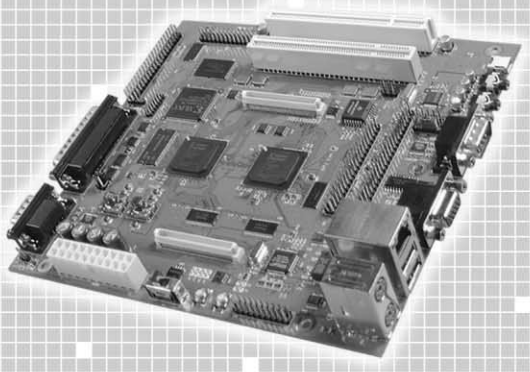
組み込みシステム開発評価キット活用通信

大牧 正知/山武 一朗



第13回 MicroBlaze新メモリ・マップ対応割り込みハンドラの作成とブレーク信号の追加

1. 新メモリ・マップ対応版サンプル・プログラムの作成

● 新メモリ・マップ版での割り込みコントローラの構造

図1に新メモリ・マップ版での割り込みコントローラの構造を示します．

BLANCAシステム・バス上での割り込みは，A/Vプロセッサの BLANCAシステム・バス上に接続するBLANCA割り込みコントローラに接続されます．I/Oプロセッサ内のコントローラの割り込み出力は，FPGAの外部I/O信号を伝わってA/Vプロセッサ内に実装したBLANCA割り込みコントローラに接続されます．この部分は旧メモリ・マップ版も同じです．

このBLANCA割り込みコントローラで1本化された割り込み信号を，旧メモリ・マップ版ではMicroBlazeの割り込み入力に直接接続していました．

新メモリ・マップ版では，EDK標準添付のMicroBlaze標準の割り込みコントローラOPB_INTCが追加され，BLANCAシステム・バス上の割り込みはこのOPB_INTCに接続することになりました．つまり割り込みコントローラが2段構成になった形となります．

OPB_INTCを追加したので，EDK標準添付のUARTやタイマの割り込みも割り込み駆動が可能になるよう，割り込み信号を接続しています．

● 割り込み使用サンプル・プログラム

新メモリ・マップの上で割り込みを使ったサンプル・プログラムを動かしてみましょう．ここでは割り込みを使うサンプル・プログラムとして最も簡単なBLANCAタイマ・コントローラ用のサンプル・プログラムを示します．

リスト1にOPB_INTCでBLANCAシステム・バスからの割り込みをハンドリングするためのCPU固有割り込み初期化および割り込み処理関数を，**リスト2**にタイマ割り込みサンプル・プログラム（Timer.C）を示します．

リスト1のCPU_IntInitialize()関数でOPB_INTCのBLANCAシステム・バスからの割り込みを受け付けるよう初期化します．EDKのバージョン9以前は，MicroBlazeの割り込み処理関数はEDKのメニュー上から関数名を入力することで設定できましたが，EDK9.1iからはこの方式がなくなりました．それに代わり，**リスト1**のCPU_IntInitialize()関数内に記述があるように，割り込みハンドラの登録関数（XIntc_Connec）などを使って，割り込み処理関数を登録する方式となりました．

そしてBLANCAシステム・バスで割り込みが発生すると，

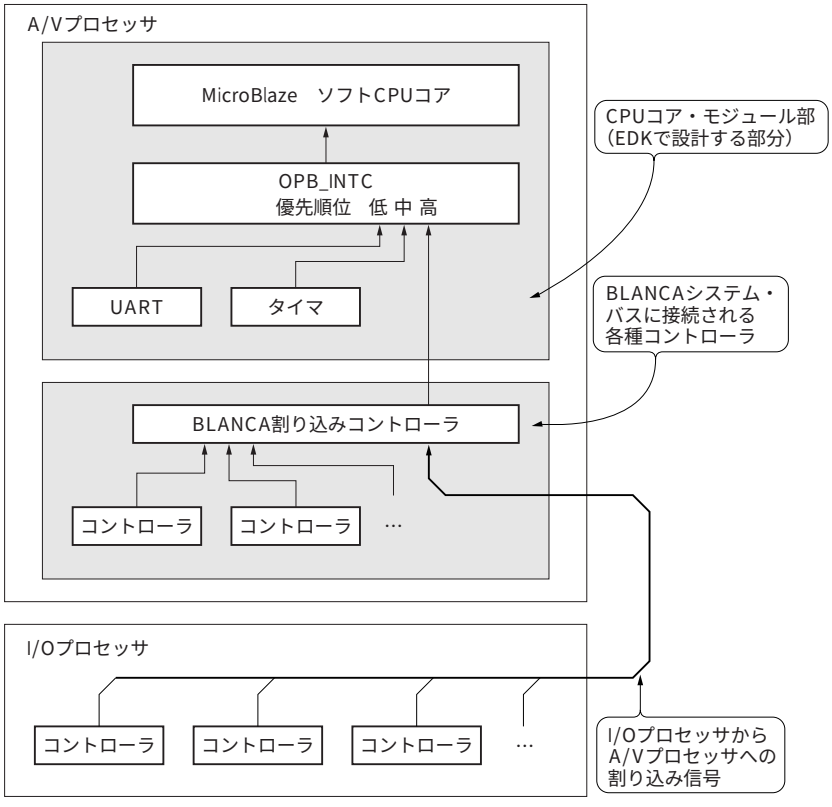


図1 MicroBlaze新メモリ・マップ版での割り込みコントローラの構造

KEYWORD ── MicroBlaze，EDK，BLANCAシステム・バス，BLANCA割り込みコントローラ，割り込みハンドラ，ブレーク信号

リスト2のBLANCA_Interrupt()関数が呼び出されます．

リスト2にはCPUに依存する部分はないので，M32R用などほかのCPU用のタイマ割り込みサンプル・プログラムも全く同じソースとなっています．

● GCCを使う場合の割り込みハンドラ

本誌2007年12月号の特集記事および付属DVD-ROMには，MicroBlaze用のGCCも収録されています．MicroBlazeのハードウェアはサンプル設計プロジェクトのものをそのまま使い，

リスト1　EDK向けCPU固有処理ルーチン（CPU_INIT.C）

```
/*** CPU各種初期化 ***/

/* MicroBlaze用各種ヘッダファイル */
#include "xintc.h"
#include "xparameters.h"
#include "mb_interface.h"
/* CPU外部バス初期化 */
void CPU_BusInitialize(void)
{
    /* 初期化の必要なし */
}

/* CPU割り込み初期化 */
void CPU_IntInitialize(void)
{
    /* 割り込みコントローラの初期化 */
    XIntc_Initialize(&Intc, XPAR_OPB_INTC_0_DEVICE_ID);
    /* 割り込みハンドラの登録 */
    XIntc_Connect(&Intc,
    XPAR_OPB_INTC_0_SYSTEM_MICROBLAZE_0_INTERRUPT_PIN_INTR,
         (XInterruptHandler)BLANCA_Interrupt, 0);
    /* 割り込みコントローラのグローバル・イネーブル */
    XIntc_Start(&Intc, XIN_REAL_MODE);
    /* 割り込みコントローラの外部割り込み入力のイネーブル */

    XIntc_Enable(&Intc,
    XPAR_OPB_INTC_0_SYSTEM_MICROBLAZE_0_INTERRUPT_PIN_INTR);
}
```

BLANCA_Interrupt関数を割り込み処理関数として登録

リスト2　BLANCAタイマ割り込みサンプル・プログラム（Timer.c）

```
/*********************************/
/*  タイマ・テスト・プログラム          */
/*********************************/

/* CPU固有機能定義ヘッダ */
#include "cpu.h"

/* CPU 初期化ルーチン・ヘッダ */
#include "cpu_init.h"

/* CPU UART入出力ヘッダ */
#include "cpu_uart.h"

/* BLANCAシステム用各種ヘッダ */
#include "blanca_bios.h"
/* 簡易ライブラリ・ヘッダ */
#include "cqlib.h"

/* タイマ・コントローラ・リソース情報構造体 */
struct BLANCA_RESOURCE TIME;
#define  TIME_Signature BLANCA_SIGN("TIME") /* 'TIME' */
#define  TIME_SysClkHz
((volatile unsigned int *)(TIME.BaseAdr+0x10))
#define  TIME_A0ctrl
((volatile unsigned int *)(TIME.BaseAdr+0x20))
#define  TIME_A0count
((volatile unsigned int *)(TIME.BaseAdr+0x24))
#define  TIME_A0compare
((volatile unsigned int *)(TIME.BaseAdr+0x28))
#define  TIME_A1ctrl
((volatile unsigned int *)(TIME.BaseAdr+0x30))
#define  TIME_A1count
((volatile unsigned int *)(TIME.BaseAdr+0x34))
#define  TIME_A1compare
((volatile unsigned int *)(TIME.BaseAdr+0x38))
#define  TIME_B0ctrl
((volatile unsigned int *)(TIME.BaseAdr+0x80))
#define  TIME_B0count
((volatile unsigned int *)(TIME.BaseAdr+0x84))
#define  TIME_B0compare
((volatile unsigned int *)(TIME.BaseAdr+0x88))
#define  TIME_B1ctrl
((volatile unsigned int *)(TIME.BaseAdr+0x90))
#define  TIME_B1count
((volatile unsigned int *)(TIME.BaseAdr+0x94))
#define  TIME_B1compare
((volatile unsigned int *)(TIME.BaseAdr+0x98))

volatile unsigned int  TimerA0_Int;   /* 割り込み発生フラグ */
volatile unsigned int  TimerA1_Int;   /* 割り込み発生フラグ */
volatile unsigned int  TimerB0_Int;   /* 割り込み発生フラグ */
volatile unsigned int  TimerB1_Int;   /* 割り込み発生フラグ */


/* 割り込み処理ルーチン */
void BLANCA_Interrupt(void)
{
    unsigned int stat;
    stat=*AVP_IntStatus;    /* AVP割り込み要因チェック */
    if (stat&(1<<TIME.ChipSelect)) {
         stat=*TIME_A0ctrl;       /* タイマA0ステータス取得 */
         if (stat&0x80000000) {   /* 割り込み発生 */
              TimerA0_Int = 1;    /* タイマA0割り込みフラグ・セット */
              *TIME_A0ctrl=stat;  /* タイマA0割り込みクリア */
         }
         stat=*TIME_A1ctrl;       /* タイマA1ステータス取得 */
         if (stat&0x80000000) {   /* 割り込み発生 */
              TimerA1_Int = 1;    /* タイマA1割り込みフラグ・セット */
              *TIME_A1ctrl=stat;  /* タイマA1割り込みクリア */
         }
         stat=*TIME_B0ctrl;       /* タイマB0ステータス取得 */
         if (stat&0x80000000) {   /* 割り込み発生 */
              TimerB0_Int = 1;    /* タイマB0割り込みフラグ・セット */
              *TIME_B0ctrl=stat;  /* タイマB0割り込みクリア */
         }
         stat=*TIME_B1ctrl;       /* タイマB1ステータス取得 */
         if (stat&0x80000000) {   /* 割り込み発生 */
              TimerB1_Int = 1;    /* タイマB1割り込みフラグ・セット */
              *TIME_B1ctrl=stat;  /* タイマB1割り込みクリア */
         }
    }
}

/* メイン・ルーチン */
int main(void)
{
    int i,h,m,s,Time_Count;
    char c;

    CPU_BusInitialize();    /* CPUバス周辺初期化 */
    CPU_UartInitialize();   /* コンソール用UART初期化 */

    i=BLANCA_BIOS_Initialize();  /* BLANCA BIOS初期化 */
    if (i<0) {
         CPU_UartPuts("\nBLANCA BIOS initialize error!\n");
         return -1;
    }
    CPU_UartPuts("\n\n");
    /* これ以降からprintf関数が使える */
    cq_printf("%s\nBLANCA BIOS initialize ...
                                          success!\n",TITLE_msg);
    if (RGBC.ChipSelect) cq_printf
                                  (" Analog RGB controller detect\n");
    if (LBUS.ChipSelect) cq_printf
                                    (" LocalBus controller detect\n");
    if (PS2H.ChipSelect) cq_printf
                          (" Intelligent PS/2 controller detect\n");

    /* タイマ・コントローラ・リソース取得 */
    BLANCA_BIOS_GetResource(TIME_Signature, &TIME, 1);
```

タイマ割り込み処理の内容