TMS320 DSP
DESIGNER'S NOTEBOOK

Addressing Peripherals as Data Structures in C

APPLICATION BRIEF: SPRA226

Nat Seshan and Mark Utter
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
April 1993

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Addressing Peripherals as Data Structures in C

Abstract

A DSP's peripheral-specific registers can be manipulated in C. Two methods are presented for handling this task. Each method has different advantages and disadvantages. Method 1 is very useful for addressing peripheral or memory buffers that are device specific. Method 2 is preferred for addressing peripherals or memory buffers which are not device specific (i.e., peripherals are user specified). Details of both methods and short code listings are used to describe the techniques.

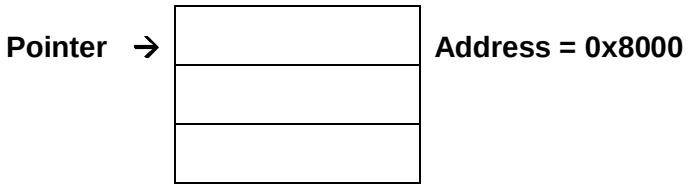
Design Problem

How can I manipulate a DSP's device peripheral-specific registers in C?

Solution

A data structure is usually assigned to `.bss` by the C compiler. A peripheral such as a serial port has control registers with an address different from `.bss`. The problem is to connect the two.

Method 1: Use a pointer to the peripheral.



Peripheral as memory locations

1.1 First declare a structure that logically represents the memory locations of the peripheral.

```
struct controller   {
                    unsigned int status;
                    ...
                    };
```

1.2 Declare a pointer to the structure and initialize it to the peripheral's address.

```
struct controller *IFperipheral = (struct controller *) 0x8000;
```

1.3 In your code, access the peripheral's memory values indirectly.

```
IFperipheral -> status = 0;
```

Method 2: Placing the structure in its own section.

2.1 Declare a peripheral instead of a pointer.

```
struct controller IFperiph;
```

2.2 Use inline assembly to give the structure its own section.

```
asm("_IFperiph .usect \"periph\", 128);
                  /* 128 is the size of struct */
```

This creates a user-defined section that can be linked to any address.

2.3 Use your linker command file to map the section to memory.

```
periph: load = 0x8000
```

2.4 Address the structure elements directly.

```
IFperiph.status = 0;
```

Both methods work. Sometimes the pointer method is most efficient. Other times, the second method is best. Method 1 is very useful for addressing peripheral or memory buffers that are device specific. Method 2 is preferred for addressing peripherals or memory buffers which are not device specific (i.e., peripherals are user specified). This method ensures the task of mapping and aligning user-specific peripherals and/or memory buffers to the linker. The choice depends on the individual application. For more information, read the TMS320C30 Peripheral Run-time Support Library Users Guide. Also see: DSP Applications Using the ’C30 EVM, “C Coding Tips for Application-Specific Processors.”