参 考 資 料

JAJA187-2009年8月

Code Composer Studio v3.3による OMAP-L137 ARM926EJ-S プログラミング入門

生駒　伸一郎　　営業・技術本部　　マーケティング/応用技術統括部

アブストラクト

この資料では弊社ソフトウェア統合開発環境“CODE COMPOSER STUDIO V3.3(以下CCS)”を使用し、OMAP-L137のARM926EJ-Sプロセッサをターゲットに、何らかのOSを使用するのではなく一からソフトウェアを構築するための導入部分を紹介します。またハードウェアの評価/デバッグなどにご活用いただけるように組込みプログラムとして最低限必要な割込み処理やキャッシュ、C674X DSPコアとのデータ受け渡しなどを、サンプル・プログラムをもとに紹介します。

目次

図

表

1 CCS と TMS470 コード生成ツールによる実行可能ファイルの作成

1.1 CCS と TMS470 コード生成ツール概要

CCSは弊社DSPコア(C2x, C5x, C6x)用のコード生成ツールだけでなく、ARMプロセッサ用のコード生成ツール“TMS470 Code Generation Tools(以下TMS470 CGT)”を含んだ形で提供されており、C/C++言語及びアセンブリ言語でのプログラミングが可能です。TMS470 CGTはARM7、ARM9、CortexA8などの複数のARMプロセッサに対応しています。またCCSのデバッグ機能はTMS470 CGTで作成した実行可能ファイルのターゲットへのロード、実行、シンボリックデバッグ（ステップ実行、変数ウォッチなど）、ARMコアレジスタの参照、メモリ内容の参照などが可能です。

なおこの資料はCCS version 3.3.80.11、TMS470 CGT version 4.5.1をもとに紹介します。各ツールの最新バージョンはCCSのアップデート・アドバイザにより、ダウンロードが可能です。

1.2 プロジェクトの作成とビルド・オプション

OMAP-L137にはARM926EJ-SプロセッサとC674x DSPコアという２つのプロセッサが搭載されていますので、CCSを起動すると“パラレル・デバッグ・マネージャ”が立ち上がります。

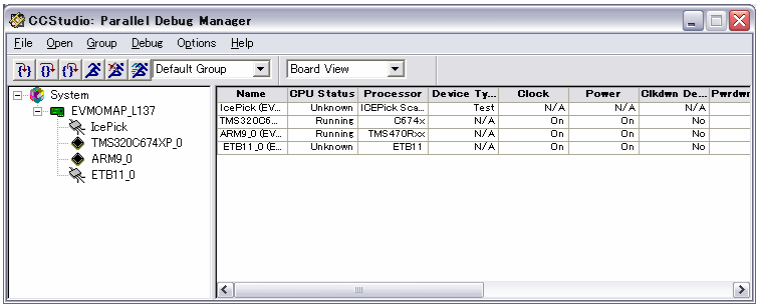

図１　パラレル・デバッグ・マネージャ

ARM926EJ-S プロセッサ向けに作業を行うには“ARM9_0”を、C674x DSP コア向けに作業を行うには“TMS320C674XP_0”をダブルクリックしてメインウィンドウを開き、メインメニューから操作を行います。

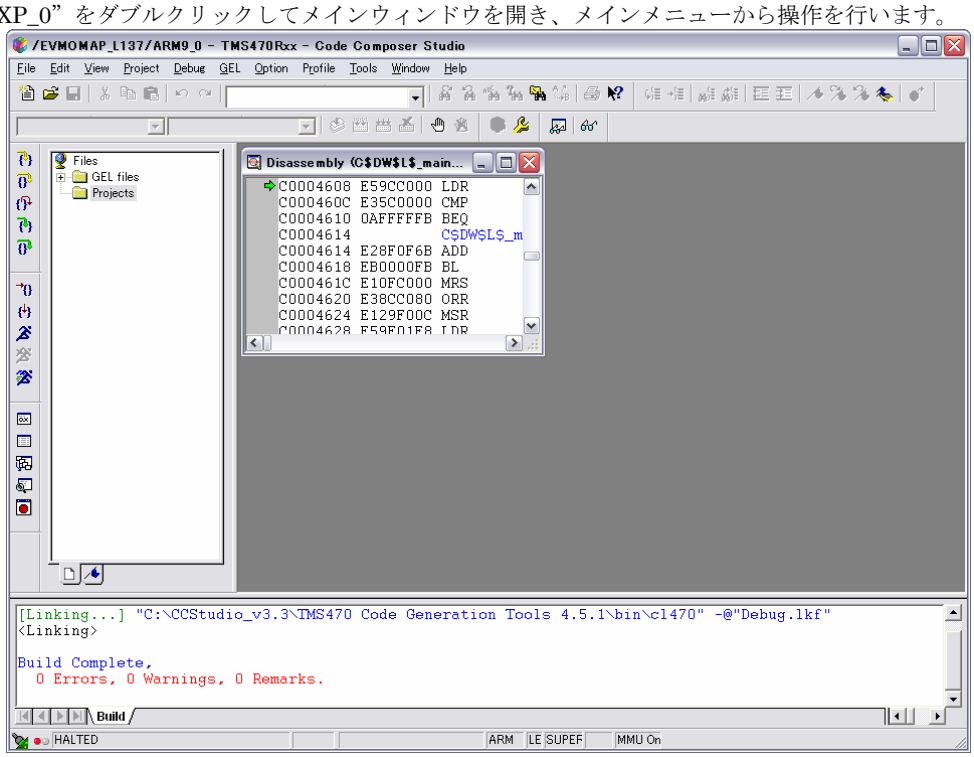

図２　CCSメインウィンドウ

CCSでのプログラミングはプロジェクトの作成から始めます。CCSのGUI（グラフィカル・ユーザー・インターフェイス）は、プロセッサの種類にかかわらず同様のインターフェイスです。CCSの一般的な使用方法はCCS付属のヘルプやCCSの入門マニュアル 日本語版などをご参照ください。

Code Composer Studio Tutorial : Code Composer Studio のメインメニュー “Help” -> “tutorial”
Code Composer Studio 開発ツール v3.3 入門マニュアル [JAJU021] : http://www.ti.com/jp/lit/jaju021

またコンパイラ、リンカなどの詳細はTMS470 CGTのマニュアルをご参照ください。この資料ではOMAP-L137向けに注意が必要な部分を紹介します。

TMS470R1x Optimizing C/C++ Compiler v4.4 User's Guide [SPNU151D] : http://www.ti.com/jp/litv/pdf/spnu151d
TMS470R1x Assembly Language Tools v4.4 User's Guide [SPNU118G] : http://www.ti.com/jp/litv/pdf/spnu118g
これらのPDFファイルはTMS470 CGTをインストールしたフォルダの下のdocsフォルダにもインストールされます。

新規のプロジェクト作成に関して、“Target Family:”はデフォルトのTMS470R2Xでかまいません。

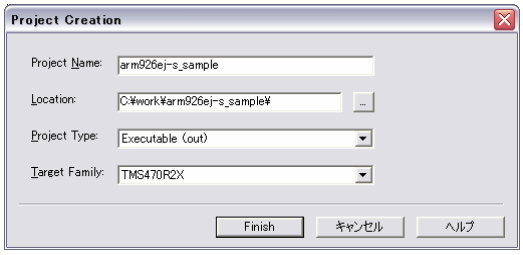

図３　新規プロジェクトの作成ダイアログ

ビルド・オプションに関して、コンパイラタブではOMAP-L137の仕様にあわせ、リトル・エンディアンのオプション“-me”を追加します。ビルド・オプションは非常にたくさんありますので、“Category:”を選択することによってGUIが切り替わるようになっています。ウィンドウ上部のテキスト・フィールドに直にオプションを入力してもかまいません。

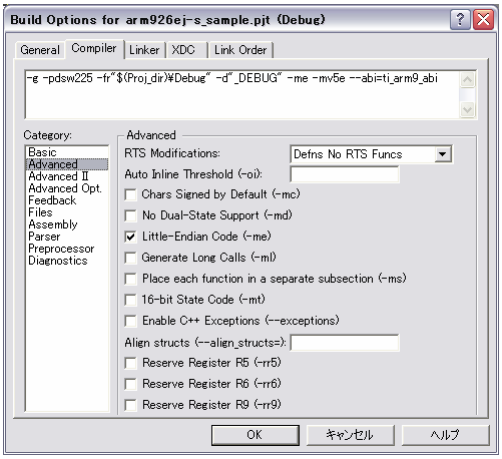

図４　ビルド・オプションダイアログ（リトル・エンディアン）

リンカタブではスタック領域、ヒープ領域のサイズの設定、及びランタイム・サポート・ライブラリの設定を追加します。ランタイム・サポート・ライブラリはTMS470 CGTの一部として提供されるC/C++言語でのプログラミングをサポートするライブラリで、各言語の標準関数や、初期化ルーチンをサポートしています。TMS470 CGTのランタイム・サポート・ライブラリ（以下RTSライブラリ）はARMプロセッサの種類やABI(Application Binary Interface)の種類、エンディアンネス、モード、C++例外サポートの有無ごとにライブラリファイル（拡張子“lib”）が用意されていますので、適切なものを選択します。この資料付属のサンプルではOMAP-L137向けにARM9、リトル・エンディアンで、モードはデフォルトの32bit(ARMモード)、ABIもデフォルトのarm9-abi(COFF形式)、C言語でのプログラミングということでC++例外サポート無を選択しました。

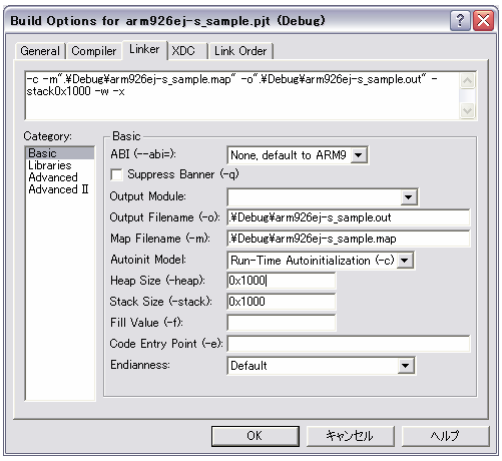

図 5　ビルド・オプションダイアログ（スタック領域、ヒープ領域のサイズ）

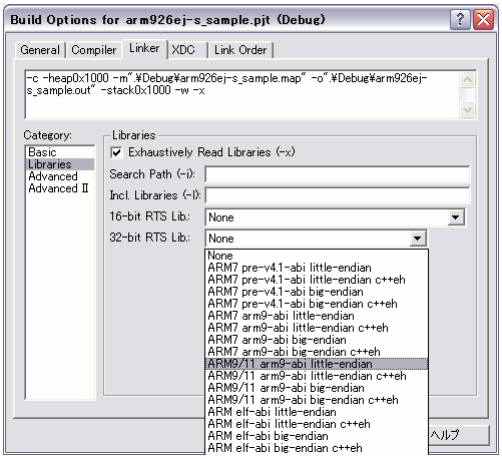

図 6　ビルド・オプションダイアログ（ランタイム・サポート・ライブラリ）

1.3　リンカ・コマンド・ファイルの作成

プログラムのメモリ配置はリンカ・コマンド・ファイル（拡張子“cmd“）で設定します。OMAP-L137では全てのRAM領域をARM926EJ-Sプロセッサからアクセス可能ですが、付属サンプルでは“OMAP-L137/TMS320C6747 浮動小数点スタータ・キット（以下OMAP-L137 OSK）”のボードにあわせARM local RAM（0xFFFF0000～0xFFFF1FFF）とSDRAMの一部（16Mbyte、0xC0000000～0xC1000000）を使用する設定にし、割込みベクタ以外のセクションは全てSDRAMへ配置することにしました。処理内容によってはShared RAM（0x80000000～0x8001FFFF）を活用するのもいいでしょう。

```
MEMORY
{
LOCAL_RAM : org = 0xFFFF0000  len = 0x1FFF
SDRAM : org = 0xC0000000  len = 0x1000000
}

SECTIONS
{
.intvecs   : {} > LOCAL_RAM  /* INTERRUPT VECTORS                  */
.text      : {} > SDRAM      /* CODE                               */
.cinit     : {} > SDRAM      /* INITIALIZATION TABLES              */
.const     : {} > SDRAM      /* CONSTANT DATA                      */
.sysmem    : {} > SDRAM      /* DYNAMIC MEMORY ALLOCATION AREA     */
.bss       : {} > SDRAM      /* GLOBAL & STATIC VARS               */
.stack     : {} > SDRAM      /* SOFTWARE SYSTEM STACK              */
}
```

リンカ・コマンド・ファイルサンプル（“linker.cmd”）

以上で基本的な設定が終わりました。簡単なmain関数のソースファイルを用意し、ソースファイルとリンカ・コマンド・ファイルをプロジェクトへ追加するとビルド、実行できるでしょう。

```
#include <stdio.h>

void main(void)
{
  printf("Hello world!\n");
}
```

簡単なmain関数のソースファイルサンプル

ランタイム・サポート・ライブラリの標準出力はCCSのOutputウィンドウ“Stdout”タブになります。

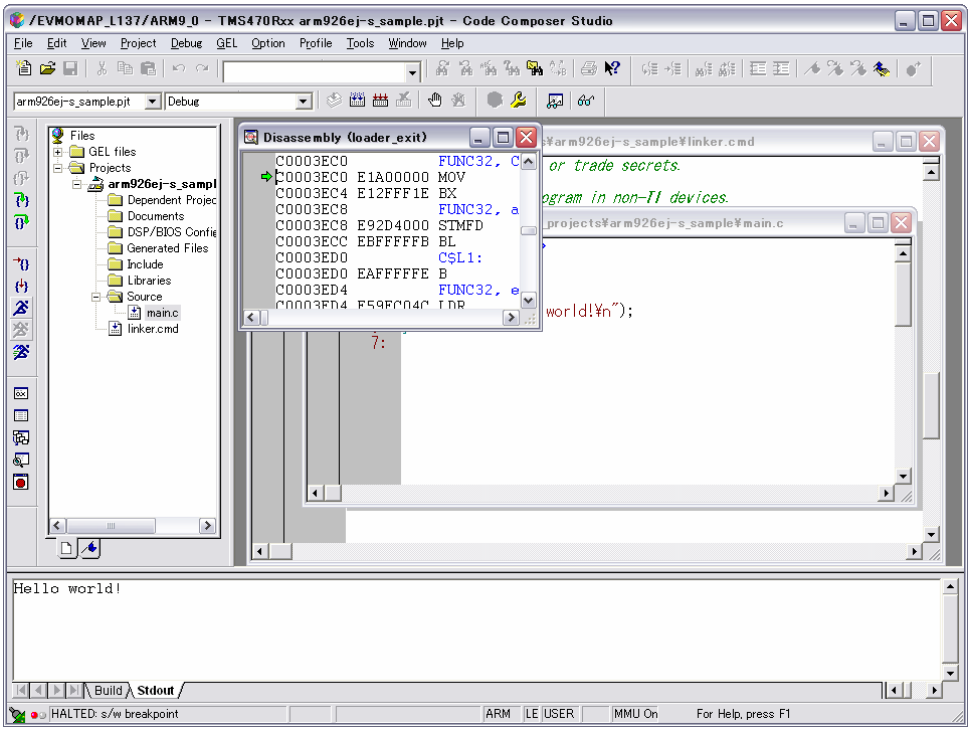

図 7　簡単なmain関数のプログラムをビルド、実行した様子

1.4　周辺機能の設定（チップ・サポート・レジスター・コンフィグレーション・レイヤー）

OMAP-L137 OSK付属のSDK（ソフトウェア・デベロップメント・キット）にはARM926EJ-Sプロセッサ向けに一からソフトウェアを構築するためのヘッダファイル等は用意されていませんが、C674x DSPコア用には周辺機能のレジスタのアドレス値や各レジスタのもつフィールド定義などが記述されているヘッダファイルが用意されています。このC674x DSPコア用ヘッダファイルを“チップ・サポート・レジスター・コンフィグレーション・レイヤー（以下CSLR）”と呼んでいます。CSLRは全てC言語で記述されおり、少しの修正でARM926EJ-Sプロセッサでも使用できます。そこで付属サンプルではSDK 1.00.00.10で提供されているCSLRに修正を施したものを同封し、プログラムの中で使用しています（修正内容に関しては“５．サンプル・プログラムについて”をご参照ください）。

CSLRでは周辺機能ごとにヘッダファイルが用意されており、各ヘッダファイルには周辺機能の持つレジスタ郡をまとめた構造体と各レジスタの持つフィールドが定義されています。例えばタイマー用のヘッダファイルはcslr_tmr.hです。またsoc_OMAPL137.hに各周辺機能のレジスタのベースアドレスが定義されています。以下に使用例とcslr_tmr.hの一部を示します。１、２行目で２つのヘッダファイルをインクルードしています。５行目で構造体を使ったポインタ型変数を用意し、ベースアドレスを代入しています（CSL_TMR_0_REGSはsoc_OMAPL137.hで定義されているベースアドレスです）。そしてこのポインタ型変数を参照しながら各レジスタへアクセスします。CSL_FINST()やCSL_FINS()はCSLRが用意しているマクロでread-modify-writeを行い、指定したフィールドへ値を書き込みます。例えば３１行目はtimer0のPRD12レジスタ内PRD12フィールドに変数timer_periodの値を書き込む記述になります。マクロはいくつか用意されており、３２ビット値を作るだけのCSL_FMK()やフィールド値を読み出すCSL_FEXT()などがCSLRのcslr.hで定義されていますのでご活用ください。

```
 1:#include <ti/pspiom/cslr/soc_OMAPL137.h>
 2:#include <ti/pspiom/cslr/cslr_tmr.h>
 3:#include "led.h"
 4:
 5:CSL_TmrRegsOvly tmr0Regs = (CSL_TmrRegsOvly)CSL_TMR_0_REGS;
 6:
 7:void tmr0_init(void)
 8:{
 9:  /*internal clock source is used. 24MHz*/
10:  /*1ms count is 24000(0x5DC0)*/
11:  unsigned int timer_period = 0x00005DC0;
12:
13:  /*reset*/
14:  tmr0Regs->TGCR = 0x0;
15:  tmr0Regs->TCR = 0x0;
16:  CSL_FINST(tmr0Regs->TIM12,TMR_TIM12_TIM12,RESETVAL);
17:  CSL_FINST(tmr0Regs->TIM34,TMR_TIM34_TIM34,RESETVAL);
18:  CSL_FINST(tmr0Regs->INTCTLSTAT,TMR_INTCTLSTAT_PRDINTSTAT12,PEND);
19:  CSL_FINST(tmr0Regs->INTCTLSTAT,TMR_INTCTLSTAT_PRDINTSTAT34,PEND);
20:  CSL_FINST(tmr0Regs->INTCTLSTAT,TMR_INTCTLSTAT_PRDINTEN12,DISABLE);
21:  CSL_FINST(tmr0Regs->INTCTLSTAT,TMR_INTCTLSTAT_PRDINTEN34,DISABLE);
22:
23:  CSL_FINST(tmr0Regs->EMUMGT,TMR_EMUMGT_FREE,FREE);
24:
25:  /* Select 32bit unchain mode   */
26:  /* Take the timer out of reset */
27:  tmr0Regs->TGCR = CSL_FMKT(TMR_TGCR_TIMMODE,32BIT_UNCHAIN)
28:    | CSL_FMKT(TMR_TGCR_TIM12RS,NO_RESET)
29:    | CSL_FMKT(TMR_TGCR_TIM34RS,NO_RESET);
30:
31:  CSL_FINS(tmr0Regs->PRD12,TMR_PRD12_PRD12,timer_period);
32:
33:  CSL_FINST(tmr0Regs->INTCTLSTAT,TMR_INTCTLSTAT_PRDINTEN12,ENABLE);
34:
35:  CSL_FINST(tmr0Regs->TCR,TMR_TCR_ENAMODE12,EN_CONT);
36:}
```

付属サンプル“timer.c”からの抜粋（行番号はこの資料のために追加）

```
/**************************************************************************\
* Register Overlay Structure
\**************************************************************************/
typedef struct  {
    volatile Uint32 REVID;
    volatile Uint32 EMUMGT;
    volatile Uint32 GPINTGPEN;
    volatile Uint32 GPDATGPDIR;
    volatile Uint32 TIM12;
    volatile Uint32 TIM34;
    volatile Uint32 PRD12;
    volatile Uint32 PRD34;
    volatile Uint32 TCR;
    volatile Uint32 TGCR;
    volatile Uint32 WDTCR;
    volatile Uint8 RSVD0[8];
    volatile Uint32 REL12;
    volatile Uint32 REL34;
    volatile Uint32 CAP12;
    volatile Uint32 CAP34;
    volatile Uint32 INTCTLSTAT;
    volatile Uint8 RSVD1[24];
    volatile Uint32 CMP0;
    volatile Uint32 CMP1;
    volatile Uint32 CMP2;
    volatile Uint32 CMP3;
    volatile Uint32 CMP4;
    volatile Uint32 CMP5;
    volatile Uint32 CMP6;
    volatile Uint32 CMP7;
} CSL_TmrRegs;

/**************************************************************************\
* Overlay structure typedef definition
\**************************************************************************/
typedef volatile CSL_TmrRegs         *CSL_TmrRegsOvly;
```

CSLR“cslr_tmr.h”からの抜粋

2 割込み処理ルーチンの作成

2.1 OMAP-L137 ARM926EJ-S の割込み概要

OMAP-L137 ARM926EJ-Sの割込み概要図を以下に示します。各周辺機能及びGPIOからの割込み要求は割込みコントローラ“AINTC”を介してARM926EJ-Sの割込み入力FIQまたはIRQにまとめられ、伝わります。AINTCは周辺機能からの個別の割込み要求フラグ、許可/禁止、FIQ/IRQへのマッピングができ、また、まとめられたFIQ/IRQをそれぞれ許可/禁止することができます。さらにARM926EJ-SはCPSR（カレント・プログラム・ステータス・レジスタ）によりFIQ/IRQを許可/禁止します。

周辺機能からの割込み要求は80個におよび、各割込み要求はハードウェア固定の番号が振られています。例えば付属サンプルで使用しているTimer0（のカウンタ下位32bit）の割込み要求は21、UART2の割込み要求は61といった具合です。この番号をシステム・インタラプト・インデックスと呼びます。一方ARM926EJ-S側にもハードウェア固定の番号が振られており、FIQが0、IRQが1です。この番号をホスト・インタラプト・インデックスと呼びます。

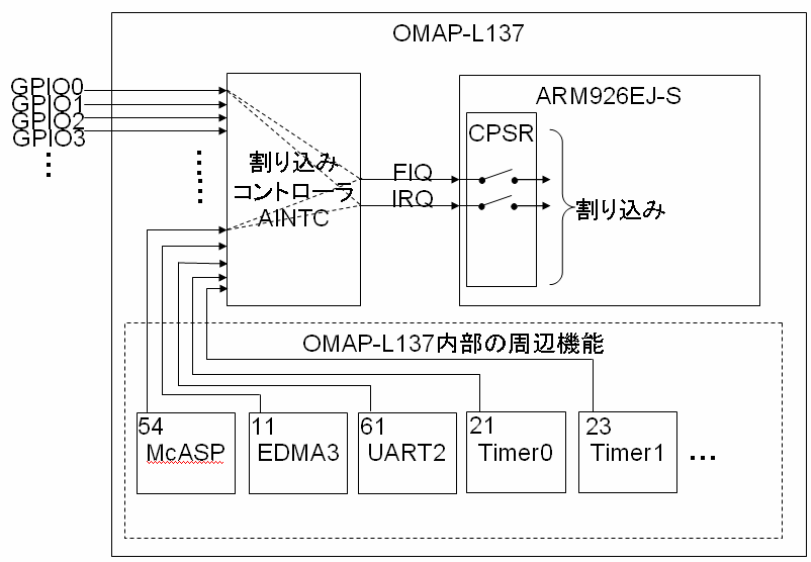

図 8　OMAP-L137割込み概要

割込み要求のFIQ/IRQへのマッピングはAINTCの持つ32個チャネルを介して行います。チャネルからFIQ/IRQへのマッピングはハードウェア固定で、チャネル0,1はFIQへ、2～31はIRQへマッピングされています。一方各周辺機能からの割込み要求のチャネルへのマッピングはソフトウェアで行います。複数の割込み要求を１つのチャネルにマッピングすることも可能です。またAINTCはこのチャネル番号を優先順位として使い、チャネル0が最も優先順位が高く、チャネル31が最も低くなります。

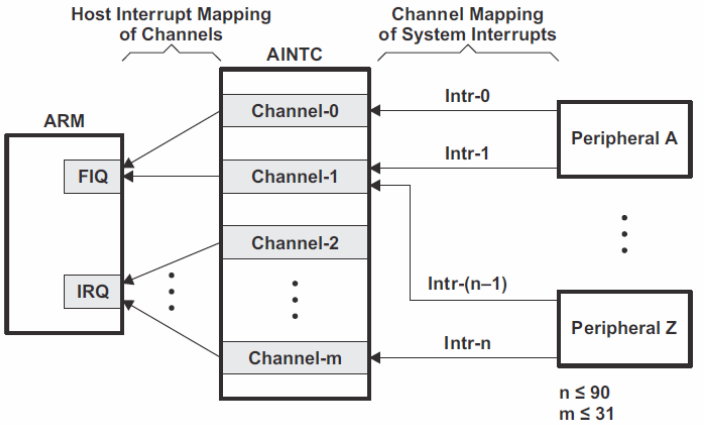

図 9　割り込み要求のマッピングとチャネル

多くの組込みシステムではリアルタイム処理を必要とし、優先順位を基準としたプリエンプティブな処理が欠かせません。AINTCは優先順位を基準とした割込み処理のネスティング（プリエンプティブな割込み処理）を実現するために、以下の３つの方法をサポートしています。

①. 全てのチャネル番号を基準としてネスティングする。ある割込みが派生したとき、その割込みと優先順位が同じかもしくは低い割込みはハードウェアで半自動的に禁止され、優先順位が高い割込み要求が許可される。

②. FIQ/IRQごとにチャネル番号を基準としてネスティングする。ある割込みが派生したとき、その割込みと優先順位が同じかもしくは低い割込みはハードウェアで半自動的に禁止され、優先順位が高い割込み要求が許可される。

③. ソフトウェアで優先順位を管理してネスティングする。ある割込みが派生したときソフトウェアの持つ優先順位を基準に各割込み要求の禁止/許可を再設定する。

①または②の方法を使用すると割込みハンドラのソフトウェア処理を減らすことができ、割込みサービスルーチンをコールする際のオーバーヘッドを軽減できます。ただしチャネル番号を優先順位として使いますので①の方法での優先順位は32段階、②の方法での優先順位はFIQに２段階、IRQに30段階になります。そのため各段階数以上の割込み要求をハンドルするシステムでは同じ優先順位（チャネル）に複数の割込み要求をマッピングする必要が生じます。AINTCの詳細は
OMAP-L137 Applications Processor System Reference Guide [SPRUG84B] : http://www.ti.com/jp/litv/pdf/sprug84b
をご参照ください。

割込み要求（を含む例外）が発生すると強制的にベクタアドレスへ処理が移ります。OMAP-L137 ARM926EJ-Sのベクタアドレスは0xFFFF0000からはじまり、オフセットは以下の通りです。Iビット、FビットはARM926EJ-S内部CPSRのビットで、IビットでIRQの禁止/許可を、FビットでFIQの禁止/許可をコントロールします。これらのビットはセット（１に）することで禁止になり、下記表で“Set”と記述のある例外(Exception)は、例外発生時にIRQ/FIQがハードウェアで自動的に禁止になることを意味します。

Vector Offset Address	Exception	Mode on entry	I Bit State on Entry	F Bit State on Entry
0h	Reset	Supervisor	Set	Set
4h	Undefined instruction	Undefined	Set	Unchanged
8h	Software interrupt	Supervisor	Set	Unchanged
Ch	Pre-fetch abort	Abort	Set	Unchanged
10h	Data abort	Abort	Set	Unchanged
14h	Reserved	—	—	—
18h	IRQ	IRQ	Set	Unchanged
1Ch	FIQ	FIQ	Set	Set

表１　OMAP-L137 ARM926EJ-Sのベクタテーブル

ARM926EJ-SプロセッサはARM v5アーキテクチャに基づき、ARM9EJ-Sコアをベースにキャッシュ やMMUなどを搭載したプロセッサです。詳細はARM社のドキュメント・ホームページhttp://infocenter.arm.com/help/index.jspより以下の資料をご参照ください。

ARM926EJ-S Technical Reference Manual
ARM9EJ-S Technical Reference Manual
ARM v5 Architecture Reference Manual

2.2　boot.asm の修正

TMS470 CGTでビルドした実行可能ファイルのプログラム開始ポイント（エントリー・ポイント）は_c_int00になります。_c_int00ではスタックポインタの設定とグローバル/スタティック変数の初期値代入を行った後main関数をコールし、以降ユーザーのプログラムが処理されます。TMS470 CGTはスタックポインタとしてARM9EJ-Sコア内部レジスタR13を使用しますのでR13にスタック領域の初期アドレスを設定しますが、ARM v5アーキテクチャではR13はプロセッサモードごとにバンク・レジスタを持っていますのでプログラムで使用するモードのR13がスタック領域を指すように_c_int00でモードを切り替えて、初期化します。

TMS470 CGTのRTSライブラリでは_c_int00の処理がUserモードで行われるようになっていますが、Userモードは Privilegedモード（特権モード）ではないためにアクセスできるレジスタに制限があり、小規模な組込みシステムとしては使いづらい面があります。そこで付属サンプルでは_c_int00のスタックポインタ初期化ルーチン“boot.asm”のソースをコピーしてSupervisorモードで_c_int00の処理を行うように修正、プロジェクトに追加しました。RTSライブラリのソースファイルはTMS470 CGTをインストールしたフォルダの下の“lib”フォルダにあるrtssrc.zipに含まれています。以下に修正したboot.asmの一部を示します。修正内容は３７行目から４３行目のUserモードへ切り替える部分のコメントアウトと、４４行目から５０行目のSupervisorモードへの切り替え処理の追加のみです。付属サンプルでは32ビットモード（ARMモード）での動作部分のみを修正していることに注意してください。

```
 1:;*****************************************************************************
 2:;*  32 BIT STATE BOOT ROUTINE                                                *
 3:;*****************************************************************************
 4:
 5:        .global __stack
 6:;*************************************************************
 7:;* DEFINE THE USER MODE STACK (DEFAULT SIZE IS 512)
 8:;*************************************************************
 9:__stack:.usect  ".stack", 0, 4
10:
11:        .global _c_int00
12:;*************************************************************
13:;* FUNCTION DEF: _c_int00
14:;*************************************************************
15:_c_int00: .asmfunc
16:
17:        .if __TI_NEON_SUPPORT__ | __TI_VFP_SUPPORT__
18:        ;*-------------------------------------------------------
19:        ;* SETUP PRIVILEGED AND USER MODE ACCESS TO COPROCESSORS
20:        ;* 10 AND 11, REQUIRED TO ENABLE NEON/VFP
21:        ;* COPROCESSOR ACCESS CONTROL REG
22:        ;* BITS [23:22] - CP11, [21:20] - CP10
23:        ;* SET TO 0b11 TO ENABLE USER AND PRIV MODE ACCESS
24:        ;*-------------------------------------------------------
25:        MRC      p15,#0x0,r0,c1,c0,#2
26:        MOV      r3,#0xf00000
27:        ORR      r0,r0,r3
28:        MCR      p15,#0x0,r0,c1,c0,#2
29:
30:        ;*-------------------------------------------------------
31:        ; SET THE EN BIT, FPEXC[30] TO ENABLE NEON AND VFP
32:        ;*-------------------------------------------------------
33:        MOV      r0,#0x40000000
34:        FMXR     FPEXC,r0
35:        .endif
36:
37:;        ;*-------------------------------------------------------
38:;        ;* SET TO USER MODE
39:;        ;*-------------------------------------------------------
40:;        MRS     r0, cpsr
41:;        BIC     r0, r0, #0x1F  ; CLEAR MODES
42:;        ORR     r0, r0, #0x10  ; SET USER MODE
43:;        MSR     cpsr, r0
44:        ;*-------------------------------------------------------
45:        ;* SET TO SVC MODE
46:        ;*-------------------------------------------------------
47:        MRS     r0, cpsr
48:        BIC     r0, r0, #0x1F  ; CLEAR MODES
49:        ORR     r0, r0, #0x13  ; SET SVC MODE
50:        MSR     cpsr, r0
51:
52:        ;*-------------------------------------------------------
53:        ;* INITIALIZE THE USER MODE STACK
54:        ;*-------------------------------------------------------
55:        LDR     sp, c_stack
56:        LDR     r0, c_STACK_SIZE
57:        ADD     sp, sp, r0
```

修正したboot.asmからの抜粋（行番号はこの資料のために追加）

修正したboot.asmをプロジェクトに追加してビルドするとRTSライブラリ内のboot.asmとシンボルが競合してしまいリンカエラーが起こります。そこでリンカのオプションに“-priority”を追加します。このオプションはリンカが先に読み込んだオブジェクトファイルおよびライブラリファイルのシンボルを優先して使うためのオプションで、デフォルトでは（プロジェクトに追加されているソースからコンパイル/アセンブルされた）オブジェクトファイルが先に読み込まれます。（もしリンカがファイルを読み込む順番をデフォルトと異なる順にする際にはビルド・オプションの“Link Order”タブで変更可能です。）

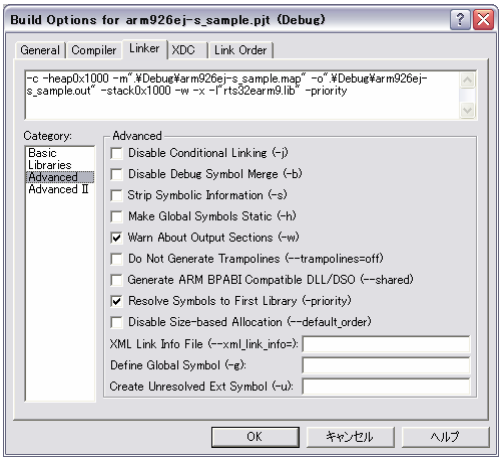

図 10　ビルド・オプションダイアログ（-priorityリンカオプションを追加した様子）

付属サンプルでは割込み処理のネスティング可能な割込みハンドラを実装しましたのでFIQ/IRQモードのスタック領域はアクセスせず、割込みサービスルーチンはSupervisorモードで処理するようにしています。そのためFIQ/IRQモードのR13は初期化していませんが、割込みレイテンシを極力少なくするためにネスティングはせず、割込み処理をFIQ/IRQモードで実行する場合は、それぞれのモードのR13を初期化し、その後Supervisorモードに移行するルーチンにしてもいいでしょう。

2.3　AINTC の初期化

OMAP-L137で割込みをARM926EJ-Sプロセッサに伝えるための初期化処理をまとめると以下のようになります。

① 周辺機能から割込みが出力されるように設定
② 割込み処理のネスティングをどの方法で行うか（またはネスティングを行わないか）を設定（
③ 周辺機能の割込みをAINTCのチャネルにマッピングする設定
④ AINTCでの周辺機能割込み（システム・インタラプト）許可
⑤ AINTCでのFIQ/IRQ割込み（ホスト・インタラプト）許可
⑥ AINTCでのグローバルな割込み許可
⑦ ARM926EJ-S CPSRでのFIQ/IRQ割込み許可

②から⑥はAINTCのレジスタで設定します。各処理で使用するレジスタは以下の通りです。

② Control Register(CR)のNESTMODEフィールド
③ Channel Map Register(CMR) 0～22のうちシステム・インタラプト・インデックスに該当するフィールド
④ System Interrupt Enable Indexed Set Register(EISR)のINDEXフィールド
⑤ Host Interrupt Enable Indexed Set Register(HIEISR)のINDEXフィールド
⑥ Global Enable Register(GER)のENABLEビット

CRのNESTMODEフィールドには、全てのチャネル番号を基準にネスティングする場合には“2”、FIQ/IRQごとにチャネル番号を基準にしてネスティングする場合には“1”、ソフトウェアで優先順位を管理してネスティングを行う場合には“3”、ネスティングを行わない場合には“0”を設定します。

CMRのフィールドはシステム・インタラプト・インデックスの数(0～90)分用意されていますので、該当するフィールドにチャネル番号(0～31)を書き込みます。CRのNESTMODEフィールドを“1”または“2”に設定した場合にはこのチャネル番号の設定が優先順位の設定にもなりますので注意してください。

EISRのINDEXフィールドには許可したいシステム・インタラプト・インデックスを、HIEISRのINDEXフィールドには許可したいホスト・インタラプト・インデックスを書き込みます。最後にGERのENABLEビットに“1”をセットします。

付属サンプルでは以上の処理をaintc.cの関数aintc_init()で行っています。

①の処理は周辺機能によって内容はさまざまですのでこの資料では言及しません。付属サンプルではTimer0とUART2の割込みを使うために、それぞれの初期化関数（timer.cの関数tmr0_init()、uart.cの関数uart_init()）で行っています。

⑦の処理はTMS470 CGTのCコンパイラの組み込み関数_enable_FIQ()および_enable_IRQ()で行うことができます。付属サンプルでは一連の初期化処理が終わった後、関数_enable_IRQ()をコールしています。

2.4　ベクタテーブルと IRQ ハンドラの作成

概要でご紹介したとおり、OMAP-L137 ARM926EJ-Sプロセッサのベクタテーブルは0xFFFF0000から始まり、４バイト（32ビット）ごとに各例外のベクタアドレスになっていますので、それぞれの例外ハンドラへの分岐命令を配置します。命令の

配置を指定することになりますので、ベクタテーブルはアセンブリで記述します。以下の例ではIRQハンドラのラベルを“_irq_handler”とし、ロード命令（ldr）を使って分岐しています。

```
        .global  c int00
        .global _irq_handler
        .sect ".intvecs"

        ldr pc, c_int00     ; reset
        .word 0xEAFFFFFE    ; undefined instruction
        .word 0xEAFFFFFE    ; software interrupt
        .word 0xEAFFFFFE    ; abort (prefetch)
        .word 0xEAFFFFFE    ; abort (data)
        .word 0xEAFFFFFE    ; reserved
        ldr pc, irq_handler ; IRQ
        .word 0xEAFFFFFE    ; FIQ

c_int00 .long _c_int00
irq_handler .long _irq_handler
```

付属サンプル“vector.asm”からの抜粋

IRQハンドラによりどの周辺機能からの割込み要求なのかを判断し、該当する割込みサービスルーチンを呼び出します。以下の例ではFIQ/IRQごとにチャネル番号を基準にしてネスティングする場合のIRQハンドラを紹介します。

ARM v5アーキテクチャではIRQベクタへ処理が移ったとき、ハードウェアで自動的にIRQが禁止になり、かつIRQモードに移行します。割込みハンドラでは割り込まれた処理に戻れるようにレジスタ値のスタックへの保存を行いますが、そのままR13を基準にレジスタ値をストアすると、IRQモードのスタック領域へアクセスしてしまいます。ネスティングを考慮し割込みサービスルーチンはSupervisorモードで処理しますので、IRQハンドラでのレジスタ値の保存はSupervisorモードで行います。25行目から35行目までがその処理です。

37，38行目でHost Interrupt Prioritized Index Register 2(HIPIR2)をリードし、割込みを発生した周辺機能のインデックスを取得します。

40行目から43行目ではHost Interrupt Nesting Level Register 2(HINLR2)へアクセスし、発生した割込みよりも優先順位の高い割込みだけが許可されるように設定しています。41行目のリードで割込み処理前の優先レベル（チャネル番号）を取得し、43行目でスタックへ保存しています。この値は割込みサービスルーチンから戻ってきた後、元の割込み優先レベルへ戻すために使います。42行目のライトで発生した割込みの優先レベル（チャネル番号）をAINTC内部回路にラッチさせ、以降その優先レベル以下の割込みは禁止されます。OVERRIDEビットを立てずに（“0”で）このライトアクセスを行っていることに注意してください。

HIPIR2には確実にインデックスのリードが行えるようにソフトウェアでの応答があるまで値を保持するモードがあります（このモードはCRのPRHOLDMODEビットにより選択します）。45行目のHIPIR2へのライトはその応答処理です。

46行目でSystem Interrupt Status Indexed Clear Register(SICR)へインデックス値を書き込み、割込みをクリアします。

47行目から49行目で割込みサービスルーチンのポインタ（開始アドレス）を取得しています。付属サンプルではisr.cにてグローバルなポインタの配列を用意しています。

51行目から56行目は何回ネスティングしているか、どの周辺機能からの割込みがネスティングしたのかを調べたデバッグルーチンですので削除してもいいでしょう。

60行目から64行目でIRQを再許可し、割込みサービスルーチンへ分岐しています。

67行目以降、割込みサービスルーチン処理後の後処理を行います。後処理はIRQ禁止の状態で行います。

69行目から72行目はデバッグルーチンですので削除してもいいでしょう。

74行目から77行目では優先レベルの再設定を行うためにスタックに保存していた値を取り出し、HINLR2へ書き込んでいます。この書き込みではOVERRIDEビットを立てて（“1”で）ライトアクセスを行っていることに注意してください。

79行目から81行目では割り込まれたルーチンのレジスタ値を復帰させ、IRQハンドラの処理を終えています。

```
SVC_MODE .equ 0x13
IRQ_MODE .equ 0x12
IRQ_BIT .equ 0x80

OFFSET_AINTC_SICR   .equ 0x24
OFFSET_AINTC_HIPIR2 .equ 0x904
AINTC_HIPIR2_MASK .equ 0x000000FF; actualy 0~90. thus, only 7bit mask is enough.

LOG_SIZE .equ 50

        .global _nesting_log_cnt
        .global _nesting_log_buffer
        .global _irq_nesting_log_init
        .global _nesting_cnt
        .global _isr_ptr_tbl
        .global _irq_handler

        .bss _nesting_cnt, 4, 4
        .bss _nesting_log_cnt, 4, 4
        .bss _nesting_log_buffer, LOG_SIZE*4, 4

        .text
_irq_handler:
        ;; under IRQ mode
        MSR     cpsr_all, #(SVC_MODE | IRQ_BIT)
        ;; under SVC mode
        STMFD   sp!, {r0-r3, r12, lr, pc}; pc is dummy at this point
        MSR     cpsr_all, #(IRQ_MODE | IRQ_BIT)
        ;; under IRQ mode
        SUB     r0, lr, #4
        MRS     r1, spsr
        MSR     cpsr_all, #(SVC_MODE | IRQ_BIT)
        ;; under SVC mode
        STR     r0, [sp, #0x18] ; set corrected pc in stuck
        STMFD   sp!, {r1}       ; push spsr

        LDR     r0, AINTC_BASE_ADDR
        LDR     r1, [r0, #OFFSET_AINTC_HIPIR2]

        LDR     r2, AINTC_HINLR2_ADDR
        LDR     r3, [r2]
        STR     r3, [r2]
        STMFD   sp!, {r3}       ; push nesting level

        STR     r1, [r0, #OFFSET_AINTC_HIPIR2]
        STR     r1, [r0, #OFFSET_AINTC_SICR]
        MOV     r1, r1, lsl #2
        LDR     r0, isr_ptr_tbl
        LDR     r0, [r0, r1]

        LDR     r2, nesting_cnt
        LDR     r3, [r2]
        ADD     r12, r3, #1
        STR     r12, [r2]
        CMP     r3, #0
        BNE     nesting_log

nesting_log_rtn:
        ;; enable IRQ
        MSR     cpsr_all, #(SVC_MODE)

        ;; call each ISR
        MOV     lr, pc
        MOV     pc, r0

```

```
 66:        ;; disable IRQ
 67:        MSR     cpsr_all, #(SVC_MODE | IRQ_BIT)
 68:
 69:        LDR     r2, nesting_cnt
 70:        LDR     r3, [r2]
 71:        SUB     r3, r3, #1
 72:        STR     r3, [r2]
 73:
 74:        LDMFD   sp!, {r2}       ; pop nesting level
 75:        LDR     r0, AINTC_HINLR2_ADDR
 76:        ORR     r2, r2, #0x80000000; AINTC_HINLR2_OVERRIDE
 77:        STR     r2, [r0]
 78:
 79:        LDMFD   sp!, {r1}       ; pop spsr
 80:        MSR     spsr_all, r1
 81:        LDMFD   sp!, {r0-r3, r12, lr, pc}^
 82:
 83:nesting_log:
 84:        LDR     r2, nesting_log_buffer
 85:        LDR     r3, nesting_log_cnt
 86:        LDR     r12, [r3]
 87:        CMP     r12, #LOG_SIZE
 88:        MOVNE   lr, r12, lsl #2
 89:        ADDNE   r12, r12, #1
 90:        MOVEQ   lr, #0
 91:        MOVEQ   r12, #0
 92:        STR     r1, [r2, lr]
 93:        STR     r12, [r3]
 94:        B       nesting_log_rtn
 95:
 96:isr_ptr_tbl     .long _isr_ptr_tbl
 97:AINTC_BASE_ADDR .long 0xFFFEE000
 98:AINTC_HINLR2_ADDR .long 0xFFFEF104
 99:nesting_cnt     .long _nesting_cnt
100:nesting_log_cnt .long _nesting_log_cnt
101:nesting_log_buffer .long _nesting_log_buffer
```

付属サンプルirq_handler.asmからの抜粋（行番号はこの資料のために追加）

レジスタ値のスタックへの保存や復帰はARM926EJ-S内のレジスタを直接アクセスするためC言語では記述できない部分です。そこで上記IRQハンドラではアセンブリで実装し、あわせてAINTCのネスティング機能のコントロールも行っています。TMS470 CGTのCコンパイラは関数にレジスタ値のスタックへの保存、復帰ルーチンを埋め込むためのプラグマ（#pragma INTERRUPT(func, interrupt_type)）が用意されていますので、ネスティングを行わない場合はこのプラグマを活用してもいいでしょう（上記IRQハンドラから呼び出される割込みサービスルーチンの関数にはINTERRUPTプラグマを付ける必要はありません）。

3 キャッシュの設定

3.1 ARM926EJ-S プロセッサの MMU 及びキャッシュ概要

OMAP-L137 ARM926EJ-Sプロセッサには16Kbyteのインストラクション・キャッシュと16Kbyteのデータ・キャッシュが搭載されています。またLinuxなどの仮想メモリ空間を必要とするOSをサポートするためにMMU(Memory Management Unit)が搭載されています。ARM926EJ-Sプロセッサは MMUの機能を停止させた状態で、データ・キャッシュを有効にすることができない仕様になっているため、キャッシュ有効エリアの設定はMMUの動作と関連しています。この資料ではシンプルな組込みプログラミング環境構築のため、仮想メモリ空間と物理メモリ空間が一致するような設定方法について紹介します。

MMUはプログラム実行に使用するアドレス（仮想アドレスVirtual Address (VA)）と物理メモリへのアクセスに使うアドレス（物理アドレスPhysical Address (PA)）の変換を、テーブルに従い自動的に行います。またこの変換テーブルはキャッシュ有効エリアかどうかの情報も含み、キャッシュコントローラは変換テーブルの内容に従いキャッシングします。ARM926EJ-Sプロセッサでは４種類の変換テーブル（“Section”、“Large page”、“Small page”、“Tiny page”）に対応していますが、付属サンプルではVA＝PAというシンプルなテーブルを作成するために“Section”を使用しています。このテーブルをメモリに配置しておくとMMUはTLB(Translation Lookaside Buffer)を介して読み込み、テーブルの内容にしたがってアドレスの変換とキャッシュのコントロールを行います。

またARM926EJ-Sプロセッサはより高速なコンテキストスイッチを実現するためにFast Context Switch Extension(FCSE)を搭載しています。FCSEは32Mbyteごとのプロセスを切り替えるためにVAからModified Virtual Address(MVA)を出力し、

MMUへ渡します。つまり正確には、MMUはMVAをPAに変換することになりますが、付属サンプルではFCSEも積極的には使用せず、VA=MVA=PAとなるシステムをMMU及びキャッシュを有効にした状態で作成しています。

MMUやキャッシュ、FCSEは“CP15”と呼ばれるコプロセッサを使いコントロールします。ARM v5アーキテクチャではコプロセッサへのアクセスに特別な命令を使い、行いますので、CP15のコントロールルーチンはアセンブリで記述します。

MMU及びキャッシュ、FCSEの詳細はARM社のドキュメント・ホームページhttp://infocenter.arm.com/help/index.jspより *ARM926EJ-S Technical Reference Manual* をご参照ください。

3.2 システム制御コプロセッサ(CP15)を設定するサブルーチンの作成

CP15には0から15まで、16個レジスタがあり、それぞれのレジスタへのリードアクセス、ライトアクセス毎に機能や設定される内容が決まっています。リードするにはMRC命令を、ライトするにはMCR命令を使います。例えばレジスタ２へアドレス値を書き込みことにより、変換テーブルのベースアドレスを設定することができ、そのときのアセンブリは

MCR　p15, #0, r0, c2, c0, #0（あらかじめr0にベースアドレス値をセットしておく）

となります。

付属サンプルのcp15_subroutine.asmでは必要なCP15レジスタの設定ルーチン呼び出しをC言語から行えるようにサブルーチン化しています。引数や戻り値の渡し方の例としてもご参照ください。ファンクションコールに関するルールの詳細は *TMS470R1x Optimizing C/C++ Compiler v4.4 User's Guide* [SPNU151D] : http://www.ti.com/jp/litv/pdf/spnu151d

をご参照ください。

3.3 変換テーブルの作成と MMU、キャッシュの初期化

“Section”タイプの変換テーブルでは1Mbyteのメモリ空間ごとにセクション・ディスクリプタと呼ばれる３２ビットのデータを用意し、メモリに連続して配置しておきます。つまりARM926EJ-Sプロセッサの持つ全メモリ空間（0x0～0xFFFFFFFF）のためにディスクリプタを4,096個、用意することになります。付属サンプルではmmu.c内、関数mmu_init()にてC言語で各ディスクリプタの内容を設定していますが、ARM926EJ-Sの仕様では、変換テーブルが16Kbyte境界でアラインメントされている必要がありますので、グローバルな領域をfirst_level_descripter.asmにてアセンブリ擬似命令を使い確保しています。また付属サンプルではEMIFB SDRAMのメモリ空間をキャッシュ有効エリアとして設定しています。ボードの構成やシステムの構成によってはEMIFAのメモリ空間やShared RAMのメモリ空間をキャッシュ有効エリアにし、パフォーマンス向上を検討してもいいでしょう。

関数mmu_init()では変換テーブルの値を用意した後、CP15へアクセスし、MMU及びキャッシュを有効にしています。

4 ARM926EJ-S プロセッサと C674x DSP コアのコミュニケーション

4.1 メモリ空間と CHIPSIG 割込み概要

OMAP-L137はARM926EJ-Sプロセッサに加えて、信号処理を得意とするC674x DSPコアも搭載しており、２つのプロセッサはそれぞれ、ほぼ全てのメモリ空間をアクセスすることが可能です。そのため周辺機能やRAMへのアクセスが競合しないように、ソフトウェアで管理することになります。付属サンプルではEMIFB SDRAMのメモリ空間のうち0xC0000000～0xC0FFFFFFまでをARM926EJ-Sプロセッサのプログラム、0xC3000000～0xC3FFFFFFまでをC674x DSPコアのプログラムで使用するようにリンカにてメモリ配置しています。また２つのプロセッサ間でのデータの受け渡しは、それぞれのプログラムが使用していない0xC1000000から始まるメモリ空間を共有空間とし、その共有空間を読み書きすることで行っています。なお周辺機能へのアクセスはDSP/BIOSが使用しているTimer1と下記CHIPSIG割込みを除き、ARM926EJ-S側プログラムで行っていますが、C674x DSP側プログラムが入出力に使用する周辺機能はC674x DSP側プログラムでコントロールしてもよいでしょう。

OMAP-L137では２つのプロセッサ間で同期を取るためにCHIPSIG割込みが用意されており、双方向に割込みを起こすことが可能です。CHIPSIG割込みはCHIPSIGレジスタの各ビットへ“１”をセットすることにより発生し、CHIPSIG_CLRレジスタへ“１”をセットすることでクリアされます。CHIPSIG/CHIPSIG_CLRレジスタにはCHIPSIG0からCHIPSIG4までビットが用意されていますが、通常、ARM926EJ-Sプロセッサへ割込みを発生させるにはCHIPSIG0、1を、C674x DSPコアに割込みを発生させるにはCHIPSIG2、3を使用します。付属サンプルでは以下のような順序でお互いに割込みを受け取り、データの受け渡しを行っています。

① ARM926EJ-S側プログラムで共有空間にデータを用意
② ARM926EJ-S側プログラムでCHIPSIG2割込みを起こし、C674x DSPコアへデータが用意できたことを伝える。そしてCHIPSIG0割込みを待つ。
③ C674x DSPコア側プログラムの割込み処理で共有空間のデータを読み出し、定数“0xCAFE0000”を加算して書き戻す。

④　C674x DSPコア側プログラムでCHIPSIG0割込みを起こし、ARM926EJ-Sプロセッサへデータが用意できたことを伝える。

⑤　ARM926EJ-S側プログラムはCHIPSIG0割込み待ちを解除し、期待の値が共有空間に書き込まれているかどうか確認。

なおCHIPSIG割込みをC674x DSPコアで処理する際にはEvent Combiner経由ではなく、C674x DSPコアの割込み入力へ直接マッピングするようにしてください。これはハードウェア上の制約です。DSP/BIOSを使用する場合は、付属サンプルのようにHWIへマッピングして処理を行います。

OMAP-L137のメモリ空間及びCHIPSIG割込みの詳細については以下の資料をご参照ください。

OMAP-L137 Datasheet [SPRS563B]：http://www.ti.com/jp/lit/gpn/omap-l137

OMAP-L137 Applications Processor System Reference Guide [SPRUG84B]：http://www.ti.com/jp/litv/pdf/sprug84b

4.2　キャッシュ・コヒーレンシの保持

２つのプロセッサにはそれぞれ専用のキャッシュがあり、それぞれのキャッシュは各プロセッサからのアクセスをもとにキャッシングを行っていますので、共有空間がキャッシュ有効エリアの場合、共有空間のデータとキャッシュメモリのデータのコヒーレンシ（一貫性）を保つようにソフトウェアで処理しなければ、誤ったデータを受け渡してしまう可能性があります。例えばC674x DSPコアが共有空間に書き込んだデータをARM926EJ-Sプロセッサが受け取る（読み出す）ことを考えると、C674x DSPコアのキャッシュはリードアロケート・ライトバックキャッシュですので、書き込みが終わった後、キャッシュメモリの内容を共有空間へライトバック（クリーン）する必要があります。またARM926EJ-Sプロセッサのキャッシュはリードアロケート・ライトスルーキャッシュ（注1）ですので、ARM926EJ-Sプロセッサが読み出すときには、共有空間のデータをキャッシングするように、キャッシュメモリの古いデータの無効化を行った後、読み出す必要があります。

付属サンプルではARM926EJ-Sプロセッサのキャッシュ無効化はCP15のレジスタ７を操作することによって行い、C674x DSPコアのライトバック及び無効化はDSP/BIOSのAPIを使用しています。両プロセッサのキャッシュともに初期化後の段階では、キャッシュメモリは空（無効化済み）ですので、最初のアクセスは普通に行い、共有空間へのアクセス後、次のアクセスを見越して無効化するような手順になっています。

注1：OMAP-L137のエラッタにより、ARM926EJ-Sプロセッサのキャッシュをライトスルーで使用しています。詳細は
OMAP-L137 Silicon Errata [SPRZ291A]：http://www.ti.com/jp/litv/pdf/sprz291a
をご参照ください。

5　サンプル・プログラムについて

JAJA187.zipを展開し、ご使用ください。このプログラムはOMAP-L137 OSKボード上で動作するプログラムです。OMAP-L137 OSKの使用法についてはキット付属のドキュメントをご参照下さい。なおサンプル・プログラムはOMAP-L137 OSK付属GELファイルを使用した環境を想定しています。また、ボード上のブートスイッチ（SW2）は“Emulation Debug”の設定にし、お使いください。ARM926EJ-S向けプロジェクトはJAJA187\arm926ej-s_sample\arm926ej-s_sample.pjt、C674x DSPコア向けプロジェクトはJAJA187\c674x_sample\c674x_sample.pjtになります。それぞれCCSにてオープンし、ご使用ください。

周辺機能のサンプルとしてTimer0の割込み処理によるLED点滅ルーチンとUART2の送受信ルーチン（送信は簡単のためポーリング処理）を実装しています。RS232-Cケーブルでホスト ＰＣと接続し、115200bsp、8ビット、パリティ無し、ストップ1ビット、フローコントロール無しに設定したターミナルソフトを起動した後、2つのプロセッサのプログラムを実行してください。実行するとLEDの点滅が始まり、ターミナルに“=== OMAP-L137 ARM…”と表示されます。その後ターミナル上で“a”をキーボード入力すると2つのプロセッサ間でのデータ受け渡しルーチンが動作し、データの受け渡しが完了すると再度“a”の入力待ちになります。

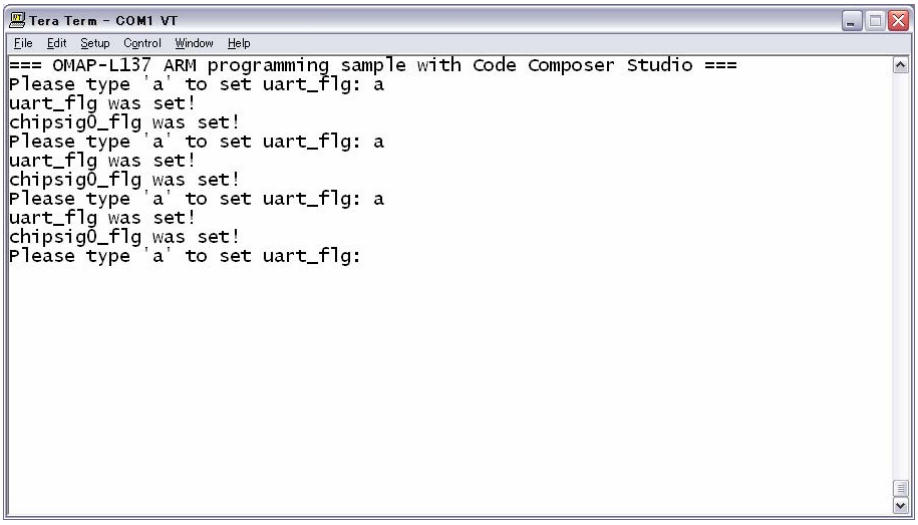

図 11　サンプル・プログラム実行時のターミナルの様子

JAJA187\ti\pspiom\cslrフォルダにはOMAP-L137/C6747 SDK 1.00.00.10より抜き出したCSLRをもとに、ARM926EJ-Sにて使用できるよう修正したCSLRがあります。修正したファイルはtistdtypes.h及びcslr_aintc.hの２つです。また修正箇所は“__TI_TMS470_V5__”がdefineされているときのみ有効になります。__TI_TMS470_V5__はビルド・オプションに-mv5eがあると、コンパイラが自動的にdefineします。

ご注意

日本テキサス・インスツルメンツ株式会社（以下TIJといいます）及びTexas Instruments Incorporated（TIJの親会社、以下TIJないしTexas Instruments Incorporatedを総称してTIといいます）は、その製品及びサービスを任意に修正し、改善、改良、その他の変更をし、もしくは製品の製造中止またはサービスの提供を中止する権利を留保します。従いまして、お客様は、発注される前に、関連する最新の情報を取得して頂き、その情報が現在有効かつ完全なものであるかどうかご確認下さい。全ての製品は、お客様とTIJとの間に取引契約が締結されている場合は、当該契約条件に基づき、また当該取引契約が締結されていない場合は、ご注文の受諾の際に提示されるTIJの標準販売契約約款に従って販売されます。

TIは、そのハードウェア製品が、TIの標準保証条件に従い販売時の仕様に対応した性能を有していること、またはお客様とTIJとの間で合意された保証条件に従い合意された仕様に対応した性能を有していることを保証します。検査およびその他の品質管理技法は、TIが当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、政府がそれ等の実行を義務づけている場合を除き、必ずしも行なわれておりません。

TIは、製品のアプリケーションに関する支援もしくはお客様の製品の設計について責任を負うことはありません。TI製部品を使用しているお客様の製品及びそのアプリケーションについての責任はお客様にあります。TI製部品を使用したお客様の製品及びアプリケーションについて想定されうる危険を最小のものとするため、適切な設計上および操作上の安全対策は、必ずお客様にてお取り下さい。

TIは、TIの製品もしくはサービスが使用されている組み合せ、機械装置、もしくは方法に関連しているTIの特許権、著作権、回路配置利用権、その他のTIの知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表明もしておりません。TIが第三者の製品もしくはサービスについて情報を提供することは、TIが当該製品もしくはサービスを使用することについてライセンスを与えるとか、保証もしくは是認するということを意味しません。そのような情報を使用するには第三者の特許その他の知的財産権に基づき当該第三者からライセンスを得なければならない場合もあり、またTIの特許その他の知的財産権に基づきTI からライセンスを得て頂かなければならない場合もあります。

TIのデータ・ブックもしくはデータ・シートの中にある情報を複製することは、その情報に一切の変更を加えること無く、かつその情報と結び付られた全ての保証、条件、制限及び通知と共に複製がなされる限りにおいて許されるものとします。当該情報に変更を加えて複製することは不公正で誤認を生じさせる行為です。TIは、そのような変更された情報や複製については何の義務も責任も負いません。

TIの製品もしくはサービスについてTIにより示された数値、特性、条件その他のパラメーターと異なる、あるいは、それを超えてなされた説明で当該TI製品もしくはサービスを再販売することは、当該TI製品もしくはサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、かつ不公正で誤認を生じさせる行為です。TIは、そのような説明については何の義務も責任もありません。

TIは、TIの製品が、安全でないことが致命的となる用途ないしアプリケーション（例えば、生命維持装置のように、TI製品に不良があった場合に、その不良により相当な確率で死傷等の重篤な事故が発生するようなもの）に使用されることを認めておりません。但し、お客様とTIの双方の権限有る役員が書面でそのような使用について明確に合意した場合は除きます。たとえTIがアプリケーションに関連した情報やサポートを提供したとしても、お客様は、そのようなアプリケーションの安全面及び規制面から見た諸問題を解決するために必要とされる専門的知識及び技術を持ち、かつ、お客様の製品について、またTI製品をそのような安全でないことが致命的となる用途に使用することについて、お客様が全ての法的責任、規制を遵守する責任、及び安全に関する要求事項を満足させる責任を負っていることを認め、かつそのことに同意します。さらに、もし万一、TIの製品がそのような安全でないことが致命的となる用途に使用されたことによって損害が発生し、TIないしその代表者がその損害を賠償した場合は、お客様がTIないしその代表者にその全額の補償をするものとします。

TI製品は、軍事的用途もしくは宇宙航空アプリケーションないし軍事的環境、航空宇宙環境にて使用されるようには設計もされていませんし、使用されることを意図されておりません。但し、当該TI製品が、軍需対応グレード品、若しくは「強化プラスティック」製品としてTIが特別に指定した製品である場合は除きます。TIが軍需対応グレード品として指定した製品のみが軍需品の仕様書に合致いたします。お客様は、TIが軍需対応グレード品として指定していない製品を、軍事的用途もしくは軍事的環境下で使用することは、もっぱらお客様の危険負担においてなされるということ、及び、お客様がもっぱら責任をもって、そのような使用に関して必要とされる全ての法的要求事項及び規制上の要求事項を満足させなければならないことを認め、かつ同意します。

TI製品は、自動車用アプリケーションないし自動車の環境において使用されるようには設計されていませんし、また使用されることを意図されておりません。但し、TIがISO/TS 16949の要求事項を満たしていると特別に指定したTI製品は除きます。お客様は、お客様が当該TI指定品以外のTI製品を自動車用アプリケーションに使用しても、TIは当該要求事項を満たしていなかったことについて、いかなる責任も負わないことを認め、かつ同意します。

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管・輸送環境、基板実装条件によっては、お客様での実装前後に破壊/劣化、または故障を起こすことがあります。

弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1. 静電気
   - 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
   - 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
   - マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
   - 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。
2. 温・湿度環境
   - 温度：0～40℃、相対湿度：40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
   - 直射日光があたる状態で保管・輸送しないこと。
3. 防湿梱包
   - 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。
4. 機械的衝撃
   - 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。
5. 熱衝撃
   - はんだ付け時は、最低限260℃以上の高温状態に、10秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）
6. 汚染
   - はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
   - はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上