IMPORTANT NOTICE

The Processors Wiki was decommissioned on January 15, 2021.

Wiki pages that were migrated are redirecting to the new location. Pages not migrated are only accessible to users on the TI network. Users off the network can not access the Wiki and will receive a 404 error. **DO NOT** share Wiki URLs with customers. They can not access the Wiki site.

Accessing c int00

From Texas Instruments Wiki
Jump to: navigation, search

Contents

[hide]

Introduction[edit]

_c_int00 is the name of the symbol of the C environment entry point. Sometimes it is necessary to obtain this address (e.g., to reset the software) or even to set c_int00 address to a fixed location (so that the host that is booting the processor can make it jump to that location). This page describes how to achieve that.

Retrieving the c_int00 Address[edit]

If you need to retrieve the address of c_int00 at run time, use the code below as an example:

```
extern void c_int00(void);

void main( void )
{
    void (*x)(void) = c_int00;
}
```

For EABI programs, the symbol's name is _c_int00:

```
extern void _c_int00(void);

void main( void )
{
    void (*x)(void) = _c_int00;
}
```

Setting the c_int00 Address to a Fixed Location[edit]

When using an external host to boot the DSP, it might need the c_int00 address to be always the same. The procedures below describe how to achieve that.

Using DSP/BIOS[edit]

If you are using DSP/BIOS, the symbol _c_int00 is placed in a section named .sysinit.

You can create a new memory section and put .sysinit in it.

To do that, go to your DSP/BIOS Configuration file (.tcf). At **System**, right click on **MEM – Memory Section Manager** and select **Insert MEM** to create a new section. Adjust the properties of the new section: base, len. You can look in your map file to see the size of the **.sysinit** section to determine the minimum length that you need to fit it in this memory section that you created. Do not forget to adjust the other memory sections to make sure that they do not overlap each other. So, for example, if you are going to put sysinit in internal memory, reduce the base and len of IRAM if necessary. You can remove the heap in this section by unchecking the option **create a heap in this memory**. At the bottom of the Properties window, select **space: code**.

After that, right click on **MEM – Memory Section Manager** and select **Properties**. In the BIOS Code tab select your new memory section for **Startup Code Section (.sysinit)**.

Thus, the linker will always put the c_int00 entry point in the same location.

If you not only need the address of c_int00 to be always in the same location, but want to specify that location, you may need to create a custom linker command (.cmd) file that references the DSP/BIOS generated file.

To do that, in the tcf file, create a new memory region (for example called RAM4CINT00). Then create your own custom .cmd file containing (example for C5000):

```
-l configcfg.cmd  /* name of your original .cmd */

SECTIONS {
   .bootcode: {
        -lbios.a55L<boot.o55L>(.sysinit)

   } > RAM4CINT00
}
```

This should only put the code for _c_int00 into the new section .bootcode. Don't forget to put the normal .sysinit back to your original memory location.

Here's an example for 64x+ core:

```
SECTIONS
{
      boot > 0x00800000
      {
      -l bios.a64P<boot.o64P>(.sysinit)
      }
}
```

In the above example we are pulling in only the code for c_int00 and hard-coding it to a specific address, 0x00800000. This might be helpful on a device such as TMS320C6455 for doing HPI boot which begins executing from address 0x00800000 after the host sends the interrupt to the DSP to begin execution.

NOTE: If you you are using a C674x DSP, use a674 and o674. If you are using huge memory model for C5000, use a55H and o55H.

Using SYS/BIOS[edit]

If you are using SYS/BIOS, system start up and initialization is handled by XDCtools. For some targets, the c_int00 code is a C function. The C code is compiled with the "-mo" flag which places the symbol _c_int00 in a subsection of .text (.text:_c_int00).

To place the .text symbol at an explicit address, add a custom linker command (.cmd) file to your project with a directive similar to:

```
 SECTIONS {
   .text:_c_int00 > 0xc3000000
 }
```

On other targets, the _c_int00 code is written in assembly language. As of XDC 3.24.02, this assembly code is placed in .text and not .text:_c_int00. This will be fixed in a future release of XDC and BIOS. But, until then, the following linker workaround can be applied to place the

_c_init00 function. You will have to update your path and library name accordingly. Check your .map file for the name of the boot library, and the generated linker.cmd filed for the full path.

```
 boot : > 0x3D8000 PAGE = 0
 {
    -
l"C:\ti\ccsv5_3_0_00042\xdctools_3_24_02_30\packages\ti\targets\rts2800\lib\b
oot.a28FP" <boot_cg.o28FP> (.text)
 }
```

Not using SYS/BIOS or DSP/BIOS[edit]

If you are not using DSP/BIOS, you should be able to specify the address using the linker command file. The general idea is to put the boot module (where _c_int00 is defined) into an output section of its own, then allocate that output section to a specific memory address. For example for C5000:

```
boot > 0x1234
{
    -l rts55.lib<boot.obj>(.text)
}
```

You can change the address 0x1234 to be whatever address that you want to put the code. Probably the start of your RAM would be a good place so that the linker doesn't have problems allocating other objects around it.

NOTE: If you are using a C64x+ DSP, replace rts55.lib with rts64plus.lib. If you you are using a C674x DSP, use rts6740.lib. If you are using huge memory model for C5000, use rts55h.lib.

Related information and forum posts[edit]

address of _c_int00

_c_int00 address

How to specify an exact address for c_int00?

{{

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum

Keystone=

- For technical support on MultiCore devices, please post your

C2000=*technica support the C200 please p your question on The C2000*

- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Accessing c int00** here.

questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Accessing c int00** here.

*Forum. Please p only commen about th article **Accessin c int00** here.*

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors
- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

Categories:

- DSPBIOS
- Compiler
- SYSBIOS

Navigation menu

Personal tools

- Log in
- Request account

Namespaces

- Page
- Discussion

Variants

Views

- Read
- View source
- View history

More

Search

Go

Navigation

- Main Page
- All pages
- All categories
- Recent changes
- Random page
- Help

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

- This page was last edited on 6 February 2013, at 09:44.