Application Brief

Runtime Stack Size Monitoring for C2000™ MCUs

Luke Chen

Introduction

Sufficient stack memory space is an important factor for stable system operation. However, too large stack size allocation wastes the memory space and too small stack causes a system crash due to stack overflow. The allocation of the stack memory size has always been a matter of discussion. This application brief discusses how to monitor the used stack size at runtime for C2000™ real-time microcontrollers, and helps us to determine the optimal stack memory size.

Implementation

The method typically used is to fill the stack with a known value (0xAA55) prior to running the code. During software development stage, you can use the fill command in the linker command file or use the Code Composer Studio™ (CCS) memory fill option to fill the entire stack with the defined known value. Then, when you run the application code, anytime you halt you can use the CCS memory browser to check the stack memory to see how much stack space has been used.

However, if you want to monitor the stack utilization in standalone mode, or generate a hex file for firmware upgrade or a standalone programmer, you should not use the fill command in the linker command file. Instead you need a method to fill the stack at runtime and to monitor the stack utilization. To achieve this, the below proposed _system_pre_init() function can be a good method.

The _system_pre_init() function is invoked after the stack pointer is initialized and before any C/C++ environment setup is performed. To keep the used stack memory size under control, create the small piece of C code in the application software filling the stack space.

Stack Address and Stack Size

Add the C code shown below to your application software, so that you are able to obtain the stack space size and start address information: either legacy COFF or EABI format.

```
01 #if defined(__TI_EABI__)
02     #define STACK_MEM_ADDR      __stack          // Symbol of stack start address
03     #define STACK_MEM_SIZE      __TI_STACK_SIZE  // Symbol of stack memory size
04 #else
05     #define STACK_MEM_ADDR      _stack           // Symbol of stack start address
06     #define STACK_MEM_SIZE      _STACK_SIZE      // Symbol of stack memory size
07 #endif
08
09 #define STACK_KNOWN_DATA        0xAA55           // Define the known data
10
11 extern int  *STACK_MEM_ADDR;
12 extern void *STACK_MEM_SIZE;
```

Fill the Stack Space

Include the C function shown below to fill the stack space with the defined known data at runtime. When caller calls this function, the Return Program Counter (RPC) register value is pushed on the stack, then the function return address is stored in the RPC register.

If a break point is set at line 10 of the _system_pre_init() function to halt, use the CCS memory browser to check stack space. You can see six words of stack space are used. Table 1 shows us the stack memory address description. Skip these used memories and fill the rest of stack space.

```
01 //
02 // Function used to runtime fill stack space with the known data
03 //
04 int _system_pre_init()
05 {
06     int *ptr_stack  = (int *)(&STACK_MEM_ADDR);        // Get stack start address
07     long stack_size = (long)(void *)(&STACK_MEM_SIZE); // Get stack memory size
08
09     // Skip used stack space
10     ptr_stack  += 6;
11     stack_size -= 6;
12
13     // Fill unused stack space with the known data
14     do
15     {
16         *ptr_stack++ = STACK_KNOWN_DATA;
17     } while(--stack_size > 0);
18
19     // return one non-zero value
20     return 1;
21 }
```

Table 1. The Stack Memory Address Description

Stack Memory Address	Memory Description
Stack Start Address + 0	RPC register data, 2 words
Stack Start Address + 2	Local variable stack_size, 2 words
Stack Start Address + 4	Local variable ptr_stack, 2 words
Stack Start Address + 6 to Stack Start Address + Stack Size - 1	Unused memory space

Check used stack size

Include the C code shown below to the application code, and periodically call the monitor_used_stack_size() function. For example, in the main routine or timer ISR, the global variable stack_mem_used shows the actual used stack size of the system.

When the system is operating in standalone mode, you can transmit this data to the terminal via the SCI port and then you can determine the maximum stack size needed for the system under all operating conditions. This information helps to optimize the stack size.

In this document, LAUNCHXL-F28379D is used to evaluate all the example C code. However, this runtime stack size monitoring example can be applied to all C28x C2000 devices.

```
01 // Global variable to monitor used stack size
02 long stack_mem_used = 0;
03
04 //
05 // Function to monitor used stack size
06 //
07 void monitor_used_stack_size(void)
08 {
09     int *ptr_stack = (int *)(&STACK_MEM_ADDR);           // Get stack start address
10     long stack_size = (long)(void *)(&STACK_MEM_SIZE);   // Get stack memory size
11
12     // Check the filled known data and determine used stack size
13     do
14     {
15         if(*ptr_stack++ == STACK_KNOWN_DATA)
16         {
17             break;
18         }
19     } while(--stack_size > 0);
20
21     // Get used stack size
22     stack_mem_used = ((long)(void *)(&STACK_MEM_SIZE) - stack_size);
23 }
```

Conclusion

This application document focuses on the application level, showing how to include the simple C code to fill stack memory with the known values at runtime, to monitor the amount of stack size required by the application code.

C2000 devices built-in the C28x Emulation Analysis Block that can be an alternate way to check the stack size needed by the application code. For details, see *Online Stack Overflow Detection on the TMS320C28x DSP*.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265