

Nios II マルチコア構成時の 共有関数のリンク方法

ver.14

Nios II マルチコア構成時の共有関数のリンク方法

目次

1. はじめに	3
2. 適用条件	3
3. 概要	4
4. セクション定義	5
5. 共有関数を使用する	6
改版履歴	8

1. はじめに

この資料は、複数の Nios[®] II を使用してマルチコア構成とした場合に、一つの関数を複数の CPU コア (Nios II) から参照する一つの手法を紹介します。プログラム・メモリの容量が足りず、関数を共有して使用したいときなどにご活用ください。

2. 適用条件

- 対応バージョン
 - Quartus[®] II 開発ソフトウェア v14.1
 - Nios II Software Build Tools (Nios II SBT) v14.1
- 検証ハードウェア
 - Cyclone V E FPGA Development Board
 - ・ FPGA : Cyclone V 5CEFA7F31I7ES

3. 概要

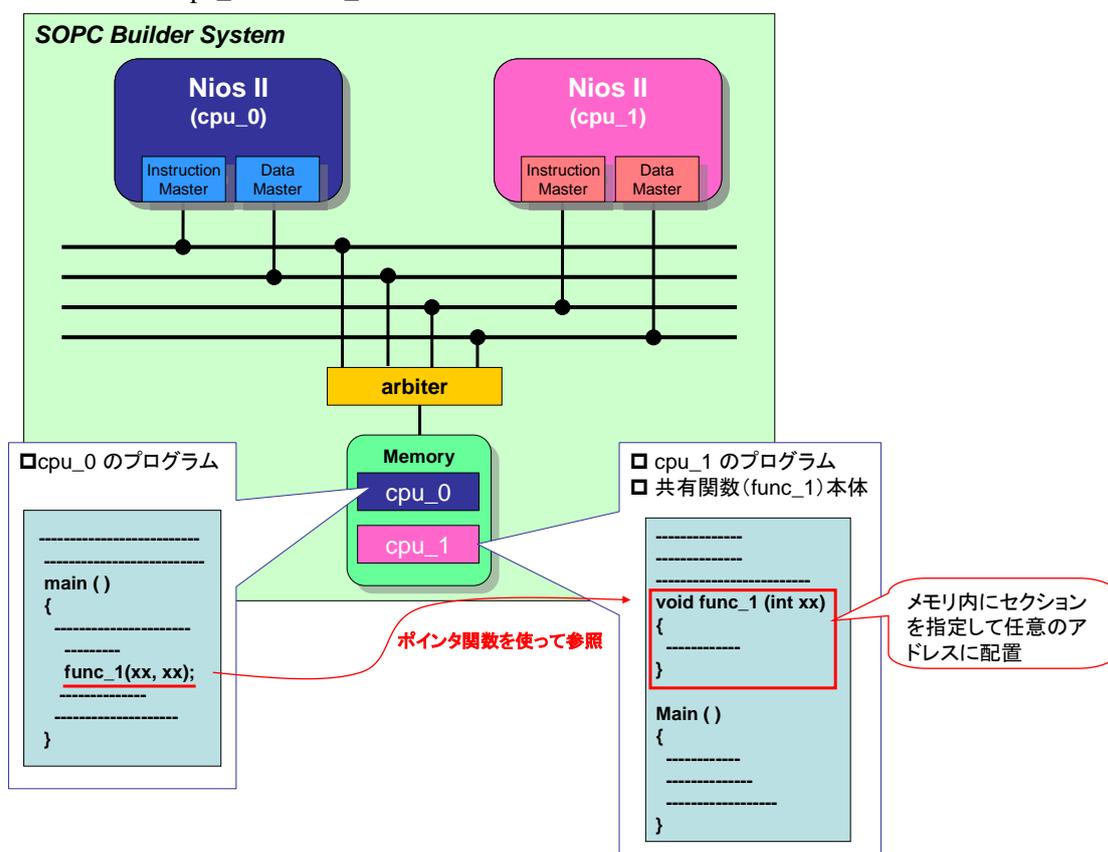
下記のような手法を用いて、複数の CPU コアから 他のコアが持つ関数へのアクセスを行います。

- 複数のコアが共有しているメモリにセクションを切り、共有したい関数(func_1)を配置します。
- 別の func_1 を持っていない関数から、関数ポインタで func_1 を呼び出して使用します。

例として下記のような構成を想定します。

システム内に Nios II が 2 つのマルチコアの構成とします。cpu_0、cpu_1 がプログラム・メモリとして一つのメモリを共有します。

cpu_1 のソース・コードにのみ共有関数(func_1)を記述します。cpu_0 のソース・コードの中には共有関数(func_1)はありませんので、cpu_1 の func_1 を参照して実行します。



4. セクション定義

cpu_1 のソフトウェア・プロジェクトのプログラム・メモリ内に任意のセクションを定義し、このセクションに共有関数を配置します。

Nios II SBT にてプロジェクトを作成した際にリンカ・スクリプトが自動生成されますが、このリンカ・スクリプトをユーザが編集し使用することが可能です。

- ① BSP プロジェクト(bsp のプロジェクト)から BSP Editor を起動します。
 <プロジェクト名>_bsp を右クリックから Nios II ⇒ BSP Editor フォルダに generated.x ファイルが生成されます。こちらがプロジェクトのリンカ・スクリプトになります。
- ② Linker Script タブを開きます。
- ③ Linker Memory Regions の “Add” ボタンをクリックして、新しい Region を追加します。以下の例では、commem_on_ssram という名前の Region を作成しています。Size を 4096 に Offset を 0x100000 に設定しています。この設定で、SSRAM の空き領域に 4096 バイトの共有メモリ領域が確保されました。サイズやオフセットをどのように設定するかは、共有関数のサイズと使用する RAM 領域の空き領域がどのくらいあるかで決定します。
- ④ 次に、Linker Section Mappings の “Add” ボタンをクリックしてに新しいセクション .sect_0 を生成します。Linker Region Name には、commem_on_ssram を選択します。

The screenshot shows the Nios II BSP Editor interface. The 'Linker Section Mappings' table is as follows:

Linker Section Name	Linker Region Name	Memory Device Name
.bss	ext_ssram	ext_ssram
.entry	reset	ext_flash
.exceptions	ext_ssram	ext_ssram
.heap	ext_ssram	ext_ssram
.rodata	ext_ssram	ext_ssram
.rwdata	ext_ssram	ext_ssram
.sect_0	commem_on_ssram	ext_ssram
.stack	ext_ssram	ext_ssram
.text	ext_ssram	ext_ssram

The 'Linker Memory Regions' table is as follows:

Linker Region Name	Address Range	Memory Device Name	Size (bytes)	Offset (bytes)
com_mem_1_2	0x09001000 - 0x09001FFF	com_mem_1_2	4096	0
com_mem_0_1	0x09000000 - 0x09000FFF	com_mem_0_1	4096	0
commem_on_ssram	0x04100000 - 0x04100FFF	ext_ssram	4096	0x100000
ext_ssram	0x04060020 - 0x040C001F	ext_ssram	393216	393248
ext_ssram_BEFORE_EXCEPTION	0x04000000 - 0x0400001F	ext_ssram	32	0
ext_flash	0x00460020 - 0x004BFFFF	ext_flash	393184	4587652
reset	0x00460000 - 0x0046001F	ext_flash	32	4587520
ext_flash BEFORE RESET	0x00000000 - 0x003FFFFF	ext_flash	4194304	0

The 'Information' pane shows the following log messages:

- ① Set setting property: altera_vic_driver.VIC_1.vec_size destination to system_h_define
- ① Set setting property: altera_vic_driver.VIC_1.vec_size description to The number of bytes in each vector table entry. Supported values are 16, 32, 64, 128, 256 and 512.
- ① Finished loading SOPC Builder system info file ".\..\nios2_system.sopcinfo [relative to settings file]"
- ① Added memory region "commem_on_ssram" with offset 68,157,440 into device "ext_ssram" and length 4,096 .
- ① Removed memory region "sect_0" with offset 1,048,576 into device "ext_ssram" and length 4,096.
- ① Changed mapped section ".sect_0" from memory region "sect_0" to memory region "commem_on_ssram".
- ① Changed memory region "commem_on_ssram" to slave "ext_ssram", offset "0x100000", and length "4096"
- ① Changed memory region "commem_on_ssram" to slave "ext_ssram", offset "0x100000", and length "4096"

5. 共有関数を使用する

cpu_1 のプログラム内にある共有関数 func_1 を cpu_0 のソース上から使用します。

- ① cpu_1 のプログラム中(cpu_1.c)に下記のように `__attribute__` 記述で、3 章で指定したセクション(.sect_0)に func_1 を配置します。func_1 の関数本体も cpu_1 のプログラム中に記述します。

cpu_1 のプログラム

```

cpu_1.c
#include <stdio.h>
#include <io.h>
#include <unistd.h>
#include "system.h"

// *****
// 共有関数 func_1
// *****
int func_1(alt_u32 led_pio_base) __attribute__((section (".sect_0")));

int func_1(alt_u32 led_pio_base)
{
    while(1)
    {
        IOWR(led_pio_base, 0, 0x0);
        usleep(500000);
        IOWR(led_pio_base, 0, 0x1);
        usleep(500000);
    }
}

```

func_1 本体

- ※ 注) 共有する関数(例:func_1)内で `printf` 等の標準関数や HAL を使用している場合には、cpu_1 のプログラムのライブラリを使用しますので、それらの関数がスレッド・セーフかどうかを確認してご使用ください。
- ※ 共有関数自体を `mutex` 等のコアを使用し、それぞれの CPU からの共有関数へのアクセスを排他制御することもできます。

- ② cpu_0 のプログラム中で func_1 を使用します。cpu_0 のプログラムは func_1 の関数を持っていないので、cpu_1 のプログラム中の func_1 を参照して使用します。① で func_1 を .sect で指定したアドレスに配置したので、このアドレスを関数ポインタとして宣言し使用します。

cpu_0 のプログラム

```

cpu_1.c  cpu_0.c ✕
#include <stdio.h>
#include <io.h>
#include <unistd.h>
#include "system.h"

int main(void)
{
    int (*func_p)(alt_u32 led_pio_base);
    func_p = (int (*)(alt_u32))0x04100000;

    printf("Hello from Nios II 0, start!!");

    (*func_p)(LED_PIO_0_BASE);
    return 0;
}
    
```

- 関数ポインタ宣言
- .sect_0 で指定したアドレスをポインタに入力

関数 func_1 を関数ポインタ *func_p を使って呼び出します

改版履歴

Revision	年月	概要
1	2015年5月	初版

免責およびご利用上の注意

弊社より資料を入手されましたお客様におかれましては、下記の使用上の注意を一読いただいた上でご使用ください。

1. 本資料は非売品です。許可無く転売することや無断複製することを禁じます。
2. 本資料は予告なく変更することがあります。
3. 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がありましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。

株式会社アルティマ ホームページ: <http://www.altima.co.jp> 技術情報サイト EDISON: <https://www.altima.jp/members/index.cfm>

株式会社エルセナ ホームページ: <http://www.elsena.co.jp> 技術情報サイト ETS : <https://www.elsena.co.jp/elspear/members/index.cfm>

4. 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
5. 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカー発行の英語版の資料もあわせてご利用ください。