To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

SuperH RISC Engine Family

Example of Using Huffman Coding Data Compression

Introduction

In any embedded system memory is a valuable resource. The use of code, compiler and linker optimisation techniques can make the most efficient use of both the Flash/ROM and RAM available in a system. In some situations it may be possible to squeeze more out of the available Flash and RAM resources by compressing code and/or data and storing it in Flash and then decompressing this information when it is needed. This is particularly suited to environments where there is limited Flash memory but an abundance of SRAM or SDRAM memory. In such a situation code can be stored as a compressed image in Flash and then decompressed at runtime into the RAM from where execution can continue. The demonstration project available with this application note shows how this can be done with the SH-2 microcontrollers and the SH-2 simulator within HEW.

The compression method used in this application note and example application is that of Huffman coding. It is not the objective of this document to explain Huffman coding in any detail as this topic is extensive covered by articles available on the Internet. A Google search for ‘Huffman coding’ should lead to many of these documents. The compression of data is performed on a Windows® based PC using a custom command line application called ‘motice_huff.exe’ provided with this apps note. This utility compresses the contents of an s-record file and formats the compressed data into a constant C structure which can be added to a project. The decompression of this data is then performed at runtime by code executing on the microcontroller, in the case of the example an SH-2 based microcontroller.

Extensive use of source code from the book "Mastering Algorithms with C" by Kyle Loudon, published by O'Reilly Media Inc has been made. It is recommended that this book is read in conjunction with this apps note. Along with data compression techniques this book is also a valuable resource of information and source code relating to data encryption, sorting, searching and data structures.

Contents

Compressing the Data

The compression of s-record data is performed using a Windows® command line application called 'motice_huff.exe' which is available with this apps note. This utility is used as follows.

```
motice_huff <source> <destination> <structname>
```

Where:

source is the filename of the Motorola S-Record file to be processed

destination is the filename of the C file to be created

structname is the name of the constant C structure contained in the destination file

The resulting C structure is of type 'rom_data' which is defined as.

```
struct rom_data {
      unsigned long start_address;
      unsigned long orig_data_length;
      unsigned long compressed_data_length;
      unsigned char data[];
};
```

Decompressing the Data

The Huffman coded data can be decompressed using the function 'huffman_uncompress' which has the following prototype.

```
int huffman_uncompress(const unsigned char *compressed, unsigned char
*original);
```

Where,

compressed is a pointer to the compressed data

original is a pointer to an area of RAM where the data can be decompressed into

Note that this function has been altered from that supplied with "Mastering Algorithms with C".

The user is responsible for ensuring that sufficient RAM is available for the decompressed data. The function returns the size of the decompressed data if decompression is successful or -1 if it is not.

Demo Application

A HEW4 workspace is available with this apps note containing two projects. ‘CompressedApplication’, as the name suggests is the application linked to RAM which is compressed by ‘motice_huff’ and the project ‘HuffCompression’ decompresses the image into RAM and calls it via the reset vector. These projects have been built for the Renesas SH-2 microcontroller family and executed using the HEW SH-2 simulator. Using the simulator means this application can be easily evaluated without the need for any hardware. The compiler used is the Renesas SH compiler toolchain version V9.00.01.001 with HEW4.

‘CompressedApplication’ Project

The code in this application is linked to address H’00200000 to emulate RAM based addresses. This application simply writes text strings to the simulated I/O window of the SH-2 simulator. The linker is configured to generate a Motorola s-record file which is then processed by ‘motice_huff.exe’ to compress the data in the file and format it as a constant C structure. The screenshot shown in figure 1 shows how ‘motice_huff’ has been added as a build phase within HEW.

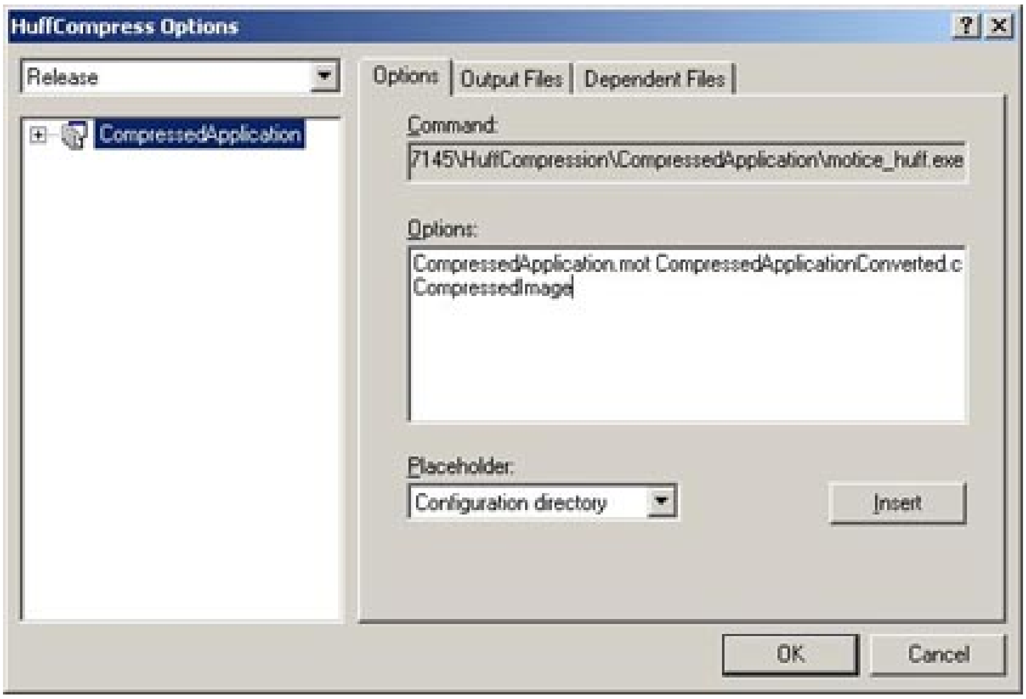

Figure 1: Motice_huff.exe Build Phase

The operation of this application should be easy to understand by reading through the source code of the project. In particular, the file 'CompressedApplication.c' should be studied.

'HuffCompression' Project

This project contains the main application and includes the file 'CompressedApplicationConverted.c' which is the output from the 'Motice_huff' build phase of the 'CompressedApplication' project. This file contains the constant C structure 'CompressedImage' of type 'rom_data' which consists of the compressed image and size and address information. Space for the decompressed image is allocated in the 'RamCode' array located in the 'RAMCODE' linker section using the #pragma directive shown below taken from the file 'HuffCompression.c'.

```
#pragma section RAMCODE
unsigned char RamCode[ RAMCODE_SIZE ];
#pragma section
```

The RAMCODE section is linked from address H'00200000 which is the same address to where the 'CompressedApplication' was linked to. It is important that this project and the RAMCODE section are linked to the same addresses. The compressed image is then decompressed by the function call shown below.

```
i = huffman_uncompress( CompressedImage.data, RamCode );
```

This function returns the number of bytes in the decompressed image and so can be used to check whether an error has occurred during the decompression by comparing the returned value with 'CompressedImage.orig_data_length'. If no error is detected the RAM based code can then be called by loading its reset vector and populating a function pointer with its contents. The new application can then be called using this pointer.

By loading and executing 'HuffCompression.abs' file in the SH-2 simulator with the simulated I/O window visible it should be observed that the decompressed 'CompressedApplication' running from RAM writes text strings into this window.

Compression Statistics

When the 'motice_huff.exe' utility runs as a build phase within HEW information relating to it operation can be observed in HEW's build window. Figure 2 below shows information relating to the size of the original and compressed data.

Compressed Application		
Original Size	Compressed Size	Saving
24692 bytes	21843 bytes	2849 bytes (11.5%)

Figure 2: Compression Statistics

From figure 2 it can be seen that the original application image of approximately 25kB has been reduced by 11.5%, a saving of 2849 bytes. If this same compression ratio was achieved with the 256kB flash memory contents of a Renesas SH7145F microcontroller an additional 25kB of code could be squeezed into the flash memory. The size of the decompression program used within the 'HuffCompression' project is approximately 7kB. So, the use of this type of compression can be beneficial when used on large program images.

The decompression code makes extensive use of dynamically allocated memory and as such a heap of 16kB is required. However, in a system where code is to run from RAM it is unlikely that this memory requirement would cause concern. The decompression time for this 25kB image stored in 32-bit 1-state flash decompressed into 16-bit 3-state (external) SRAM with the heap in the same external SRAM was approximately 500ms for a 50MHz SH-2.

Conclusion

In embedded systems where an application is too large to fit into the available flash/ROM resources and large amounts of RAM are available an alternative to moving to a device with more flash could be to compress the flash/ROM contents and decompress them to RAM at runtime. Even in systems where RAM resources are limited parts of the flash/ROM based code could be decompressed into RAM when required.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com