DSP MultiCore Application Dynamic Loader : Invoker

Invoker

GongGa

Version: 1.0

Last update: October 11, 2019

Abstract

This document describes the usage of DSP MultiCore Application Dynamic Loader : **Invoker**.

1 Motivation for Invoker

- Need to deploy multiple applications on multiple cores.
- Need to deploy an application dynamically on a core
- Need to deploy applications automatic
- Need to control the startup sequence of applications

2 Invoker infrastructure components

The Invoker infrastructure provides a set of utilities to help achieve the above mentioned needs. There are two major utilities used by Invoker. The HEX Decoder and the DSP Loader.

2.1 Hex Decoder

The dynamic deploy application is build by CCS, the Hex Decoder is use to crop and decode the *.out file generate by CCS.

- COFF executable program analysis.
- Program segmentation analysis.
- Program information collection.

2.2 DSP Loader

The DSP Loader is used to load the COFF file generate by CCS to DSP6678.

- Program memory address space allocation.

- Program loading.
- Program entry address setting.
- Program start sequence control.
- Multi-core reset control.

3 Build program

The Invoker does not do any address assignment for the application segments, the application developer has taken of mulicore address assigment for the applications and we highly recommended that the application is build under the memory partition(2G DDR3) advice provide by Invoker(show in **Table** 1).

Table 1: Memory Partition

CoreID	Start Addr	End Addr	Byte
Core 0	0x80000000	0x8FFFFFFF	256M
Core 1	0x90000000	0x9FFFFFFF	256M
Core 2	0xA0000000	0xAFFFFFFF	256M
Core 3	0xB0000000	0xBFFFFFFF	256M
Core 4	0xC0000000	0xCFFFFFFF	256M
Core 5	0xD0000000	0xDFFFFFFF	256M
Core 6	0xE0000000	0xEFFFFFFF	256M
Core 7	0xF0000000	0xFFFFFFFF	256M

To prevent memory conflicts between multiple programs, the memory address space is divided into 8 parts. Each program should build under the partition rules, otherwise the program may be destroyed by another program.

3.1 Using .cmd file

In CCS, program can use a .cmd file to config the section allocation.

```
MEMORY
{
    MSMCSRAM   o = 0x0C000000 l = 0x00400000 /* 4MB   Multicore shared memory (DSP C6678) */
    DDR3       o = 0xA0000000 l = 0x10000000 /* 1GB   DDR3 (DSP C6678)                    */
}

SECTIONS
{
    .text:_c_int00  >  0xA0000000   /* C program entry point                            */
    .text           >  DDR3         /* Executable code and constants                    */
    .cinit          >  DDR3         /* Initialize the table                             */
    .const          >  DDR3         /* Global and static constants                      */
    .switch         >  DDR3         /* Jump table                                       */
    .stack          >  DDR3         /* stack                                            */
    .far            >  DDR3         /* Global and static variables                      */
    .fardata        >  DDR3         /* Initialized non-static global and static variables*/
    .cio            >  DDR3         /* C Input and output buffer                        */
    .sysmem         >  DDR3         /* Dynamic memory allocation area                   */
}
```

3.2 Using .cfg file

Or use the .cfg file to do the section allocation.

```
Program.sectMap[".const"]     = "DDR3";
Program.sectMap[".switch"]    = "DDR3";
Program.sectMap[".cinit"]     = "DDR3";
Program.sectMap[".cio"]       = "DDR3";
Program.sectMap[".vecs"]      = "DDR3";
Program.sectMap[".DDR3_0"]    = "DDR3";
Program.sectMap["SystemHeap"] = "DDR3";
Program.sectMap[".far"]       = "DDR3";
Program.sectMap[".fardata"]   = "DDR3";
Program.sectMap[".stack"]     = "DDR3";
Program.sectMap[".bss"]       = "DDR3";
Program.sectMap[".text"]      = "DDR3";
Program.sectMap[".rodata"]    = "DDR3";
Program.sectMap[".neardata"]  = "DDR3";
Program.sectMap[".system"]       = "DDR3";
```

4 COFF file

The executable programs on DSP exist as files in COFF format, want more information, google it.