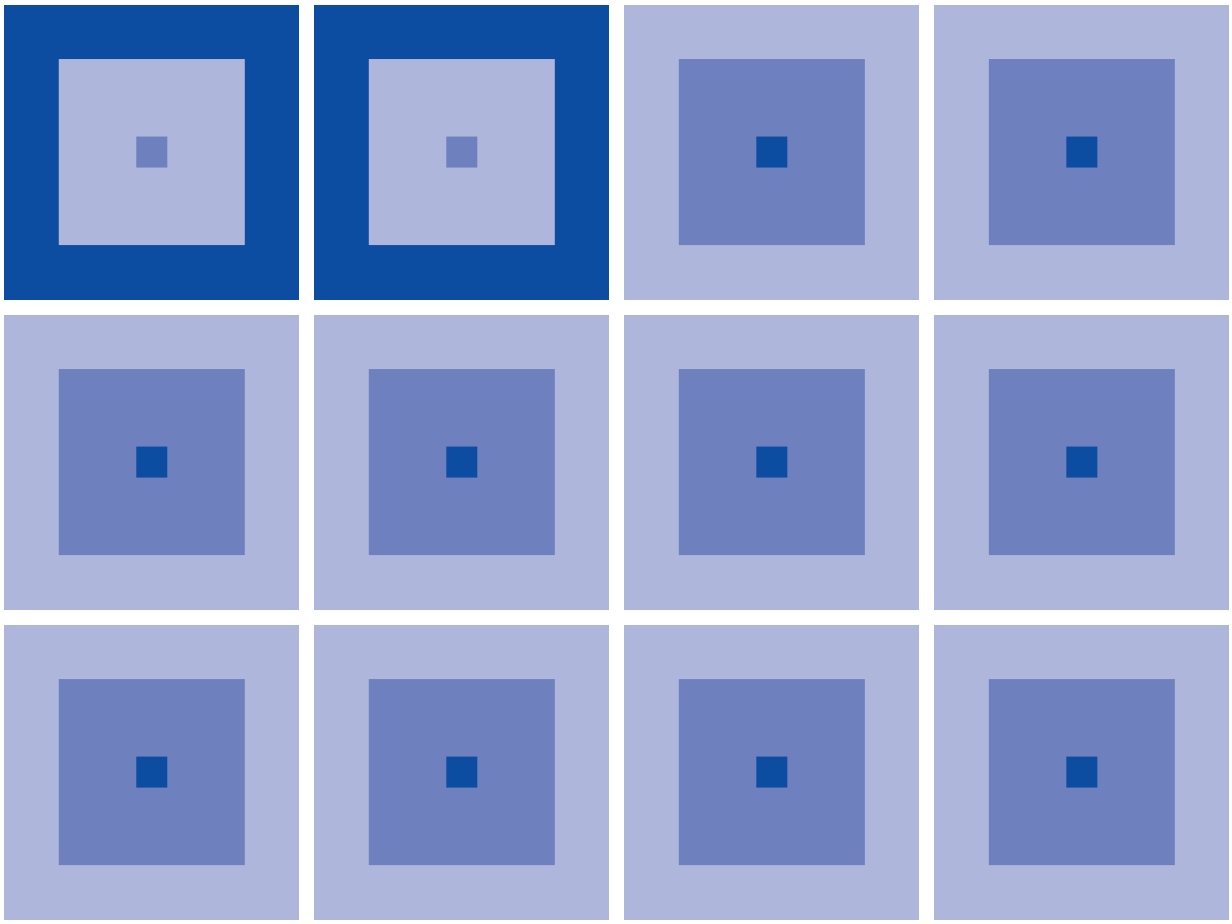
EPSON
EXCEED YOUR VISION

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER

S1C33 Family
スタートアップマニュアル



セイコーエプソン株式会社

本資料のご使用につきましては、次の点にご留意願います。

1. 本資料の内容については、予告なく変更することがあります。
2. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りします。
3. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これらに起因する第三者の権利(工業所有権を含む)侵害あるいは損害の発生に対し、弊社は如何なる保証を行うものではありません。また、本資料によって第三者または弊社の工業所有権の実施権の許諾を行うものではありません。
4. 特性表の数値の大小は、数直線上の大小関係で表しています。
5. 本資料に掲載されている製品のうち、「外国為替及び外国貿易法」に定める戦略物資に該当するものについては、輸出する場合、同法に基づく輸出許可が必要です。
6. 本資料に掲載されている製品は、一般民生用です。生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本(当該)製品をこれらの用途に用いた場合の如何なる責任についても負いかねます。

製品型番体系

●デバイス

S1　C　33209　F　00E1　00

- 00 ■ 梱包仕様
  - 00 : テープ&リール以外
  - 0A : TCP BL　2方向
  - 0B : テープ&リール BACK
  - 0C : TCP BR　2方向
  - 0D : TCP BT　2方向
  - 0E : TCP BD　2方向
  - 0F : テープ&リール FRONT
  - 0G : TCP BT　4方向
  - 0H : TCP BD　4方向
  - 0J : TCP SL　2方向
  - 0K : TCP SR　2方向
  - 0L : テープ&リール LEFT
  - 0M : TCP ST　2方向
  - 0N : TCP SD　2方向
  - 0P : TCP ST　4方向
  - 0Q : TCP SD　4方向
  - 0R : テープ&リール RIGHT
  - 99 : 梱包仕様未定
- 00E1 ■ 仕様
- F ■ 形状
  - [D: ベアチップ、F: QFP、B: BGA]
- 33209 ■ 機種番号
- C ■ 機種名称
  - [C: マイコン、デジタル製品]
- S1 ■ 製品分類
  - [S1: 半導体]

●開発ツール

S5U1　C　33000　H2　1　00

- 00 ■ 梱包仕様
  - [00: 標準梱包]
- 1 ■ バージョン
  - [1: Version 1]
- H2 ■ ツール種類
  - Hx : ICE
  - Dx : 評価ボード
  - Ex : ROMエミュレーションボード
  - Mx : 外部ROM用エミュレーションメモリ
  - Tx : 実装用ソケット
  - Cx : コンパイラパッケージ
  - Sx : ミドルウェアパッケージ
- 33000 ■ 対応機種番号
  - [33L01: S1C33L01用]
- C ■ ツール分類
  - [C: マイコン用]
- S5U1 ■ 製品分類
  - [S5U1: 半導体用開発ツール]

ー はじめに ー

S1C33 Familyマイクロプロセッサ/コントローラはセイコーエプソンオリジナルの CMOS 32ビットRISCコアを中心に、ROM、RAM、DMA、タイマ、SIO、PLL、A/D変換器などを1チップに内蔵しています。高速動作、低消費電流、省コードサイズ、積和機能を特長とし、OA機器から携帯機器まで幅広く対応できます。また、ASICマイコン、カスタムマイコンへの展開も可能となっています。
本書はS1C33 Familyマイクロプロセッサ/コントローラを使用するアプリケーション開発者向けのマニュアルで、S1C33チップ(主にS1C33301)の基本的な組み込み用プログラミング、周辺機能のプログラミング方法などを説明します。
本書をお読みになるには、以下の内容が予備知識として必要です。

- C言語(ANSI C準拠)に関する知識およびCソースプログラムの作成方法
- GNU C、binutil、GNU makeおよびGNUリンカ(ld)用リンカスクリプトに関する知識
- アセンブリ言語に関する一般的な知識
- Cコンパイラおよびアセンブラを使用したプログラム開発全般の基礎知識
- Windows 2000またはWindows XPの基本的な操作方法

なお、本書に掲載のプログラム例は、S1C33 Family C/C++コンパイラパッケージ(S5U1C33001C)のVer. 3以降に含まれるサンプルから抜粋しています。また、Cコンパイラの詳細な仕様については、“S5U1C33001C Manual”を参照してください。

〈マニュアルの構成〉

本マニュアルは、以下に示す4つの章から構成されています。
1章では、組み込み用ソフトウェアを作成するための基礎知識を掲載しています。
2章では、S1C33 MCUの基本的なプログラミングについて、サンプルプログラムを使用して説明しています。
3章では、S1C33 MCUの周辺回路のサンプルプログラムについて説明しています。
4章では、S1C33 MCUのプログラミングにおける技術的な項目や注意事項について説明しています。

〈関連マニュアル〉

関連マニュアルは以下のとおりです。

- S1C33000コアCPUマニュアル
- S1C33 Family C33 ADVコアCPUマニュアル
- S5U1C33000C Manual(S1C33 Family Cコンパイラパッケージ)(Ver. 4)
- S5U1C33001C Manual(S1C33 Family C/C++コンパイラパッケージ)(Ver. 3)
- S1C33 Family各機種別テクニカルマニュアル

〈WEB公開資料〉

以下の資料をWEBに公開しています。ただし、参照にはユーザIDとパスワードが必要です。
次のアドレスよりログインしてください。
http://www.epsondevice.com/webapp/MCUToolsDownload/entry.jsp

1. S5U1C33001C(GNU33 Ver. 2, Ver. 3)とS5U1C33000C(CC33)との違いについて
   CC33、GNU33 Ver. 2、GNU33 Ver. 3の相違点がまとめられています。
2. ツール対応表
   S1C33 FamilyのCPUコア、MCU、ICD、リファレンスボード、OS、ミドルウェアの対応についてまとめています。また、各コアにおけるオプション、ライブラリ、アセンブラなどの相違点をまとめています。
3. FAQ
   よくいただく質問と回答を記載しています。

ー 目 次 ー

1 組み込みの基礎知識

本章では、初めて組み込みソフトウェア開発をする方を対象に、最初に理解していただきたい、プログラムが動くための基本メカニズムやスタートアップルーチンによる初期化など、組み込みソフトウェア開発では極めて重要な考え方について説明します。

1.1 プログラムが動くための基本メカニズム

はじめに、S1C33プロセッサ(以下MCU)が動き始めるときの動作(基本メカニズム)について説明します(図1.1.1)。

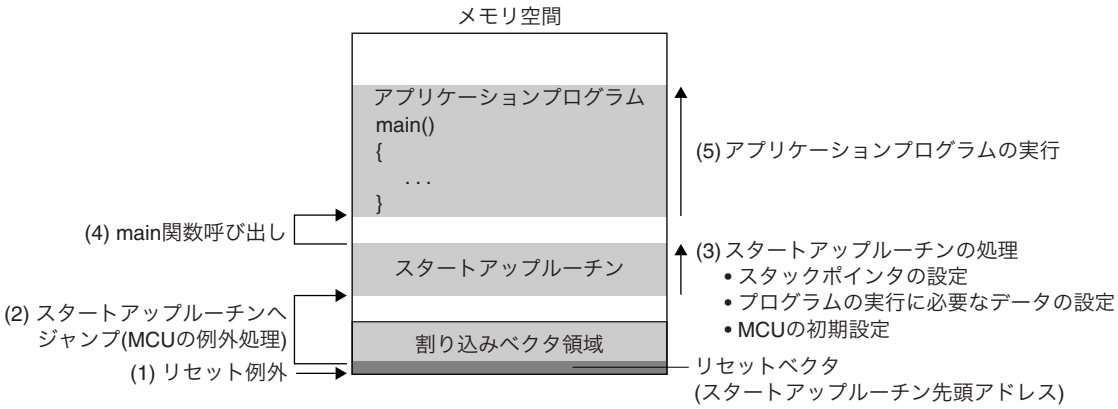


図1.1.1 S1C33プロセッサ起動時の基本メカニズム

(1) 組み込み機器に電源を投入するとリセット例外が発生し、MCUは割り込みベクタ領域(トラップテーブル)内のベースアドレスの内容を読み込みます。割り込みベクタ領域は、各種例外/割り込み処理ルーチン(関数)のアドレスを書き込んでおくテーブルで、例外や割り込み発生時はここからアドレスが読み出され、対応する処理ルーチンにジャンプするようになっています。ベースアドレスにはリセット時に処理するスタートアップルーチンへのベクタ(開始アドレス)を書き込んでおきます。

(2) MCUは(1)で読み出した内容(アドレス)にジャンプすることで、スタートアップ(初期化設定)ルーチンを呼び出します。

(3) スタートアップルーチンは、まずスタックの設定やプログラムの実行に必要な初期化処理を行います。

(4) 初期化処理を終了後、スタートアップルーチンはmain関数を呼び出します。

組み込みアプリケーションのプログラムはmain関数からではなく、その前段階としてスタートアップルーチンというものが存在します。組み込みソフトウェアを開発するにあたって、プログラムを動かすためにはスタートアップルーチンを理解することが必要です。

1.2 スタートアップ(初期化設定)ルーチン

スタートアップルーチンは、mainルーチンを実行する前に必要な初期化などを行います。
WindowsやLinux/UNIXのアプリケーションプログラムを開発するときは、プログラムの実行環境がいつも同じなので(毎回変更する必要がないので)、Cコンパイラが自動的にスタートアップルーチンをリンクしています。そのため、スタートアップルーチンを意識することがありません。
しかし、組み込み機器の場合、どのようなデバイスが接続されているか、メモリがどれだけ搭載されているかなどは機器に依存します。そのため、デバイスの初期化と一言でいっても機器によってそれぞれ処理が異なります。さらに、デバイスの初期化の方法についても使用するMCUによって異なるため、組み込みシステムにおいてスタートアップルーチンの役割は重要となります。
組み込みソフトウェアのスタートアップルーチンでは、一般に次のような処理を行います。

- スタックポインタの設定
- PSRの初期化と割り込み(IE)許可
- MCU内蔵周辺回路の初期化
  - BCU設定の初期化
  - 割り込み設定の初期化
  - I/Oレジスタ設定の初期化
- プログラムの実行に必要なデータの設定
  - ROM領域からRAM領域への初期値データの転送(.dataセクションのコピー)
  - 初期値を持たないメモリ領域の0クリア(.bssセクションの0クリア)

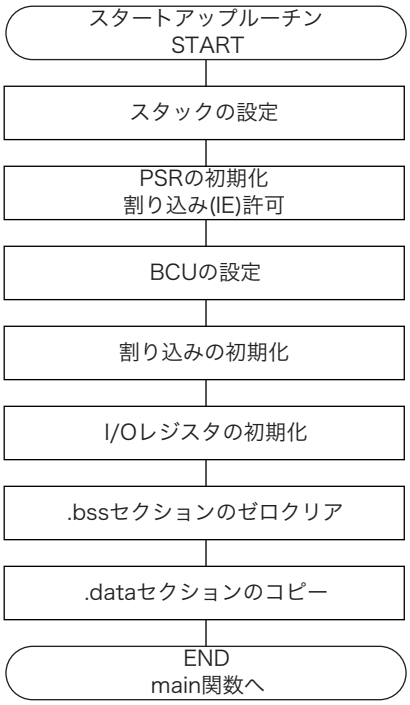


図1.2.1 スタートアップルーチン

スタックはサブルーチンや関数を呼び出す際に、処理中のデータや戻りアドレスなどを一時的に退避するのに使うRAM領域です。割り込みや例外もスタックを使用するため、スタートアップルーチンでスタック領域を確保します。
PSR(プロセッサステータスレジスタ)はCPUの状態を保持する32ビットレジスタで、実行した命令の結果によって変化します。プログラムの実行にも影響を与えるため、割り込みや例外の発生時には処理ルーチンへの分岐前にPSRの内容はスタックへ退避し、リターン後にPSRへ戻されます。スタートアップルーチンでは各ビットに0をセットして初期化を行います。またIE(割り込みイネーブル)に1をセットし、マスク可能な外部割り込みを許可します。

PSR (C33 STDコア)

31~12	11~8	7	6	5	4	3	2	1	0
Reserved	IL	MO	DS	–	IE	C	V	Z	N

IL: 割り込みレベル (0~15: 割り込み許可レベル)
MO: MACオーバーフローフラグ (1: オーバーフローあり, 0: なし)
DS: 被除数符号フラグ (1: 負, 0: 正)
IE: 割り込み許可 (1: 許可, 0: 禁止)
Z: ゼロフラグ (1: ゼロ, 0: ゼロ以外)
N: ネガティブフラグ (1: 負, 0: 正)
C: キャリーフラグ (1: キャリー/ボローあり, 0: なし)
V: オーバーフローフラグ (1: オーバーフローあり, 0: なし)

図1.2.2 PSR

これら以外にも、プログラムの実行には初期値を持たないグローバル変数の初期化が必要になります。これらの設定はリセットによって不定となるため、初期化もしくは0クリアする必要があります(.bssセクションのクリア)。また初期値を持つグローバル変数は、初期値データをROMからRAMにコピーする必要があります(.dataセクションのコピー)。
MCUの内蔵周辺機能や割り込みに関する設定は、使用する機能や割り込みに対応して初期化を行ってください。C言語の標準ライブラリを使用する場合は、main関数を呼び出す前にライブラリの初期化を行う必要があります。
そのほかに、ソフトウェアの実行に関わる初期化だけでなく、MCUやその他のハードウェアを使用するための初期化も必要です。これらの初期化は、MCUやハードウェアのマニュアルを参照しながらコーディングを行ってください。
組み込みアプリケーションでは、このようにスタートアップルーチンを実行した後にmain関数が呼び出されます(図1.2.1)。

以上のことを念頭におき、組み込み機器のプログラムの開発を行ってください。

このページはブランクです。

2 S1C33用プログラムの書き方

本章では、S1C33 Family共通のプログラムの書き方について説明します。

1章で説明したように、組み込みアプリケーションではmain関数を実行する前処理として、スタートアップルーチンを実行する必要があります。以下、スタートアップルーチンを含め、main関数を呼び出すまでの処理の流れについて、サンプルプログラムを参考に説明します。
説明内のサンプルプログラムや周辺機能は、特に指定したものを除きS1C33301の例です。機種によっては、機能や制御レジスタのアドレス等が異なる場合がありますので注意してください。

2.1 ベクタテーブルとスタートアップルーチン

S1C33用プログラムの実行には、ベクタテーブルとブートルーチンが最低限必要になります。
ベクタテーブルは、プログラム実行中に割り込みや例外が発生した場合に実行するトラップ(割り込み)処理ルーチンへのベクタ(分岐先アドレス)を配列として格納したテーブルです。そのため、トラップテーブルとも呼ばれます。

	ベクタアドレス
リセット	base + 0
Reserved	base + 4~12
ゼロ除算	base + 16
Reserved	base + 20
アドレス不整例外	base + 24
NMI	base + 28
Reserved	base + 32~44
ソフトウェア例外0	base + 48
:	:
ソフトウェア例外3	base + 60
マスク可能な外部割り込み0	base + 64
:	:
マスク可能な外部割り込み215	base + 924

base: トラップテーブル先頭アドレス
= 0x0080000 (内蔵ROMよりのブート時)
= 0x0c00000 (外部ROMよりのブート時)

図2.1.1 ベクタテーブルの構成

一方、スタートアップルーチンはブートルーチンとも呼ばれ、イニシャルリセット時のトラップ処理ルーチンとして呼び出されます。
S1C33 FamilyのMCUではパワーオン時にコールドリセットがかかり、リセット例外が発生します。C33コアはリセット例外に対応するベクタをベクタテーブルから読み出してそのアドレス(スタートアップルーチン)にジャンプし、main関数を実行するために必要な各種初期化の処理を開始します。
図2.1.2にS1C33 FamilyのMCUによるmain関数呼び出しまでの流れを示します。

図2.1.2 main関数呼び出しまでの実行フロー

C言語によるベクタテーブルとスタートアップルーチンの記述例として、GNU33¥sample_ide¥std¥dmt33301¥8timer¥src¥vector.c(8ビットタイマサンプルプログラム)の内容を以下に示します。

ベクタテーブルとスタートアップルーチン

```
/* Prototype */
void boot() __attribute__((interrupt_handler));
void div0() __attribute__((interrupt_handler));
void unalign() __attribute__((interrupt_handler));
void dummy() __attribute__((interrupt_handler));
void nmi() __attribute__((interrupt_handler));

extern void int_8timer0();
extern void int_8timer1();
extern void int_8timer3();
extern void init_bcu(void);
extern void init_int(void);
extern void init_sys(void);
extern void init_ram(void);
extern void exit(void);

/* vector table */
const unsigned long vector[] = {
        (unsigned long)boot,        // 0    0   Reset
        (unsigned long)dummy,       // 4    1   Reserved
        (unsigned long)dummy,       // 8    2   Reserved
        (unsigned long)dummy,       // 12   3   Reserved
        (unsigned long)div0,        // 16   4   Division by zero
        (unsigned long)dummy,       // 20   5   Reserved
        (unsigned long)unalign,     // 24   6   Address misaligned exception
        (unsigned long)nmi,         // 28   7   NMI
        (unsigned long)dummy,       // 32   8   Reserved
        (unsigned long)dummy,       // 36   9   Reserved
        (unsigned long)dummy,       // 40   10  Reserved
        (unsigned long)dummy,       // 44   11  Reserved
        (unsigned long)dummy,       // 48   12  Software exception 0
        (unsigned long)dummy,       // 52   13  Software exception 1
        (unsigned long)dummy,       // 56   14  Software exception 2
        (unsigned long)dummy,       // 60   15  Software exception 3
```

```
(unsigned long)dummy,       // 64   16  Port input interrupt 0
(unsigned long)dummy,       // 68   17  Port input interrupt 1
(unsigned long)dummy,       // 72   18  Port input interrupt 2
(unsigned long)dummy,       // 76   19  Port input interrupt 3
(unsigned long)dummy,       // 80   20  Key input interrupt 0
(unsigned long)dummy,       // 84   21  Key input interrupt 1
(unsigned long)dummy,       // 88   22  High-speed DMA Ch.0
(unsigned long)dummy,       // 92   23  High-speed DMA Ch.1
(unsigned long)dummy,       // 96   24  High-speed DMA Ch.2
(unsigned long)dummy,       // 100  25  High-speed DMA Ch.3
(unsigned long)dummy,       // 104  26  Intelligent DMA
(unsigned long)dummy,       // 108  27
(unsigned long)dummy,       // 112  28
(unsigned long)dummy,       // 116  29
(unsigned long)dummy,       // 120  30  16bit timer 0 comp B
(unsigned long)dummy,       // 124  31  16bit timer 0 comp A
(unsigned long)dummy,       // 128  32
(unsigned long)dummy,       // 132  33
(unsigned long)dummy,       // 136  34  16bit timer 1 comp B
(unsigned long)dummy,       // 140  35  16bit timer 1 comp A
(unsigned long)dummy,       // 144  36
(unsigned long)dummy,       // 148  37
(unsigned long)dummy,       // 152  38  16bit timer 2 comp B
(unsigned long)dummy,       // 156  39  16bit timer 2 comp A
(unsigned long)dummy,       // 160  40
(unsigned long)dummy,       // 164  41
(unsigned long)dummy,       // 168  42  16bit timer 3 comp B
(unsigned long)dummy,       // 172  43  16bit timer 3 comp A
(unsigned long)dummy,       // 176  44
(unsigned long)dummy,       // 180  45
(unsigned long)dummy,       // 184  46  16bit timer 4 comp B
(unsigned long)dummy,       // 188  47  16bit timer 4 comp A
(unsigned long)dummy,       // 192  48
(unsigned long)dummy,       // 196  49
(unsigned long)dummy,       // 200  50  16bit timer 5 comp B
(unsigned long)dummy,       // 204  51  16bit timer 5 comp A
(unsigned long)int_8timer0, // 208  52  8bit timer 0
(unsigned long)int_8timer1, // 212  53  8bit timer 1
(unsigned long)dummy,       // 216  54  8bit timer 2
(unsigned long)int_8timer3, // 220  55  8bit timer 3
(unsigned long)dummy,       // 224  56  Serial interface Ch.0
(unsigned long)dummy,       // 228  57  Serial interface Ch.0
(unsigned long)dummy,       // 232  58  Serial interface Ch.0
(unsigned long)dummy,       // 236  59
(unsigned long)dummy,       // 240  60  Serial interface Ch.1
(unsigned long)dummy,       // 244  61  Serial interface Ch.1
(unsigned long)dummy,       // 248  62  Serial interface Ch.1
(unsigned long)dummy,       // 252  63
(unsigned long)dummy,       // 256  64  A/D converter
(unsigned long)dummy,       // 260  65  RTC
(unsigned long)dummy,       // 264  66
(unsigned long)dummy,       // 268  67
(unsigned long)dummy,       // 272  68  Port input interrupt 4
(unsigned long)dummy,       // 276  69  Port input interrupt 5
(unsigned long)dummy,       // 280  70  Port input interrupt 6
(unsigned long)dummy,       // 284  71  Port input interrupt 7
(unsigned long)dummy,       // 288  72  8bit timer 4
(unsigned long)dummy,       // 292  73  8bit timer 5
(unsigned long)dummy,       // 296  74
(unsigned long)dummy,       // 300  75
(unsigned long)dummy,       // 304  76  Serial interface Ch.2
(unsigned long)dummy,       // 308  77  Serial interface Ch.2
(unsigned long)dummy,       // 312  78  Serial interface Ch.2
(unsigned long)dummy,       // 316  79
(unsigned long)dummy,       // 320  80  Serial interface Ch.3
(unsigned long)dummy,       // 324  81  Serial interface Ch.3
(unsigned long)dummy,       // 328  82  Serial interface Ch.3
(unsigned long)dummy,       // 332  83
(unsigned long)dummy,       // 336  84
(unsigned long)dummy,       // 340  85
(unsigned long)dummy,       // 344  86
(unsigned long)dummy,       // 348  87
```

```
        (unsigned long)dummy,       // 352  88
        (unsigned long)dummy,       // 356  89
        (unsigned long)dummy,       // 360  90
        (unsigned long)dummy,       // 364  91
        (unsigned long)dummy,       // 368  92
        (unsigned long)dummy,       // 372  93
        (unsigned long)dummy,       // 376  94
        (unsigned long)dummy,       // 380  95
        (unsigned long)dummy,       // 384  96
        (unsigned long)dummy,       // 388  97
        (unsigned long)dummy,       // 392  98
        (unsigned long)dummy,       // 396  99
        (unsigned long)dummy,       // 400  100
        (unsigned long)dummy,       // 404  101
        (unsigned long)dummy,       // 408  102
        (unsigned long)dummy,       // 412  103
        (unsigned long)dummy,       // 416  104
        (unsigned long)dummy,       // 420  105
        (unsigned long)dummy,       // 424  106
        (unsigned long)dummy,       // 428  107
        (unsigned long)dummy,       // 432  108
        (unsigned long)dummy,       // 436  109
        (unsigned long)dummy,       // 440  110
        (unsigned long)dummy,       // 444  111
        (unsigned long)dummy,       // 448  112 FIFO Serial interface Ch.0
        (unsigned long)dummy,       // 452  113 FIFO Serial interface Ch.0
        (unsigned long)dummy        // 456  114 FIFO Serial interface Ch.0
};

/*****************************************************************************
 * boot
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Boot program.
 *****************************************************************************/
void boot(void)
{
      asm("xld.w  %r15,0x2000");    // Set SP in end of 8KB internal RAM     (1)
      asm("ld.w   %sp,%r15");
      asm("ld.w   %r15,0x0");       // Initialize PSR                        (2)
      asm("ld.w   %psr,%r15");
      asm("ld.w   %r15,0b10000");   // Set PSR to interrupt enable           (3)
      asm("ld.w   %psr,%r15");

      init_bcu();                   // Initialize BCU on boot time           (4)
      init_int();                   // Initialize interrupt controller       (5)
      init_sys();                   // Initialize for sys.c                  (6)
      init_ram();                   // Initialize bss section & data section (7)

      main();                       // Call main                             (8)

      exit();                       // In last, go to exit                   (9)
}

/*****************************************************************************
 * dummy
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Dummy interrupt program.
 *****************************************************************************/
void dummy(void)
{
INT_LOOP:
      goto  INT_LOOP;                                                         (10)
}

/*****************************************************************************
 * div0
 *   Type :     void
 *   Ret val :  none
```

```
 *   Argument : void
 *   Function : Division by zero exception program.
 ********************************************************************************/
void div0(void)
{
INT_LOOP:
      goto  INT_LOOP;
}

/********************************************************************************
 * unalign
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Address misaligned exception program.
 ********************************************************************************/
void unalign(void)
{
INT_LOOP:
      goto  INT_LOOP;
}

/********************************************************************************
 * nmi
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : NMI interrupt program.
 ********************************************************************************/
void nmi(void)
{
INT_LOOP:
      goto  INT_LOOP;
}
```

ベクタテーブルはconst型の32ビット配列として定義しています。このようにして、32ビットの分岐先アドレスをROM上に格納することができます。各ベクタのコメント(//*x y z*)は、テーブル先頭からのオフセットアドレス(*x*)とベクタ番号(*y*)を示す10進数と割り込み処理の説明(*z*)です。上記のサンプルプログラムでは、使用する8ビットタイマの割り込み処理関数を記述しています。また、使用しない割り込みについてはダミールーチン(dummy)を用意し、そこへ飛ぶようになっています。
スタートアップルーチンは、リセット例外処理関数(boot)として呼び出されます。ブートルーチンではスタックポインタの初期化(1)、PSRの初期化と割り込み許可(2、3)、BCUの初期化(4)、割り込みコントローラの初期化(5)、入力用バッファの初期化(6)、初期値を持たないグローバル変数(.bssセクション)の0クリア、初期値を持つグローバル変数(.dataセクション)の初期値データのRAMへのコピー(7)を行った後、main関数をコールします(8)。必ずしもすべてを実行しなければならない訳ではありませんので、必要に応じて処理してください。ここでは最後にすべての処理が終了したことを確認するためにexit関数を用意しています(9)。また、不当な割り込みが入るとINT_LOOPに入るように設定(10)しておくと、デバッグ時にブレークポイントをセットすることでデバッグが容易になります。

2.2 割り込み処理

2.2.1 プロトタイプ宣言

C/C++コンパイラ(S5U1C33001C)では以下の形式で__attribute__((interrupt_handler))による関数のプロトタイプ宣言を行うことで、その関数を割り込み処理関数に設定することができます。

***<型> <関数名>* __attribute__((interrupt_handler));**

割り込み処理関数をプロトタイプ宣言することによって、汎用レジスタの退避命令とreti命令がC/C++コンパイラによって自動的に組み込まれます。また、乗除算や積和演算が行われた場合は%ahr(算術演算ハイレジスタ)と%alr(算術演算ローレジスタ)の退避命令も自動的に組み込まれます。特にreti命令は、割り込み発生時にハードウェアが退避させたPSRとPCの内容を戻して処理ルーチンからリターンするため、プロトタイプ宣言をしない場合は必ずreti命令を割り込み処理関数の最後に記述してください。
C言語での記述例は次のようになります。また、コンパイル後のアセンブラコードも記載します。
以下のサンプルプログラムは、GNU33¥sample_ide¥std¥dmt33301¥8timer¥src¥drv_8timer.c内の8ビットタイマ0の割り込み関数です。

Cコード

```
/* Prototype */
void int_8timer0() __attribute__((interrupt_handler));

/* 8bit timer0 interrupt function */
void int_8timer0(void)
{
      ***8ビットタイマ0割り込み処理***
}
```

アセンブラコード

```
/* 8bit timer0 interrupt function */
{
      pushn %r14          ┐
      ld.w  %0, %alr      │
      ld.w  %r1, %ahr     │ C/C++コンパイラによって組み込まれるコード
      pushn %r1           ┘

      ***8ビットタイマ0割り込み処理***

      popn  %1            ┐
      ld.w  %alr, %0      │
      ld.w  %ahr, %1      │ C/C++コンパイラによって組み込まれるコード
      popn  %r14          │
      reti                ┘
}
```

上記のように割り込み処理関数をプロトタイプ宣言すると、C/C++コンパイラが自動的にレジスタの退避やreti命令を組み込みます。不測の事態を避けるためにも、割り込み処理関数は必ずプロトタイプ宣言をするようにしてください。

2.2.2 NMI

割り込みにはマスク可能な割り込みとマスク不可能な割り込みがあります。
マスク不可能な割り込みをNMI(ノンマスカブル割り込み)といいます。NMIは他の割り込みに優先して、無条件にCPUに受け付けられます。
ただし、SP(スタックポインタ)設定前にNMIが発生すると誤動作するため、イニシャルリセット後はSPが設定されるまで、NMIを含む割り込みはすべてハードウェアによってマスクされます。

2.2.3 例外

例外はプログラムの実行中に何らかの異常が発生することをいい、S1C33 Familyでは0による除算や不正なメモリアクセスなどが例外に割り当てられています。C33コアは例外が発生するとベクタテーブルを参照し、例外処理を行います。
この中で、アドレス不正例外は16ビットのメモリリード/ライト時に奇数アドレスにアクセスした場合、または32ビットのメモリリード/ライト時にワード境界アドレス以外にアクセスした場合に発生します。C33コアではこのようなアクセスを禁止していますので注意してください。

2.2.4 ソフトウェア割り込み

マスク可能な割り込みにはハードウェア割り込みとソフトウェア割り込み(ソフトウェア例外)があります。タイマなど周辺回路による割り込みはハードウェア割り込みです。一方ソフトウェア割り込みは、int命令を実行することによって、任意の箇所で割り込みを発生させることができます。ソフトウェア割り込みが発生すると、C33コアはベクタテーブル内の対応するベクタ(ソフトウェア割り込み処理関数)へ飛び、割り込み処理を実行します。
以下にソフトウェア割り込みのサンプルプログラムを示します。

ソフトウェア割り込み例

```
void softint1 __attribute__((interrupt_handler));

int int_num;

const unsigned long vector[] = {
            :
(unsigned long)softint1,            // 52   13   software interrupt 1        (1)
            :
}

int main()
{
            :
      asm("int 1");                 // software interrupt 1                  (2)
            :
}

void softint1()
{
      int_num = 5;                                                            (3)
}
```

上記サンプルプログラムではmain関数の“asm("int 1")”がソフトウェア割り込み1を発生させ(2)、ベクタテーブルを介してソフトウェア割り込み1に対応する関数へ飛びます(1)。割り込み処理関数softint1では“int_num = 5”という割り込み処理を実行し、main関数へ戻ります(3)。

2.2.5 割り込み要因フラグ

マスク可能なハードウェア割り込みには、それぞれ割り込み要因フラグと割り込みイネーブルレジスタが用意されています。割り込みの発生には、割り込みイネーブルレジスタにより割り込みが許可されている状態で割り込み要因フラグがセットされる必要があります。
割り込み要因フラグは、対応する割り込み要因が発生すると割り込みが発生するしない(割り込みイネーブルレジスタの設定)にかかわらず、ハードウェアによって1にセットされます。この要因フラグによって割り込み要因の発生を知ることができます。また、セットされた割り込み要因フラグはソフトウェアでリセットする必要があります。割り込み発生後、割り込み要因フラグをリセットせずにメインルーチンに戻ると再度割り込みが発生しますので、必ず割り込み処理の中でリセットしてください。また、イニシャルリセット時も割り込み要因フラグは初期化されないため、使用する前に必ずソフトウェアでリセットしてください。
割り込み要因フラグのリセット方式には、リセットオンリーとリード/ライトの2種類があり、フラグセット/リセット方式選択レジスタ(0x4029F)で選択します。イニシャルリセット時にはリセットオンリー方式に設定されます。各方式の違いは以下のとおりです。

● リセットオンリー方式

割り込み要因フラグは1の書き込みによってリセットされます。0を書き込んだ場合、セットもリセットもされません。したがって、特定の要因フラグのみを確実にリセットすることができます。

● リード/ライト方式

割り込み要因フラグは他のレジスタと同様に、0を書き込むとリセットされ、1を書き込むとセットされます。この場合、0を書き込んだすべての要因フラグがリセットされ、1を書き込んでしまった場合は割り込み要因が発生してしまいますので注意してください。

図2.2.5.1に割り込み処理の基本フローを示します。

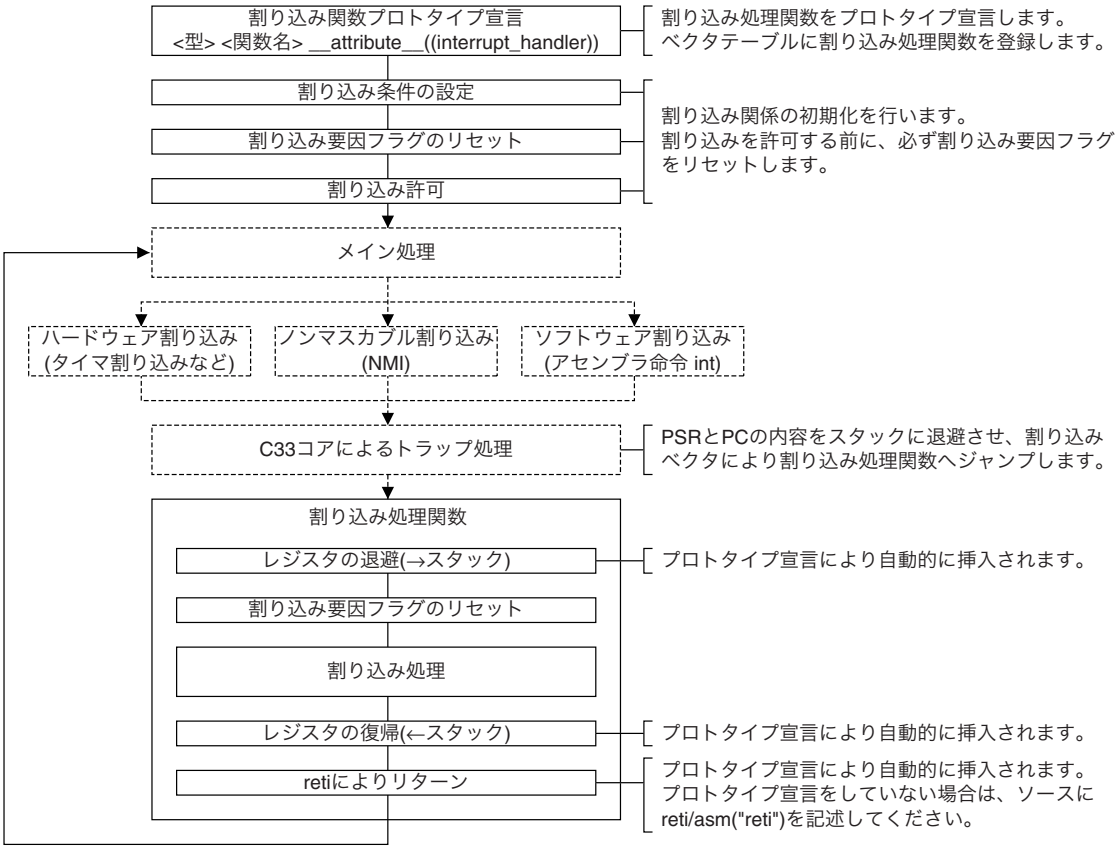


図2.2.5.1 基本的な割り込み処理フロー

2.3 Cコンパイラとコードの最適化

2.3.1 変数のアクセス

変数には外部変数(RAM上の変数、ROM上の定数値、static変数といった絶対アドレスを持つもの)とauto変数(スタック上に置かれる変数)があります。通常、外部変数はレジスタに変数の32ビットメモリアドレスをロードしてアクセスします。一方、auto変数はSP(スタックポインタ) + オフセットのみでアクセスできるため、以下のようなメリットがあります。

- 外部変数に比べてアクセスに必要な命令数が少なく、処理速度も速い。
- スタック上に一時的に置かれるため、RAMを常時占有せずRAMを節約できる。
- レジスタへの割り付けや不要なアクセスの消去といったCコンパイラの最適化の恩恵を受けやすい。

デメリットとしてはスタックが大きくなり、かつ上限値が予想しにくくなることが挙げられますが、あるルーチンで一時的に使用する変数はauto変数へ割り付ける方がより高いコード効率が得られます。

2.3.2 volatile修飾

コンパイラとプログラムによる変数へのアクセスに影響を及ぼす修飾子として、volatileがあります。一般的にCコンパイラはより高速で効率的なコードを作成するため、メモリへのアクセス回数を減らし、レジスタにロードした値をできる限り再利用して処理を行うことで最適化を図っています。ただし、この最適化によって、プログラムに記述されているメモリアクセスが実際には省略されてしまう可能性があります。この場合、ハードウェアによって変更された制御レジスタ値や、割り込み処理で変更された変数は、現在の値が反映されずに処理されてしまいます。volatile修飾子は、変数が変更されている可能性があるものとして、変数の参照のたびにその内容を取得し直すことをコンパイラに指定します。つまり、volatile宣言付きで定義した変数は、参照時に必ずメモリアクセスが発生し、最新の内容に更新された上で処理されます。
以下に、volatile宣言の有無によるアセンブラコードを比較した例を示します。

Cコード

```
int i_Normal_Flg;             //volatile宣言なし
volatile int i_Volatile_Flg;  //volatile宣言あり

while( 1 ){
      /* (1)volatile宣言なし */
      if( i_Normal_Flg == 1 )
      {
            break;
      }

while( 1 ){
      /* (2)volatile宣言あり */
      if( i_Volatile_Flg == 2 )
      {
            break;
      }
}
```

アセンブラコード

```
/* (1)volatile宣言なし */
cmp   %r4,0x1      ← レジスタの値を再利用して比較
jrne  0xff

/* (2)volatile宣言あり */
ld.w  %r4,[%r5]    ← 毎回メモリの値を参照して比較
cmp   %r4,0x2
jrne  0xfe
```

上記のようなプログラムの場合、たとえば割り込み処理関数内で変数の値が変更されたとき、(1)では実際のメモリを参照しないためメモリの値が更新されてもレジスタの値は更新されず、意図した結果が得られません。一方(2)ではメモリの値を確実に反映させることができます。

2.3.3 ポインタ型構造体・配列

前述のとおり、通常の外部変数へのアクセスはレジスタに変数の32ビットアドレスをロードして行うため、必要命令数が増えます。そこで、外部変数を構造体や配列で宣言します。これにより、構造体や配列の先頭をポインタとして各要素にはオフセットのみでアクセスできますので、外部変数への効率良いアクセスが可能となります。

2.3.4 Cコンパイラの最適化オプション

Cコンパイラは-O0、-O、-O2、-O3、-Osいずれか1つのスイッチの指定に従って最適化処理を行います。ただし、-O2/-O3は速度のみ、-Osはサイズのみを最適化の対象にしています。スイッチが指定されない場合は最適化は行われません。-Oの数値が上がるほどコード効率が高まりますが、デバッグ情報が一部出力されないなどの問題が発生することがありますので注意してください。正常に実行できないときは、最適化の数値を下げてください。また-O2/-O3は速度の最適化を目的としていますので、-Oよりもサイズが大きくなることがあります。
通常は、-Oでコンパイルすることを推奨します。
最適化オプションについての詳しい説明は、同梱マニュアル“S5U1C33001C Manual”の“6.3.2 コマンドラインオプション”の最適化を参照してください。

3 S1C33基本周辺機能のプログラミング

本章では、S1C33チップの基本的な周辺機能モジュールの初期設定を中心に、プログラム方法についてサンプルプログラムを参考にしながら説明します。
以下、サンプルプログラムや周辺機能は特に指定したものを除きS1C33301の例です。機種によっては、機能や制御レジスタアドレスなどが異なる場合がありますので注意してください。
なお以下のサンプルプログラムで使用されている定数は、GNU33¥sample_ide¥std¥dmt33301¥xxx¥includeにあるヘッダファイル内に宣言されています(xxxは周辺回路名)。

3.1 BCU

BCUでは、メモリエリアごとに接続するメモリや周辺I/Oデバイスの種類、サイズ、その他アクセス条件が設定できます。以下、GNU33¥sample_ide¥std¥dmt33301¥bcu¥srcにあるサンプルプログラムを参考に、SRAMとFLASHを接続した場合の設定について説明します。

● 外部メモリマップとチップイネーブル

アドレス空間のエリア4～10は外部システムに解放され、エリア個々に#CE(チップイネーブル)端子が設けられています。C33 STDコアでは#CE出力端子が7本に制限されていますがエリア11～18のメモリ空間もサポートしており、#CE信号とエリアの対応をDRAMタイミング設定レジスタ(0x48130)のCEFUNC[1:0](D[A:9])で切り換えられるようになっています。

表3.1.1 #CE出力の切り換え

端子	CEFUNC = 00	CEFUNC = 01	CEFUNC = 1x
#CE4	#CE4	#CE11	#CE11+#CE12
#CE5	#CE5	#CE15	#CE15+#CE16
#CE6	#CE6	#CE6	#CE7+#CE8
#CE7/#RAS0	#CE7/#RAS0	#CE13/#RAS2	#CE13/#RAS2
#CE8/#RAS1	#CE8/#RAS1	#CE14/#RAS3	#CE14/#RAS3
#CE9	#CE9	#CE17	#CE17+#CE18
#CE10EX	#CE10EX	#CE10EX	#CE9+#CE10EX

(デフォルト: CEFUNC = 00)

● SRAM、ROM、FLASH用の設定

SRAM、ROM、FLASH用の設定は、BCUレジスタで以下のエリアごとに行えます。

設定エリア

18–17、16–15、14–13、12–11、10–9、8–7、6、5–4

設定内容

a) デバイスサイズ: 8または16ビット
エリア6はアドレス範囲により8、16ビットが切り換わります。
b) ウェイト数: 0～7サイクル
書き込み時は、0ウェイトに設定しても1以上となります。
c) 出力ディセーブル遅延時間: 0.5～3.5サイクル
異なるエリア間のアクセス時に挿入されるウェイトサイクルです。

● FLASH、SRAM接続プログラム

S1C33301のエリア5と8にFLASH、エリア10にSRAMが接続されている場合のBCU設定のサンプルプログラムです。

BCU設定部

```
void init_bcu(void)
{
      /* Set area 4-5,6 0x4812a <- 0x1111 */
      /* Device size 16 bits, output disable delay 1.5, wait control 1 in area 4-5,6 */
      *(volatile unsigned short *)BCU_A4_A5_A6_ADDR =
      BCU_DFH_15 | BCU_WTH_1 | BCU_SZL_16 | BCU_DFL_15 | BCU_WTL_1;          (1)

      /* Set area 7-8 0x48128 <- 0x0011 */
      /* Device size 16 bits, output disable delay 1.5, wait control 1, */
      /* DRAM is not used in area 7-8 */
      *(volatile unsigned short *)BCU_A7_A8_ADDR =
      BCU_DRAH_NOT | BCU_DRAL_NOT | BCU_SZL_16 | BCU_DFL_15 | BCU_WTL_1;     (2)

      /* Set area 9-10 setting 0x48126 <- 0x0017 */
      /* Device size 16 bits, disable delay 1.5, wait control 1, burst ROM not used */
      *(volatile unsigned short *)BCU_A9_A10_ADDR =
      BCU_BROH_NOT | BCU_BROL_NOT | BCU_SZL_16 | BCU_DFL_15 | BCU_WTL_1;     (3)
}
```

このサンプルプログラムでは、エリア4–5と6(1)、7–8(2)、9–10(3)を、デバイスサイズ = 16ビット、ウェイトサイクル = 1、出力ディセーブル遅延時間 = 1.5サイクルに設定しています。
また、エリア7–8(2)ではDRAMを未使用、エリア9–10(3)ではバーストROMを未使用に設定しています。
なお、2個の8ビットSRAMで16ビット幅にしている場合は上記の設定で構いませんが、16ビットSRAMを使用している場合は外部インタフェース方式設定(0x4812E•D3)を#BSL(1)にする必要があります。また、一つの設定が適用されるエリアに8ビットと16ビットのデバイスを混在して使用することはできません。

3.2 8ビットプログラマブルタイマ

ここでは、GNU33¥sample_ide¥std¥dmt33301¥8timer¥src内のサンプルプログラムを例に、8ビットタイマを使用した基本的な割り込み制御プログラムについて説明します。
8ビットタイマは、8ビットプリセッタブルダウンカウンタのアンダーフロー信号を割り込みや周辺回路、IC外部に出力することができます。

● 8ビットタイマ割り込み制御プログラム

サンプルプログラムでは8ビットタイマ0、1、3からそれぞれ一定周期で割り込みを発生させ、その時点のカウンタ値を取得します。すべてのタイマの割り込みが発生後、取得したカウンタ値を表示します。ここではタイマ0のプログラムを抜粋して解説します。

8ビットタイマ割り込みベクタ部

```
      (unsigned long)int_8timer0,         // 208  52  8bit timer 0
      (unsigned long)int_8timer1,         // 212  53  8bit timer 1
      (unsigned long)dummy,               // 216  54  8bit timer 2
      (unsigned long)int_8timer3,         // 220  55  8bit timer 3
```

ベクタテーブルの設定
　割り込みルーチンをベクタテーブルに登録します。

8ビットタイマの初期設定部

```
/* Prototype */
void init_8timer0(void);

/*****************************************************************************
 * init_8timer0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize 8bit timer0.
 *****************************************************************************/
void init_8timer0(void)
{
      unsigned char temp;

      /* Set 8bit timer0 interrupt enable on interrupt controller 0x40275 */
      *(volatile unsigned char *)INT_E8TU_ADDR &= ~INT_E8TU0;                (1)

      /* Set 8bit timer0 control 0x40160, timer stop */
      *(volatile unsigned char *)T8P_PTRUN0_ADDR &= 0xfe;                    (2)

      /* Set 8bit timer0 prescaler 0x4014d */
      temp = *(volatile unsigned char *)PRESC_P8TS0_P8TS1_ADDR;              (3)
      temp &= 0xF0;
      temp |= PRESC_PTONL_ON | PRESC_CLKDIVL_SEL7;
      *(volatile unsigned char *)PRESC_P8TS0_P8TS1_ADDR = temp;

      /* Set 8bit timer0 reload data 0x40161 */
      *(volatile unsigned char *)T8P_RLD0_ADDR = 0x27;                       (4)
      // Set reload data (0x27 -> 1ms on OSC3 clock 40MHz)

      /* Set 8bit timer0 clock output off, preset and timer stop 0x40160 */
      *(volatile unsigned char *)T8P_PTRUN0_ADDR =
            T8P_PTOUT_OFF | T8P_PSET_ON | T8P_PTRUN_STOP;                    (5)

      /* Set 8bit timer0 interrupt CPU request on interrupt controller 0x40292 */
      *(volatile unsigned char *)INT_R16T5_R8TU_RS0_ADDR &= ~INT_R8TU0;      (6)
      // IDMA request disable and CPU request enable

      /* Set 8bit timer0 interrupt priority level 3 on interrupt controller 0x40269 */
      temp = *(volatile unsigned char *)INT_P8TM_PSIO0_ADDR;                 (7)
      temp &= 0xF0;
      temp |= INT_PRIL_LVL3;
      *(volatile unsigned char *)INT_P8TM_PSIO0_ADDR = temp;
```

```
      /* Reset 8bit timer0 interrupt factor flag on interrupt controller 0x40285 */
      *(volatile unsigned char *)INT_F8TU_ADDR = INT_F8TU0;                    (8)
      // Reset 8bit timer0 underflow interrupt factor flag

      /* Set 8bit timer0 interrupt enable on interrupt controller 0x40275 */
      *(volatile unsigned char *)INT_E8TU_ADDR |= INT_E8TU0;                   (9)
      // Set 8bit timer0 underflow interrupt enable
}

/************************************************************************
 * run_8timer
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned long reg       8bit timer run/stop register address
 *   Function : Run 8bit timer.
 ************************************************************************/
void run_8timer(unsigned long reg)
{
      /* run timer0 0x40160 */
      *(volatile unsigned char *)reg |= 0x01;                                  (10)
}
```

(1)割り込みの禁止
誤動作防止のため、8ビットタイマ0割り込みを禁止します。

(2)8ビットタイマカウンタ停止
タイマの入力クロックを設定するため、一度タイマを停止します。

(3)入力クロックの設定、タイマへのクロック供給を開始
プリスケーラの分周クロックを設定し、タイマへのクロック供給を開始します。
サンプルプログラムではプリスケーラで動作クロックを1/256に分周しています。

(4)リロードデータの設定
タイマがダウンカウントするデータを設定します。この値によってタイマがアンダーフローを発生する周期が決まります。アンダーフロー周期は次の式で求められます。

$$\text{アンダーフロー周期} = \frac{(\text{リロードデータ} + 1)}{(\text{プリスケーラ入力クロック周波数} \times \text{プリスケーラ分周比})}$$

サンプルプログラムではリロードデータを0x27に設定していますので、動作クロックが40MHzの場合、約250µsごとにアンダーフローが発生します。

(5)アンダーフロー制御の設定、リロードデータのプリセット
タイマのアンダーフロー信号をIC外部に出力するか設定します。サンプルプログラムでは外部出力はしません。
同時に(4)で設定したリロードデータをダウンカウンタへプリセットします。

(6)割り込み/IDMAリクエストの設定
アンダーフローを割り込み要因としてCPUに割り込みを要求するか、IDMAを起動するかを選択します。サンプルプログラムでは割り込み要求を行います。

(7)プライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(8)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(9)割り込みの許可
タイマの割り込みを許可します。これ以降タイマを動作させると一定周期で割り込みが発生します。

(10)タイマスタート
タイマをスタートさせます。タイマをストップさせるまで一定周期でアンダーフローを出力します。

割り込み処理部

```
/* Prototype */
void int_8timer0(void) __attribute__((interrupt_handler));

/************************************************************************
 * int_8timer0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : 8bit timer0 underflow interrupt function.
 *                Read 8bit timer3 counter data and stop 8bit timer0.
 ************************************************************************/
void int_8timer0(void)
{
      extern volatile unsigned char timer0;
      extern volatile int t8int0_flg;

      /* interrupt operation */
      timer0 = read_8timer_cnt(T8P_PTD0_ADDR);                          (1)
      stop_8timer(T8P_PTRUN0_ADDR);
      t8int0_flg = TRUE;

      /* Reset 8bit timer0 interrupt factor flag register 0x40285 */
      *(volatile unsigned char *)INT_F8TU_ADDR = INT_F8TU0;             (2)
}
```

(1)割り込み処理の実行
　割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。
　サンプルプログラムでは割り込みが発生した時点のカウンタ値を保存し、タイマを止めています。
　また、割り込み確認用のソフトウェアフラグをセットしています。

(2)割り込み要因フラグのクリア
　割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

3.3 16ビットプログラマブルタイマ

ここではGNU33¥sample_ide¥std¥dmt33301¥16timer¥src内のサンプルプログラムを例に、16ビットタイマを使用した基本的な割り込み制御プログラムについて説明します。
16ビットタイマにはコンペアデータレジスタAおよびBが設けられています。このレジスタの設定値とタイマ内のアップカウンタの値が一致するとコンペアAおよびB信号が出力され、割り込みやタイマ出力を制御します。このコンペアデータレジスタにより、割り込みの周期や出力信号のデューティ比をプログラマブルに設定できます。

● 16ビットタイマ割り込み制御プログラム

サンプルプログラムは16ビットタイマ0、1、2、3の順に割り込みが発生するように設定して4つのタイマを動作させます。タイマ0、1、2の割り込みが発生した時点でタイマ3のカウント値を記録して、割り込みを発生したタイマを停止します。タイマ3の割り込みが発生したところで、タイマ0、1、および2の割り込み発生時に記録したタイマ3のカウント値を表示します。以下、タイマ0のプログラムを抜粋してタイマの制御方法を解説します。

16ビットタイマ割り込みベクタ部

```
      (unsigned long)dummy,               // 120  30  16bit timer 0 comp B
      (unsigned long)int_16timer_c0,      // 124  31  16bit timer 0 comp A
      (unsigned long)dummy,               // 128  32
      (unsigned long)dummy,               // 132  33
      (unsigned long)int_16timer_u1,      // 136  34  16bit timer 1 comp B
      (unsigned long)dummy,               // 140  35  16bit timer 1 comp A
      (unsigned long)dummy,               // 144  36
      (unsigned long)dummy,               // 148  37
      (unsigned long)dummy,               // 152  38  16bit timer 2 comp B
      (unsigned long)int_16timer_c2,      // 156  39  16bit timer 2 comp A
      (unsigned long)dummy,               // 160  40
      (unsigned long)dummy,               // 164  41
      (unsigned long)int_16timer_u3,      // 168  42  16bit timer 3 comp B
      (unsigned long)dummy,               // 172  43  16bit timer 3 comp A
```

ベクタテーブルの設定
　割り込みルーチンをベクタテーブルに登録します。

16ビットタイマ初期設定部

```
/* Prototype */
void init_16timer0(void);

/*****************************************************************************
 * init_16timer0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize 16bit timer0.
 *****************************************************************************/
void init_16timer0(void)
{
      unsigned char temp;

      /* Set 16bit timer0 interrupt enable 0x40272, compare A interrupt disable */
      *(volatile unsigned char *)INT_E16T0_E16T1_ADDR &= ~INT_E16TC0;        (1)

      /* Set 16bit timer0 control 0x48186, timer stop */
      *(volatile unsigned char *)T16P_PRUN0_ADDR &= 0xfe;                   (2)

      /* Set 16bit timer0 prescaler 0x40147, prescaler on, CLK/256 */
      *(volatile unsigned char *)PRESC_P16TS0_ADDR =
            PRESC_PTONL_ON | PRESC_CLKDIVL_SEL5;                            (3)

      /* Set 16bit timer0 comparison match A data 0x48180, compare data A 0x4e */
      *(volatile unsigned short *)T16P_CR0A_ADDR = 0x9c;                    (4)

      /* Set 16bit timer1 comparison match B data 0x48182, compare data B 0x9c */
      *(volatile unsigned short *)T16P_CR0B_ADDR = 0x138;
```

```
        /* Set 16bit timer0 mode 0x48186 */
        /* fine-mode normal, compare buffer disable, output normal, clock internal, */
        /* clock output off, timer reset, timer stop */
        *(volatile unsigned char *)T16P_PRUN0_ADDR =
              T16P_SELFM_NOR | T16P_SELCRB_DIS | T16P_OUTINV_NOR | T16P_CKSL_INT |
              T16P_PTM_OFF | T16P_PSET_ON | T16P_PRUN_STOP;                    (5)

        /* Set 16bit timer0 interrupt CPU request 0x40290, compare A,B CPU request */
        temp = *(volatile unsigned char *)INT_RP0_RHDM_R16T0_ADDR;              (6)
        temp &= 0x3f;
        temp |= INT_RIDMA_DIS;
        *(volatile unsigned char *)INT_RP0_RHDM_R16T0_ADDR = temp;

        /* Set 16bit timer0 interrupt priority level 3 on interrupt controller 0x40266 */
        temp = *(volatile unsigned char *)INT_P16T0_P16T1_ADDR;                 (7)
        temp &= 0xf0;
        temp |= INT_PRIL_LVL3;
        *(volatile unsigned char *)INT_P16T0_P16T1_ADDR = temp;

        /* Reset 16bit timer0 interrupt factor flag 0x40282, */
        /* Reset compare A interrupt flag */
        *(volatile unsigned char *)INT_F16T0_F16T1_ADDR = INT_F16TC0;           (8)

        /* Set 16bit timer0 interrupt enable 0x40272, compare A interrupt enable */
        *(volatile unsigned char *)INT_E16T0_E16T1_ADDR |= INT_E16TC0;          (9)

}

/*****************************************************************************
 * run_16timer
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned long reg       16bit timer run/stop register address
 *   Function : Run 16bit timer.
 *****************************************************************************/
void run_16timer(unsigned long reg)
{
        *(volatile unsigned char *)reg |= 0x01;                                 (10)
}
```

(1)割り込みの禁止
　誤動作防止のため、16ビットタイマ0割り込みを禁止します。

(2)16ビットタイマ0カウンタ停止
　タイマの入力クロックを設定するため、一度タイマを停止します。

(3)入力クロックの設定、タイマへのクロック供給を開始
　プリスケーラの分周クロックを設定し、タイマへのクロック供給を開始します。
　サンプルプログラムではプリスケーラで動作クロックを1/256に分周しています。

(4)コンペアデータの設定
　アップカウンタと比較するデータを設定します。
　コンペアデータA、Bとアップカウンタの値が一致すると、それぞれコンペアA割り込み、コンペアB割り込みが発生します。サンプルプログラムでは、コンペアデータAを0x9c、コンペアデータBを0x138に設定していますので、動作クロックが40MHzの場合は約1msでコンペアA割り込みが、約2msでコンペアB割り込みが発生します。コンペアB割り込みが発生するとアップカウンタはリセットされます。

(5)16ビットタイマ制御の設定
　16ビットタイマの外部出力や入力クロックの設定を行います。
　外部出力に関しては出力モードと信号反転の設定、外部出力開始の制御を行います。
　入力クロックに関しては内部/外部クロックの選択、Run/Stopの制御を行います。
　サンプルプログラムでは外部出力は行わず、入力クロックは内部クロックを選択しています。

(6)割り込み/IDMAリクエストの設定
コンペア信号を割り込み要因としてCPUに割り込みを要求するか、IDMAを起動するかを選択します。サンプルプログラムでは割り込み要求を行います。

(7)プライオリティレベルの設定
割り込みのプライオリティレベルを設定します。プライオリティレベルが高い割り込みが優先されます。サンプルプログラムではプライオリティレベルを3に設定しています。

(8)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(9)割り込みの許可
タイマの割り込みを許可します。これ以降タイマを動作させると一定周期で割り込みが発生します。

(10)タイマスタート
タイマをスタートさせます。ストップさせるまでタイマは一定周期でコンペア信号を出力します。

割り込み処理部

```
/* Prototype */
void int_16timer_c0(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_16timer_c0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : 16bit timer0 comparison match A interrupt function.
 *****************************************************************************/
void int_16timer_c0(void)
{
      extern volatile int timer0;

      /* interrupt operation */
      timer0 = read_16timer_cnt(T16P_TC3_ADDR);                          (1)
      stop_16timer(T16P_PRUN0_ADDR);

      /* Reset 16bit timer0 compare A interrupt factor flag */
      *(volatile unsigned char *)INT_F16T0_F16T1_ADDR = INT_F16TC0;       (2)
}
```

(1)割り込み処理の実行
割り込みが発生するとベクターテーブルで記述されている割り込み処理関数を実行します。
サンプルプログラムでは割り込みが発生した時点のカウンタ値を保存し、タイマを止めています。

(2)割り込み要因フラグのクリア
割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

3.4 ウォッチドッグタイマ

ここではGNU33¥sample_ide¥std¥dmt33301¥wdt¥src内のサンプルプログラムを例に、CPUの暴走を検出するウォッチドッグタイマの制御プログラムについて説明します。
ウォッチドッグタイマは16ビットタイマ0を用いて実現します。この機能を有効にすると、タイマ0のコンペアB信号によってNMIが発生します。つまり、ソフトウェアで定期的にタイマ0をリセットしてNMIが発生しないようにすることで、その処理ルーチンを通らないようなプログラムの暴走を検出することができます。

● ウォッチドッグタイマ制御プログラム

このサンプルプログラムではメイン処理ループを用意し、その中で16ビットタイマ0をリセットしています。
万一タイマ0がリセットされずにコンペアB信号が出力された場合はNMIが発生しますので、CPUが暴走したと見なしてNMI処理を行います。

ウォッチドッグタイマの初期設定部

```
/* Prototype */
void init_wdt(void);

/************************************************************************
 * init_wdt
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize WatchDogTimer function.
 ************************************************************************/
void init_wdt(void)
{
      /* watchdog timer write protection 0x40170, EWD write enable */
      *(volatile unsigned char *)WDT_WRWD_ADDR = WDT_WRWD_WRT;              (1)

      /* watchdog timer enable 0x40171, NMI enable */
      *(volatile unsigned char *)WDT_EWD_ADDR = WDT_EWD_ENABLE;             (2)
}
```

(1) ウォッチドッグタイマ書き込み保護の解除
不要なNMIの発生を防止するため、ウォッチドッグタイマイネーブル(EWD)ビットは書き込みが禁止されています。そこで、書込み保護レジスタに1をセットし、書き込みを許可します。

(2) ウォッチドッグタイマイネーブル
ウォッチドッグタイマ機能を有効にします。
これ以降16ビットタイマ0のコンペアB信号が出力されると、NMIが発生します。

メイン処理部

```
/*****************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : 16bit timer demonstration program.
 *****************************************************************************/
int main(void)
{
      for (;;)
      {
            /* Main process */
                  . . . . .                                                    (1)

            /* Reset 16bit timer0 */
            *(volatile unsigned char *)T16P_PRUN0_ADDR |= T16P_PSET_ON;       (2)
      }
}
```

(1) メイン処理

メインルーチンの処理を実行します。
サンプルプログラムでは、このメイン処理ループを繰り返し実行するものとしています。

(2) 16ビットタイマ0のリセット

メインルーチンの処理が終わったところで16ビットタイマ0をリセットします。リセットする前に16ビットタイマ0のコンペアB信号によってNMIが発生した場合は、CPUが暴走したと判断できます。

16ビットタイマの設定については“3.3 16ビットプログラマブルタイマ”を参考にしてください。
通常動作時にウォッチドッグタイマによるNMIが発生することのないように、コンペアマッチB信号の出力周期はメインルーチン処理の周期より長く設定しておく必要があります。

割り込み処理部

```
/* Prototype */
void nmi(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * nmi
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : NMI interrupt program.
 *****************************************************************************/
void nmi(void)
{
      /* Reset 16bit timer0 interrupt flag 0x40282 */
      *(volatile unsigned char *)INT_F16T0_F16T1_ADDR = INT_F16TU0;

      /* Stop 16bit timer0 */
      stop_16timer(T16P_PRUN0_ADDR);

      t16i0_flg = TRUE;

      write_str("*** NMI occurred by Watchdog timer ***¥n");
      write_str("¥n");
}
```

割り込み処理

CPUが暴走した際の処理を記述します。
サンプルプログラムでは16ビットタイマ0の割り込み要因フラグをリセットしてタイマを停止し、NMIが発生したかを判定するソフトウェアフラグをセットしています。

3.5 計時タイマ

ここではGNU33¥sample_ide¥std¥dmt33301¥ct¥src内のサンプルプログラムを例に、計時タイマの制御プログラムについて説明します。
計時タイマはOSC1分周信号を入力クロックとする時間カウンタを持っており、各カウンタのデータをソフトウェアによって読み出すことができます。またカウントアップによる割り込みと日時指定によるアラームを発生することができます。

● 計時タイマ制御プログラム

このサンプルプログラムは1Hzカウンタのカウントアップによる割り込みを制御します。以下、計時タイマの初期化設定と割り込み処理について解説します。

計時タイマ割り込みベクタ部

```
      (unsigned long)int_ct,        // 260  65  RTC
```

割り込みベクタの設定
　割り込みルーチンをベクタテーブルに登録します。

計時タイマ初期設定部

```
/* Prototype */
void init_ct(void);

/************************************************************************
 * init_ct
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize clock timer to use real time clock.
 ************************************************************************/
void init_ct(void)
{
      unsigned char temp;

      /* Set clock timer interrupt enable 0x40277, clock timer interrupt disable */
      *(volatile unsigned char *)INT_EADE_ECTM_EP4_ADDR &= 0xfd;             (1)

      /* Stop clock timer 0x40151 */
      *(volatile unsigned char *)CT_TCRUN_ADDR = 0x00;                       (2)

      /* Reset clock timer 0x40151 */
      *(volatile unsigned char *)CT_TCRUN_ADDR |= CT_TCRST_RST;              (3)

      /* Set clock timer interrupt control 0x40152, */
      /* timer interrupt 1hz, timer alarm disable, factor flag reset */
      *(volatile unsigned char *)CT_TCAF_ADDR =
            CT_TCISE_1HZ | CT_TCASE_NONE | CT_TCIF_RST | CT_TCAF_RST;        (4)

      /* Set clock timer interrupt priority level 3 on interrupt controller 0x4026b */
      temp = *(volatile unsigned char *)INT_PCTM_ADDR;                       (5)
      temp |= INT_PRIL_LVL3;
      *(volatile unsigned char *)INT_PCTM_ADDR = temp;

      /* Reset clock timer interrupt factor flag on interrupt controller 0x40287 */
      *(volatile unsigned char *)INT_FADE_FCTM_FP4_ADDR = INT_FCTM;          (6)
      // Reset clock timer interrupt factor flag

      /* Set clock timer interrupt enable on interrupt controller 0x40277 */
      *(volatile unsigned char *)INT_EADE_ECTM_EP4_ADDR |= INT_ECTM;         (7)
      // Set clock timer interrupt enable
}
```

```
/*****************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Clock timer demonstration program.
 *****************************************************************************/
void main()
{
      /* Run clock timer */
      *(volatile unsigned char *)CT_TCRUN_ADDR |= 0x01;                    (8)
}
```

(1)割り込みの禁止
　誤動作防止のため、計時タイマ割り込みを禁止します。

(2)計時タイマの停止
　誤動作防止のため、計時タイマを停止します。

(3)計時タイマカウンタをリセット
　計時タイマのカウンタをリセットします。計時タイマの各カウンタはソフトウェアでのみリセットされます。イニシャルリセットによっては、リセットされませんので注意してください。また動作中はリセットされず、停止中にのみ受け付けられます。
　リセット時、設定によっては割り込みが発生する可能性があります。したがって、先に計時タイマ割り込みを禁止してリセットしてください。

(4)割り込み要因の選択
　割り込み発生要因とする信号を選択します。
　32Hz、8Hz、2Hz、1Hz、1分、1時間、1日から選択した信号の立ち下がりエッジで割り込み要因が発生します。

(5)プライオリティレベルの設定
　割り込みのプライオリティレベルを設定します。プライオリティレベルが高い割り込みが優先されます。サンプルプログラムではプライオリティレベルを3に設定しています。

(6)割り込み要因フラグのリセット
　イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(7)割り込みの許可
　タイマの割り込みを許可します。これ以降タイマを動作させると指定周期で割り込みが発生します。

(8)計時タイマスタート
　計時タイマをスタートさせます。選択した割り込み周期で、もしくは指定したアラーム日時になると割り込みが発生します。

割り込み処理部

```
/* Prototype */
void int_ct(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_ct
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Clock timer interrupt function.
 *****************************************************************************/
void int_ct(void)
{
      extern volatile int ctint_flg;

      /* interrupt operation */
      ctint_flg = TRUE;                                                    (1)

      /* Reset clock timer interrupt factor flag 0x40287 */
      *(volatile unsigned char *)INT_FADE_FCTM_FP4_ADDR = INT_FCTM;        (2)
}
```

(1)割り込み処理の実行
割り込みが発生するとベクタテーブルで記述されている割り込み処理関数を実行します。
サンプルプログラムでは、割り込み発生の確認用に用意されたソフトウェアフラグをセットしています。

(2)割り込み要因フラグのリセット
割り込みの発生により割り込み要因フラグが1になっているので、これをリセットします。

3.6 シリアルインタフェース

ここでは、GNU33¥sample_ide¥std¥dmt33301¥sif_asyn/slv/mst内のサンプルプログラムを例にシリアルインタフェースの制御プログラムを説明します。
シリアルインタフェースはクロック同期式転送方式と調歩同期式転送方式に対応しています。クロック同期式転送では送受信ユニットに共通のクロックを使用し、そのクロックに同期させてデータを転送します。調歩同期式では各データの前後に同期のためのスタートビットとストップビットを付加して転送を行います。

● シリアルインタフェース(クロック同期式スレーブモード)プログラム

GNU33¥sample_ide¥std¥dmt33301¥sif_slv¥src内のサンプルプログラムでは、クロック同期式転送方式でスレーブ側(本チップ)からマスタ側へ、割り込みを用いてデータを連続転送します。ここでは、シリアルインタフェースCh.1のスレーブモードへの設定や、その他の初期設定について解説します。

シリアルインタフェース(クロック同期式スレーブモード)割り込みベクタ部

```
      (unsigned long)dummy,               // 240  60  Serial interface Ch.1
      (unsigned long)dummy,               // 244  61  Serial interface Ch.1
      (unsigned long)int_sif_empty,       // 248  62  Serial interface Ch.1
```

割り込みベクタの設定
　割り込みルーチンをベクタテーブルに登録します。

シリアルインタフェース(クロック同期式スレーブモード)初期設定部

```
/* Prototype */
void init_sync_sif1(void);

/*****************************************************************************
 * init_sync_sif1
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize synchronous serial channel 1.
 *****************************************************************************/
void init_sync_sif1(void)
{
      unsigned char temp;

      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.1 interrupt disable */
      *(volatile unsigned char *)INT_ES_ADDR &=
            ~(INT_ESTX1 | INT_ESRX1 | INT_ESERR1);                          (1)

      /* Set serial control 0x401e8, send disable, receive disable */
      *(volatile unsigned char *)SIF_SMD1_ADDR = SIF_TXEN_DIS | SIF_RXEN_DIS;   (2)

      /* Set serial interface I/O port 0x402d0, #SRDY1, #SCLK1, SOUT1, SIN1 */
      *(volatile unsigned char *)IO_CFP0_ADDR |=
            IO_CFP07_SRDY1 | IO_CFP06_SCLK1 | IO_CFP05_SOUT1 | IO_CFP04_SIN1;   (3)

      /* Set IrDA control 0x401e9, i/f mode normal */
      *(volatile unsigned char *)SIF_IRMD1_ADDR = SIF_IRMD_ORD;               (4)

      /* Set serial control 0x401e8, transfer-mode clock-syn slave */
      *(volatile unsigned char *)SIF_SMD1_ADDR = SIF_SMD_SLA;                 (5)

      /* Set serial control 0x401e8, send enable, receive disable, clock #SCLK */
      *(volatile unsigned char *)SIF_SMD1_ADDR |=
            SIF_TXEN_ENA | SIF_RXEN_DIS | SIF_SSCK_SCLK;                    (6)

      /* Clear serial status */
      *(volatile unsigned char *)SIF_RDBF1_ADDR = SIF_ERR_NON;                (7)

      /* Set serial interface ch.1 and A/D converter interrupt priority register */
      /* 0x4026a */
      temp = *(volatile unsigned char *)INT_PSIO1_PAD_ADDR;                   (8)
      temp &= 0xf0;
      temp |= INT_PRIL_LVL3;
```

```
        *(volatile unsigned char *)INT_PSIO1_PAD_ADDR = temp;

        /* Set serial ch.0,1 interrupt flag 0x40286, clear serial ch.1 interrupt flag */
        *(volatile unsigned char *)INT_FS_ADDR =
              INT_FSTX1 | INT_FSRX1 | INT_FSERR1;                                    (9)

        /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.1 interrupt enable */
        *(volatile unsigned char *)INT_ES_ADDR |=
              INT_ESTX1   | INT_ESRX1 | INT_ESERR1;                                  (10)
}
```

(1)割り込みの禁止
誤動作防止のため、シリアルインタフェースの割り込みを禁止します。

(2)送受信の禁止
動作中の設定変更は誤作動の原因になるため、シリアルインタフェースの送受信を禁止します。

(3)入出力端子の設定
入出力ポート端子の機能をシリアルインタフェース用に切り換えます。
クロック同期式モードではSINx、SOUTx、#SCLKx、#SRDYx端子を使用します。

(4)インタフェースモードの設定
通常のインタフェースかIrDAインタフェースを選択します。
イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
サンプルプログラムでは通常のインタフェースを選択します。

(5)転送モードの設定
シリアルインタフェースの転送モードを選択します。
サンプルプログラムではクロック同期式転送のスレーブとして使用します。

(6)内部クロック、送受信許可の設定
クロック同期式のスレーブ側はマスタデバイスが出力するクロックで動作するため、動作クロックを外部クロックに設定します。よって、プリスケーラやタイマの設定をする必要はありません。
また、同時にシリアルインタフェースの送受信許可の設定を行います。
サンプルプログラムではスレーブとして送信を行いますので送信のみ許可します。
クロック同期式転送では送信と受信を同時に許可することはできません。

(7)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(8)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(9)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(10)割り込みの許可
シリアルインタフェースの割り込みを許可します。
送信時は送信バッファエンプティ割り込みが発生します。

シリアルインタフェース(クロック同期式スレーブモード)割り込み処理部

```
/* Prototype */
void int_sif_empty(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_sif_empty
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Serial I/F ch.1 sending buffer empty interrupt function.
 *****************************************************************************/
void int_sif_empty(void)
{
      extern volatile int int_empty_flg;
      extern volatile unsigned char str[10];
      extern volatile int i;

      if (i > 9)
      {
            int_empty_flg = TRUE;                                          (1)
      }
      else
      {
            /* write sending data */
            *(volatile unsigned char *)SIF_TXD1_ADDR = str[i];              (1)
            i++;
      }

      /* clear serial sending buffer empty interrupt flag */
      *(volatile unsigned char *)INT_FS_ADDR = INT_FSTX1;                  (2)
}
```

(1)割り込み処理

シリアルインタフェースでは、送信データが送信データレジスタからシフトレジスタに転送されると送信バッファエンプティ割り込み要因が発生します。割り込みが許可されていれば割り込みが発生し、割り込み処理を行います。

サンプルプログラムでは送信データを送信データレジスタへ書き込んでいます。

また、一定数のデータを送信するとソフトウェアフラグをTRUEにします。

(2)割り込み要因フラグのリセット

割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

● シリアルインタフェース(クロック同期式マスタモード)プログラム

GNU33¥sample_ide¥std¥dmt33301¥sif_mst¥src内のサンプルプログラムでは、クロック同期式転送方式でスレーブ側からマスタ側(本チップ)へ送られてくるデータを、割り込みを用いて連続受信します。ここでは、シリアルインタフェースCh.1のマスタモードへの設定や、その他の初期設定について解説します。

シリアルインタフェース(クロック同期式マスタモード)割り込みベクタ部

```
      (unsigned long)int_sif_error,       // 240  60  Serial interface Ch.1
      (unsigned long)int_sif_full,        // 244  61  Serial interface Ch.1
      (unsigned long)dummy,               // 248  62  Serial interface Ch.1
```

割り込みベクタの設定

割り込みルーチンをベクタテーブルに登録します。

シリアルインタフェース(クロック同期式マスタモード)初期設定部

```
/* Prototype */
void init_sync_sif1(void);

/*************************************************************************
 * init_sync_sif1
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize synchronous serial channel 1.
 *************************************************************************/
void init_sync_sif1(void)
{
      unsigned char temp;

      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.1 interrupt disable */
      *(volatile unsigned char *)INT_ES_ADDR &=
            ~(INT_ESTX1 | INT_ESRX1 | INT_ESERR1);                              (1)

      /* Set serial control 0x401e8, send disable, receive disable */
      *(volatile unsigned char *)SIF_SMD1_ADDR = SIF_TXEN_DIS | SIF_RXEN_DIS;   (2)

      /* Set serial interface I/O port 0x402d0, #SRDY1, #SCLK1, SOUT1, SIN1 */
      *(volatile unsigned char *)IO_CFP0_ADDR |=
            IO_CFP07_SRDY1 | IO_CFP06_SCLK1 | IO_CFP05_SOUT1 | IO_CFP04_SIN1;   (3)

      /* Set IrDA control 0x401e9, i/f mode normal */
      *(volatile unsigned char *)SIF_IRMD1_ADDR = SIF_IRMD_ORD;                   (4)

      /* Set serial control 0x401e8, transfer-mode clock-syn master */
      *(volatile unsigned char *)SIF_SMD1_ADDR = SIF_SMD_MAS;                     (5)

      /* Set serial control 0x401e8, send disable, receive enable, clock internal */
      *(volatile unsigned char *)SIF_SMD1_ADDR |=
            SIF_TXEN_DIS | SIF_RXEN_ENA | SIF_SSCK_INT;                         (6)

      /* Set 8bit timer3 prescaler, prescaler on, CLK/4 */
      *(volatile unsigned char *)PRESC_P8TS2_P8TS3_ADDR =
            RESC_PTONH_ON | PRESC_CLKDIVH_SEL1;                                 (7)

      /* Set 8bit timer3 reload data 0x4016d, reload data 0x40 */
      *(volatile unsigned char *)T8P_RLD3_ADDR = 0x20;                            (8)

      /* Set 8bit timer3 for serial interface ch.1 0x4016c, */
      /* clock output, reload data pre-set, timer run */
      *(volatile unsigned char *)T8P_PTRUN3_ADDR =
            T8P_PTOUT_ON | T8P_PSET_ON | T8P_PTRUN_RUN;                          (9)

      /* Clear serial status */
      *(volatile unsigned char *)SIF_RDBF1_ADDR = SIF_ERR_NON;                    (10)

      /* Set serial interface ch.1 and A/D converter interrupt priority register */
      /* 0x4026a */
      temp = *(volatile unsigned char *)INT_PSIO1_PAD_ADDR;                       (11)
      temp &= 0xf0;
      temp |= INT_PRIL_LVL3;
      *(volatile unsigned char *)INT_PSIO1_PAD_ADDR = temp;

      /* Set serial ch.0,1 interrupt flag 0x40286, clear serial ch.1 interrupt flag */
      *(volatile unsigned char *)INT_FS_ADDR =
            INT_FSTX1 | INT_FSRX1 | INT_FSERR1;                                 (12)

      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.1 interrupt enable */
      *(volatile unsigned char *)INT_ES_ADDR |=
            INT_ESTX1 | INT_ESRX1 | INT_ESERR1;                                 (13)
}
```

(1)割り込みの禁止
誤動作防止のため、シリアルインタフェースの割り込みを禁止します。

(2)送受信の禁止
動作中の設定変更は誤作動の原因になるため、シリアルインタフェースの送受信を禁止します。

(3)入出力端子の設定
入出力ポート端子の機能をシリアルインタフェース用に切り換えます。
クロック同期式モードではSINx、SOUTx、#SCLKx、#SRDYx端子を使用します。

(4)インタフェースモードの設定
通常のインタフェースかIrDAインタフェースを選択します。
イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
サンプルプログラムでは通常のインタフェースを選択します。

(5)転送モードの設定
シリアルインタフェースの転送モードを選択します。
サンプルプログラムではクロック同期式転送のマスタとして使用します。

(6)内部クロック、送受信の許可設定
クロック同期式のマスタモードでは内部発生したクロックで動作するため、動作クロックを内部クロックに設定します。同時にシリアルインタフェースの送受信許可の設定を行います。
サンプルプログラムではマスタとして受信を行いますので受信のみ許可します。
クロック同期式転送では送信と受信を同時に許可することはできません。

(7)～(9)入力クロック(8ビットタイマ)の設定
クロック同期式マスタモードは内部発生したクロックで動作します。
各チャネルのクロック源は次のとおりです。
Ch.0: 8ビットプログラマブルタイマ2の出力クロック
Ch.1: 8ビットプログラマブルタイマ3の出力クロック
Ch.2: 8ビットプログラマブルタイマ4の出力クロック
Ch.3: 8ビットプログラマブルタイマ5の出力クロック

ここではシリアルインタフェースCh.1を使用しますので、8ビットタイマ3の設定を行います。
(9)でクロック出力制御をOnすることで、シリアルインタフェースへクロックを供給することができます。
そのほかの8ビットタイマの設定については“3.2 8ビットプログラマブルタイマ”を参照してください。

(10)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(11)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(12)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(13)割り込みの許可
シリアルインタフェースの割り込みを許可します。
受信時はシリアルエラー割り込み、受信バッファフル割り込みが発生します。

シリアルインタフェース(クロック同期式マスタモード)割り込み処理部

```
/* Prototype */
void int_sif_error(void) __attribute__((interrupt_handler));
void int_sif_full(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_sif_error
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Serial I/F ch.0 receiving error interrupt function.
 *****************************************************************************/
void int_sif_error(void)
{
      extern volatile int int_error_flg;

      int_error_flg = TRUE;                                                    (1)

      /* clear serial error interrupt flag */
      *(volatile unsigned char *)INT_FS_ADDR = INT_FSERR0;                     (3)
}

/*****************************************************************************
 * int_sif_full
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Serial I/F ch.0 receiving buffer full interrupt function.
 *****************************************************************************/
void int_sif_full(void)
{
      extern volatile unsigned char str[10];
      extern volatile int i;

      /* read receiving data */
      str[i] = *(volatile unsigned char *)SIF_RXD1_ADDR;                       (2)
      //write_hex((unsigned long)str[i]);

      /* clear serial receiving buffer full interrupt flag */
      *(volatile unsigned char *)INT_FS_ADDR = INT_FSRX1;                      (3)

      i++;
}
```

(1)シリアルエラー割り込み処理
シリアルインタフェースによるデータ受信の際にパリティエラー、フレーミングエラー、またはオーバーランエラーが検出されると受信エラー割り込み要因フラグがセットされます。割り込みが許可されていれば、割り込みが発生し割り込み処理を行います。クロック同期式モードではパリティエラーとフレーミングエラーは発生しません。
サンプルプログラムでは割り込み処理として、割り込み確認用のソフトウェアフラグをTRUEにしています。

(2)受信バッファフル割り込み処理
シリアルインタフェースでは受信が完了して、受信データがシフトレジスタから受信データレジスタに転送されると受信バッファフル割り込み要因が発生します。割り込みが許可されていれば割り込みが発生し、割り込み処理を行います。
サンプルプログラムでは受信データレジスタのデータを読み出しています。

(3)割り込み要因フラグのリセット
割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

以上の設定の後、クロック同期式転送を開始できます。データ送受信開始後のシリアルインタフェースの動作は以下のとおりです。

受信（マスタモード）

1. スレーブ側からの#SRDY信号がLowになるまで待機します。
2. #SRDYがLowであればシリアルインタフェースへの同期クロックの入力を開始します。
3. スレーブ側からのデータをシフトレジスタへ取り込み、MSBを受信すると受信データレジスタへ転送します。
4. 受信データレジスタへ転送されると、受信バッファフル割り込みが発生し割り込み処理を行います。

送信（スレーブモード）

1. 送信データレジスタへデータをセットすると#SRDY信号がLowになり、マスタ側からのクロック入力を待ちます。
2. #SCLKから同期クロックが入力されると送信データレジスタからシフトレジスタへ送信データを転送します。
3. データがシフトレジスタに転送されると、送信バッファエンプティ割り込みが発生し割り込み処理を行います。この時点で#SRDY信号はHighになります。
4. シフトレジスタのデータはクロックに同期してマスタ側へ出力されます。
   MSBをマスタ側へ出力すると#SRDY信号はLowになります。

サンプルプログラムでは、送信ステップ3の割り込み処理で再度データを送信データレジスタに書き込むことにより、繰り返し上記の手順でデータの連続送信を行います。

● シリアルインタフェース（調歩同期式転送）プログラム

GNU33¥sample_ide¥std¥dmt33301¥sif_asyn¥src内のサンプルプログラムでは、調歩同期式転送方式を用いて外部シリアルデバイスとの連続データ転送を行います。ここでは調歩同期式シリアルインタフェースCh.0の初期設定について解説します。

シリアルインタフェース（調歩同期式転送）割り込みベクタ部

```
      (unsigned long)int_sif_error,       // 224  56  Serial interface Ch.0
      (unsigned long)int_sif_full,        // 228  57  Serial interface Ch.0
      (unsigned long)int_sif_empty,       // 232  58  Serial interface Ch.0
```

割り込みベクタの設定
　割り込みルーチンをベクタテーブルに登録します。

シリアルインタフェース（調歩同期式受信）初期設定部

```
/* Prototype */
void init_async_sif0(void);

/*****************************************************************************
 * init_async_sif0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize asynchronous serial channel 0.
 *****************************************************************************/
void init_async_sif0(void)
{
      unsigned char temp;

      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch0 interrupt disable */
      *(volatile unsigned char *)INT_ES_ADDR &=
            ~(INT_ESTX0 | INT_ESRX0 | INT_ESERR0);                              (1)

      /* Set serial control 0x401e3, send disable, receive disable */
      *(volatile unsigned char *)SIF_SMD0_ADDR = SIF_TXEN_DIS | SIF_RXEN_DIS;   (2)

      /* Set serial interface I/O port 0x402d0, #SRDY0, #SCLK0, SOUT0, SIN0 */
      *(volatile unsigned char *)IO_CFP0_ADDR |= IO_CFP01_SOUT0 | IO_CFP00_SIN0; (3)

      /* Set IrDA control 0x401e4, i/f mode normal */
      *(volatile unsigned char *)SIF_IRMD0_ADDR = SIF_IRMD_ORD;                 (4)
```

```
        /* Set serial control 0x401e3, transfer-mode clock-syn master */
        *(volatile unsigned char *)SIF_SMD0_ADDR = SIF_SMD_8BIT;                    (5)

        /* Set IrDA control 0x401e4, dividing frequency 1/16 */
        *(volatile unsigned char *)SIF_IRMD0_ADDR |= SIF_DIVMD_16;                  (6)

        /* Set serial control 0x401e3, send disable, receive enable, */
        /* parity off, parity-mode even, stop-bit 1, clock internal */
        *(volatile unsigned char *)SIF_SMD0_ADDR |=
              SIF_TXEN_DIS | SIF_RXEN_ENA | SIF_EPR_OFF | SIF_PMD_EVEN |
              SIF_STPB_1 | SIF_SSCK_INT;                                            (7)

        /* Set 8bit timer2 prescaler, prescaler on, CLK/4 */
        *(volatile unsigned char *)PRESC_P8TS2_P8TS3_ADDR =
              PRESC_PTONL_ON |PRESC_CLKDIVL_SEL1;                                   (8)

        /* Set 8bit timer2 reload data 0x40169, reload data 0x40 */
        *(volatile unsigned char *)T8P_RLD2_ADDR = 0x20;                            (9)

        /* Set 8bit timer2 for serial interface ch.0 0x40168, */
        /* clock output, reload data pre-set, timer run */
        *(volatile unsigned char *)T8P_PTRUN2_ADDR =
              T8P_PTOUT_ON | T8P_PSET_ON | T8P_PTRUN_RUN;                           (10)

        /* Clear serial status */
        *(volatile unsigned char *)SIF_RDBF1_ADDR = SIF_ERR_NON;                    (11)

        /* Set serial interface ch.0 and 8bit timer interrupt priority */
        /* register 0x40269 */
        temp = *(volatile unsigned char *)INT_P8TM_PSIO0_ADDR;                      (12)
        temp &= 0x0f;
        temp |= INT_PRIH_LVL3;
        *(volatile unsigned char *)INT_P8TM_PSIO0_ADDR = temp;

        /* Set serial ch.0,1 interrupt flag 0x40286, clear serial ch.0 interrupt flag */
        *(volatile unsigned char *)INT_FS_ADDR =
              INT_FSTX0 | INT_FSRX0 | INT_FSERR0;                                   (13)

        /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.0 interrupt enable */
        *(volatile unsigned char *)INT_ES_ADDR |=
              INT_ESTX0 | INT_ESRX0 | INT_ESERR0;                                   (14)
}
```

(1)割り込みの禁止
　誤動作防止のため、シリアルインタフェースの割り込みを禁止します。

(2)送受信の禁止
　動作中の設定変更は誤作動の原因になるため、シリアルインタフェースの送受信を禁止します。

(3)入出力端子の設定
　入出力ポート端子の機能をシリアルインタフェース用に切り換えます。
　調歩同期式モードではSINx、SOUTx端子を使用します。

(4)インタフェースモードの設定
　通常のインタフェースかIrDAインタフェースを選択します。
　イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
　サンプルプログラムでは通常のインタフェースを選択します。

(5)転送モードの設定
　シリアルインタフェースの転送モードを選択します。ここでは8ビット調歩同期式として使用します。

(6)調歩同期クロックの設定
　IrDAインタフェース入出力の設定と調歩同期クロックの分周比を設定します。
　サンプルプログラムではIrDAを使用せず、クロックを1/16に設定しています。

(7) 入力クロック、送受信許可、データフォーマットの設定
調歩同期式では入力クロックとして内部クロックか外部クロックを選択します。また、同時にシリアルインタフェースの送受信許可とデータフォーマットを設定します。データフォーマットとしてデータ長、ストップビット、パリティビットが設定可能です。
サンプルプログラムでは内部クロック、データ長 = 8ビット、ストップビット = 1ビット、パリティビットなし、受信許可に設定しています。

(8)～(10)8ビットタイマの設定
入力クロックとして内部クロックを選択した場合は8ビットタイマで発生したクロックで動作するため、8ビットタイマのクロック出力をOnします。各チャネルのクロック源は“● シリアルインタフェース(クロック同期式マスタモード)プログラム”を参照してください。
8ビットタイマの設定については“3.2 8ビットプログラマブルタイマ”を参照してください。

(11)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(12)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(13)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(14)割り込みの許可
シリアルインタフェースの割り込みを許可します。
受信時はシリアルエラー割り込み、受信バッファフル割り込みが発生します。

シリアルインタフェース(調歩同期式受信)割り込み処理部

調歩同期式転送の受信割り込み(受信バッファフル、シリアルエラー)については、“シリアルインタフェース(クロック同期式マスタモード)割り込み処理部”を参照してください。

シリアルインタフェース(調歩同期式送信)初期化設定部

```
/* Prototype */
void set_sif_mode(unsigned char);

/*************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Serial interface demonstration program.
 *************************************************************************/
int main(void)
{
      unsigned char sif_mode;

      /* Set serial interface ch.0 transmit mode */
      sif_mode = SIF_TXEN_ENA | SIF_RXEN_DIS | SIF_EPR_OFF |
            SIF_PMD_EVEN | SIF_STPB_1 | SIF_SSCK_INT | SIF_SMD_8BIT;          (1)

      set_sif_mode(sif_mode);

/*************************************************************************
 * set_sif_mode
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned char mode      Serial mode
 *   Function : Set serial interface mode.
 *************************************************************************/
void set_sif_mode(unsigned char mode)
{
      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch0 interrupt disable */
      *(volatile unsigned char *)INT_ES_ADDR &=
```

```
            ~(INT_ESTX0 | INT_ESRX0 | INT_ESERR0);                                    (2)

      /* Set serial control 0x401e3, send disable, receive disable */
      *(volatile unsigned char *)SIF_SMD0_ADDR = SIF_TXEN_DIS | SIF_RXEN_DIS;    (3)

      /* Clear serial status */
      *(volatile unsigned char *)(SIF_RDBF0_ADDR) = SIF_ERR_NON;                  (4)

      /* Set serial control register */
      *(volatile unsigned char *)(SIF_SMD0_ADDR) = mode;                          (5)
      // Set serial control register

      /* Set serial ch.0,1 interrupt flag 0x40286, clear serial ch.0 interrupt flag */
      *(volatile unsigned char *)INT_FS_ADDR =
            INT_FSTX0 | INT_FSRX0 | INT_FSERR0;                                   (6)

      /* Set serial ch.0,1 interrupt enable 0x40276, serial ch.0 interrupt enable */
      *(volatile unsigned char *)INT_ES_ADDR |=
            INT_ESTX0 | INT_ESRX0 | INT_ESERR0;                                   (7)
}
```

(1)転送モード、入力クロック、送受信許可、データフォーマットの設定
シリアルインタフェースの転送モードを選択します。調歩同期式では入力クロックとして内部クロックか外部クロックを選択します。また、同時にシリアルインタフェースの送受信許可とデータフォーマットを設定します。データフォーマットとしてデータ長、ストップビット、パリティビットが設定可能です。
サンプルプログラムでは転送モードを8ビット調歩同期式に設定し、内部クロック、データ長 = 8ビット、ストップビット = 1ビット、パリティビットなし、送信許可に設定しています。

(2)割り込みの禁止
誤動作防止のため、シリアルインタフェースの割り込みを禁止します。

(3)送受信の禁止
動作中の設定変更は誤作動の原因になるため、シリアルインタフェースの送受信を禁止します。

(4)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(5)送信パラメータのセット
(1)で用意した送信パラメータをレジスタに設定します。

(6)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(7)割り込みの許可
シリアルインタフェースの割り込みを許可します。
送信時は送信バッファエンプティ割り込みが発生します。

シリアルインタフェース(調歩同期式送信)割り込み処理部

調歩同期式転送の送信割り込み(送信バッファエンプティ)については、“シリアルインタフェース(クロック同期式スレーブモード)割り込み処理部”を参照してください。

以上の設定の後、調歩同期式転送を開始できます。データ送受信開始後のシリアルインタフェースの動作は以下のとおりです。

受信

1. スタートビットの入力によりサンプリングを開始します。
2. スタートビットがサンプリングされると、各データビットをLSBからMSBまでシフトレジスタへ取り込みます。
3. MSBが取り込まれると、続いてパリティビットとストップビットを取り込みます。
4. ストップビットをサンプリングするとシフトレジスタのデータは受信データレジスタへ転送されます。また、受信データレジスタへの転送時にはパリティチェックが行われ、データが転送されると受信バッファフル割り込みが発生して割り込み処理を行います。

送信

1. サンプリングクロックに同期してデータを送信データレジスタからシフトレジスタへ転送します。また、同時にSOUTxよりスタートビットを送信します。
2. データがシフトレジスタに転送されると、送信バッファエンプティ割り込みが発生し割り込み処理を行います。
3. スタートビットの送信後、クロックの立ち上がりに同期して各データビットをLSBからMSBまで送信します。
4. MSB送信後、パリティビットとストップビットを送信します。

サンプルプログラムでは、送信ステップ2の割り込み処理で再度データを送信データレジスタに書き込むことにより、繰り返し上記の手順でデータの連続送信を行います。

3.7 FIFO付きシリアルインタフェース

ここではGNU33¥sample_ide¥std¥dmt33301¥fsif_asyn/slv/mst内のサンプルプログラムを例に、FIFO付きシリアルインタフェースの制御プログラムについて説明します。
FIFO付きシリアルインタフェースでは、シリアルインタフェースの機能に加え、4バイト受信データバッファおよび2バイト送信データバッファを使用した連続受信、連続送信が可能です。

● FIFO付きシリアルインタフェース(クロック同期式転送スレーブモード)プログラム

GNU33¥sample_ide¥std¥dmt33301¥fsif_slv¥src内のサンプルプログラムは、クロック同期式転送方式でスレーブ側(本チップ)からマスタ側へ、割り込みを用いてデータを連続転送します。ここでは、FIFO付きシリアルインタフェースCh.0のスレーブモードへの設定や、その他の初期設定について解説します。

FIFO付きシリアルインタフェース(クロック同期式スレーブモード)割り込みベクタ部

```
      (unsigned long)dummy,              // 448  112 FIFO Serial interface Ch.0
      (unsigned long)dummy,              // 452  113 FIFO Serial interface Ch.0
      (unsigned long)int_fsif_empty      // 456  114 FIFO Serial interface Ch.0
```

割り込みベクタの設定
　割り込みルーチンをベクタテーブルに登録します。

FIFO付きシリアルインタフェース(クロック同期式スレーブモード)初期設定部

```
/* Prototype */
void init_fifo_bcu(void);
void init_sync_sif0(void);

/*****************************************************************************
 * init_fifo_bcu
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize BCU for FIFO serial I/F.
 *****************************************************************************/
void init_fifo_bcu(void)
{
      unsigned short temp;

      /* Set area 5-18 endian and external/internal access control 0x48132 */
      temp = *(volatile unsigned short *)BCU_EC_IO_ADDR;                    (1)
      temp &= 0xfdfd;
      temp |= BCU_A6IO_INT;
      *(volatile unsigned short *)BCU_EC_IO_ADDR = temp;

      /* Set area 4-6 BCU register, fifo i/f selection 0x4812a */
      temp = *(volatile unsigned short *)BCU_A4_A5_A6_ADDR;                 (2)
      temp &= 0xf0ff;
      temp |= BCU_WTH_1;
      *(volatile unsigned short *)BCU_A4_A5_A6_ADDR = temp;
}

/*****************************************************************************
 * init_sync_sif0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize synchronous FIFO serial channel 0.
 *****************************************************************************/
void init_sync_sif0(void)
{
      /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
      /* FIFO serial ch.0 interrupt disable */
      *(volatile unsigned char *)INT_EFS_ADDR &=
            ~(INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0);                       (3)

      /* Set FIFO serial interface control, each mode disable 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR =
                  FSIF_TXEN_DIS | FSIF_RXEN_DIS;                            (4)
```

```
        /* Set P00-P03 extended function, #FSRDY0, #FSCLK0, FSOUT0, FSIN0 0x300040 */
        *(volatile unsigned char *)FSIF_EFP_ADDR =
              FSIF_EFP03_FSRDY | FSIF_EFP02_FSCLK | FSIF_EFP01_FSOUT |
              FSIF_EFP00_FSIN;                                                    (5)

        /* Set FIFO serial interface IrDA register, i/f mode normal 0x300204 */
        *(volatile unsigned char *)FSIF_IRMD0_ADDR = FSIF_IRMD_ORD;               (6)

        /* Set FIFO serial interface control register, */
        /* transfer mode clock-sync slave 0x300203 */
        *(volatile unsigned char *)FSIF_SMD0_ADDR = FSIF_SMD_SLA;                 (7)

        /* Set FIFO serial interface IrDA register, #FSRDY0 normal 0x300204 */
        *(volatile unsigned char *)FSIF_IRMD0_ADDR |= FSIF_SRDYL_NML;             (8)

        /* Set FIFO serial interface control register, transfer enable, */
        /* #FSCLK0 0x300203 */
        *(volatile unsigned char *)FSIF_SMD0_ADDR |=
              FSIF_TXEN_ENA | FSIF_RXEN_DIS | FSIF_SSCK_FSCLK;                    (9)

        /* Set FIFO serial interface status register, error flag clear 0x300202 */
        *(volatile unsigned char *)FSIF_RDBF0_ADDR &= 0xe3;                       (10)

        /* Set FIFO serial interface interrupt priority level 3 on interrupt */
        /* controller 0x402b0 */
        *(volatile unsigned char *)INT_PFSIO_ADDR = INT_PRIH_LVL3;                (11)

        /* Set FIFO serial ch.0 interrupt flag 0x402b2, */
        /* clear serial ch.0 interrupt flag */
        *(volatile unsigned char *)INT_FFS_ADDR =
              INT_FFSTX0 | INT_FFSRX0 | INT_FFSERR0;                              (12)

        /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
        /* serial ch.0 interrupt enable */
        *(volatile unsigned char *)INT_EFS_ADDR |=
              INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0;                              (13)
}
```

(1), (2)BCUの設定
FIFO付きシリアルインタフェースの制御レジスタはエリア6の0x300200～0x300209に割り付けられているため、アクセス前にBCUレジスタを設定し、エリア6の内部デバイスがアクセスされるようにしておきます。

(3)割り込みの禁止
誤動作防止のため、FIFO付きシリアルインタフェースの割り込みを禁止します。

(4)送受信の禁止
動作中の設定変更は誤作動の原因になるため、FIFO付きシリアルインタフェースの送受信を禁止します。

(5)入出力端子の設定
入出力ポート端子の機能をFIFO付きシリアルインタフェース用に切り換えます。
クロック同期式モードではFSIN0、FSOUT0、#FSCLK0、#FSRDY0端子を使用します。

(6)インタフェースモードの設定
通常のインタフェースかIrDAインタフェースを選択します。
イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
サンプルプログラムでは通常のインタフェースを選択します。

(7)転送モードの設定
FIFO付きシリアルインタフェースの転送モードを選択します。
サンプルプログラムではクロック同期式転送のスレーブとして使用します。

(8)#FSRDY0制御の設定
受信バッファフル時の#FSRDY0信号の制御内容を設定します。マスクを選択すると、受信バッファフル時に#FSRDY0信号をHighにして、オーバーランエラーの発生を回避することができます。

(9)内部クロック、送受信許可の設定
クロック同期式のスレーブ側はマスタデバイスが出力するクロックで動作するため、動作クロックを外部クロックに設定します。よって、プリスケーラやタイマの設定をする必要はありません。
また、同時にFIFO付きシリアルインタフェースの送受信許可の設定を行います。
サンプルプログラムではスレーブとして送信を行いますので送信のみ許可します。
クロック同期式転送では送信と受信を同時に許可することはできません。

(10)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(11)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(12)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(13)割り込みの許可
FIFO付きシリアルインタフェースの割り込みを許可します。
送信時は送信バッファエンプティ割り込みが発生します。

FIFO付きシリアルインタフェース(クロック同期式スレーブモード)割り込み処理部

```
/* Prototype */
void int_fsif_empty(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_fsif_empty
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : FIFO serial I/F ch.0 sending buffer empty interrupt function.
 *****************************************************************************/
void int_fsif_empty(void)
{
      extern volatile int int_empty_flg;
      extern volatile unsigned char str[10];
      extern volatile int i;

      if (i > 9)
      {
            int_empty_flg = TRUE;                                          (1)
      }
      else
      {
            /* write sending data */
            *(volatile unsigned char *)FSIF_TXD0_ADDR = str[i];             (1)
            i++;
      }

            /* clear serial sending buffer empty interrupt flag */
            *(volatile unsigned char *)INT_FFS_ADDR = INT_FFSTX0;           (2)
}
```

(1)割り込み処理
FIFO付きシリアルインタフェースでは、送信データが送信データレジスタからシフトレジスタに転送されると送信バッファエンプティ割り込み要因が発生します。割り込みが許可されていれば割り込みが発生し、割り込み処理を行います。
サンプルプログラムでは送信データを送信データレジスタへ書き込んでいます。
また、一定数のデータを送信するとソフトウェアフラグをTRUEにします。

(2)割り込み要因フラグのリセット
割り込みの発生によって割り込み要因フラグが1になっていますので、これをリセットします。

● FIFO付きシリアルインタフェース(クロック同期式マスタモード)プログラム

GNU33¥sample_ide¥std¥dmt33301¥fsif_mst¥src内のサンプルプログラムでは、クロック同期式転送方式でスレーブ側からマスタ側(本チップ)へ送られてくるデータを、割り込みを用いて連続受信します。ここでは、FIFO付きシリアルインタフェースCh.0のマスタモードへの設定や、その他の初期設定について解説します。

FIFO付きシリアルインタフェース(クロック同期式マスタモード)割り込みベクタ部

```
      (unsigned long)int_fsif_error,      // 448  112 FIFO Serial interface Ch.0
      (unsigned long)int_fsif_full,       // 452  113 FIFO Serial interface Ch.0
      (unsigned long)dummy                // 456  114 FIFO Serial interface Ch.0
```

割り込みベクタの設定

割り込みルーチンをベクタテーブルに登録します。

FIFO付きシリアルインタフェース(クロック同期式マスタモード)初期設定部

```
/* Prototype */
void init_fifo_bcu(void);
void init_sync_sif0(void);

/*****************************************************************************
 * init_fifo_bcu
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize BCU for FIFO fifo-serial I/F.
 *****************************************************************************/
void init_fifo_bcu(void)
{
      unsigned short temp;

      /* Set area 5-18 endian control and external/internal access control 0x48132 */
      temp = *(volatile unsigned short *)BCU_EC_IO_ADDR;                    (1)
      temp &= 0xfdfd;
      temp |= BCU_A6IO_INT;
      *(volatile unsigned short *)BCU_EC_IO_ADDR = temp;

      /* Set area 4-6 BCU register, fifo i/f selection 0x4812a */
      temp = *(volatile unsigned short *)BCU_A4_A5_A6_ADDR;                 (2)
      temp &= 0xf0ff;
      temp |= BCU_WTH_1;
      *(volatile unsigned short *)BCU_A4_A5_A6_ADDR = temp;
}

/*****************************************************************************
 * init_sync_sif0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize synchronous FIFO serial channel 0.
 *****************************************************************************/
void init_sync_sif0(void)
{
      /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
      /* FIFO serial ch.0 interrupt disable */
      *(volatile unsigned char *)INT_EFS_ADDR &=
            ~(INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0);                       (3)

      /* Set FIFO serial interface control, each mode disable 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR =
            FSIF_TXEN_DIS | FSIF_RXEN_DIS;                                  (4)

      /* Set P00-P03 extended function, fifo i/f port selection 0x300040 */
      *(volatile unsigned char *)FSIF_EFP_ADDR =
            FSIF_EFP03_FSRDY | FSIF_EFP02_FSCLK | FSIF_EFP01_FSOUT |
            FSIF_EFP00_FSIN;                                                (5)

      /* Set FIFO serial interface IrDA register, i/f mode selection 0x300204 */
      *(volatile unsigned char *)FSIF_IRMD0_ADDR = FSIF_IRMD_ORD;           (6)
```

```
        /* Set FIFO serial interface control register, */
        /* transfer mode selection 0x300203 */
        *(volatile unsigned char *)FSIF_SMD0_ADDR = FSIF_SMD_MAS;              (7)

        /* Set FIFO serial interface IrDA register, i/f mode setting 0x300204 */
        *(volatile unsigned char *)FSIF_IRMD0_ADDR |=
              FSIF_SRDYL_NML | FSIF_FINT0_LV2 | FSIF_DIVMD_16;                 (8)

        /* Set FIFO serial interface control register, transfer mode setting 0x300203 */
        *(volatile unsigned char *)FSIF_SMD0_ADDR |=
                    FSIF_TXEN_DIS | FSIF_RXEN_ENA | FSIF_SSCK_INT;             (9)

        /* Set FIFO serial interface baudrate reload data 0x300206,0x300207, */
        /* 9600bps(40M) */
        *(volatile unsigned char *)FSIF_BRTRD_LSB = 0x80;                      (10)
        *(volatile unsigned char *)FSIF_BRTRD_MSB = 0x00;

        /* Set FIFO serial interface status register, error flag clear 0x300202 */
        *(volatile unsigned char *)FSIF_RDBF0_ADDR &= 0xe3;                    (11)

        /* Set FIFO serial interface interrupt priority level 3 on interrupt */
        /* controller 0x402b0 */
        *(volatile unsigned char *)INT_PFSIO_ADDR = INT_PRIH_LVL3;             (12)

        /* Set FIFO serial ch.0 interrupt flag 0x402b2, */
        /* clear serial ch.0 interrupt flag */
        *(volatile unsigned char *)INT_FFS_ADDR =
              INT_FFSTX0 | INT_FFSRX0 | INT_FFSERR0;                           (13)

        /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
        /* serial ch.0 interrupt enable */
        *(volatile unsigned char *)INT_EFS_ADDR |=
              INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0;                           (14)

        /* Set FIFO serial interface baudrate control register 0x300205 */
        *(volatile unsigned char *)FSIF_BRTRUN_ADDR = FSIF_BTRUN_ON;           (15)
}
```

(1), (2)BCUの設定
FIFO付きシリアルインタフェースの制御レジスタはエリア6の0x300200～0x300209に割り付けられているため、アクセス前にBCUレジスタを設定し、エリア6の内部デバイスがアクセスされるようにしておきます。

(3)割り込みの禁止
誤動作防止のため、FIFO付きシリアルインタフェースの割り込みを禁止します。

(4)送受信の禁止
動作中の設定変更は誤作動の原因になるため、FIFO付きシリアルインタフェースの送受信を禁止します。

(5)入出力端子の設定
入出力ポート端子の機能をFIFO付きシリアルインタフェース用に切り換えます。
クロック同期式モードではFSIN0、FSOUT0、#FSCLK0、#FSRDY0端子を使用します。

(6)インタフェースモードの設定
通常のインタフェースかIrDAインタフェースを選択します。
イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
サンプルプログラムでは通常のインタフェースを選択します。

(7)転送モードの設定
FIFO付きシリアルインタフェースの転送モードを選択します。
サンプルプログラムではクロック同期式転送のマスタとして使用します。

(8)#FSRDY0制御、受信FIFOレベルの設定
受信バッファフル時の#FSRDY0信号の制御内容と受信FIFOレベルの設定をします。#FSRDY0信号制御では、マスクを選択すると受信バッファフル時に#FSRDY0信号をHighにして、オーバーランエラーの発生を回避することができます。また、受信FIFOレベルを設定することで受信バッファフルを発生させる受信バッファのデータ数を設定できます。
サンプルプログラムでは#FSRDY0の通常出力を選択し、受信FIFOレベルを2に設定しています。

(9)内部クロック、送受信許可の設定
クロック同期式マスタモードは内部発生したクロックで動作するため、動作クロックを内部クロックに設定します。同時にFIFO付きシリアルインタフェースの送受信許可の設定を行います。
サンプルプログラムではマスタとして受信を行いますので受信のみ許可します。
クロック同期式転送では送信と受信を同時に許可することはできません。

(10)入力クロック(ボーレート)の設定
クロック同期式マスタモードは内部発生したクロックで動作します。FIFO付きシリアルインタフェースでは専用のボーレートタイマ(10ビットプログラマブルタイマ)を使用して入力クロックを生成します。ここでは、動作クロックが40MHzの場合にボーレートが9600bpsとなるように、ボーレートタイマのリロードデータレジスタに初期値を設定しています。

(11)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(12)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(13)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(14)割り込みの許可
FIFO付きシリアルインタフェースの割り込みを許可します。
受信時はシリアルエラー割り込み、受信バッファフル割り込みが発生します。

(15)ボーレートタイマのスタート
ボーレート制御レジスタを設定し、ボーレートタイマの動作を開始させます。

FIFO付きシリアルインタフェース(クロック同期式マスタモード)割り込み処理部

```
/* Prototype */
void int_fsif_error(void) __attribute__((interrupt_handler));
void int_fsif_full(void) __attribute__((interrupt_handler));

/************************************************************************
 * int_fsif_error
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : FIFO serial I/F ch.0 receiving error interrupt function.
 ************************************************************************/
void int_fsif_error(void)
{
      extern volatile int int_error_flg;

      int_error_flg = TRUE;                                          (1)

      /* clear serial error interrupt flag */
      *(volatile unsigned char *)INT_FFS_ADDR = INT_FFSERR0;         (3)
}

/************************************************************************
 * int_fsif_full
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : FIFO serial I/F ch.0 receiving buffer full interrupt function.
 ************************************************************************/
```

```
void int_fsif_full(void)
{

      extern volatile unsigned char str[10];
      extern volatile int i;

      /* read receiving data */
      while ((*(volatile unsigned char *)FSIF_RDBF0_ADDR & 0x01)!=0)
      {
            str[i] = *(volatile unsigned char *)FSIF_RXD0_ADDR;              (1)
            //write_hex((unsigned long)str[i]);
            i++;
      }

      /* clear serial receiving buffer full interrupt flag */
      *(volatile unsigned char *)INT_FFS_ADDR = INT_FFSRX0;                  (2)

}
```

(1)シリアルエラー割り込み処理

FIFO付きシリアルインタフェースによるデータ受信の際にパリティエラー、フレーミングエラー、またはオーバーランエラーが検出されると受信エラー割り込み要因フラグがセットされます。割り込みが許可されていれば、割り込みが発生し割り込み処理を行います。クロック同期式モードではパリティエラーとフレーミングエラーは発生しません。

サンプルプログラムでは割り込み処理として、割り込み確認用のソフトウェアフラグをTRUEにしています。

(2)受信バッファフル割り込み処理

FIFO付きシリアルインタフェースでは受信が完了して、シフトレジスタから受信データバッファ内に指定数以上のデータ数が転送されると、受信バッファフル割り込み要因が発生します。割り込みが許可されていれば割り込みが発生し、割り込み処理を行います。

サンプルプログラムでは受信データバッファが空になるまで、受信データレジスタのデータを読み出しています。

(3)割り込み要因フラグのリセット

割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

以上の設定の後、クロック同期式転送を開始できます。データ送受信開始後のFIFO付きシリアルインタフェースの動作は以下のとおりです。

受信(マスタモード)

1. スレーブ側からの#FSRDY0信号がLowになるまで待機します。
2. #FSRDY0がLowであればFIFO付きシリアルインタフェースへの同期クロックの入力を開始します。
3. スレーブ側からのデータをシフトレジスタへ取り込み、MSBを受信すると受信データバッファへ転送します。受信データバッファへ転送されるとデータが読み出せる状態になります。
4. 受信データバッファ内のデータ数が指定数以上になると、受信バッファフル割り込みが発生し割り込み処理を行います。

送信(スレーブモード)

1. 送信データバッファへデータをセットすると#FSRDY0信号がLowになり、マスタ側からのクロック入力を待ちます。送信データバッファには、2バイトまでデータの書き込みが可能です。
2. #FSCLK0から同期クロックが入力されると送信データバッファからシフトレジスタへ送信データを転送します。この時点で#FSRDY0信号はHighになります。
3. シフトレジスタのデータはクロックに同期してマスタ側へ出力されます。
   MSBをマスタ側へ出力すると#FSRDY0信号はLowになります。
   送信データバッファ内のデータがすべて出力されるまで、(2)と(3)を繰り返します。
4. 送信データバッファ内の最後のデータがシフトレジスタに転送されると、送信バッファエンプティ割り込みが発生し割り込み処理を行います。

サンプルプログラムでは、送信ステップ4の割り込み処理で再度データを送信データバッファに書き込むことにより、繰り返し上記の手順でデータの連続送信を行います。

● FIFO付きシリアルインタフェース(調歩同期式転送)プログラム

GNU33¥sample_ide¥std¥dmt33301¥fsif_asyn¥src内のサンプルプログラムでは、調歩同期式転送方式を用いて外部シリアルデバイスとの連続データ転送を行います。ここでは、FIFO付きシリアルインタフェースCh.0の調歩同期式モードの設定や、その他の初期設定について解説します。

FIFO付きシリアルインタフェース(調歩同期式転送)割り込みベクタ部

```
      (unsigned long)int_fsif_error,      // 448  112 FIFO Serial interface Ch.0
      (unsigned long)int_fsif_full,       // 452  113 FIFO Serial interface Ch.0
      (unsigned long)int_fsif_empty       // 456  114 FIFO Serial interface Ch.0
```

割り込みベクタの設定

割り込みルーチンをベクタテーブルに登録します。

FIFO付きシリアルインタフェース(調歩同期式受信)初期設定部

```
/* Prototype */
void init_fifo_bcu(void);
void init_async_fsif0(void);

/*****************************************************************************
 * init_fifo_bcu
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize BCU for FIFO fifo-serial I/F.
 *****************************************************************************/
void init_fifo_bcu(void)
{
      unsigned short temp;

      /* Set area 5-18 endian control and external/internal access control 0x48132 */
      temp = *(volatile unsigned short *)BCU_EC_IO_ADDR;                    (1)
      temp &= 0xfdfd;
      temp |= BCU_A6IO_INT;
      *(volatile unsigned short *)BCU_EC_IO_ADDR = temp;

      /* Set area 4-6 BCU register, fifo i/f selection 0x4812a */
      temp = *(volatile unsigned short *)BCU_A4_A5_A6_ADDR;                 (2)
      temp &= 0xf0ff;
      temp |= BCU_WTH_1;
      *(volatile unsigned short *)BCU_A4_A5_A6_ADDR = temp;
}

/*****************************************************************************
 * init_async_fsif0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize asynchronous FIFO serial channel 0.
 *****************************************************************************/
void init_async_fsif0(void)
{
      /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
      /* FIFO serial ch.0 interrupt disable */
      *(volatile unsigned char *)INT_EFS_ADDR &=
            ~(INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0);                       (3)

      /* Set FIFO serial interface control, each mode disable 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR =
            FSIF_TXEN_DIS | FSIF_RXEN_DIS;                                  (4)

      /* Set P00-P03 extended function, fifo i/f port selection 0x300040 */
      *(volatile unsigned char *)FSIF_EFP_ADDR =
            FSIF_EFP01_FSOUT | FSIF_EFP00_FSIN;                             (5)

      /* Set FIFO serial interface IrDA register, i/f mode selection 0x300204 */
      *(volatile unsigned char *)FSIF_IRMD0_ADDR = FSIF_IRMD_ORD;           (6)

      /* Set FIFO serial interface control register, transfer mode 8bit 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR = FSIF_SMD_8BIT;            (7)
```

```
        /* Set FIFO serial interface IrDA register, i/f mode setting 0x300204 */
        *(volatile unsigned char *)FSIF_IRMD0_ADDR |=
              FSIF_SRDYL_NML | FSIF_FINT0_LV2 | FSIF_DIVMD_16;                    (8)

        /* Set FIFO serial interface control register, transfer mode setting 0x300203 */
        *(volatile unsigned char *)FSIF_SMD0_ADDR |=
              FSIF_TXEN_DIS | FSIF_RXEN_ENA | FSIF_EPR_OFF |
              FSIF_PMD_EVEN | FSIF_STPB_1 | FSIF_SSCK_INT;                        (9)

        /* Set FIFO serial interface status register, error flag clear 0x300202 */
        *(volatile unsigned char *)FSIF_RDBF0_ADDR &= 0xe3;                       (10)

        /* Set FIFO serial interface baudrate reload data 0x300206,0x300207, */
        /* 9600bps(40M) */
        *(volatile unsigned char *)FSIF_BRTRD_LSB = 0x80;                         (11)
        *(volatile unsigned char *)FSIF_BRTRD_MSB = 0x00;

        /* Set FIFO serial interface interrupt priority level 3 on interrupt */
        /* controller 0x402b0 */
        *(volatile unsigned char *)INT_PFSIO_ADDR = INT_PRIH_LVL3;                (12)

        /* Set FIFO serial ch.0 interrupt flag 0x402b2, */
        /* clear serial ch.0 interrupt flag */
        *(volatile unsigned char *)INT_FFS_ADDR =
              INT_FFSTX0 | INT_FFSRX0 | INT_FFSERR0;                              (13)

        /* Set FIFO serial ch.0 interrupt enable 0x402b1, serial ch.0 interrupt enable */
        *(volatile unsigned char *)INT_EFS_ADDR |=
              INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0;                              (14)

        /* Set FIFO serial interface baudrate control register 0x300205 */
        *(volatile unsigned char *)FSIF_BRTRUN_ADDR = FSIF_BTRUN_ON;              (15)
}
```

(1), (2)BCUの設定
FIFO付きシリアルインタフェースの制御レジスタはエリア6の0x300200～0x300209に割り付けられているため、アクセス前にBCUレジスタを設定し、エリア6の内部デバイスがアクセスされるようにしておきます。

(3)割り込みの禁止
誤動作防止のため、FIFO付きシリアルインタフェースの割り込みを禁止します。

(4)送受信の禁止
動作中の設定変更は誤作動の原因になるため、FIFO付きシリアルインタフェースの送受信を禁止します。

(5)入出力端子の設定
入出力ポート端子の機能をFIFO付きシリアルインタフェース用に切り換えます。
調歩同期式モードではFSIN0、FSOUT0端子を使用します。

(6)インタフェースモードの設定
通常のインタフェースかIrDAインタフェースを選択します。
イニシャルリセット時、この設定は不定になりますので初期化する必要があります。
サンプルプログラムでは通常のインタフェースを選択します。

(7)転送モードの設定
FIFO付きシリアルインタフェースの転送モードを選択します。
サンプルプログラムでは8ビット調歩同期式モードとして使用します。

(8)#FSRDY0制御、受信FIFOレベル、クロック分周比の設定
受信バッファフル時の#FSRDY0信号の制御内容と受信FIFOレベル、調歩同期クロックの分周比を設定します。#FSRDY0信号制御では、マスクを選択すると受信バッファフル時に#FSRDY0信号をHighにして、オーバーランエラーの発生を回避することができます。また、受信FIFOレベルを設定することで受信バッファフルを発生させる受信バッファのデータ数を設定できます。ここでは同時に調歩同期クロックの分周比も設定します。
サンプルプログラムでは#FSRDY0の通常出力を選択し、受信FIFOレベルを2、分周比を1/16に設定しています。

(9)入力クロック、送受信許可、データフォーマットの設定
調歩同期式では入力クロックとして内部クロックか外部クロックを選択します。また、同時にFIFO付きシリアルインタフェースの送受信許可とデータフォーマットを設定します。データフォーマットとしてデータ長、ストップビット、パリティビットが設定可能です。
サンプルプログラムでは内部クロック、データ長 = 8ビット、ストップビット = 1ビット、パリティビットなし、受信許可に設定しています。

(10)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(11)入力クロック(ボーレート)の設定
内部クロックを選択した場合、FIFO付きシリアルインタフェースは専用のボーレートタイマ(10ビットプログラマブルタイマ)を使用して入力クロックを生成します。ここでは、動作クロックが40MHzの場合にボーレートが9600bpsとなるように、ボーレートタイマのリロードデータレジスタに初期値を設定しています。

(12)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(13)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(14)割り込みの許可
FIFO付きシリアルインタフェースの割り込みを許可します。
受信時はシリアルエラー割り込み、受信バッファフル割り込みが発生します。

(15)ボーレートタイマのスタート
ボーレート制御レジスタを設定し、ボーレートタイマの動作を開始させます。

FIFO付きシリアルインタフェース(調歩同期式受信)割り込み処理部

調歩同期式転送の受信割り込み(受信バッファフル、シリアルエラー)については、"FIFO付きシリアルインタフェース(クロック同期式マスタモード)割り込み処理部"を参照してください。

FIFO付きシリアルインタフェース(調歩同期式送信)初期化設定部

```
/* Prototype */
void set_fsif_mode(unsigned char);

/*****************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : FIFO serial interface demonstration program.
 *****************************************************************************/
int main(void)
{
      unsigned char sif_mode;

      /* Set FIFO serial interface ch.0 transmit mode */
      sif_mode = FSIF_TXEN_ENA | FSIF_RXEN_DIS | FSIF_EPR_OFF |
            FSIF_PMD_EVEN | FSIF_STPB_1 | FSIF_SSCK_INT | FSIF_SMD_8BIT;      (1)
```

```
      set_fsif_mode(sif_mode);
}

/*************************************************************************
 * set_fsif_mode
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned char mode      Serial mode
 *   Function : Set FIFO serial interface mode.
 *************************************************************************/
void set_fsif_mode(unsigned char mode)
{
      /* Set FIFO serial ch.0 interrupt enable 0x402b1, */
      /* FIFO serial ch.0 interrupt disable */
      *(volatile unsigned char *)INT_EFS_ADDR &=
            ~(INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0);                    (2)

      /* Set FIFO serial interface control, each mode disable 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR =
                  FSIF_TXEN_DIS | FSIF_RXEN_DIS;                         (3)

      /* Set FIFO serial control register 0x300203 */
      *(volatile unsigned char *)FSIF_SMD0_ADDR = mode;                  (4)
      // Set serial control register

      /* Clear FIFO serial status 0x300202 */
      *(volatile unsigned char *)FSIF_RDBF0_ADDR = FSIF_ERR_NON;
      // Clear serial status                                             (5)

      /* Set FIFO serial ch.0 interrupt flag 0x402b2, */
      /* clear serial ch.0 interrupt flag */
      *(volatile unsigned char *)INT_FFS_ADDR =
            INT_FFSTX0 | INT_FFSRX0 | INT_FFSERR0;                       (6)

      /* Set FIFO serial ch.0 interrupt enable 0x402b1, serial ch.0 interrupt enable */
      *(volatile unsigned char *)INT_EFS_ADDR |=
            INT_EFSTX0 | INT_EFSRX0 | INT_EFSERR0;                       (7)
}
```

(1)転送モード、入力クロック、送受信許可、データフォーマットの設定
FIFO付きシリアルインタフェースの転送モードを選択します。調歩同期式では入力クロックとして内部クロックか外部クロックを選択します。また、同時にFIFO付きシリアルインタフェースの送受信許可とデータフォーマットを設定します。データフォーマットとしてデータ長、ストップビット、パリティビットが設定可能です。
サンプルプログラムでは転送モードを8ビット調歩同期式に設定し、内部クロック、データ長 = 8ビット、ストップビット = 1ビット、パリティビットなし、送信許可に設定しています。

(2)割り込みの禁止
誤動作防止のため、FIFO付きシリアルインタフェースの割り込みを禁止します。

(3)送受信の禁止
動作中の設定変更は誤作動の原因になるため、FIFO付きシリアルインタフェースの送受信を禁止します。

(4)送信パラメータのセット
(1)で用意した送信パラメータをレジスタに設定します。

(5)ステータスレジスタのリセット
ステータスレジスタのエラーフラグを、0の書き込みによりリセットします。

(6)割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(7)割り込みの許可
FIFO付きシリアルインタフェースの割り込みを許可します。
送信時は送信バッファエンプティ割り込みが発生します。

FIFO付きシリアルインタフェース(調歩同期式送信)割り込み処理部

調歩同期式転送の送信割り込み(送信バッファエンプティ)については、“FIFO付きシリアルインタフェース(クロック同期式スレーブモード)割り込み処理部”を参照してください。

以上の設定の後、調歩同期式転送を開始できます。データ送受信開始後のFIFO付きシリアルインタフェースの動作は以下のとおりです。

受信

1. スタートビットの入力によりサンプリングを開始します。
2. スタートビットがサンプリングされると、各データビットをLSBからMSBまでシフトレジスタへ取り込みます。
3. MSBが取り込まれると、続いてパリティビットとストップビットを取り込みます。
4. ストップビットをサンプリングするとシフトレジスタのデータは受信データバッファへ転送されます。受信データバッファへ転送されるとデータが読み出せる状態になります。また、受信データバッファへの転送時にはパリティチェックが行われます。
5. 受信データバッファ内のデータ数が指定数以上になると、受信バッファフル割り込みが発生し割り込み処理を行います。

送信

1. サンプリングクロックに同期してデータを送信データバッファからシフトレジスタへ転送します。また、同時にFSOUT0よりスタートビットを送信します。
2. スタートビットの送信後、クロックの立ち上がりに同期して各データビットをLSBからMSBまで送信します。
3. MSB送信後、パリティビットとストップビットを送信します。
   送信データバッファ内のデータがすべて出力されるまで、(1)～(3)を繰り返します。
4. 送信データバッファ内の最後のデータがシフトレジスタに転送されると、送信バッファエンプティ割り込みが発生し割り込み処理を行います。

サンプルプログラムでは、送信ステップ4の割り込み処理で再度データを送信データレジスタに書き込むことにより、繰り返し上記の手順でデータの連続送信を行います。

3.8 ポート割り込み

ここではGNU33¥sample_ide¥std¥dmt33301¥gpio(gpio_key)¥src内のサンプルプログラムを例に、入力割り込みプログラムについて説明します。入力割り込みにはポート入力割り込みとキー入力割り込みがあります。ポート入力割り込みは、選択したポートの入力信号のエッジ、またはレベルによって発生します。キー入力割り込みは、既定の組み合わせの中から選択したポートの状態と入力比較レジスタの内容との不一致により発生します。

● ポート入力割り込み制御プログラム

このサンプルプログラムは、K50ポート入力の立ち上がりエッジによって、ポート入力割り込み(FPT0)が発生するように設定しています。以下、ポート入力割り込みの初期設定と割り込み処理について解説しています。

ポート入力割り込みベクタ部

```
      (unsigned long)int_io0,       // 64   16  Port input interrupt 0
      (unsigned long)dummy,         // 68   17  Port input interrupt 1
      (unsigned long)dummy,         // 72   18  Port input interrupt 2
      (unsigned long)dummy,         // 76   19  Port input interrupt 3

      (unsigned long)dummy,         // 272  68  Port input interrupt 4
      (unsigned long)dummy,         // 276  69  Port input interrupt 5
      (unsigned long)dummy,         // 280  70  Port input interrupt 6
      (unsigned long)dummy,         // 284  71  Port input interrupt 7
```

ポート入力割り込み初期設定部

```
/* Prototype */
void init_port(void);

/*****************************************************************************
 * init_port
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize I/O port function.
 *****************************************************************************/
void init_port(void)
{
      unsigned char temp;

      /* input port0-3 and key input0,1 interrupt enable 0x40270, */
      /* input port0 interrupt disable */
      *(volatile unsigned char *)INT_EP0_EK_ADDR &= ~INT_EP0;                (1)

      /* Set input/output port function 0x402c0, port K50 */
      *(volatile unsigned char *)IN_CFK5_ADDR &= 0xfe;                       (2)

      /* FPT0-FPT3 interrupt port selection 0x402c6, FPT0 K50 */
      temp = *(volatile unsigned char *)IN_SPT0_SPT3_ADDR;                   (3)
      temp &= 0xfc;
      temp |= 0x1;
      *(volatile unsigned char *)IN_SPT0_SPT3_ADDR |= temp;

      /* FPT0-FPT7 interrupt input polarity selection 0x402c8, */
      /* FPT0 high level or edge rising */
      *(volatile unsigned char *)IN_SPP_ADDR |= 0x01;                        (4)

      /* FPT0-FPT7 interrupt edge/level selection 0x402c9, FPT0 edge */
      *(volatile unsigned char *)IN_SEP_ADDR |= 0x01;                        (5)

      /* input port0-3, HSDMA and 16bit timer0 IDMA request 0x40290, */
      /* input port0 CPU request */
      *(volatile unsigned char *)INT_RP0_RHDM_R16T0_ADDR &= 0xfe;            (6)

      /* port input0,1 interrupt priority register 0x40260, */
      /* input port0 priority level 3 */
      temp = *(volatile unsigned char *)INT_PP0_PP1_ADDR;                    (7)
      temp &= 0xf0;
```

```
    temp |= INT_PRIL_LVL3;
    *(volatile unsigned char *)INT_PP0_PP1_ADDR = temp;

    /* input port0-3 and key input0,1 interrupt factor flag 0x40280, */
    /* input port0 reset interrupt flag */
    *(volatile unsigned char *)INT_FP0_FK_ADDR = INT_FP0;                    (8)

    /* input port0-3 and key input0,1 interrupt enable 0x40270, */
    /* input port0 interrupt enable */
    *(volatile unsigned char *)INT_EP0_EK_ADDR |= INT_EP0;                   (9)
}
```

(1)割り込みの禁止
誤動作防止のため、ポート入力割り込みを禁止します。

(2)入力ポートの設定
使用するポートの機能を選択します。
サンプルプログラムではK50ポート機能として入力ポートK50を選択します。

(3)入力端子の選択
ポート入力割り込み用端子を選択します。
各割り込み要因のポート割り当ては以下のとおりです。

表3.8.1 ポート入力割り込み用入力端子の選択

割り込み要因	制御ビット	SPT設定			
		11	10	01	00
FPT7	SPT7[1:0] (0x402C7•D[7:6])	P27	P07	P33	K67
FPT6	SPT6[1:0] (0x402C7•D[5:4])	P26	P06	P32	K66
FPT5	SPT5[1:0] (0x402C7•D[3:2])	P25	P05	P31	K65
FPT4	SPT4[1:0] (0x402C7•D[1:0])	P24	P04	K54	K64
FPT3	SPT3[1:0] (0x402C6•D[7:6])	P23	P03	K53	K63
FPT2	SPT2[1:0] (0x402C6•D[5:4])	P22	P02	K52	K62
FPT1	SPT1[1:0] (0x402C6•D[3:2])	P21	P01	K51	K61
FPT0	SPT0[1:0] (0x402C6•D[1:0])	P20	P00	K50	K60

サンプルプログラムではK50(FPT0)を割り込み要因として選択します。

(4), (5)割り込み発生条件の設定
ポート入力割り込みを発生させる条件を選択します。ポート入力割り込みは入力信号のレベル、もしくはエッジで発生させることができます。割り込み発生条件の設定内容は以下のとおりです。

表3.8.2 ポート入力割り込み発生条件

SEPTx (0x402C9•Dx)	SPPTx (0x402C8•Dx)	FPTx割り込み条件
1	1	立ち上がりエッジ
1	0	立ち下がりエッジ
0	1	Highレベル
0	0	Lowレベル

サンプルプログラムではFPT0の割り込み発生条件に立ち上がりエッジを選択しています。

(6)割り込み/IDMAリクエストの設定
割り込み要因によってCPUに割り込みを要求するか、IDMAを起動するかを選択します。
サンプルプログラムでは割り込み要求を行います。

(7)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(8)割り込み要因フラグのリセット
誤動作防止のため、割り込みを許可する前に割り込み要因フラグをリセットします。

(9)割り込みの許可
ポート入力割り込みを許可します。
以降、K50ポートの立ち上がりエッジで割り込みが発生します。

ポート入力割り込み処理部

```
/* Prototype */
void int_io0(void) __attribute__((interrupt_handler));

/*************************************************************************
 * int_io0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : I/O port FPT0 interrupt function.
 *************************************************************************/
void int_io0(void)
{
      unsigned int j;
      volatile extern unsigned char int_io0_flg;

      int_io0_flg = TRUE;                                                    (1)

      /* input port0-3 and key input0-1 interrupt factor flag 0x40280, */
      /* reset interrupt factor flag */
      *(volatile unsigned char *)INT_FP0_FK_ADDR = INT_FP0;                  (2)

}
```

(1)割り込み処理
選択したポートが割り込み発生条件を満たすと割り込みが発生します。割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。
サンプルプログラムでは、割り込み発生の確認用に用意されたソフトウェアフラグをセットしています。

(2)割り込み要因フラグのリセット
割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

● キー入力割り込み制御プログラム

このサンプルプログラムでは、K50ポートの内容と入力比較レジスタの設定値との不一致によりキー入力割り込み(FPK0)が発生するように設定しています。以下、キー入力割り込みの初期設定と割り込み処理について解説します。

キー入力割り込みベクタ部

```
      (unsigned long)int_key0,      // 80   20  Key input interrupt 0
      (unsigned long)dummy,         // 84   21  Key input interrupt 1
```

キー入力割り込み初期設定部

```
/* Prototype */
void init_port(void);

/*************************************************************************
 * init_port
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Initialize I/O port function.
 *************************************************************************/
void init_port(void)
{
      unsigned char temp;

      /* Address for input port0-3 and key input0,1 */
      /* interrupt enable register 0x40270 */
      *(volatile unsigned char *)INT_EP0_EK_ADDR &= ~INT_EK0;                (1)

      /* Set input/output port function 0x402c0, port K50 */
      *(volatile unsigned char *)IN_CFK5_ADDR &= 0xfe;                       (2)

      /* Address for FPK0-FPK1 interrupt port selection register 0x402ca, port K50 */
      temp = *(volatile unsigned char *)IN_SPPK_ADDR;                        (3)
      temp &= 0xfc;
```

```
    temp |= 0x0;
    *(volatile unsigned char *)IN_SPPK_ADDR |= temp;

    /* FPK0 input mask register 0x402ce, FPK00 enable */
    *(volatile unsigned char *)IN_SMPK0_ADDR |= 0x1;                          (4)

    /* FPK0 input comparison register 0x402cc, FPK00 high */
    *(volatile unsigned char *)IN_SCPK0_ADDR |= 0x1;                          (5)

    /* Address for port input0,1 interrupt priority register 0x40260 */
    temp = *(volatile unsigned char *)INT_PK0_PK1_ADDR;                       (6)
    temp &= 0xf0;
    temp |= INT_PRIL_LVL3;
    *(volatile unsigned char *)INT_PK0_PK1_ADDR = temp;

    /* Address for input port0-3 and key input0,1 */
    /* interrupt factor flag register 0x40280 */
    *(volatile unsigned char *)INT_FP0_FK_ADDR = INT_FK0;                     (7)

    /* Address for input port0-3 and key input0,1 */
    /* interrupt enable register 0x40270 */
    *(volatile unsigned char *)INT_EP0_EK_ADDR |= INT_EK0;                    (8)
}
```

(1)割り込みの禁止
　誤動作防止のため、キー入力割り込みを禁止します。

(2)入力ポートの設定
　使用するポートの機能を選択します。
　サンプルプログラムではK50ポート機能として入力ポートK50を選択します。

(3)入力端子の選択
　キー入力割り込みに使用する入力端子を選択します。
　各割り込み要因のポート割り当ては以下のとおりです。

表3.8.3 キー入力割り込み用入力端子の選択

割り込み要因	制御ビット	SPPK設定			
		11	10	01	00
FPK1	SPPK1[1:0] (0x402CA•D[3:2])	P2[7:4]	P0[7:4]	K6[7:4]	K6[3:0]
FPK0	SPPK0[1:0] (0x402CA•D[1:0])	P2[4:0]	P0[4:0]	K6[4:0]	K5[4:0]

　サンプルプログラムではK5[4:0](FPK0)を割り込み要因として選択します。

(4)入力マスクレジスタの設定
　キー入力割り込みに使用するポートを選択します。
　ここで選択されたポートが割り込み条件を満たした場合に割り込みが発生します。
　サンプルプログラムではK50(SMPK00)のみ割り込みを有効にしています。

(5)入力比較レジスタの設定
　(4)で割り込みが許可されたポートの割り込み条件を設定します。
　サンプルプログラムでは立ち上がりエッジを選択しています。

(6)割り込みプライオリティレベルの設定
　割り込みのプライオリティレベルを設定します。
　割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
　サンプルプログラムではプライオリティレベルを3に設定しています。

(7)割り込み要因フラグのリセット
　誤動作防止のため、割り込みを許可する前に割り込み要因フラグをリセットします。

(8)割り込みの許可
　キー入力割り込みを許可します。
　以降、K50ポートの立ち上がりエッジで割り込みが発生します。

キー入力割り込み処理部

```
/* Prototype */
void int_key0(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_key0
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : I/O port FPK0 interrupt function.
 *****************************************************************************/
void int_key0(void)
{
      unsigned int j;
      volatile extern unsigned char int_key0_flg;

      int_key0_flg = TRUE;                                                    (1)

      /* Reset key input0 interrupt factor flag 0x40280 */
      *(volatile unsigned char *)INT_FP0_FK_ADDR = INT_FK0;                   (2)
}
```

(1)割り込み処理

選択したポートと入力比較レジスタの内容に不一致が生じると、割り込みが発生します。割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。
サンプルプログラムでは、割り込み発生の確認用に用意されたソフトウェアフラグをセットしています。

(2)割り込み要因フラグのリセット

割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

3.9 A/D変換

ここではGNU33¥sample_ide¥std¥dmt33301¥ad¥src内のサンプルプログラムを例に、ソフトウェアトリガによるA/D変換プログラムについて説明します。チップ内蔵のA/D変換器により、指定した端子からのアナログ入力をA/D変換することができます。

● A/D変換制御プログラム

このサンプルプログラムは、AD0とVss端子間に入力された電圧のA/D変換を、連続モードで128回繰り返し実行します。以下、A/D変換器の初期設定と割り込み処理について解説します。

A/D変換割り込みベクタ部

```
      (unsigned long)int_ad,        // 256  64  A/D converter
```

ベクタテーブルの設定
　割り込みルーチンをベクタテーブルに登録します。

A/D変換器初期設定部

```
/* Prototype */
void init_ad(void);

void init_ad(void)
{
      unsigned char temp;

      /* Set A/D converter interrupt enable on interrupt controller 0x40277 */
      *(volatile unsigned char *)INT_EADE_ECTM_EP4_ADDR &= ~INT_EADE;        (1)
      // Set A/D converter interrupt disable

      /* Set A/D converter enable 0x40244 */
      *(volatile unsigned char *)AD_OWE_ADDR &= ~AD_ADE_ENA;                 (2)
      // A/D disable

      /* Set A/D converter port select 0x402c3 */
      *(volatile unsigned char *)IN_CFK6_ADDR = IN_CFK60_AD0;                (3)
      // A/D AD0 port

      /* Set A/D operation mode select 0x4025f */
      *(volatile unsigned char *)AD_CADV_ADDR = AD_CADV_CMP;                 (4)
      // 209 compatible mode

      /* Set A/D converter prescaler set 0x4014f */
      *(volatile unsigned char *)PRESC_PSAD_ADDR =
            PRESC_PTONL_ON | PRESC_CLKDIVL_SEL4;                             (5)
      // Set A/D converter prescaler(CLK/32)

      /* Set A/D converter status register 0x40242, 0x40243, 0x40244, 0x40245 */
      *(volatile unsigned char *)AD_CH_ADDR = AD_MS_CON | AD_TS_SOFT;        (6)
      // A/D converter software trigger and continuous mode
      *(volatile unsigned char *)AD_CS_ADDR = AD_CS_0 | AD_CE_0;             (7)
      // A/D converter start channel AD0 and A/D end channel AD0
      *(volatile unsigned char *)AD_OWE_ADDR =
            AD_ADE_ENA | AD_ADST_STOP | AD_OWE_NOERR;                        (8)
      // A/D converter enable, A/D converter stop,
      // A/D converter over write error clear
      *(volatile unsigned char *)AD_ST_ADDR = AD_ST_9;                        (9)
      // A/D converter sampling 9 clocks

      /* Set A/D converter interrupt CPU request on interrupt controller 0x40293 */
      *(volatile unsigned char *)INT_RS1_RADE_RP4_ADDR = ~INT_RADE;          (10)
      // IDMA request disable and CPU request enable

      /* Set A/D converter interrupt priority level 3 on interrupt controller */
      /* 0x4026a */
      temp = *(volatile unsigned char *)INT_PSIO1_PAD_ADDR;                  (11)
      temp &= 0x0f;
      temp |= INT_PRIH_LVL3;
      *(volatile unsigned char *)INT_PSIO1_PAD_ADDR = temp;
```

```
    /* Reset A/D converter interrupt factor flag on interrupt controller 0x40287 */
    *(volatile unsigned char *)INT_FADE_FCTM_FP4_ADDR = INT_FADE;          (12)
    // Reset A/D converter interrupt factor flag

    /* Set A/D converter interrupt enable on interrupt controller 0x40277 */
    *(volatile unsigned char *)INT_EADE_ECTM_EP4_ADDR |= INT_EADE;         (13)
    // Set A/D converter interrupt enable
}
```

(1)割り込みの禁止
誤動作を防止するためにA/D変換割り込みを禁止します。

(2)A/D変換の停止
A/D変換器の設定をする間、誤動作防止のためA/D変換を停止します。

(3)入力端子の選択
A/D変換器で使用する端子を設定します。
サンプルプログラムではK60(AD0)を使用します。

(4)動作モードの設定
従来のC33アナログブロックとの互換性を持つ33209互換モードにするか、拡張機能を使用可能な機能拡張モードにするか選択します。
動作モードによる相違点は以下のとおりです。

表3.9.1 33209互換モードと機能拡張モードの相違点

機　　能	33209互換モード	機能拡張モード
変換結果の読み出し	全チャネル共通の変換結果レジスタから読み出します。複数チャネルの変換を行う場合、次のチャネルの変換終了前に変換結果レジスタを読み出す必要があります。	チャネルごとに用意されている変換結果バッファから読み出すことができます。複数チャネルの変換を行う場合、次のチャネルの変換が終了しても、現在のチャネルの変換結果が失われることはありません。
変換終了フラグ、オーバーライトエラーフラグ	それぞれ1ビットを全チャネル共通に使用します。	チャネルごとにそれぞれのフラグが用意されています。
上限値/下限値の比較	この機能はありません。	上限値と下限値を設定し、指定チャネルの変換結果がその範囲内かどうかを確認できます。
割り込み	変換終了割り込みのみ発生可能です。割り込みをチャネル単位にマスクすることはできません。	変換終了割り込みと上下限割り込みを発生可能です。 チャネル単位で変換終了割り込みをマスク可能です。

サンプルプログラムでは33209互換モードを選択します。

(5)入力クロックの選択、クロック供給の開始
プリスケーラの分周クロックを選択し、A/D変換器へクロック供給を開始します。
サンプルプログラムではプリスケーラで1/32に分周した入力クロックを使用します。

(6)A/D変換モードとトリガの設定
A/D変換器のモードと変換開始のトリガを選択します。
A/D変換モードは通常モードおよび連続モードから選択します。通常モードでは選択したチャネル範囲のA/D変換を一回終えると停止するのに対し、連続モードではソフトウェアにより停止するまで選択したチャネル範囲のA/D変換を連続的に実行します。
A/D変換を開始するトリガ方式は外部トリガ、8ビットタイマ0、16ビットタイマ0、ソフトウェアトリガから選択します。
サンプルプログラムでは連続モード、ソフトウェアトリガを選択しています。

(7)アナログ変換開始/終了チャネルの設定
アナログ入力に設定したチャネルの中から、A/D変換の開始チャネルと終了チャネルを選択します。1回の変換動作で開始チャネルから終了チャネルまでのA/D変換を連続的に行います。
サンプルプルグラムではチャネル0(AD0)のみを選択しています。

(8) A/Dイネーブル、エラーフラグリセット
A/D変換を許可するA/Dイネーブルビットを有効にします。
同時にオーバーライトエラーフラグもリセットします。

(9) サンプリング時間の設定
アナログ信号の入力サンプリング時間を設定します。
4段階設定可能ですが、デフォルト(9クロック)で使用してください。

(10) 割り込み/IDMAリクエストの設定
A/D変換終了を割り込み要因としてCPUに割り込みを要求するか、もしくはIDMAを起動するか選択します。
サンプルプログラムでは割り込み要求を行います。

(11) プライオリティレベルの設定
割り込みのプライオリティレベルを設定します。プライオリティレベルが高い割り込みが優先されます。サンプルプログラムではプライオリティレベルを3に設定しています。

(12) 割り込み要因フラグのリセット
イニシャルリセット後、割り込み要因フラグは不定となるため、割り込みを許可する前にリセットします。

(13) 割り込みの許可
A/D変換割り込みを許可します。これ以降、A/D変換が終了すると割り込みが発生します。

割り込み処理

```
/* Prototype */
void int_ad(void) __attribute__((interrupt_handler));

/************************************************************************
 * int_ad
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : A/D converter interrupt function.
 *                Read A/D converter status and A/D convert data.
 ************************************************************************/
void int_ad(void)
{
      extern volatile unsigned short *ad_adr; // A/D data address
      extern volatile int ad_int_flg;         // A/D converter interrupt flag     (1)
      *ad_adr = read_ad_data();               // Read A/D converter data
      write_hex(*ad_adr);                     // Output A/D converter data
      ad_adr++;
      ad_int_flg = TRUE;                      // A/D converter interrupt flag on  (2)
      /* Reset A/D converter interrupt factor flag 0x40287 */
      *(volatile unsigned char *)INT_FADE_FCTM_FP4_ADDR = INT_FADE;
}
```

(1) 割り込み処理の実行
割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。
サンプルプログラムは、変換結果を読み出し、[Simulated I/O]ウィンドウに表示します。
さらに、割り込み確認用のソフトウェアフラグをセットします。

(2) 割り込み要因フラグのクリア
割り込みの発生により割り込み要因フラグが1になりますので、これをリセットします。

3.10 HSDMA転送

ここではGNU33¥sample_ide¥std¥dmt33301¥hsdma¥src内のサンプルプログラムを例に、HSDMA転送プログラムについて説明します。HSDMAはDMA要求に対して即座に対応してデータ転送を行います。

● HSDMA転送プログラム

このサンプルプログラムは、HSDMA Ch.1をソフトウェアトリガで起動し、転送元アドレスのデータを転送先アドレスへ転送します。以下、初期設定部と割り込み処理について解説しています。

HSDMA割り込みベクタ部

```
      (unsigned long)dummy,          // 88   22  High-speed DMA Ch.0
      (unsigned long)int_hsdma1,     // 92   23  High-speed DMA Ch.1
      (unsigned long)dummy,          // 96   24  High-speed DMA Ch.2
      (unsigned long)dummy,          // 100  25  High-speed DMA Ch.3
```

ベクタテーブルの設定

割り込みルーチンをベクタテーブルに登録します。

HSDMA転送初期設定部

```
/* Prototype */
void init_hsdma(unsigned char, unsigned long, unsigned long);

/*****************************************************************************
 * init_hsdma
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned char   HSDMA channel number
 *              unsigned char   HSDMA trigger mode
 *              unsigned long   HSDMA mode and transfer count
 *              unsigned long   HSDMA source address setting
 *              unsigned long   HSDMA destination address setting
 *   Function : Initialize HSDMA setting.
 *****************************************************************************/
void init_hsdma(unsigned char ch, unsigned long src, unsigned long dst)
{
      /* Set HSDMA interrupt enable on interrupt controller 0x40271 */
      *(volatile unsigned char *)INT_EHDM_EIDM_ADDR &= ~INT_EHSDMA1;          (1)

      /* Disable HSDMA transfer 0x4823c, disable HSDMA transfer */
      *(volatile unsigned char *)HSDMA_HS1EN_ADDR &= ~HSDMA_HSEN_ENA;         (2)

      /* HSDMA trigger mode 0x40298, software trigger */
      *(volatile unsigned char *)HSDMA_HSD0S_HSD1S_ADDR = HSDMA_HSD1_SOFT;    (3)

      /* HSDMA mode and control 0x48232, dual mode */
      *(volatile unsigned short *)(HSDMA_TC0H_D0DIR_DUALM0_ADDR + ch * 0x10) =
            HSDMA_DUAL_DUAL;                                                  (4)

      /* HSDMA destination address 0x48238, destination address increment, */
      /* destination address 0xcd0000 */
      *(volatile unsigned long *)(HSDMA_D0ADRL_ADDR + ch * 0x10) =
            HSDMA_DMOD_BLK | HSDMA_INC_INIT | dst;                            (5)

      /* HSDMA source address 0x48234, HWsize, increment, source address 0xcc0000 */
      *(volatile unsigned long *)(HSDMA_S0ADRL_ADDR + ch * 0x10) =
            HSDMA_DATSIZE_HALF | HSDMA_INC_INIT | src;                        (6)

      /* HSDMA transfer count and block size 0x48230, count 1, size 0x80 */
      *(volatile unsigned short *)(HSDMA_BLKLEN0_TC0L_ADDR + ch * 0x10) =
            0x0100 | 0x0080;                                                  (7)

      /* Set HSDMA interrupt priority level 3 on interrupt controller 0x40263 */
      *(volatile unsigned char *)INT_PHSD0_PHSD1_ADDR = INT_PRIH_LVL3;        (8)

      /* Reset HSDMA interrupt flag on interrupt controller 0x40281 */
      *(volatile unsigned char *)INT_FHDM_FIDM_ADDR = INT_FHSDMA1;            (9)

      /* Set HSDMA interrupt enable on interrupt controller 0x40271 */
```

```
    *(volatile unsigned char *)INT_EHDM_EIDM_ADDR |= INT_EHSDMA1;          (10)

    /* Enable IDMA transfer 0x4823c, Enable HSDMA transfer */
    *(volatile unsigned char *)HSDMA_HS1EN_ADDR |= HSDMA_HSEN_ENA;         (11)
}
```

(1)割り込みの禁止
誤動作を防止するため、HSDMA転送の割り込みを禁止します。

(2)HSDMAチャネルの禁止
コントロール情報を設定する前にHSDMAチャネルの動作を禁止します。

(3)トリガ要因の設定
HSDMAを起動するトリガ要因を選択します。割り込み要因をトリガ要因に選択すると、HSDMAは選択した割り込み要因の発生によって起動します。発生した割り込みは要因フラグがセットされますが、HSDMAはこの要因フラグをリセットしませんので注意してください。また、トリガ要因となった割り込みが許可されていれば、1回のDMA転送終了時点でCPUに対して割り込み要求を行います。
サンプルプログラムではソフトウェアトリガを選択しています。

(4)アドレスモード、転送カウンタ(上位ビット)、転送方向(シングルアドレスモードのみ)の設定
シングルアドレスモードかデュアルアドレスモードを選択します。シングルアドレスモードの場合は転送方向も選択します。
このアドレスの転送カウンタ設定ビットは、シングル/連続転送モードの場合は転送カウンタのD[23:16]、ブロック転送モードの場合は転送カウンタのD[15:8]を設定します。
サンプルプログラムではデュアルアドレスモードを選択し、ブロック転送用に転送カウンタを設定しています。

(5)転送先アドレス、転送モードの設定
データの転送先アドレスと転送先アドレスの制御方法を設定します。転送先アドレスの制御方法はインクリメント/デクリメント/固定から選択します。また、HSDMAの転送モードをシングル/連続/ブロック転送から選択します。
サンプルプログラムでは転送先アドレス0xcd0000番地、ブロック転送モード、インクリメント制御を設定しています。

(6)転送データサイズ、転送元アドレスの設定
転送データ1個分のサイズと転送元アドレス、転送元アドレスの制御方法を設定します。
サンプルプログラムでは転送データサイズ = 16ビット、転送元アドレス0xcc0000番地、インクリメント制御を設定しています。

(7)ブロックサイズ、転送カウンタ(下位ビット)の設定
データの転送回数(転送カウンタの下位ビット)を設定します。このアドレスの転送カウンタ設定ビットは、シングル/連続転送モードの場合は転送カウンタのD[15:0]、ブロック転送モードの場合は転送カウンタのD[7:0]を設定します。また、ブロック転送モードの場合は1回に転送されるブロックのサイズも設定します。必ず0以外を設定するようにしてください。
サンプルプログラムでは転送回数1回、ブロックサイズ0x80に設定しています。

(8)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(9)割り込み要因フラグのリセット
誤動作防止のため、割り込みを許可する前に割り込み要因フラグをリセットします。

(10)割り込みの許可
HSDMA割り込みを許可します。
以降、HSDMA転送終了時に割り込みが発生します。

(11)HSDMA転送の許可
HSDMA転送を許可します。
これ以降、トリガ要因が発生するとHSDMA転送を行います。

HSDMA割り込み処理部

```
/* Prototype */
void int_hsdma1(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_hsdma1
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : HSDMA ch.1 interrupt function.
 *****************************************************************************/
void int_hsdma1(void)
{
      extern volatile int hdmaint_flg;

      hdmaint_flg = TRUE;                                                    (1)

      /* Reset HSDMA ch.1 interrupt factor flag 0x40281 */
      *(volatile unsigned char *)INT_FHDM_FIDM_ADDR = INT_FHSDMA1;           (2)
}
```

(1)割り込み処理

HSDMA転送が完了すると割り込みが発生します。割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。

サンプルプログラムでは、割り込み発生の確認用に用意されたソフトウェアフラグをセットしています。

(2)割り込み要因フラグのリセット

割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

3.11 IDMA転送

ここではGNU33¥sample_ide¥std¥dmt33301¥idma¥src内のサンプルプログラムを例に、IDMA転送プログラムについて説明します。
IDMAはRAM上に用意されたコントロール情報によって動作します。

● IDMA転送プログラム

このサンプルプログラムはIDMAをソフトウェアトリガで起動し、転送元アドレスのデータを転送先アドレスへ転送します。以下、Ch.0を例に解説します。

IDMA割り込みベクタ部

```
      (unsigned long)int_idma,      // 104  26  Intelligent DMA
```

ベクタテーブルの設定
　割り込みルーチンをベクタテーブルに登録します。

IDMAコントロール情報設定部

```
/*****************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : IDMA demonstration program.
 *****************************************************************************/
int main(void)
{
      /* Set IDMA ch.0 */
      first_wd = IDMA_LNKEN_ENA | IDMA_LINK | IDMA_CNT | IDMA_BSIZE;          (1)

      second_wd = IDMA_DINTEN_ENA | IDMA_DATSIZ_HW|
            IDMA_SRINC_INC | IDMA_SRC_START0;                                (2)

      third_wd = IDMA_DMOD_BLOCK | IDMA_DSINC_INC | IDMA_DST_START0;          (3)

      write_idma_info((unsigned long *)
            (&dma_control[0]),first_wd,second_wd,third_wd);                  (4)
```

(1)コントロール情報の設定(1st word)
　IDMAコントロール情報を設定します。第1ワードではIDMAリンクイネーブル、IDMAリンクフィールド、転送回数カウンタ、ブロックサイズ(ブロック転送モードの場合)を設定します。IDMAリンクをイネーブルにすると、このチャネルの転送後、IDMAリンクフィールドで指定したチャネルを起動します。ブロック転送モードの場合は、必ずブロックサイズを0以外に設定してください。

(2)コントロール情報の設定(2nd word)
　IDMAコントロール情報を設定します。第2ワードでは終了割り込みイネーブル、データサイズコントロール、転送元アドレスコントロール、転送元アドレスを設定します。データサイズコントロールは転送データ1個分のサイズを設定します。転送元アドレスコントロールは転送後のソースアドレス更新方式を設定します。

(3)コントロール情報の設定(3rd word)
　IDMAコントロール情報を設定します。第3ワードでは転送モード､転送先アドレスコントロール､転送先アドレスを設定します。転送先アドレスコントロールは転送後の転送先アドレス更新方式を設定します。

(4)コントロール情報の書き込み
　上記(1)～(3)で設定したIDMAコントロール情報をRAM上のベースアドレスへ書き込みます。

IDMA初期設定部

```
/* Prototype */
void init_idma(unsigned long, unsigned char);

/*************************************************************************
 * init_idma
 *   Type :     void
 *   Ret val :  none
 *   Argument : unsigned long addr      IDMA control information start address
 *              unsigned char ch        IDMA start channel
 *   Function : Initialize IDMA control information start address and channel.
 *************************************************************************/
void init_idma(unsigned long addr, unsigned char ch)
{
      /* Set IDMA interrupt enable on interrupt controller 0x40271 */
      *(volatile unsigned char *)INT_EHDM_EIDM_ADDR &= ~INT_EIDMA;          (1)

      /* Set IDMA control information address 0x48200 */
      *(volatile unsigned long *)IDMA_DBASEL_ADDR = addr;                   (2)

      /* Set IDMA start channel 0x48204 */
      *(volatile unsigned char *)IDMA_DCHN_ADDR = ch;                       (3)

      /* Set IDMA interrupt priority level 3 on interrupt controller 0x40265 */
      *(volatile unsigned char *)INT_PDM_ADDR = INT_PRIL_LVL3;              (4)

      /* Reset IDMA interrupt flag on interrupt controller 0x40281 */
      *(volatile unsigned char *)INT_FHDM_FIDM_ADDR = INT_FIDMA;            (5)

      /* Set IDMA interrupt enable on interrupt controller 0x40271 */
      *(volatile unsigned char *)INT_EHDM_EIDM_ADDR |= INT_EIDMA;           (6)
}
```

(1)割り込みの禁止
誤動作を防止するため、IDMA転送の割り込みを禁止します。

(2)ベースアドレスの設定
コントロール情報の先頭アドレスをIDMAベースレジスタに設定します。IDMAベースレジスタを設定するアドレスは必ずワード(32ビット)境界で指定してください。

(3)IDMAスタートチャネルの設定
IDMAのチャネル番号を設定します。これによりRAMに書き込むIDMAのコントロール情報のアドレスが決まります。

(4)割り込みプライオリティレベルの設定
割り込みのプライオリティレベルを設定します。
割り込みが同時に発生した場合、プライオリティレベルが高い割り込みが優先されます。
サンプルプログラムではプライオリティレベルを3に設定しています。

(5)割り込み要因フラグのリセット
誤動作防止のため、割り込みを許可する前に割り込み要因フラグをリセットします。

(6)割り込みの許可
IDMA割り込みを許可します。
以降、IDMA転送終了時に割り込みが発生します。

IDMA転送メイン部

```
/*****************************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : IDMA demonstration program.
 *****************************************************************************/
int main(void)
{
      /* Disable IDMA transfer 0x48205 */
      *(volatile unsigned char*)IDMA_DMAEN_ADDR &= 0xfe;                    (1)

      /* Set IDMA ch.0 */                                                   (2)
      first_wd = IDMA_LNKEN_ENA | IDMA_LINK | IDMA_CNT | IDMA_BSIZE;
      second_wd = IDMA_DINTEN_ENA | IDMA_DATSIZ_HW|
            IDMA_SRINC_INC | IDMA_SRC_START0;
      third_wd = IDMA_DMOD_BLOCK | IDMA_DSINC_INC | IDMA_DST_START0;
      write_idma_info((unsigned long *)
            (&dma_control[0]), first_wd, second_wd, third_wd);

      /* Enable IDMA transfer */
      *(volatile unsigned char*)IDMA_DMAEN_ADDR |= 0x01;                    (3)

      /* Start IDMA transfer */
      *(volatile unsigned char *)IDMA_DCHN_ADDR |= 0x80;                    (4)
}
```

(1) IDMA転送の禁止
　誤動作を防止するため、コントロール情報を設定する間はIDMA転送を禁止します。

(2) コントロール情報の設定
　前記の手順でコントロール情報をRAMへ書き込みます。

(3) IDMA転送の許可
　IDMA転送を許可します。

(4) IDMA転送の開始
　トリガ要因であるソフトウェアトリガによりIDMA転送を開始します。

IDMA割り込み処理部

```
/* Prototype */
void int_idma(void) __attribute__((interrupt_handler));

/*****************************************************************************
 * int_idma
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : IDMA interrupt function.
 *****************************************************************************/
void int_idma(void)
{
      extern volatile int idmaint_flg;

      idmaint_flg = TRUE;                                                   (1)

      /* HSDMA ch.0,1 and IDMA interrupt factor flag 0x40281, */
      /* Reset interrupt factor flag */
      *(volatile unsigned char *)INT_FHDM_FIDM_ADDR = INT_FIDMA;            (2)
}
```

(1) 割り込み処理
　IDMA転送が完了すると割り込みが発生します。割り込みが発生するとベクタテーブルに記述されている割り込み処理関数を実行します。サンプルプログラムでは、割り込み発生の確認用に用意されたソフトウェアフラグをセットしています。

(2) 割り込み要因フラグのリセット
　割り込みの発生により割り込み要因フラグが1になっていますので、これをリセットします。

3.12 SLEEP

ここではGNU33¥sample_ide¥std¥dmt33301¥slp¥src内のサンプルプログラムを例に、スタンバイモードであるSLEEPモードへの移行方法を説明します。SLEEPモードではCPU動作に加え、高速発振回路(OSC3)も停止します。そのため低速発振回路(OSC1)と計時タイマを除くすべての周辺回路が停止し、省電力化が実現できます。

● SLEEPモードプログラム

このサンプルプログラムは、SLEEPモードからの復帰時に必要な設定を行った上でslp命令を実行します。SLEEPモードはNMIによって解除されます。SLEEPモード解除後は8ビットタイマで高速発振回路(OSC3)の発振安定を待ち、クロックの供給を再開します。

slp実行部

```
/* Prototype */
void slp_exe(void);
void set_8timer1(void);
void int_8timer1() __attribute__((interrupt_handler));

/*************************************************************************
 * slp_exe
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : slp execution
 *************************************************************************/
void slp_exe(void)
{
      /* Set for stabilizing OSC3 by 8bit timer1 */
      set_8timer1();                                                    (1)

      /* 8bit timer1 interrupt disable */
      *(volatile unsigned char *)INT_E8TU_ADDR &= ~INT_E8TU1;           (2)

      /* 8bit timer1 on for slp */
      *(volatile unsigned char *)OSC_PF1ON_ADDR &= ~OSC_8T1ON_OFF;      (3)

      /* 8bit timer1 run */
      *(volatile unsigned char *)T8P_PTRUN1_ADDR |= T8P_PTRUN_RUN;      (4)

      /* slp */
      asm("slp");                                                       (5)

      /* Reset 8bit timer1 interrupt factor flag on interrupt controller 0x40285 */
      *(volatile unsigned char *)INT_F8TU_ADDR = INT_F8TU1;             (6)
}
```

(1)8ビットタイマの設定
SLEEPモード解除後、高速発振回路(OSC3)は発振安定までにある程度の時間を要します。そのため、SLEEP解除後の高速発振回路(OSC3)の発振安定待ちに8ビットタイマを使用します。
ここでは8ビットタイマに発振待ち時間を設定します。

(2)割り込みの禁止
8ビットタイマを発振安定待ちに使用するため、8ビットタイマ割り込みを禁止します。

(3)高速発振(OSC3)待ち機能の設定
SLEEP解除後の高速発振(OSC3)待ち機能を有効にします。この機能を有効にするとSLEEP解除後、(1)で設定した8ビットタイマがアンダーフローするまでクロックはCPUに供給されません。

(4)8ビットタイマをスタート
8ビットタイマの動作を開始させます。

(5)slp命令の実行
slp命令を実行してSLEEPモードに移行します。これにより、CPU動作、高速発振回路(OSC3)が停止します。

(6)割り込み要因フラグのリセット
SLEEPから復帰後、アンダーフローによって1になった8ビットタイマの割り込み要因フラグをリセットします。

このページはブランクです。

4 テクニカルリファレンス

本章では、S1C33 Familyの開発において必要となる技術的な項目や注意事項について解説します。
説明内のサンプルプログラムや周辺機能は、特に指定したものを除きS1C33301の例です。

4.1 ブートについて

4.1.1 外部RAMからのブート

S1C33 Familyのプロセッサはイニシャルリセット後、ベクタテーブルにあるリセットベクタを読み出します。このベクタテーブルは通常ブートアドレス(S1C33301の場合、0xc00000)に配置されています。
もし、外部RAMからブートしたいときは、ベクタテーブルのベースアドレスをTTBRレジスタ(0x48134～0x48137)によって変更し、外部RAMに配置することによって実現することが可能です。TTBRレジスタに設定するベースアドレスは任意の1KB境界アドレスで指定できます。デバッガからベースアドレスを変更するためのコマンドは以下のとおりです。

TTBRの変更(デバッガコマンド)

```
set {char}0x4812d=0x59        //enable TTBR          (1)
set {long}0x48134=0x0600000   //set address          (2)
```

(1)TTBR書き込み保護の解除
通常、TTBRへの書き込みは禁止されています。書き込み保護を解除するには、TTBR書き込み保護レジスタに0b01011001(0x59)を書き込む必要があります。

(2)ベースアドレスの設定
ベクタテーブルのベースアドレスを設定します。ここで設定したアドレスを先頭にベクタテーブルが配置されます。
上記の例ではベースアドレスを0x600000番地に変更します。

4.1.2 フラッシュメモリからのブート

フラッシュメモリからブートしたいときは、以下の手順でデバッガからフラッシュメモリにブートプログラムを書き込む必要があります。フラッシュメモリ書き込み/消去プログラム“am29f800.elf”はGNU33¥tool¥fls33gに含まれています。

フラッシュメモリへの書き込み(デバッガコマンド)

```
# load symbol information
file am29f800.elf                                          (1)

# decide debugger mode and its port
target icd usb                                             (2)

# load to memory
load am29f800.elf                                          (3)

// flash memory setting
c33 fls 0x0C00000 0x0Cfffff FLASH_ERASE FLASH_LOAD         (4)
c33 fle 0x0C00000 0 0                                      (5)

# load symbol information
file 33301serial_syn_slv.elf                               (6)

# decide debugger mode and its port                        (7)
target icd usb

# load to memory
load                                                       (8)
```

(1) デバッグ情報の読み込み
elf形式のオブジェクトファイルからデバッグ情報のみを読み込みます。
ここでは、ターゲットシステムに搭載されたフラッシュメモリに対応する書き込み/消去プログラムのデバッグ情報を読み込みます。

(2) ターゲットシステムの接続
ターゲットシステムとの接続を確立し、デバッグモードを設定します。
ここでは、USBインタフェースを介してS5U1C33001H(ICD Ver. 3)との接続を確立しています。

(3) プログラムのロード
elf形式のオブジェクトファイルからプログラムを読み込み、ターゲットシステムのメモリにロードします。
ここでは、ターゲットシステムに搭載されたフラッシュメモリに対応する書き込み/消去プログラムをロードしています。

(4) フラッシュメモリの設定
フラッシュメモリにデータを書き込むための設定を行います。設定内容はフラッシュメモリの先頭/終端アドレスと書き込み/消去プログラムのアドレスです。
ここでは、まだフラッシュメモリへの実際の書き込みは行われません。

(5) フラッシュメモリの消去
フラッシュメモリの内容を消去します。指定内容は最初がフラッシュメモリのコントロールアドレスで、後の2つは消去範囲の先頭/終端ブロックです。先頭/終端ブロックに0を指定するとフラッシュメモリの全ブロックが消去されます。

(6) デバッグ情報の読み込み
elf形式のオブジェクトファイルからデバッグ情報のみを読み込みます。
ここでは、フラッシュに書き込むプログラムのデバッグ情報を読み込みます。

(7) ターゲットシステムの接続
ターゲットシステムとの接続を確立し、デバッグモードを設定します。
ここでは、USBインタフェースを介してS5U1C33001H(ICD Ver. 3)との接続を確立しています。

(8) プログラムのロード
elf形式のオブジェクトファイルからプログラムを読み込み、ターゲットシステムにロードします。
ターゲットシステム上のフラッシュメモリ書き込みプログラムにより、ロードしたプログラムがフラッシュメモリに書き込まれます。

以上により、ターゲットシステムはフラッシュメモリ上のブートプログラムで起動可能となります。

4.2 リンカスクリプトについて

4.2.1 .dataセクションの使用法

.dataセクションには、初期値を持ちリード/ライト可能なデータが配置されます。.dataセクションは下記のように初期値をROMに書き込み、プログラムでRAMに転送して使用します。セクションの詳細については、同梱マニュアル“S5U1C33001C Manual”の“3.8.2 セクション”を参照してください。

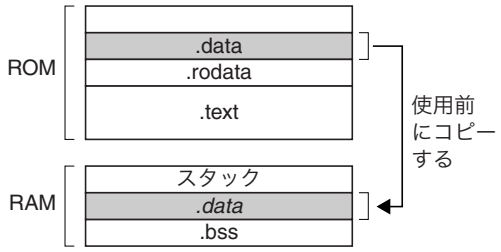


図4.2.1.1 .dataセクション

実際に.dataセクションを使用するには、以下の3つの条件を満たす必要があります。
(1) 変数の初期値をROMに書き込んでおくこと
(2) ROMのデータをRAMに展開すること
(3) RAMに展開されたデータをもとにプログラムが動作すること

ただし、.dataセクション初期値のROMへの配置はリンカが自動的に行い、リンクの設定を記述するリンカスクリプトファイルもGNU33 IDEによって自動生成されます。
したがって、ユーザはROMデータのRAMへの転送のみを、リンカが生成したセクションシンボルを使用して初期化ルーチンで行う必要があります。この処理は、GNU33¥sample_ide¥std¥dmt33301¥xxx¥common¥init.cに含まれるブートプログラムサンプルを使用することで、容易に実現できます(xxxは周辺回路名)。
以下に、Cコードおよびアセンブラコードのブートプログラムサンプルを記載します。

● データ転送プログラム

以下の例のようにセクションシンボルを使用してデータをRAMに転送してから、プログラムの実行を開始する必要があります。GNU33 IDEでリンカスクリプトファイルを生成すると、転送元アドレス、転送先の先頭/終端アドレスを示すシンボルが作成されますので、それを利用してデータのコピーを行います。

アセンブラコードの例

```
/* DATA section copy */
asm("xld.w  %r4,__START_data");     ←RAM上のデータ展開エリアの始点
asm("xld.w  %r5,__END_data");       ←RAM上のデータ展開エリアの終点
asm("xld.w  %r6,__START_data_lma"); ←ROM上の格納エリアの始点

/* data copy */
asm("dat_cpy:");
asm("cmp    %r4,%r5");
asm("jreq   ram_exit");
asm("ld.b   %r7,[%r6]+");
asm("ld.b   [%r4]+,%r7");
asm("jp     dat_cpy");
asm("ram_exit:");
```

Cコードの例

```
extern char __START_data;
extern char __END_data;
extern char __START_data_lma;

char *src = &__START_data_lma;
char *dst = &__START_data;

while (dst < &__END_data)
{
      *dst++ = *src++;
}
```

4.2.2 プログラムの内蔵RAMへのキャッシュ法

外部ROMやフラッシュメモリにプログラムがある場合、1〜2ウェイトのアクセスとなります。この処理速度を上げるため、下記のようにプログラムを内蔵RAMにコピーして0ウェイトで高速実行することができます。

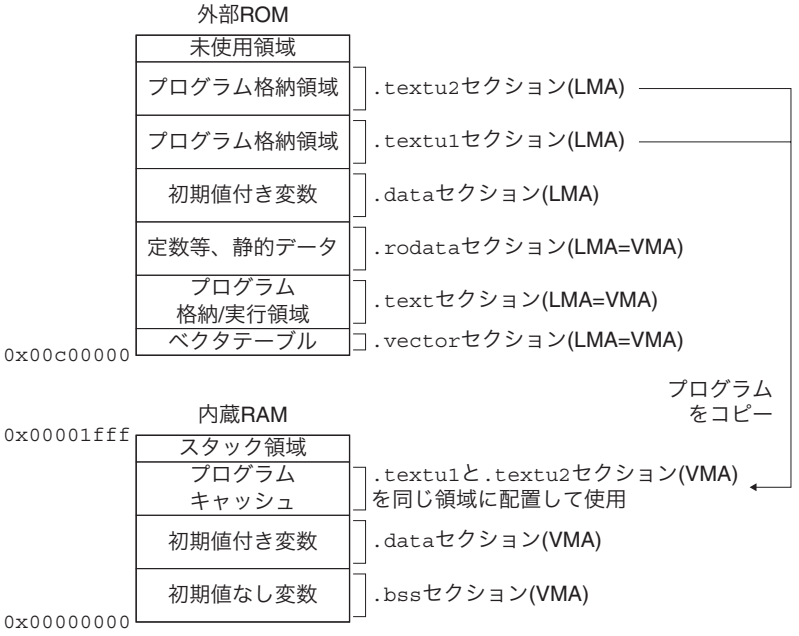


図4.2.2.1 内蔵RAM上でのプログラム実行

前述の.dataセクションと同様、リンカスクリプトファイルの設定が必要です。リンカスクリプトファイルの設定については、同梱マニュアル“S5U1C33001C Manual”の“5.7.6 リンカスクリプトの編集”および“9 リンカ”を参照してください。

● データ転送プログラム

以下のように、セクションシンボルを使用してプログラムをRAMに転送してから、実行を開始させます。また、同じRAM領域を複数のオブジェクトで共用する場合は、同じように各オブジェクトを実行する前にRAMに転送してください。GNU33 IDEでリンカスクリプトファイルを生成すると、転送元アドレス、転送先の先頭/終端アドレスを示すシンボルが作成されますので、それを利用してデータのコピーを行います。

データ転送プログラム例

```
/* TEXTU1 section copy */
asm("xld.w  %r4,__START_textu1");          ←RAM上のデータ展開エリアの始点
asm("xld.w  %r5,__END_textu1");            ←RAM上のデータ展開エリアの終点
asm("xld.w  %r6,__START_textu1_lma");      ←ROM上の格納エリアの始点

/* data copy */
asm("dat_cpy:");
asm("cmp    %r4,%r5");
asm("jreq   ram_exit");
asm("ld.b   %r7,[%r6]+");
asm("ld.b   [%r4]+,%r7");
asm("jp     dat_cpy");
asm("ram_exit:");
```

4.3 Cコンパイラについて

4.3.1 引数

Cコンパイラは汎用レジスタを以下のように使用します。

表4.3.1.1 汎用レジスタの使用法

レジスタ	使用法
%r0	フレームポインタとして使用されるレジスタ
	関数呼び出し時に値を保存する必要のあるレジスタ
%r1	関数呼び出し時に値を保存する必要のあるレジスタ
%r2	
%r3	
%r4	戻り数格納用レジスタ(8/16/32ビットデータ、double型データの下位32ビット)
%r5	戻り数格納用レジスタ(double型データの上位32ビット)
%r6	引数渡し用レジスタ(第1ワード)
%r7	引数渡し用レジスタ(第2ワード)
%r8	引数渡し用レジスタ(第3ワード)
%r9	引数渡し用レジスタ(第4ワード)
%r10	スクラッチレジスタ/未使用
%r11	
%r12	
%r13	
%r14	
%r15	デフォルトデータエリアポインタレジスタ*
%dp	デフォルトデータエリアポインタレジスタ(-mc33advオプション(C33 ADVコア)指定時)

* -medda32オプション(デフォルトデータエリア未使用)未指定時

この表から分かるように、関数呼び出し時の引数渡し用に%r6～%r9の4つのレジスタが割り当てられています。引数がレジスタ数を上回る場合(double、long long型はレジスタを2個使用)、レジスタが足りない分はスタックを介して関数に渡されます。この際、引数渡し用のレジスタがスタックのアドレスを指します。これにはメモリ操作が発生するため、引数が4個以下の場合に比べオーバーヘッドが大きくなります。コード効率を考慮した場合、関数呼び出し時の引数は4個以下にする方が良い結果が得られます。

4.3.2 代入

複数の機能が割り当てられているI/Oレジスタを設定する場合、変更したくないビットをマスクして値を代入する必要があります。このときのマスクと代入の仕方によって、コンパイルされたアセンブラコードのコード効率が変化します。
以下に、8ビットタイマの割り込みプライオリティレベルを設定するルーチンを例に、この違いを説明します。

(1)テンポラリ変数に一旦I/Oレジスタ値をロードし、マスク/値代入後にメモリにストアした場合

Cコード

```
temp = *(volatile unsigned char *)INT_P8TM_PSIO0_ADDR;
temp &= 0xF0;
temp |= INT_PRIL_LVL3;
*(volatile unsigned char *)INT_P8TM_PSIO0_ADDR = temp;
```

アセンブラコード

```
sub   %r5,0x29
ld.ub %r4,[%r5]
ext   0x3
xand  %r4,0xf0
or    %r4,0x3
ld.b  [%r5],%r4
```

(2)メモリ上のI/Oレジスタに直接マスクと値代入の操作を行った場合

Cコード

```
*(volatile unsigned char *)INT_P8TM_PSIO0_ADDR &= 0x0F;
*(volatile unsigned char *)INT_P8TM_PSIO0_ADDR |= INT_PRIL_LVL3;
```

アセンブラコード

```
ld.ub %r4,[%r5]
xand  %r4,0xfffffff0    (符号拡張)
ld.b  [%r5],%r4
ld.ub %r4,[%r5]
or    %r4,0x3
ld.b  [%r5],%r4
```

上記例の大きな違いはメモリ操作になります。コードとしては(1)の方が複雑に見えますが、コンパイル後のアセンブラコードを見ると(2)の方がメモリ操作が多いことが分かります。メモリ操作はオーバーヘッドや依存関係が発生するため、コード効率を考慮する場合は(1)の方が効率の良いコードといえます。

4.4 DMA転送について

HSDMAはデュアルアドレス転送とシングルアドレス転送の2つのデータ転送モードに対応しています。

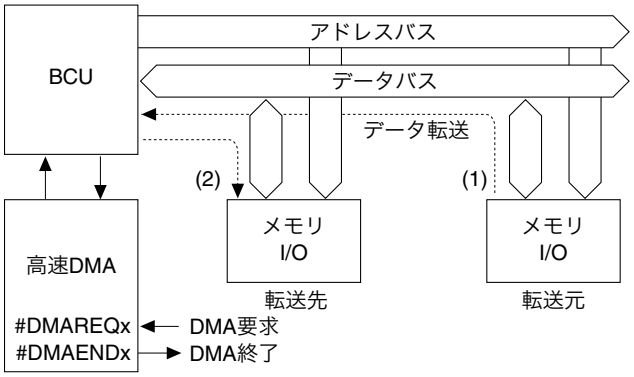


図4.4.1 デュアルアドレス転送

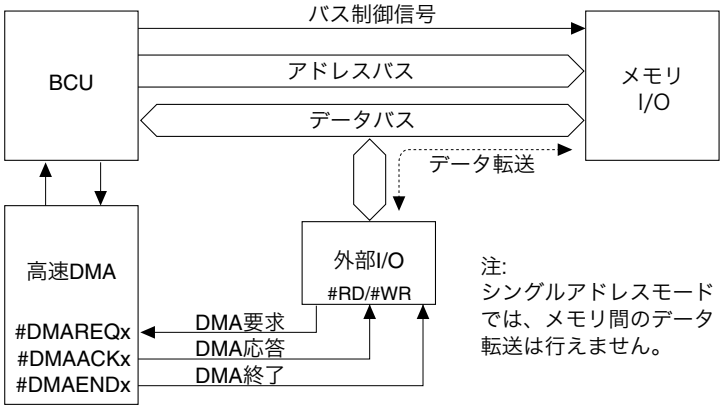


図4.4.2 シングルアドレス転送

デュアルアドレス転送モードでは転送元アドレスからチップ上のテンポラリレジスタにデータを読み出し、次にテンポラリレジスタから転送先アドレスへデータを書き込みます。これによりメモリ間のデータ転送を可能にしています。
一方、シングルアドレス転送では、外部バス上で転送元からの読み出しと転送先への書き込みを一度に実行します。このため、メモリ間のデータ転送は行えませんが、テンポラリレジスタが不要なため、より高速なデータ転送が行えます。

ただし、DMAはC33コアによるメモリアクセスを中止してデータ転送を行うため、C33コアがプログラム実行中にDMAが発生するとC33コアが一時的にアイドル状態になってしまいます。このため、場合によってはDMAを使用しない方が高速になる可能性もあります。

S1C33 Family
スタートアップマニュアル

セイコーエプソン株式会社
半導体事業部　IC営業部

〈IC国内営業グループ〉

東京　〒191-8501　東京都日野市日野421-8
TEL（042）587-5313（直通）　FAX（042）587-5116

大阪　〒541-0059　大阪市中央区博労町3-5-1　エプソン大阪ビル15F
TEL（06）6120-6000（代表）　FAX（06）6120-6100

インターネットによる電子デバイスのご紹介　http://www.epson.jp/device/semicon/

ドキュメントコード：410918600

2007年2月作成 ©A