お客様各位

資料中の「ラピスセミコンダクタ」等名称の ラピステクノロジー株式会社への変更

2020 年 10 月 1 日をもって、ラピスセミコンダクタ株式会社の LSI 事業部門は、ラピステクノロジー株式会社に分割承継されました。従いまして、本資料中にあります「ラピスセミコンダクタ株式会社」、「ラピスセミ」、「ラピス」といった表記に関しましては、全て「ラピステクノロジー株式会社」に読み替えて適用するものとさせていただきます。なお、会社名、会社商標、ロゴ等以外の製品に関する内容については、変更はありません。以上、ご理解の程よろしくお願いいたします。

2020年10月1日
ラピステクノロジー株式会社

Dear customer

LAPIS Semiconductor Co., Ltd. ("LAPIS Semiconductor"), on the 1st day of October, 2020, implemented the incorporation-type company split (shinsetsu-bunkatsu) in which LAPIS established a new company, LAPIS Technology Co., Ltd. (“LAPIS Technology”) and LAPIS Technology succeeded LAPIS Semiconductor’s LSI business.

Therefore, all references to "LAPIS Semiconductor Co., Ltd.", "LAPIS Semiconductor" and/or "LAPIS" in this document shall be replaced with "LAPIS Technology Co., Ltd."

Furthermore, there are no changes to the documents relating to our products other than the company name, the company trademark, logo, etc.

Thank you for your understanding.

LAPIS Technology Co., Ltd.
October 1, 2020

ROHM GROUP
LAPIS
SEMICONDUCTOR

ラピスセミコンダクタ 8/16 ビットマイコン

ソフトウェア安全設計

第 2 版　　発行日 2018 年 6 月 29 日

ROHM
SEMICONDUCTOR

ご注意

1) 本資料の記載内容は改良などのため予告なく変更することがあります。

2) ラピスセミコンダクタは常に品質・信頼性の向上に取り組んでおりますが，半導体製品は種々の要因で故障・誤作動する可能性があります。
万が一，本製品が故障・誤作動した場合であっても，その影響により人身事故，火災損害等が起こらないようご使用機器でのディレーティング，冗長設計，延焼防止，バックアップ，フェイルセーフ等の安全確保をお願いします。定格を超えたご使用や使用上の注意書が守られていない場合，いかなる責任もラピスセミコンダクタは負うものではありません。

3) 本資料に記載されております応用回路例やその定数などの情報につきましては，本製品の標準的な動作や使い方を説明するものです。したがいまして，量産設計をされる場合には，外部諸条件を考慮していただきますようお願いいたします。

4) 本資料に記載されております技術情報は，本製品の代表的動作および応用回路例などを示したものであり，それをもって，当該技術情報に関するラピスセミコンダクタまたは第三者の知的財産権その他の権利を許諾するものではありません。したがいまして，上記技術情報の使用に起因して第三者の権利にかかわる紛争が発生した場合，ラピスセミコンダクタはその責任を負うものではありません。

5) 本製品は，一般的な電子機器（AV機器，OA機器，通信機器，家電製品，アミューズメント機器など）および本資料に明示した用途への使用を意図しています。

6) 本資料に掲載されております製品は，耐放射線設計はなされておりません。

7) 本製品を下記のような特に高い信頼性が要求される機器等に使用される際には，ラピスセミコンダクタへ必ずご連絡の上，承諾を得てください。
・ 輸送機器（車載，船舶，鉄道など），幹線用通信機器，交通信号機器，防災・防犯装置，安全確保のための装置，医療機器，サーバー，太陽電池，送電システム

8) 本製品を極めて高い信頼性を要求される下記のような機器等には，使用しないでください。
・ 航空宇宙機器，原子力制御機器，海底中継機器

9) 本資料の記載に従わないために生じたいかなる事故，損害もラピスセミコンダクタはその責任を負うものではありません。

本資料に記載されております情報は，正確を期すため慎重に作成したものですが，万が一，当該情報の誤り・誤植に起因する損害がお客様に生じた場合においても，ラピスセミコンダクタはその責任を負うものではありません。

10) 本製品のご使用に際しては，RoHS 指令など適用される環境関連法令を遵守の上ご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して，ラピスセミコンダクタは一切の責任を負いません。本製品の RoHS 適合性などの詳細につきましては，セールス・オフィスまでお問合せください。

11) 本製品および本資料に記載の技術を輸出又は国外へ提供する際には，「外国為替及び外国貿易法」，
「米国輸出管理規則」など適用される輸出関連法令を遵守し，それらの定めにしたがって必要な手続を行ってください。

12) 本資料の一部または全部をラピスセミコンダクタの許可なく，転載・複写することを堅くお断りします。

ラピスセミコンダクタ株式会社
〒222-8575 神奈川県横浜市港北区新横浜 2-4-8
http://www.lapis-semi.com

目次

1. はじめに

本書は，ソフトウェア設計時の安全対策の考え方，および対処方法についてまとめたものです。
本書では，以下のラピスセミコンダクタ 8/16 ビットマイコンを対象に記載しています。

ML610Q100
ML610Q300
ML610Q400
ML620Q100
ML620Q500
ML620Q400
ML62Q1000 シリーズ

1.1. ソフトウェア安全設計とは

マイコンが組み込まれたシステムにおいて，ノイズ等の影響によりマイコンが誤動作してしまうと，システムの誤動作や故障等につながる恐れがあります。
ソフトウェア安全設計とは，マイコンに組み込むソフトウェアにノイズ等の影響を受けにくくする対策を施すことによって，マイコンの誤動作を防止する，あるいはマイコンの異常を検知して正常動作に復帰させることを目的とするものです。

1.2. ソフトウェア対策一覧

以下に, 本書に記載するソフトウェア対策の一覧を示します。

【マイコンの誤動作を防ぐ】

以下の項目はいずれのマイコンでも適用できる項目となります。

No.	項 目	説 明	章
1	SFR のリフレッシュ	メインルーチンにて SFR 設定をリフレッシュする	2.1
2	STOP モードへの移行処理	外部割込み禁止状態での STOP モードへの移行を防止する	2.2

【マイコンの異常を検知し, 正常動作に復帰する】

前半の No.1～8 はいずれのマイコンでも適用できる項目となります。後半の No.9 以降は, 対象の機能が搭載されたマイコンに適用できる項目となります。

No.	項 目	説 明	章
1	WDT クリア処理	WDT カウンタクリアの多用を避ける	3.1
2	WDT 割込み処理	WDT 割込みが発生した場合は WDT リセットを発生させる	3.2
3	BRK 命令の処理	BRK 命令が実行されたら WDT リセットを発生させる	3.3
4	スタックポインタ (SP) の確認	スタックポインタ (SP) の異常を検知し WDT リセットを発生させる	3.4
5	ELEVEL の確認	ELEVEL の異常を検知し WDT リセットを発生させる	3.5
6	初期化ルーチンの確認	プログラムが正常に起動したことを確認する	3.6
7	未使用割込みの処理	未使用割込みが発生した場合 WDT リセットを発生させる	3.7
8	フラッシュ書き換え処理	誤動作によるフラッシュ誤消去･誤書き換えを防止する	3.8
9	現割込みレベルの確認 ※割込みレベル機能を搭載したマイコンのみ対象	割込みレベルの異常を検知し WDT リセットを発生させる ML620Q131 ～ ML620Q136 , ML620Q131B ～ ML620Q136B , ML620Q151A～ML620156A, ML620Q151B～ML620156B, および ML62Q1000 シリーズが該当します。 それ以外のマイコンは該当しません。	3.9
10	電圧レベル監視リセット有無の確認 ※電圧レベル監視機能を搭載したマイコンのみ対象	電圧レベル監視リセットの発生を検知し, WDT リセットを発生させる ML610Q418/419, ML610Q101/102, ML610Q111/112, ML620Q131 ～ ML620Q136 , ML620Q131B ～ ML620Q136B , ML620Q151A～ML620156A, ML620Q151B～ML620156B, および ML62Q1000 シリーズが該当します。 それ以外のマイコンは該当しません。	3.10
11	LSI 起動異常有無の確認 ※LSI 起動異常検知機能を搭載したマイコンのみ対象	LSI 起動時の異常発生を検知し, WDT リセットを発生させる。 ML62Q1000 シリーズが該当します。 それ以外のマイコンは該当しません。	3.11
12	コードオプションレジスタの確認 ※コードオプションレジスタ (SFR) を搭載したマイコンのみ対象	コードオプションレジスタの異常を検知し, WDT リセットを発生させる ML610Q418/419, ML610Q172/173, ML610Q174, ML610Q178, ML620Q151A～ML620156A, ML620Q151B～ML620156B が該当します。 それ以外のマイコンは該当しません。	3.12

1.3. 関連資料

本書に関連するマニュアル類を以下に示します。併せて参照してください。

- 各マイクロコントローラのユーザーズマニュアル
- nX-U8/100 コア インストラクションマニュアル
- nX-U16/100 コア インストラクションマニュアル
- MACU8 アセンブラパッケージ ユーザーズマニュアル
- CCU8 ユーザーズマニュアル
- CCU8 プログラミングガイド
- CCU8 ランゲージリファレンス
- DTU8 ユーザーズマニュアル
- IDEU8 ユーザーズマニュアル
- uEASE ユーザーズマニュアル
- EASE1000 ユーザーズマニュアル

2. ソフトウェア対策例（マイコンの誤動作を防ぐ）

ここでは, マイコンの誤動作を防ぐための対策例を示します。

2.1. SFRのリフレッシュ

2.1.1. ソフトウェア対策の概要

LSI の初期化不良により初期化ルーチンが処理されずにメインループ処理が始まってしまった場合, LSI の異常動作が検知できない可能性があります。このため, メインループ内で SFR およびマスタ・インタラプト・イネーブル・フラグ（MIE）を常に再設定することを推奨します。

2.1.2. プログラム記述例

●main.c

```
void main( void )
{
     (中略)

    for(;;) {   // メインループ
       (中略)
       /*-------------------------*/
       /*   メイン処理    */
       /*-------------------------*/

        Initialize_SFR();             // メインループ内で SFR の設定
       __EI();                        // MIE = 1

       (中略)

       }
}
```

2.2. STOPモードへの移行処理

2.2.1. ソフトウェア対策の概要

STOP モードを解除する外部割込みが許可されない状態で STOP モードに移行すると LSI はフリーズしてしまいます。このため, ストップコードアクセプタで STOP モードへの移行を許可した後に, 外部割込みを再設定し, STOP モードに移行することを推奨します。

2.2.2. プログラム記述例

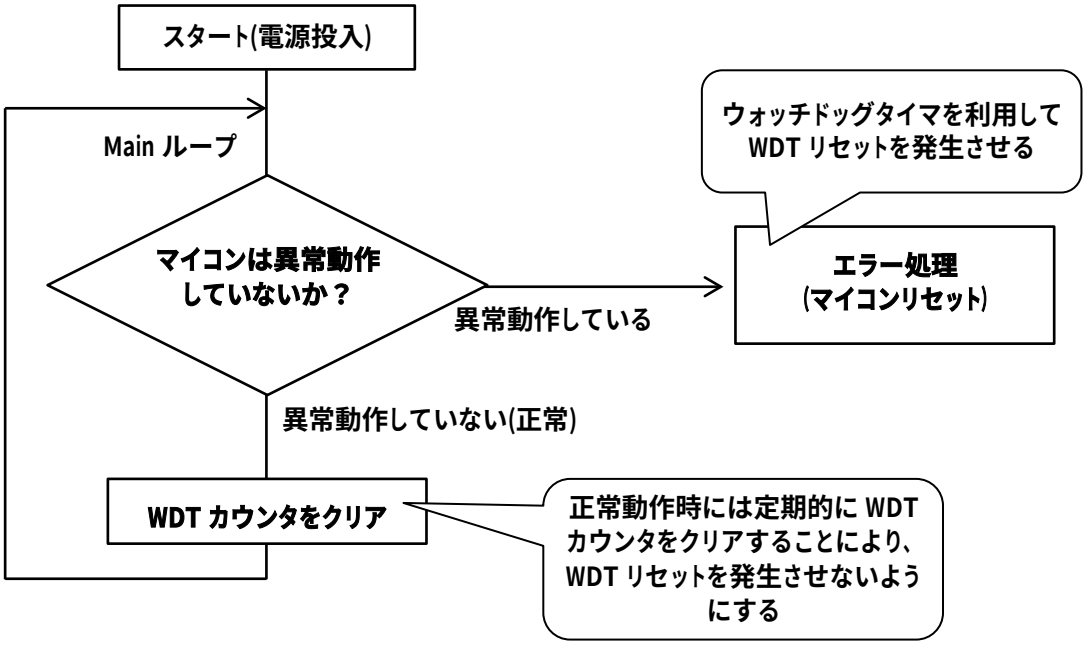
●main.c

```
void main( void )
{
     (中略)

     STPACP = 0x50;          // ストップコードアクセプタ許可
     STPACP = 0xA0;

     (外部割込み端子設定)
     (外部割込み許可)

     STP = 1;                // STOP モード移行
     __asm("nop");
     __asm("nop");

     (中略)
}
```

3. ソフトウェア対策例（マイコンの異常を検知し，復帰させる）

ここでは，マイコンの異常を検知し，正常動作に復帰させるための対策例を示します。

具体的には，レジスタやフラグ等を定期的にチェックすることでマイコンが異常動作していないかを確認し，異常動作している場合にはエラー処理としてマイコンをリセットします。マイコンのリセットには，ウォッチドッグタイマ（WDT）[*1] を利用して WDT リセットを発生させます。
WDT リセットを発生させることにより，CPU だけでなく周辺回路（ポートやタイマなど）やハードウェア（電源回路や発振回路など）などを初期化することができます。



[*1]：WDT は，マイコンが正常に動作しているかを監視するためのタイマです。
WDT のカウンタがオーバーフローすると，マイコンにリセットを発生させます。
WDT のカウンタがオーバーフローしないようにするためには，WDT カウンタをクリアします。

3.1. ウォッチドッグタイマ（WDT）のクリア処理

3.1.1. ソフトウェア対策の概要

ウォッチドッグタイマ（WDT）のクリア処理を多用した場合， LSI のフリーズを WDT で検出できない可能性が高くなってしまいます。このため，メインループ内の一箇所で WDT をクリアすることを推奨します。

3.1.2. プログラム記述例

●main.c

```
void main( void )
{
    （中略）
     for(;;) {   // メインループ
       /*------------------------*/
       /*   メイン処理           */
       /*------------------------*/
       （中略）

       wdt_clear();   // WDT はメインループ内の一箇所でクリア

    }
}
```

3.2. ウォッチドッグタイマ（WDT）割込み処理

3.2.1. ソフトウェア対策の概要

ウォッチドッグタイマカウンタの一度目のオーバフローによりノンマスカブルの WDT 割込みが発生し, 二度目のオーバフローにより WDT リセットが発生します。
一度目のオーバフローによる WDT 割込みで BRK 命令による CPU の初期化を行うと CPU は初期化されますが, 周辺機能は初期化されないため, 正常に復帰できない可能性があります。このため, 一度目の WDT 割込み処理で WDT リセットを発生させ確実に LSI を初期化することを推奨します。

3.2.2. プログラム記述例

●irq_Sample.c

```
extern void error_proc(void);
/*##############################################################################*/
/*#                          Prototype                                         #*/
/*##############################################################################*/
static void s_handlerWDTINT( void );
 （中略）
/*=== set Interrupt Vector ===*/
/* If enables multiple interrupts, */
/* specify '2' in the INTERRUPT category field. */
#pragma INTERRUPT   s_handlerWDTINT    0x0008  2
/*##############################################################################*/
/*#                         Interrupt handler                                  #*/
/*##############################################################################*/
static void s_handlerWDTINT( void )
{
       error_proc();                          // WDT 割込みが発生したらエラー処理へ移行
}
```

●main.c

```
/*******************************************
 * 異常を検出したら, 無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
   __DI();
   WDTMOD = 0x00;                             // WDT のオーバフロー時間を最短に設定
   while(1);
}
```

3.3. ブレーク（BRK）命令の処理

3.3.1. ソフトウェア対策の概要

プログラムが暴走しプログラム･カウンタ（PC）が未使用のプログラム･メモリ空間に分岐してしまった場合，未使用領域に設定された“0xFFFF”データ（BRK 命令コードに該当）により BRK 命令が実行されます。BRK 命令によって CPU は初期化されますが，周辺回路は初期化されません。このため，BRK 命令が実行された場合は WDT リセットを発生させ，周辺回路を含め確実に LSI を初期化することを推奨します。

3.3.2. プログラム記述例

●スタートアップファイル

```
;------------------------------------------------------------
;       ブレークリセットベクタ定義
;------------------------------------------------------------
        cseg  at 0:4h
        dw      $$brk_reset
;------------------------------------------------------------
;       brk 命令実行時の割込み処理
;------------------------------------------------------------
$$brk_reset:
        mov     r0,   #0
        st      r0,   WDTMOD    ; ウォッチドッグタイマ周期を最短に設定
        mov     psw,  #02h      ; 割込みレベルを 2 に設定
        bal     $               ; 無限ループし WDT リセットが発生
```

3.4. スタックポインタ（SP）の確認

3.4.1. ソフトウェア対策の概要

スタックポインタ（SP）が正常に初期化されなかった場合やプログラム動作中に何らかの原因でスタックポインタ（SP）が壊れてしまうとプログラムが正常に動作しません。このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的にスタックポインタ（SP）の値を確認することを推奨します。

3.4.2. プログラム記述例（スタートアップファイルのSP確認）

●スタートアップファイル

```
        type(ML621367)
        model   small
        romwindow       0, 0dfffh
        extrn    code: _main
        extrn    data near: _$$SP
          （中略）
        extrn    code: _error_proc
          （中略）
        public    $$start_up

        cseg     at 0:0h
        dw       offset _$$SP
          （中略）
$$start_up:
        bal      $begin
          （中略）
$begin:
;-------------------------------------------------------------------
;       スタックポインタの異常検知処理
;-------------------------------------------------------------------
        lea   0h
        l     er0, [ea]
        mov   er2 , sp
        cmp   er0 , er2                ; フラッシュ･メモリの 0h 番地の内容と SP を比較
        beq   _StackPointerCheck_OK    ; 0h 番地の内容と SP が一致したら正常
        b     _error_proc              ; 0h 番地の内容と SP が不一致の場合はエラー処理へ移行
_StackPointerCheck_OK:
          （略）
```

●main.c

```
/*****************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *****************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.4.3. プログラム記述例（メインループのSP確認）

●main.c

```
void main( void )
{
    unsigned short i_sp;
    i_sp = get_sp();        // マイコンの異常検知処理用：main 関数の最初で SP の値を取得 …①

    （中略）

    for(;;) {   // メインループ
        /*-------------------------------------------------*/
        /* スタックポインタの異常検知処理    */
        /*-------------------------------------------------*/
        if (get_sp() != i_sp) {      // SP が①の値と異なる場合は異常と判断し，
            error_proc();            // エラー処理へ移行
        }

        （中略）
    }
}
/*******************************************
 * スタックポインタを取得
 *******************************************/
unsigned short get_sp(void)
{
#asm
    mov  er0, sp
    rt
#endasm
    return 0;
}
/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.5. ELEVELの確認

3.5.1. ソフトウェア対策の概要

CPU のプログラム･ステータス･ワード（PSW）の ELEVEL が正常に初期化されなかった場合やプログラム動作中に何らかの原因で ELEVEL が壊れてしまうと割込みが正常に動作しません。このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的に ELEVEL の値を確認することを推奨します。

3.5.2. プログラム記述例（スタートアップファイルのELEVEL確認）

●スタートアップファイル

```
        type(ML621367)
        model   small
        romwindow    0, 0dfffh
        extrn   code: _main
        extrn   data near: _$$SP
          （中略）
        extrn   code: _error_proc
          （中略）

        public   $$start_up
        cseg    at 0:0h
        dw      offset _$$SP
          （中略）
$$start_up:
        bal     $begin
          （中略）
$begin:
;-----------------------------------------------------------------
;       ELEVEL の異常検知処理
;-----------------------------------------------------------------
        mov     r0,  psw
        and     r0,   #3H
        beq     _ELEVEL_Check_OK      ; ELEVEL が 0 の場合は正常
        b       _error_proc           ; ELEVEL が 0 以外の場合はエラー処理へ移行
_ELEVEL_Check_OK:
          （略）
```

●main.c

```
/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.5.3. プログラム記述例（メインループのELEVEL確認）

●main.c

```
void main( void )
{
     （中略）

    for(;;) {   // メインループ
        /*-----------------------------------*/
        /* ELEVEL の異常検知処理 */
        /*-----------------------------------*/
        if((get_psw() & 0x03) != 0) {        // ELEVEL が 0 以外の場合は異常と判断し，
              error_proc();                  // エラー処理へ移行
        }

         （中略）
    }
}
/*******************************************
 * PSW の値を取得
 *******************************************/
unsigned char get_psw(void)
{
#asm
    mov  r0,  psw
    rt
#endasm
    return 0;
}
/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
     __DI();
     WDTMOD = 0x00;                          // WDT のオーバフロー時間を最短に設定
     while(1);
}
```

3.6. 初期化ルーチンの確認

3.6.1. ソフトウェア対策の概要

LSI の初期化不良により初期化ルーチンが処理されずにメインループ処理が始まってしまった場合, LSI の異常動作が検知できない可能性があります。このため, プログラムの先頭にフラグを設定し, メインループ内でこのフラグを確認し, LSI が正常起動したことを確認することができます。

3.6.2. プログラム記述例

●main.c

```
#define MAGIC_CODE  (0x12345678)
long init_flag ;                    // 異常検知フラグ

void main( void )
{
   init_flag = MAGIC_CODE;          // フラグを設定

    (中略)

   for(;;) {   //メインループ
      /*--------------------------------*/
      /* フラグの異常検知処理 */
      /*--------------------------------*/
      if(init_flag != MAGIC_CODE) {// 先頭で設定したフラグの値が異なる場合,
          error_proc ();           // エラー処理へ移行
      }

       (中略)
   }
}
/*******************************************
 * 異常を検出したら, 無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.7. 未使用割込みの処理

3.7.1. ソフトウェア対策の概要

フェイルセーフとして未使用の割込みが発生した場合は LSI の誤動作と判断し, WDT リセットを発生させることを推奨します。

3.7.2. プログラム記述例

●irq_Sample.c

```
extern void error_proc(void);
/*****************************************************************************/
/*#                          Prototype                                      #*/
/*****************************************************************************/
static void s_handlerWDTINT( void );
 (中略)
static void s_handlerEXI0INT ( void );             // 未使用割込み
 (中略)
static void s_handlerLTBC0INT ( void );            // 未使用割込み
static void s_handlerLTBC1INT ( void );

/*=== set Interrupt Vector ===*/
/* If enables multiple interrupts, */
/* specify '2' in the INTERRUPT category field. */
#pragma INTERRUPT   s_handlerWDTINT   0x0008  2
#pragma INTERRUPT   s_handlerEXI0INT  0x0010  2     // 未使用割込み
 (中略)
#pragma INTERRUPT   s_handlerLTBC0INT  0x0074  1    // 未使用割込み
#pragma INTERRUPT   s_handlerLTBC1INT  0x0078  1

/*****************************************************************************/
/*#                      Interrupt handler                                  #*/
/*****************************************************************************/
(中略)
static void s_handlerEXI0INT ( void )
{
       error_proc();    // 未使用割込みが発生したらエラー処理へ移行
}
(中略)
static void s_handlerLTBC0INT ( void )
{
       error_proc();    //未使用割込みが発生したらエラー処理へ移行

}
static void s_handlerLTBC1INT ( void )
{
      // 割込み処理
}
```

●main.c

```
/*****************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *****************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.8. フラッシュ・メモリ書換え処理

3.8.1. ソフトウェア対策の概要

フラッシュ･メモリの誤書き換えを防止するため，フラッシュ･アクセプタ（FLASHACP）を許可した後に，書き換え対象アドレスを設定し，また正常にプログラムが実行されていることを確認した上で，フラッシュの消去･書き換え命令を実行することを推奨します。
なお，フラッシュ･メモリの誤書き換えを防止するための具体的なプログラムの記述方法は，弊社提供の「リファレンスソフトウェア」の「Software¥examples¥Flash¥DataFlashWriteSample」を参照してください。
フラッシュ･メモリの書き換え処理のフローについては，マイコンのユーザーズマニュアルを参照してください。

3.8.2. プログラム記述例

【電圧レベル監視リセット機能を持つマイコンの場合】

```
#define MAGIC_CODE          (0x12345678)
#define FLASH_MAGIC_CODE    (0x98ABCDEF)

long init_flag ;                    // 異常検知フラグ
long Flash_flag ;                   // フラッシュ･メモリ書き換えフラグ

●main.c
int main(void)
{
    init_flag = MAGIC_CODE;         // 異常検知フラグを設定

       （中略）

    /*********************************/
    /*    フラッシュ･メモリ消去処理     */
    /*********************************/
    if(init_flag != MAGIC_CODE) {   // 先頭で設定したフラグの値が異なる場合，
        error_proc();               // エラー処理へ移行
    }
    Flash_flag = FLASH_MAGIC_CODE;  // フラッシュ･メモリ書き換えフラグをセット

    FSELF = 1;                      // フラッシュ･メモリ書き換え許可

    FLASHACP = 0xFA;                // フラッシュアクセプタ許可
    FLASHACP = 0xF5;

    FLASHSEG = 0x1F;                // セグメントアドレス設定
    FLASHA = 0x0;                   // フラッシュアドレス設定

    if(Flash_flag != FLASH_MAGIC_CODE) {// フラッシュ･メモリ書き換えフラグの値が異なる場合，
        error_proc();               // エラー処理へ移行
    }

        （中略）
    FERS = 1;                       // ブロック消去
        （中略）

    Flash_flag = 0x00000000;        // フラッシュ･メモリ書き換えフラグをクリア
    FSELF = 0;                      // フラッシュ･メモリ書き換え禁止

        （中略）
}
```

【電圧レベル監視機能を持たないマイコンの場合】

電圧レベル監視機能を持たない機種の場合，低電圧･不安定な状態ではプログラムが誤動作する可能性が高くなります。このような場合、異常検知フラグに固定値を代入するだけでなく，フラグの値を更新していきながらチェックしていくことにより、より確実にフラッシュ･メモリの誤書き換えを防ぐことができます。

```
#define MAGIC_CODE          (0x12345678)
#define FLASH_MAGIC_CODE    (0x98ABCDEF)
#define FLASH_MAGIC_CODE2   (0x3CC35AA5)
#define CHECK_VALUE1        (FLASH_MAGIC_CODE ^ MAGIC_CODE)
#define CHECK_VALUE2        (FLASH_MAGIC_CODE2 ^ CHECK_VALUE1)

long init_flag ;                    // 異常検知フラグ
long Flash_flag ;                   // フラッシュ･メモリ書き換えフラグ

●main.c
void main(void)
{
    init_flag = MAGIC_CODE;         // 異常検知フラグを設定

       (中略)

    Flash_flag ^= FLASH_MAGIC_CODE; // フラッシュ･メモリ書き換えフラグを更新

    /*********************************/
    /*    フラッシュ･メモリ消去処理     */
    /*********************************/
    if((Flash_flag ^ init_flag) != CHECK_VALUE1) {  // フラグの値が想定と異なる場合，
         error_proc();              // エラー処理へ移行
    }
    Flash_flag ^= FLASH_MAGIC_CODE2; // フラッシュ･メモリ書き換えフラグを更新

    FSELF = 1;                      // フラッシュ･メモリ書き換え許可

    FLASHACP = 0xFA;                // フラッシュアクセプタ許可
    FLASHACP = 0xF5;

    FLASHSEG = 0x2;                 // セグメントアドレス設定
    FLASHA = 0x0;                   // フラッシュアドレス設定

    if((Flash_flag ^ init_flag) != CHECK_VALUE2) {  //フラグの値が想定と異なる場合，
         error_proc();              // エラー処理へ移行
    }

        (中略)
    FERS = 1;                       // ブロック消去 (セクタ消去の場合 FSERS=1;)
        (中略)


    Flash_flag = 0x00000000;        // フラッシュ･メモリ書き換えフラグをクリア
    FSELF = 0;                      // フラッシュ･メモリ書き換え禁止

        (中略)
```

3.9. 現割込みレベルの確認

3.9.1. ソフトウェア対策の概要

割込みレベルを設定できる機種において割込みレベルを使用しない場合，現在の割込みレベルを管理するためのレジスタ（CIL[*1]）が正常に初期化されなかったり，プログラム動作中に何らかの原因で CIL が壊れてしまうと割込みが正常に動作しません。このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的に CIL の値を確認することを推奨します。

[*1]：ML62Q1000 シリーズのレジスタ名称です。それ以外の機種では CILL として読み替えてください。

3.9.2. プログラム記述例（スタートアップファイルの CIL 確認）

●スタートアップファイル

```
       type(ML621367)
       model   small
       romwindow  0, 0dfffh
       extrn    code: _main
       extrn    data near: _$$SP
        （中略）
       extrn    code: _error_proc
        （中略）

       public   $$start_up
       cseg     at 0:0h
       dw       offset _$$SP
        （中略）
$$start_up:
       bal      $begin
        （中略）
$begin:
;--------------------------------------------------------------
;      CIL の異常検知処理
;--------------------------------------------------------------
      l        r0, CIL
      beq      _CIL_Check_OK         ; CIL が 0 の場合は正常
      b        _error_proc           ; CIL が 0 以外の場合はエラー処理へ移行
_CIL_Check_OK:
        （略）
```

●main.c

```
/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
    __DI();
    WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.9.3. プログラム記述例（メインループのCIL確認）

```
●main.c
void main( void )
{
     （中略）
    for(;;) {   // メインループ

       /*-----------------------------------*/
       /*  CIL の異常検知処理  */
       /*-----------------------------------*/
       if(CILL!=0) {                // CIL が 0 以外の場合は異常と判断し，
            error_proc();           // エラー処理へ移行
       }

       （中略）
    }
}
/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
     __DI();
     WDTMOD = 0x00;                 // WDT のオーバフロー時間を最短に設定
     while(1);
}
```

3.10. 電圧レベル監視リセット有無の確認

3.10.1. ソフトウェア対策の概要

電圧レベル監視（VLS：Voltage Level Supervisor）[*1] 機能の電圧レベル監視リセットは，電圧レベル監視リセットが発生した後も LSI の起動を繰り返す仕様です。電源電圧が低下し LSI が正常に動作しない電圧まで低下した後に電源が再起動した場合は LSI が正常に起動できない可能性があります。このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的にリセットステータスレジスタ（RSTAT）の VLS0R ビット[*2] を確認し，VLS0R ビットが“1”となっていた場合は，再度 WDT リセットを発生して確実に LSI を初期化することを推奨します。

[*1]：電圧レベル監視（VLS：Voltage Level Supervisor）機能は，ML62Q1000 シリーズでの機能の名称です。マイコンによっては，同様または類似の機能を電圧レベル検出回路（VLS），または LLD（Low Level Detector）などのように表現されているものもあります。ご使用のマイコンの機能名に読み替えてください。

[*2]：ML62Q1000 シリーズでのビットシンボル名です。マイコンによっては LLDR ビット，または VLSR ビットなどのように表現されているものもあります。ご使用のマイコンのビットシンボル名に読み替えてください。

3.10.2. プログラム記述例（スタートアップファイルの VLS0R ビット確認）

●スタートアップファイル

```
        type(ML621367)
        model   small
        romwindow       0, 0dfffh
        extrn    code: _main
        extrn    data near: _$$SP
          （中略）
        extrn    code: _error_proc
          （中略）
        public   $$start_up
        cseg    at 0:0h
        dw      offset _$$SP
          （中略）
$$start_up:
        bal     $begin
          （中略）
$begin:
;-----------------------------------------------------------------
;       VLS0R の異常検知処理
;-----------------------------------------------------------------
        tb      VLS0R
        bz      _VLSR_Check_OK          ; VLS0R が 0 の場合は正常
                                        ; VLS0R が 1 の場合は異常と判断，
        sb      VLS0R                   ; VLS0R に 1 を書き込むことによりクリアし
        b       _error_proc             ; エラー処理へ移行
_VLSR_Check_OK:
          （略）
```

```
●main.c
/*****************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *****************************************/
void error_proc(void)
{
    __DI();
     WDTMOD = 0x00;                  // WDT のオーバフロー時間を最短に設定
    while(1);
}
```

3.10.3. プログラム記述例（メインループのVLS0R確認）

```
●main.c

void main( void )
{
    （中略）
   for(;;) {   // メインループ

      /*-----------------------------------------------*/
      /* VLS0R の異常検知処理  */
      /*-----------------------------------------------*/
      if ( VLS0R != 0 ) {           // VLS0R が 1 の場合は異常と判断，
           VLS0R=1;                 // VLS0R に 1 を書き込むことによりクリアし
           error_proc();            // エラー処理へ移行
      }

      （中略）
   }
}
/*****************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *****************************************/
void error_proc(void)
{
    __DI();
   WDTMOD = 0x00;                   // WDT のオーバフロー時間を最短に設定
   while(1);
}
```

3.11. LSI起動異常有無の確認

3.11.1. ソフトウェア対策の概要

ML62Q1000 シリーズのマイコンには, LSI 起動時の異常発生を検知する機能が搭載されており, LSI 起動時に異常が発生すると, RSTAT レジスタの INITE ビットに“1”がセットされます。
LSI 起動で異常が発生した場合, LSI が正常に動作できない可能性があります。このため, INITE ビットに“1”がセットされた場合の基本的な考え方としては, ポートに信号を出すなどして外部に異常を通知する, システムとしては可能な限り動作させない, ということになります。
具体的には, ある端子に信号を出力する, ストップモードに移行する, WDT リセットを繰り返し発生させる等です。
なお, INITE ビットに“1”がセットされた場合は, 電源の再投入, またはリセット端子への入力によりリセットを発生させて LSI を再起動してください。

以下のプログラム記述例では, プログラム動作開始時（スタートアップファイル）にリセットステータスレジスタ（RSTAT）の INITE ビットを確認し, INITE ビットが“1”となっていた場合は, ポートへ信号を出力後, プログラムをストップモードへ移行させる例を示しています。

3.11.2. プログラム記述例（スタートアップファイルのINITEビット確認）

●スタートアップファイル

```
        type(ML621367)
        model   small
        romwindow       0, 0dfffh
        extrn    code: _main
        extrn    data near: _$$SP
          （中略）
        extrn    code: _error_proc
          （中略）
        public   $$start_up
        cseg    at 0:0h
        dw      offset _$$SP
          （中略）
$$start_up:
        bal     $begin
          （中略）
$begin:
;---------------------------------------------------------------
;       INITE の異常検知処理
;---------------------------------------------------------------
        tb      INITE
        bz      _INITE_Check_OK       ; INITE が 0 の場合は正常
                                      ; INITE が 1 の場合は異常と判断し，
        b       _LSI_init_error       ; エラー処理へ移行
_INITE_Check_OK:
          （略）
```

```
●main.c
/*******************************************
 * 異常を検出したら，ポートへ信号を出力し，STOP モードへ移行
 *******************************************/
void LSI_init_error (void)
{
    __DI();
    STPACP = 0x50;          // ストップコードアクセプタ許可
    STPACP = 0xA0;

    (ポート出力設定)
    (ポートへの信号出力)

    STP = 1;                // STOP モードへ移行
    __asm("nop");
    __asm("nop");
}
```

3.11.3. プログラム記述例 (メインループのINITE確認)

```
●main.c

void main( void )
{
    (中略)
    for(;;) {   // メインループ

      /*-----------------------------------------------*/
      /* INITE の異常検知処理  */
      /*-----------------------------------------------*/
      if ( INITE != 0 ) {           // INITE が 1 の場合は異常と判断，
           LSI_init_error ();       // エラー処理へ移行
      }

      (中略)
    }
}
```

3.12. コードオプションレジスタの確認

3.12.1. ソフトウェア対策の概要

コードオプションレジスタ（CODEOP0）が正常に設定されなかった場合やプログラム動作中に何らかの原因で CODEOP0 が壊れてしまうと LSI が正常に動作しません。このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的に CODEOP0 の値を確認することを推奨します。

3.12.2. プログラム記述例（スタートアップファイルのCODEOP0 確認）

●スタートアップファイル

```
        type(ML620156B)
        model   small
        romwindow       0, 0dfffh
        extrn    code: _main
        extrn    data near: _$$SP
          （中略）
        extrn    code: _error_proc
          （中略）
        public   $$start_up
        cseg    at 0:0h
        dw      offset _$$SP
          （中略）
$$start_up:
        bal     $begin
          （中略）

;-----------------------------------------------------------------
;       コードオプションレジスタの異常検知処理
;-----------------------------------------------------------------
        lea      0fde0h                ; コードオプションのアドレス設定（ご使用のマイコンの
                                       ; コードオプションのアドレスを設定してください）
        l        r0, 8:[ea]            ; フラッシュ・メモリからコードオプションの値を取得
        l        r1, CODEOP0           ; CODEOP0 の値を取得
        cmp      r0, r1                ; コードオプションの値と CODEOP0 の値を比較
        beq      _CODEOP0_Check_OK     ; 値が一致した場合は正常
        b        _error_proc           ; 値が不一致の場合はエラー処理へ移行
_CODEOP0_Check_OK:
        st      r0, _CodeOptionVal     ; メインループでの CODEOP0 の異常検知処理のため，
                                       ; コードオプションの値を変数（RAM）に保存
          （略）
```

3.12.3. プログラム記述例（メインループのCODEOP0 確認）

●main.c

```
unsigned char CodeOptionVal;   // 異常検知用コードオプション保持変数
void main( void )
{
    （中略）
   for(;;) {   // メインループ

      /*-----------------------------------------------*/
      /* コードオプションの異常検知処理  */
      /*-----------------------------------------------*/
      if ( CODEOP0 != CodeOptionVal ) {     // CODEOP0 とスタートアップで取得した値が
           error_proc();                    //異なる場合は異常と判断，エラー処理へ移行
      }

      （中略）
   }
}

/*******************************************
 * 異常を検出したら，無限ループに入り WDT リセット
 *******************************************/
void error_proc(void)
{
    __DI();
   WDTMOD = 0x00;                           // WDT のオーバフロー時間を最短に設定
   while(1);
}
```

改版履歴

ドキュメント No.	発行日	ページ		変更内容
		改版前	改版後	
FJXT_MCU_SAFETYSOFT-01	2018.5.31	ー	ー	初版発行
FJXT_MCU_SAFETYSOFT-02	2018.6.29	17,18	17,18	3.8.2. プログラム記述例を修正
		23	23	3.11.1. ソフトウェア対策の概要を変更 変更前) LSI 起動で異常が発生した場合，LSI が正常に動作できない可能性があります。 このため，プログラム動作開始時（スタートアップファイル），およびメインルーチン処理中に定期的にリセットステータスレジスタ（RSTAT）の INITE ビットを確認し，INITE ビットが“1”となっていた場合は，再度 WDT リセットを発生して LSI を初期化することを推奨します。 変更後) LSI 起動で異常が発生した場合，LSI が正常に動作できない可能性があります。このため，INITE ビットに“1”がセットされた場合の基本的な考え方としては，ポートに信号を出すなどして外部に異常を通知する，システムとしては可能な限り動作させない，ということになります。 具体的には，ある端子に信号を出力する，ストップモードに移行する，WDT リセットを繰り返し発生させる等です。 なお，INITE ビットに“1”がセットされた場合は，電源の再投入，またはリセット端子への入力によりリセットを発生させて LSI を再起動してください。 以下のプログラム記述例では，プログラム動作開始時（スタートアップファイル）にリセットステータスレジスタ（RSTAT）の INITE ビットを確認し，INITE ビットが“1”となっていた場合は，ポートへ信号を出力後，プログラムをストップモードへ移行させる例を示しています。
		24	24	プログラム記述例を前頁の概要に合わせて変更
		ー	25.26	3.12. コードオプションレジスタの確認を追加