



SRAM Performance Gains

Application Note

Table of Contents

SRAM PERFORMANCE GAINS	1
INTRODUCTION	3
NOTE FOR SRAM ON THE MCF5272 BASED PRODUCTS:	3
MODIFYING SRAM USAGE	4
SYSTEM GLOBAL VARIABLES	4
RTOS SYSTEM TASK STACKS	5
SYSTEM BUFFERS	6
USER SRAM USAGE	7
REBUILDING SYSTEM FILES	8

Introduction

This application note describes how to increase application performance by locating frequently used data in the processor's on-chip zero wait state static RAM (SRAM). This application note is applicable for all NetBurner products and requires the NetBurner Network Development Kit (NNDK) 2.2 or higher. This feature replaces the older "Fast Buffer" performance feature. The SRAM performance feature is enabled by default but can be reconfigured to best suite any specific application.

The NNDK directly uses the on-chip SRAM embedded on the processor to store highly-used / critical data. The speed that the CPU can access this SRAM is much faster than external SDRAM because a 32-bit value can be accessed in a single clock cycle. The performance gains over older pre-SRAM versions of the NNDK are up to 10X faster RTOS task switches and 200+% gains for network bandwidth.

The SRAM on the ColdFire processor has always been available to the user however in older releases of the NNDK it was never directly used by the system. The memory map for the processors SRAM is now recognized by the linker script for all products. This allows users to easily add and remove data from the SRAM by using attributes in the variable declaration.

Note for SRAM on the MCF5272 based products:

The MCF5272 has a smaller amount of on-chip SRAM then other ColdFire processors used in NetBurner products: 4KB rather than 64KB. This limits the types of data that can be used in this fast SRAM. Any mention of RTOS task stacks are not applicable to this processor since the default stack size is more then twice the size of the available SRAM. Any mention of network buffers are also not applicable to this processor since only a single buffer can fit in the SRAM.

Modifying SRAM usage

The NNDK uses SRAM for various sections of system data and any one of these sections can be enabled or disabled by editing the source file: `\nburn\include\constants.h`. The main global enable for SRAM is “`#define ENABLE_SRAM_SYS`”. If this line is commented out of `constants.h` and the system files are rebuilt then the only part of SRAM used by the NNDK will be the initial 1KB for the vector table. This is how the NNDK was configured in older software releases. For information on rebuilding system files refer to the “Rebuilding System Files” section of this application note.

Important: Any changes to a system file, such as `constants.h`, require the libraries be rebuilt. Otherwise your change will have no effect.

The system SRAM usage is broken down into sub-sections that can be enabled or disabled individually. The main system sections used in SRAM are global variables, system task stacks and network buffers.

System Global Variables

Most system global variables should be left in SRAM as they are by default. This is due to the fact that they take up very little room compared to stacks and also account for a significant gain in overall performance. Any individual section of variables can be disabled in the same fashion that the entire SRAM can be disabled, by commenting out the `#define` and recompiling the system libraries. The following is a list of system variables that are all located in SRAM by default:

`#define FAST_SYSTEM_VARIABLES`

This section of variables is directly related to the RTOS. When this section is defined there will be increased performance for task/interrupt switching. This section is enabled by default.

`#define FAST_ETHERNET_VARIABLES`

This section of variables is directly related to the Ethernet driver. When this section is defined there will be increased performance for all network activities. This section is enabled by default.

`#define FAST_BUFFERS_VARIABLES`

This section of variables is directly related to the buffer system. When this section is defined there will be increased performance for all code that use buffers, which includes all network and serial transmissions. This section is enabled by default.

`#define FAST_IP_VARIABLES`

This section of variables is directly related to the IP stack. When this section is defined there will be increased performance when using the network stack. This section is enabled by default.

`#define FAST_TCP_VARIABLES`

This section of variables is directly related to the TCP task of the IP stack. When this section is defined there will be increased performance of TCP. This section is enabled by default.

RTOS System Task Stacks

System stacks in SRAM account for major part of system performance. Task stacks should be placed in SRAM with more care since they take up a large amount of space. The default stack sizes are 8KB and there is 63KB of total SRAM available. Attention should also be paid when increasing the stack size of any stack in SRAM as it may now be too large to fit in the available SRAM space.

When a task stack is located in SRAM, any local variable in that task will also be located in SRAM since they are stored in the stack. Any function called in this task will also speed up since the call stack is now single cycle accessible. Switch time for both tasks and interrupts will have significant performance boost since a task state is stored or restored in its stack for both task and interrupt switches. When an interrupt occurs it will also be using the current tasks stack for any local variables. There is a significant change in the interrupt performance when an interrupt occurs during a task with a stack in SRAM compared to a task with a stack in SDRAM. This puts importance on placing stacks with the biggest slice of CPU time SRAM. The following is a list of system stacks that can be placed in SRAM:

`#define FAST_IDLE_STACK`

This section places the system Idle task stack in SRAM. The idle stack is used when all tasks are blocking. This section is NOT enabled by default. If you determine the CPU spends a significant amount of time in the idle task then it will benefit your interrupt performance and task switch time if you place this stack in SRAM.

`#define FAST_MAIN_STACK`

This section places the system Main task stack in SRAM, this is the stack used by UserMain. This section is enabled by default.

`#define FAST_ETHERNET_STACK`

This section places the system Ethernet task stack in SRAM which is used for all network activity. This section is enabled by default.

`#define FAST_IP_STACK`

This section places the system IP task stack in SRAM which is used for most network processing. This section is enabled by default.

```
#define FAST_TCP_STACK
```

This section places the system TCP task stack in SRAM which is used for all TCP based communication (HTTP, Telnet, Email, FTP, ext...). This section is enabled by default.

The following are additional system stacks that can be placed in SRAM but are NOT enabled by default:

```
#define FAST_HTTP_STACK
```

```
#define FAST_FTP_STACK
```

```
#define FAST_WIFI_STACK
```

```
#define FAST_PPP_STACK
```

```
#define FAST_COMMAND_STACK
```

System Buffers

System buffers will have a large performance impact on all network activities when placed in SRAM. System buffers are enabled in SRAM by default and can be disabled in constants.h by commenting out “#define FAST_BUFFERS”. This feature works in a similar fashion to the FAST_BUFFERS feature that existed in older releases of the NNDK. An additional primary buffer pool is created in SRAM. When a buffer is obtained the system will first look at the SRAM pool, if none are available then it will get one from the standard SDRAM buffer pool. The size of the buffer pool in SRAM is determined at runtime initialization. When the buffer pool is created it will automatically fill all remaining SRAM with as many buffers as possible. If less SRAM space is used by variables and stacks, more space will be used for the fast buffer pool. Nearly all applications should leave this section enabled since it will only use SRAM if there is unused space available.

USER SRAM Usage

There are attribute keywords in the NNDK that allow user task stacks or global variables to be added into SRAM. This can greatly increase the performance of an application when used on highly accessed data.

The attribute used for a task stack is “FAST_USER_STK”. To use this attribute you simply add it to the end of your stack declaration:

```
DWORD MyTaskStack[MY_STACK_SIZE]__attribute__((aligned( 4 ))) FAST_USER_STK;
```

The attribute “*__attribute__((aligned(4)))*” has always been required for stacks in SDRAM but is not required if the stack remains in SRAM. The NNDK will automatically align stacks that use the FAST_USER_STK attribute but it is still a good idea to include the alignment attribute in case you remove the stack from SRAM.

The attribute that should be used for user global data is “FAST_USER_VAR”. This attribute is used exactly like the stack attribute:

```
DWORD MyData[MY_DATA_SIZE] FAST_USER_VAR;
```

NOTE: Data that is placed in SRAM will ALWAYS be initialized to 0. Any initialization for SRAM variables must be done at runtime. For example:

```
int MyCount FAST_USER_VAR = 1; // Invalid initialization. MyCount still initializes to 0
```

In constants.h there are two defines that allow you to enable or disable user SRAM:

```
#define FAST_USER_STACK
```

This section enables user task stacks to be placed in SRAM instead of SDRAM. When this is defined, any stack declared with the FAST_USER_STK attribute will be placed in SRAM.

When this line is commented out the FAST_USER_STK attributes will be ignored and these stacks will be placed in SDRAM. This section is enabled by default.

```
#define FAST_USER_VARIABLES
```

This section enables global user data to be placed in SRAM instead of SDRAM. When this is defined, any global variables declared with the FAST_USER_VAR attribute will be placed in SRAM. When this line is commented out the FAST_USER_VAR attributes will be ignored and these stacks will be placed in SDRAM. This section is enabled by default.

Rebuilding system files

Any changes to a system file, such as constants.h to enable or disable SRAM sections, will require rebuilding of the NNDK system files and your project. The following two methods can be used to rebuild the system files:

The NBEclipse IDE

1. Make sure your project is set to the correct platform and highlighted in the C/C++ projects window
2. From the 'NBEclipse' menu select 'Rebuild System Files'
3. From the 'Project' menu select 'clean'
4. From the 'Project' menu select 'build project'

Command Line

1. Type set PLATFORM = <your platform>
2. Set the directory to the nburn system directory (default is C:\nburn\system)
3. Type 'make clean'
4. Type 'make'
5. Set the directory to the platform system directory (default is C:\nburn<your platform>\system)
6. Type 'make clean'
7. Type 'make'