TMS320 DSP
DESIGNER'S NOTEBOOK

Interrupts in C on the TMS320C3x

APPLICATION BRIEF: SPRA227

Tim Grady
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
June 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

Interrupts in C on the TMS320C3x

Abstract

Writing interrupt routines in C is straightforward as long as you follow the simple rules set out in this note. There are several parts to the problem: (1) writing the ISR, (2) initializing the interrupt vector table, and (3) linking the parts together in the linker command file. You must make sure to generate the interrupt vector table and to provide the linker with all the necessary information to link the ISRs, vector table, and section names into the correct locations.

Clearly there are variations on this theme. Some ISRs can be written in C and some in assembly so long as the naming conventions and vector tables are followed and initialized.

Design Problem

How do I use interrupts from C?

Solution

There are several parts to this problem: (1) writing the ISR, (2) initializing the interrupt vector table, and (3) linking the parts together in the linker command file.

A C Language ISR

The C compiler requires that each ISR be named as follows:

```
void c_int0n(void)  /* n is the int number */
{
                  /* a C function that is an ISR */
}
```

The interrupt may not return a value and has no arguments. The C compiler recognizes this naming convention and treats it as a normal ISR, which means it performs a context save where needed and returns from the routine via a RETI instruction.

A good practice is to include the interrupts in a separate file called `ints.c` or something similar. This makes for a more modular style, simpler maintenance, and easier to understand software.

The Interrupt Vector Table

The first 40h addresses are reserved for the interrupt and trap vectors. Address 0 (zero) holds the address of the reset routine. If using C linker options, the RTS30.lib function `boot.asm` takes care of defining the reset function, but the vector table initialization is left to the user. You can do so with either C or assembly language.

An assembly language routine might look like this.

```
; file name is vectors.asm
;
;  .sect “vectors”;a new section begins here
   .word _c_int00 ;the address of the reset vector
   .word _c_int01 ;the ISR for interrupt 0
   .word _c_int02 ;the ISR for interrupt 1
;  etc.
;  end
```

This routine creates a new section that is merely a list of addresses where the interrupt routines can be found. It can be written in C by encapsulating each line in an `asm` statement.

For example:

```
asm(“ .sect \”vectors\” ”);
a C function that is an ISR.
```

Linking Them Together

The linker command file provides the mechanism for including the `vectors.asm` object and the `ints.c` object.

```
/* file name == mylink.cmd */
vectors.obj
ints.obj
```

The MEMORY section needs to identify the location of the int vectors.

```
MEMORY
{
   VECTORS: origin = 0h, length = 40h
   ...
}
```

The SECTIONS section needs to map the user-defined section called “vectors” to the memory location.

```
SECTIONS
{       vectors : > VECTORS
   ...
}
```

Summary

Writing interrupt routines in C is straightforward as long as you follow the simple rules set out in this note. You must also make sure to generate the interrupt vector table and to provide the linker with all the necessary information to link the ISRs, vector table, and section names into the correct locations.

Clearly there are variations on this theme. Some ISRs can be written in C and some in assembly so long as the naming conventions and vector tables are followed and initialized.