保持しつつ、タイムベースタイマ、外部割込みと低電圧検出リセットを除くすべての機能を停止します。 タイムベースタイマモードはメインクロックモード、およびメイン CR クロックモードにおいて使用可能です。
時計モード	時計モードではサブクロック、サブ CR クロック、および時計プリスケアラのみが動作します。このモードでは CPU と周辺機能の動作クロックが停止します。デバイスは時計モードへ遷移する直前に存在しているレジスタと RAM

	<p>の内容を保持しつつ，デバイスは外部割込みと低電圧検出リセットを除くすべての機能を停止します。</p> <p>時計モードはサブクロックモードおよびサブ- CR クロックモードにおいて使用可能です。</p>
ストップモード	<p>ストップモードでは，メインクロック，メインCR クロック，およびサブクロックは停止となります。このモードでは，デバイスはストップモードへ遷移する直前に存在しているレジスタとRAMの内容を保持しつつ，外部割込みと低電圧検出リセットを除くすべての機能を停止します。</p>

図 6.2-1 スタンバイモードとクロック供給状態

6.3 クロックモードとスタンバイモードの組み合わせ

図 6.3-1 と図 6.3-2 は，クロックモードとスタンバイモードの組み合わせ，およびそれぞれの内部回路動作状態の一覧です。

詳細については，MCU MB95200H/210H シリーズのハードウェアマニュアル第 6 章を参照してください。

機能	RUN				スリープ			
	メイン クロック モード	メイン CR クロック モード	サブ クロック モード (2 系統外部 クロック品)	サブ CR クロック モード	メイン クロック モード	メイン CR クロック モード	サブ クロック モード (2 系統外部 クロック品)	サブ CR クロック モード
メイン クロック	動作	停止 *1	停止		動作	停止 *1	停止	
メイン CR クロック	停止 *2	動作	停止		停止 *2	動作	停止	
サブクロック	動作 *3		動作	動作 *3	動作 *3		動作	動作 *3
サブ CR クロック	動作 *4		動作 *4	動作	動作 *4		動作 *4	動作
CPU	動作		動作		停止		停止	
ROM	動作		動作		値保持		値保持	
RAM			動作		値保持		値保持	
I/O ポート	動作		動作		出力保持		出力保持	
タイムベース タイマ	動作		停止		動作		停止	
時計プリス ケアラ	動作 *3, *4		動作		動作 *3*4		動作	
外部割込み	動作		動作		動作		動作	
ハードウェア ウォッチドッ グタイマ	動作		動作		動作 *5		動作 *5	
ソフトウェア ウォッチドッ グタイマ	動作		動作		停止		停止	
低電圧検出 リセット	動作		動作		動作		動作	
その他の周辺 機能	動作		動作		動作		動作	

図 6.3-1 スタンバイモードとクロックモードの組み合わせ、および内部動作状態

機能	タイムベースタイマ		時計プリスケラ		ストップ			
	メイン クロック モード	メイン CR クロック モード	サブ クロック モード (2 系統外部 クロック品)	サブ CR クロック モード	メイン クロック モード	メイン CR クロック モード	サブ クロック モード (2 系統外部 クロック品)	サブ CR クロック モード
メイン クロック	動作	停止 *1	停止		停止			
メイン CR クロック	停止 *2	動作	停止		停止			
サブクロック	動作 *3		動作	動作 *3	動作 *3		停止	
サブ CR クロック	動作 *4		動作 *4	動作	動作 *4		停止	
CPU	停止		停止		停止			
ROM	値保持		値保持		値保持			
RAM								
I/O ポート	出力保持 /Hi-Z		出力保持 /Hi-Z		出力保持 /Hi-Z			
タイムベース タイマ	動作		停止		停止			
時計プリス ケラ	動作 *3, *4		動作		動作 *3, *4		停止	
外部割込み	動作		動作		動作			
ハードウェア ウォッチドッ グタイマ	動作 *5		動作 *5		動作 *5			
ソフトウェア ウォッチドッ グタイマ	停止		停止		停止			
低電圧検出 リセット	動作		動作		動作			
その他の周辺 機能	停止		停止		停止			

図 6.3-2 スタンバイモードとクロックモードの組み合わせ、および内部動作状態

注：スタンバイモードへの遷移は、クロックモードへの遷移が完了したあとに設定します。

6.4 スリープモード

6.4.1 はじめに

スリープモードは、CPU とソフトウェアウォッチドッグタイマの動作を停止します。

6.4.2 スリープモードにおける動作

スリープモードは、CPU とソフトウェアウォッチドッグタイマの動作クロックを停止します。このモードでは、CPU が停止する一方、スリープモードへ遷移する直前に存在していたレジスタと RAM の内容は維持されます。ウォッチドッグタイマ以外の周辺リソースは動作を継続します。

6.4.3 移行と解除

スタンバイ制御レジスタのスリープビット(STBC : SLP) に"1"を書き込むと、デバイスはスリープモードに入ります。

サンプルコードは、RUN 状態からスリープモードへ遷移させる STBC の設定例です。

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

注：上記のコードは、Mode_Change プロジェクトから抜粋したものです。付録を参照してください。リセットまたは周辺機能からの割込みによって、デバイスはスリープモードから解除されます。

6.5 ストップモード

6.5.1 はじめに

ストップモードはメインクロック、メイン CR クロック、サブクロックを停止します。

6.5.2 ストップモードにおける動作

ストップモードは、メインクロック、メイン CR クロック、およびサブクロックを停止します。このモードでは、デバイスは、外部割込みと低電圧検出リセット以外のすべての機能を停止します。一方、ストップモードへ遷移する直前に存在していたレジスタと RAM の内容は維持されます。

6.5.3 移行と解除

スタンバイ制御レジスタのストップビット(STBC : STP) に"1"を書き込むと、デバイスはストップモードに入ります。このとき、スタンバイ制御レジスタの端子状態設定ビット(STBC : SPL)が"0"のとき外部端子の状態は維持され、(STBC : SPL)ビットが"1"のとき外部端子の状態はHiインピーダンスとなります。(プルアップ設定レジスタでプルアップ抵抗を

選択している端子はプルアップ状態になります)

サンプルコードは、ストップモードへ遷移させる **STBC** の設定例です。

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

注：上記のコードは、**Mode_Change** プロジェクトから抜粋したものです。付録を参照してください。

デバイスは、リセットまたは外部割込みに対応してストップモードから解除されます。動作途中でストップモードとなった周辺機能はストップモードに遷移した時点の動作から再開します。よって、ストップモードから復帰した場合、必要に応じて各周辺リソースを初期化してください。

6.6 タイムベースタイマモード

6.6.1 はじめに

タイムベースタイマモードでは、メインクロック発振、サブクロック発振、タイムベースタイマ、および時計プリスケアラのみが機能します、このモードでは、**CPU** と周辺リソースの動作クロックは停止します。

6.6.2 タイムベースタイマモードにおける動作

タイムベースタイマモードでは、タイムベースタイマを除き、メインクロック供給は停止します。デバイスは、タイムベースタイマ、外部割込み、および低電圧検出リセット以外のすべての機能を停止します。一方、タイムベースタイマモードへ遷移する直前に存在していたレジスタと **RAM** の内容は維持されます。

6.6.3 移行と解除

タイムベースタイマモードへの遷移はデバイスのクロックモードがメインクロックまたはメイン **CR** クロックの場合に、スタンバイ制御レジスタの時計ビット (**STBC : TMD**)に"1"を書き込みます。

サンプルコードは、タイムベースタイマモードへ遷移させる **STBC** の設定例です。

```
SYCC2_MOSCE = 1; //Enable Main Clock
IO_SYCC2.bit.RCS1 = 1;
IO_SYCC2.bit.RCS0 = 1;
WATR = 0x0F; // About 4.10 ms Wait Time
while (!STBC_MRDY); //Wait Clock stable
SYCC2_SOSCE = 0; //Disable Sub Clock
STBC_TMD = 1;
```

注：上記のコードは、**Mode_Change** プロジェクトから抜粋したものです。付録を参照して

ください。

デバイスはリセット、タイムベースタイマ割込み、または外部割込みによりタイムベースタイマモードから解除されます。動作途中でタイムベースタイマモードとなった周辺機能はタイムベースタイマモードに遷移した時点の動作から再開します。よって、タイムベースタイマモードから復帰した場合、必要に応じて各周辺リソースを初期化してください。

6.7 時計モード

6.7.1 はじめに

時計モードでは、サブクロック、サブ CR クロック、および時計プリスケアラのみが機能します。このモードでは、CPU と周辺リソースの動作クロックは停止します。

6.7.2 時計モードにおける動作

時計モードでは、CPU と周辺リソースの動作クロックは停止します。デバイスは、時計プリスケアラ、外部割込み、および低電圧検出リセット以外のすべての機能を停止します。一方、時計モードへ遷移する直前に存在していたレジスタと RAM の内容は維持されます。

6.7.3 移行と解除

時計モードへの遷移はデバイスのクロックモードがサブクロックまたはサブ CR クロックの場合に、スタンバイ制御レジスタの時計ビット (STBC : TMD) に"1"を書き込みます。

時計モードへの遷移はデバイスのクロックモードがサブクロックモード、またはサブ CR クロックモードのときのみ遷移できます。

サンプルコードは、時計モードへ遷移させる STBC の設定例です。

```
SYCC2_SOSCE = 1;           //Enable Sub Clock
IO_SYCC2.bit.RCS1 = 0;
IO_SYCC2.bit.RCS0 = 1;
WATR = 0xF0;               //(215-2)/FCL About 1.00s
while (!SYCC_SRDY);
STBC_TMD = 1;  // Causes the device to enter time base timer mode
```

注：上記のコードは、Mode_Change プロジェクトから抜粋したものです。付録を参照してください。

デバイスは、リセット、時計割込み、または外部割込みに対応して時計モードから解除されます。動作途中で時計モードとなった周辺機能は時計モードに遷移した時点の動作から再

開します。よって、時計モードから復帰した場合、必要に応じて各周辺リソースを初期化してください。

6.8 スタンバイモードへの遷移サンプルコード

次のサンプルコードは、ストップモードや時計モードといったスタンバイモード間のモード遷移を実現できます(switchmode)は、任意のモードに変更できます。

```
//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()
{
    MCU_initialization();
    while(1)
    {
        switch (switchmode)
        {
            case normal: break;
                SYCC2_MOSCE = 1;      //Enable Main Clock
                IO_SYCC2.bit.RCS1 = 1;
                IO_SYCC2.bit.RCS0 = 1;
                WATR = 0x0F;          // About 4.10 ms Wait Time
                while (!STBC_MRDY);   //Wait Clock stable

            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
                IO_SYCC2.bit.RCS1 = 0;
                IO_SYCC2.bit.RCS0 = 1;
                WATR = 0xF0;          //(215-2)/FCL About 1.00s
                while (!SYCC_SRDY);
                STBC_TMD = 1; //Causes the device to enter watch mode
                break;               //normal to watch mode
```



```

    case Timebase:
        SYCC2_MOSCE = 1;      //Enable Main Clock
        IO_SYCC2.bit.RCS1 = 1;
        IO_SYCC2.bit.RCS0 = 1;
        WATR = 0x0F;          // About 4.10 ms Wait Time
        while (!STBC_MRDY);    //Wait Clock stable
        SYCC2_SOSCE = 0;      //Disable Sub Clock

        STBC_TMD = 1; //Causes the device to enter timebase timer mode
        break;                //normal to timebase time mode
    default: break;
}

    WDTC=0x35;                //Clear watch dog timer
    vDelay(10);                //delay the time
    led_display();             //lighten three LED
}
}

```

注：上記のコードは, **Mode_Change** プロジェクトから抜粋したものです。付録を参照してください。

7 その他の情報

富士通マイクロコントローラ製品に関するその他の情報については、下記の **Web** サイトをご覧ください。

<http://jp.fujitsu.com/microelectronics/>

8 付録

8.1 図の索引

図 4.2-1 クロックコントローラのブロックダイアグラム	12
図 4.3-1 システムクロック制御レジスタ (SYCC) の構成	13
図 4.3-2 発振安定待ち時間設定レジスタ (WATR) の構成	13
図 4.3-3 スタンバイ制御レジスタ (STBC)	13
図 4.3-4 システムクロック制御レジスタ 2 (SYCC2) の構成	14
図 6.2-1 スタンバイモードとクロック供給状態	23
図 6.3-1 スタンバイモードとクロックモードの組み合わせ, および内部動作状態	24

8.2 サンプルコード

8.2.1 Project 1 Name: Basic_Initialization

NAME : Start.asm

FUNCTION : Initialization MCU

```

;=====
; F2MC-8FX Family SOFTUNE C Compiler sample startup routine,
; ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 2008
; LICENSED MATERIAL - PROGRAM PROPERTY OF FUJITSU LIMITED
;=====
; 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
        .IMPORT      LMENTOMEM
        .IMPORT      LMEMCLEAR
        .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
;-----
//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     LMENTOMEM
#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
#enddif

```

```

;-----
; reset vector
;-----

        .SECTION    RESET, CONST, LOCATE=0xFFFFC
        .DATA.B      0xFF
        .DATA.B      0
        .DATA.H      __start
        .END         __start

```

NAME:	vector.c
FUNCTION:	Interrupt level (priority) setting and Interrupt vector definition
<pre> ;***** ; ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 2008 ; LICENSED MATERIAL - PROGRAM PROPERTY OF FUJITSU LIMITED ;***** 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 // IRQ1: external interrupt ch.5 // IRQ2: external interrupt ch.2/ch.6 // IRQ3: external interrupt ch.3/ ch.7 ILR1 = 0xFF; // IRQ4: - // 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: - // IRQ10: - // IRQ11: - ILR3 = 0xFF; // IRQ12: - // IRQ13: - // IRQ14: 8/16-bit timer chl (upper) // IRQ15: - </pre>	


```

        ILR4 = 0xFF;    // IRQ16: -
                        // IRQ17: -
                        // IRQ18: 8/10-bit A/D-converter
                        // IRQ19: Time base timer
        ILR5 = 0xFF;    // IRQ20: Watch prescaler
                        // IRQ21: -
                        // IRQ22: 8/16-bit timer ch.1
                        // IRQ23: Flash
    }
//-----
//  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-bit A/D-converter
#pragma intvect DefaultIRQHandler 19 //IRQ19: Timebase timer
#pragma intvect DefaultIRQHandler 20 //IRQ20: Watch prescaler
#pragma intvect DefaultIRQHandler 22 //IRQ22: 8/16-bit timer ch.1

```

```
#pragma intvect DefaultIRQHandler 23 //IRQ23: Flash
__interrupt void DefaultIRQHandler (void)
{
    __DI ();                                // disable interrupts
    While (1)
        __wait_nop ();                      // halt the system
}
```

NAME: Main.c

Function: external transition interrupt as key input

```

/* -----*/
// THIS SAMPLE CODE IS PROVIDED AS IS. FUJITSU MICROELECTRONICS
// ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR OMISSIONS
// Date: 20081008          Version: 1.0
/* -----*/
/*****
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
    }
}

```

8.2.2 Project 2 Name: Clock_Change

NAME: Main.c

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

```

/* -----*/
// THIS SAMPLE CODE IS PROVIDED AS IS. FUJITSU MICROELECTRONICS
// ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR OMISSIONS
// Date: 20081008          Version: 1.0
/* -----*
/*****/
//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();

    /*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

/*-----*/
/*  SYSC Enable the External Clock input */
/*-----*/

    IO_SYSC.bit.PFSEL = 0;    //Enable the external Main clock input
    IO_SYSC.bit.PGSEL = 0;    //Enable the external Sub clock input

/*-----*/
//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
/*-----*/
    IO_SYCC.byte = 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
        IO_SYCC2.bit.RCS1 = 1;
        IO_SYCC2.bit.RCS0 = 1;
        //update Oscillation stabilization wait time
        //when Main Oscillation Clock FCH=4MHz
        WATR = 0X0F;
        // Oscillation stabilization wait time mean
        // (214-2)/FCH About 4.10 ms
        while (!STBC_MRDY);

        IO_SYCC2.byte = IO_SYCC2.byte & 0xF4;
        break;

    case SUB:
        // enable the sub-clock oscillation
        SYCC2_SOSCE = 1;
        // Clock Mode Selection, select the sub-clock mode
        IO_SYCC2.bit.RCS1 = 0;
        IO_SYCC2.bit.RCS0 = 1;
        //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);

        IO_SYCC2.byte = IO_SYCC2.byte & 0xF8;
        break;
    }
```

```

case MAIN_CR:
    // enable the main CR clock oscillation
    SYCC2_MCRE = 1;
    // Clock Mode Selection, select the main CR clock mode
    IO_SYCC2.bit.RCS1 = 1;
    IO_SYCC2.bit.RCS0 = 0;
    //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);

    IO_SYCC2.byte = IO_SYCC2.byte & 0xF1;
    break;

case SUB_CR:
    // enable the Sub-CR Clock oscillation
    SYCC2_SCRE = 1;
    // Clock Mode Selection, select the Sub-CR Clock mode
    IO_SYCC2.bit.RCS1 = 0;
    IO_SYCC2.bit.RCS0 = 0;
    //update the Oscillation stabilization wait time
    //FCRHS represents the main clock frequency
    //2^5/FCRL = Sub-CR oscillation stabilization wait time
    //Indicates sub-CR clock oscillation being stable
    while (!STBC_SCRDY);

    IO_SYCC2.byte = IO_SYCC2.byte & 0xF2;
    break;

default:
    switchclock = MAIN;
    SYCC2=0x34;
    //update Oscillation stabilization wait time
    WATR = 0x03;
    //Indicates main-clock oscillation being stable

```



```

        while (!STBC_MRDY);
        break;
    }
    ++switchclock;
    if (switchclock > 3)
        switchclock = MAIN;
}

/*****
NAME:  main ()
FUNCTION:  lighten three LED
*****/

void main()
{
    MCU_initialization();
    while(1)
    {
        vDelay(100);           //delay the time
        led_display();         //lighten three LED
        WDTCT=0x35;            //Clear watch dog timer
    }
}

```

NAME: vector.c

FUNCTION: Interrupt level setting and interrupt vector definition

```

/* -----*/

// THIS SAMPLE CODE IS PROVIDED AS IS. FUJITSU MICROELECTRONICS
// ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR OMISSIONS
// Date: 20081008               Version: 1.0

/* -----*/

/*-----
VECTORS.C

- Interrupt level (priority) setting
- Interrupt vector definition

-----*/

#include "mb95200.h"

/*-----

InitIrqLevels()

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.

NOTE: value 0xFF disables the interrupt and value 0 sets highest priority.

NOTE: For all resource interrupts exists 3 Interrupt level registers
(ILRx).

Each register sets the level for 4 different resources (IRQx).

NOTE: This list is prepared for the 8FX chip MB95200.

Not all resources will be supported by all 8FX-devices

-----*/

```

```

void InitIrqLevels(void)
{
    /*      ILRx =                                IRQs defined by ILRx */

    ILR0 = 0xCF;  // IRQ0:  external interrupt ch0 | ch4
                  // IRQ1:  external interrupt ch1 | ch5
                  //      IRQ2:      external interrupt ch2 | ch6
    <<<---level 00
                  // IRQ3:  external interrupt ch3 | ch7

    ILR1 = 0xFF;  // IRQ4:  -
                  // IRQ5:  8/16-bit timer ch0 (lower)
                  // IRQ6:  8/16-bit timer ch0 (upper)
                  // IRQ7:  LIN-UART (reception)

    ILR2 = 0xFF;  // IRQ8:  LIN-UART (transmission)
                  // IRQ9:  -
                  // IRQ10: -
                  // IRQ11: -

    ILR3 = 0xFF;  // IRQ12: -
                  // IRQ13: -
                  // IRQ14: 8/16-bit timer ch1 (upper)
                  // IRQ15: -

    ILR4 = 0xFF;  // IRQ16: -
                  // IRQ17: -
                  // IRQ18: 8/10-bit A/D-converter
                  // IRQ19: Timebase timer

    ILR5 = 0xFF;  // IRQ20: Watch prescaler
                  // IRQ21: -

```

```

        // IRQ22: 8/16-bit timer ch1 (lower)
        // IRQ23: Flash
    }

/*-----
    Prototypes

    Add your own prototypes here. Each vector definition needs is proto-
    type. Either do it here or include a header file containing them.
    -----*/
__interrupt void DefaultIRQHandler (void);
__interrupt void external_int06 (void);
/*-----

    Vector definiton

    Use following statements to define vectors.

    All resource related vectors are predefined.

    Remaining software interrupts can be added hereas well.
    -----*/

#pragma intvect DefaultIRQHandler 0 // IRQ0: external interrupt ch0 |
ch4
#pragma intvect DefaultIRQHandler 1 // IRQ1: external interrupt ch1 |
ch5
#pragma intvect external_int06 2 // IRQ2: external interrupt ch2 | ch6
#pragma intvect DefaultIRQHandler 3 // IRQ3: external interrupt ch3 |
ch7

#pragma intvect DefaultIRQHandler 4 // IRQ4: -
#pragma intvect DefaultIRQHandler 5 // IRQ5: 8/16-bit timer ch0 (lower)
#pragma intvect DefaultIRQHandler 6 // IRQ6: 8/16-bit timer ch0 (upper)

```

```
#pragma intvect DefaultIRQHandler 7 // IRQ7: LIN-UART (reception)
#pragma intvect DefaultIRQHandler 8 // IRQ8: LIN-UART (transmission)
#pragma intvect DefaultIRQHandler 9 // IRQ9: -
#pragma intvect DefaultIRQHandler 10 // IRQ10: -
#pragma intvect DefaultIRQHandler 11 // IRQ11: -
#pragma intvect DefaultIRQHandler 12 // IRQ12: -
#pragma intvect DefaultIRQHandler 13 // IRQ13: -
#pragma intvect DefaultIRQHandler 14 // IRQ14: 8/16-bit timer ch1 (upper)
#pragma intvect DefaultIRQHandler 15 // IRQ15: -
#pragma intvect DefaultIRQHandler 16 // IRQ16: -
#pragma intvect DefaultIRQHandler 17 // IRQ17: -
#pragma intvect DefaultIRQHandler 18 // IRQ18: 8/10-bit A/D-converter
#pragma intvect DefaultIRQHandler 19 // IRQ19: Timebase timer
#pragma intvect DefaultIRQHandler 20 // IRQ20: Watch prescaler
#pragma intvect DefaultIRQHandler 21 // IRQ21: -
#pragma intvect DefaultIRQHandler 22 // IRQ22: 8/16-bit timer ch1 (lower)
#pragma intvect DefaultIRQHandler 23 // IRQ23: Flash

/*-----

DefaultIRQHandler()

This function is a placeholder for all vector definitions.

Either use your own placeholder or add necessary code here

(the real used resource interrupt handlers should be defined in the main.c).

-----*/

__interrupt void DefaultIRQHandler (void)
{
```

```
__DI(); // disable interrupts
while(1)
    __wait_nop(); // halt system
}
```

8.2.3 Project 3 Name: Mode_Change

NAME: Main.c

Function: standby mode change

```

/* -----*/
// THIS SAMPLE CODE IS PROVIDED AS IS. FUJITSU MICROELECTRONICS
// ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR OMISSIONS
// Date: 20081008          Version: 1.0
/* -----*/
/*****
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 watch 0x03
#define Timebase      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();
    /*-----*/
    /*  SYSC Enable the External Clock input */
    /*-----*/
        SYSC = 0x03;
    /*-----*/
    /*  External Main input */

```

```

/*-----*/

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 <= 4)
        switchmode++;
    if(switchmode >= 5)

```

```

        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;
                           SYCC2_MOSCE = 1;      //Enable Main Clock
                           IO_SYCC2.bit.RCS1 = 1;
                           IO_SYCC2.bit.RCS0 = 1;
                           WATR = 0x0F;           // About 4.10 ms Wait Time
                           while (!STBC_MRDY);    //Wait Clock stable

            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
        IO_SYCC2.bit.RCS1 = 0;
        IO_SYCC2.bit.RCS0 = 1;
        WATR = 0xF0;               //(215-2)/FCL About 1.00s
        while (!SYCC_SRDY);

        STBC_TMD = 1; //Causes the device to enter watch mode
        break;                     //normal to watch mode
    case Timebase:
        SYCC2_MOSCE = 1;           //Enable Main Clock
        IO_SYCC2.bit.RCS1 = 1;
        IO_SYCC2.bit.RCS0 = 1;
        WATR = 0x0F;               // About 4.10 ms Wait Time
        while (!STBC_MRDY);        //Wait Clock stable
        SYCC2_SOSCE = 0;           //Disable Sub Clock

        STBC_TMD = 1; //Causes the device to enter timebase timer
mode
        break;                     //normal to timebase time mode
    default: break;
}

    WDTCT=0x35;                    //Clear watch dog timer
    vDelay(10);                     //delay the time
    led_display();                  //lighten three LED
}
}

```

NAME: vector.c

FUNCTION: Interrupt level setting and interrupt vector definition

```

/* -----*/
// THIS SAMPLE CODE IS PROVIDED AS IS. FUJITSU MICROELECTRONICS
// ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR OMISSIONS
// Date: 20081008               Version: 1.0
/* -----*/
/*-----
VECTORS.C
- Interrupt level (priority) setting
- Interrupt vector definition
-----*/

#include "mb95200.h"

/*-----
InitIrqLevels()

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.

NOTE: value 0xFF disables the interrupt and value 0 sets highest priority.
NOTE: For all resource interrupts exists 3 Interrupt level registers
(ILRx).

Each register sets the level for 4 different resources (IRQx).
NOTE: This list is prepared for the 8FX chip MB95200.

Not all resources will be supported by all 8FX-devices
-----*/

void InitIrqLevels(void)
{
/*       ILRx =                               IRQs defined by ILRx */

```

```

ILR0 = 0x1F; // IRQ0: external interrupt ch0 | ch4
           // IRQ1: external interrupt ch1 | ch5
           //   IRQ2:   external interrupt ch2 | ch6
<<<---level 01
           //   IRQ3:   external interrupt ch3 | ch7
<<<---level 00

ILR1 = 0xFF; // IRQ4: -
           // IRQ5: 8/16-bit timer ch0 (lower)
           // IRQ6: 8/16-bit timer ch0 (upper)
           // IRQ7: LIN-UART (reception)
ILR2 = 0xFF; // IRQ8: LIN-UART (transmission)
           // IRQ9: -
           // IRQ10: -
           // IRQ11: -
ILR3 = 0xFF; // IRQ12: -
           // IRQ13: -
           // IRQ14: 8/16-bit timer ch1 (upper)
           // IRQ15: -
ILR4 = 0xFF; // IRQ16: -
           // IRQ17: -
           // IRQ18: 8/10-bit A/D-converter
           // IRQ19: Timebase timer
ILR5 = 0xFF; // IRQ20: Watch prescaler
           // IRQ21: -
           // IRQ22: 8/16-bit timer ch1 (lower)
           // IRQ23: Flash
}

/*-----
Prototypes

Add your own prototypes here. Each vector definition needs is proto-
type. 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 following statements to define vectors.
    All resource related vectors are predefined.
    Remaining software interrupts can be added hereas well.
-----*/

#pragma intvect DefaultIRQHandler 0    // IRQ0:  external interrupt ch0 |
ch4
#pragma intvect DefaultIRQHandler 1    // IRQ1:  external interrupt ch1 |
ch5
#pragma intvect external_int06 2      // IRQ2:  external interrupt ch2 | ch6
#pragma intvect external_int07 3      // IRQ2:  external interrupt ch3 | ch7


#pragma intvect DefaultIRQHandler 4    // IRQ4:  -
#pragma intvect DefaultIRQHandler 5    // IRQ5:  8/16-bit timer ch0 (lower)
#pragma intvect DefaultIRQHandler 6    // IRQ6:  8/16-bit timer ch0 (upper)
#pragma intvect DefaultIRQHandler 7    // IRQ7:  LIN-UART (reception)
#pragma intvect DefaultIRQHandler 8    // IRQ8:  LIN-UART (transmission)
#pragma intvect DefaultIRQHandler 9    // IRQ9:  -
#pragma intvect DefaultIRQHandler 10   // IRQ10: -
#pragma intvect DefaultIRQHandler 11   // IRQ11: -
#pragma intvect DefaultIRQHandler 12   // IRQ12: -
#pragma intvect DefaultIRQHandler 13   // IRQ13: -
#pragma intvect DefaultIRQHandler 14   // IRQ14: 8/16-bit timer ch1 (upper)
#pragma intvect DefaultIRQHandler 15   // IRQ15: -
#pragma intvect DefaultIRQHandler 16   // IRQ16: -
#pragma intvect DefaultIRQHandler 17   // IRQ17: -
#pragma intvect DefaultIRQHandler 18   // IRQ18: 8/10-bit A/D-converter
#pragma intvect DefaultIRQHandler 19   // IRQ19: Timebase timer
#pragma intvect DefaultIRQHandler 20   // IRQ20: Watch prescaler

```

```
#pragma intvect DefaultIRQHandler 21 // IRQ21: -
#pragma intvect DefaultIRQHandler 22 // IRQ22: 8/16-bit timer ch1 (lower)
#pragma intvect DefaultIRQHandler 23 // IRQ23: Flash

/*-----
   DefaultIRQHandler()

   This function is a placeholder for all vector definitions.
   Either use your own placeholder or add necessary code here
   (the real used resource interrupt handlers should be defined in the main.c).
   -----*/

__interrupt void DefaultIRQHandler (void)
{
    __DI(); // disable interrupts
    while(1)
        __wait_nop(); // halt system
}
```