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

Renesas MCU

Utility To Convert S-Record Data To A C-Structure

Introduction

In embedded applications it is often desirable to store program code in one area or type of memory but execute it from another. An example would be routines to program internal Flash memory. Such programming routines cannot execute from the same Flash memory that is being programmed and so must be run from elsewhere. In single chip systems the only other memory available is internal RAM. In such cases the program code must be stored in non-volatile Flash memory and then copied into RAM at runtime for execution. Here lies a problem. For correct execution the program code must be linked for the memory addresses where it will run (e.g. RAM) but moved into non-volatile memory (e.g. Flash) at build time. The utility, 'Motice_cl', described in this document provides one method of achieving this. It should be noted that there are many other approaches to tackling this problem.

Contents

1. Motice_cl

'Motice_cl' is a command line (DOS) application that converts the data presented in a Motorola s-record file into a C structure. The idea being that the function to be relocated is processed as follows:

1. Function is compiled to an object file.
2. Compiled object file is linked to RAM address.
3. S-record file representing linked absolute data is produced.
4. S-record file is processed using 'motice_cl' to produce a C structure containing constant data.
5. C structure is added to final application, compiled and linked.
6. At runtime the data in the structure is copied to RAM.
7. Relocated function is called using a function pointer.

This results in code that has been linked for RAM addresses being stored in non-volatile memory and then copied to and run from the original RAM addresses.

2. Calling Mechanism

The command line interface for the utility is:

MOTICE_CL source destination structname

Source: Specifies the name and path of the s-record file to be processed.

Destination: Specifies the name and path of the C file to be produced.

Structname: Specifies the name of the C structure to be generated.

3. Output Format

The C file output by 'motice_cl' has contains the following structure definition.

```
struct rom_data {
    unsigned long start_address;
    unsigned long data_length;
    unsigned char data[4708];
};
```

The size of the 'data' array is dependent on the amount of data in the processed s-record.

The format of the structure should be self explanatory. The 'start_address' is the base address where the byte 'data' should be loaded to prior to execution and 'data_length' is the number of data bytes in the 'data' array in the structure.

An example of the structure, in part, is shown below.

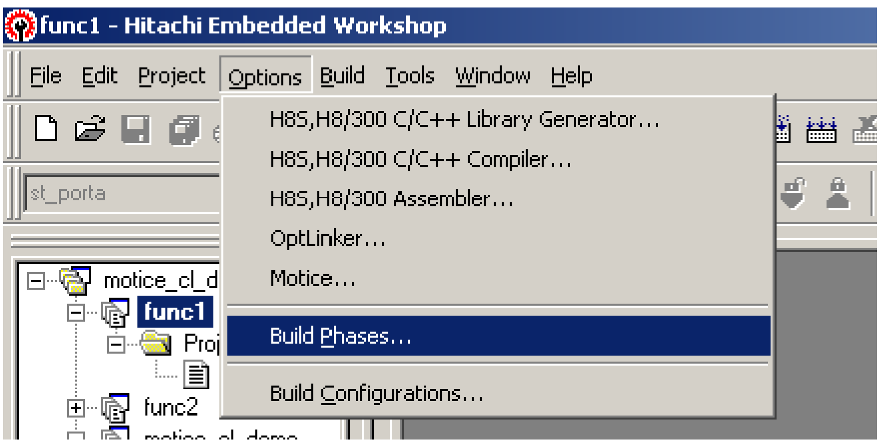
```
const struct rom_data datastruct = {
    0x00000000,
    0x00001264,
    0x00, 0x00, 0x08, 0x00, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00,
    0x08, 0x00, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x04, 0x00,
    0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x04, 0x00,
    0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
……
};
```

This example shows the data starting at address zero.

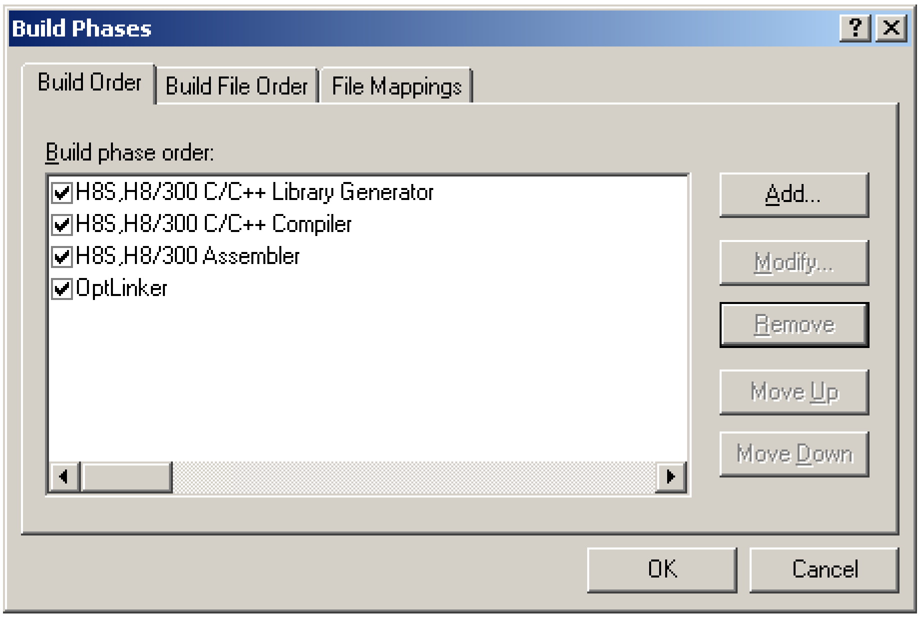
4. HEW Integration

One of the features of HEW (High-Performance Embedded Workshop) is the ability to produce custom build phases. This lends itself well to 'Motice_cl' as a separate build phase can run to call the utility each time an application is built. The following screenshots show the process of setting up the build phase.

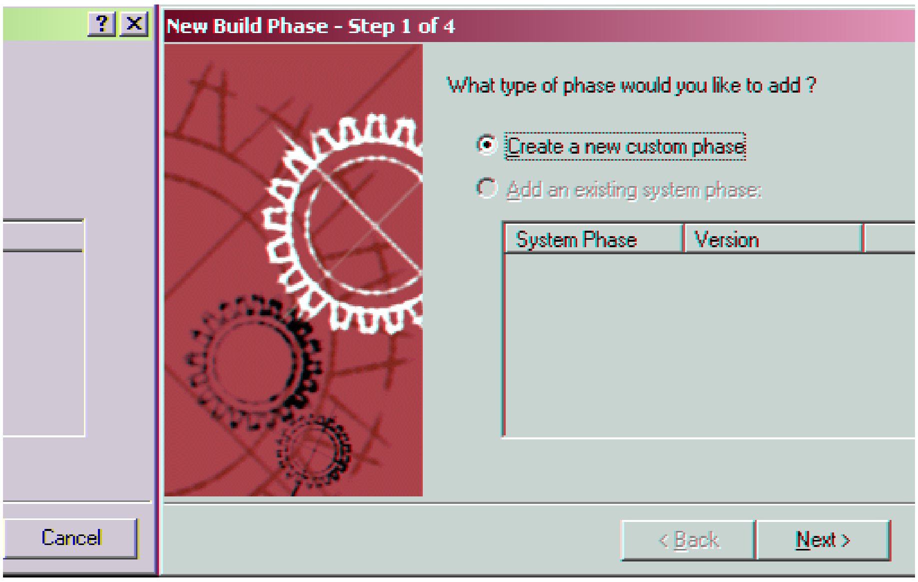
1. Select 'Build Phases' from the 'Options' menu.



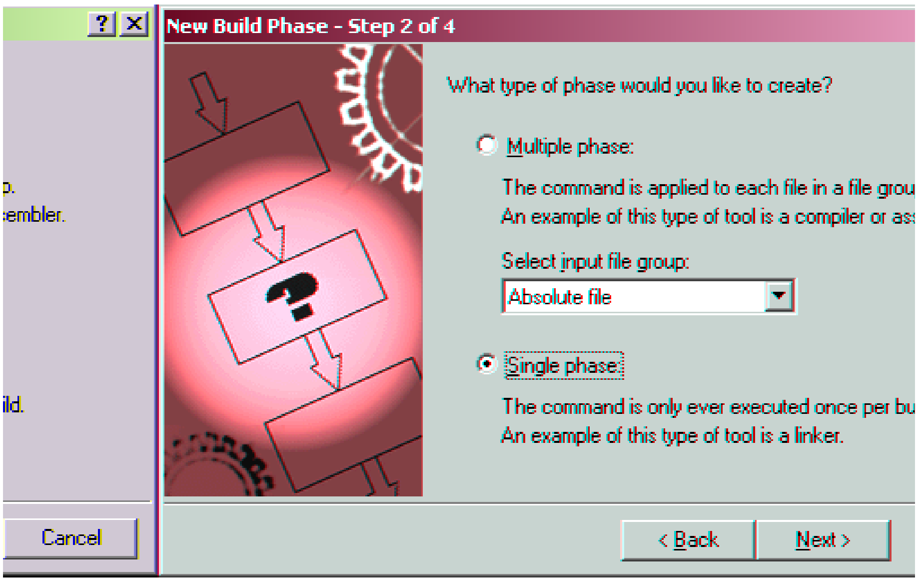
2. Click on the ‘Add’ button.



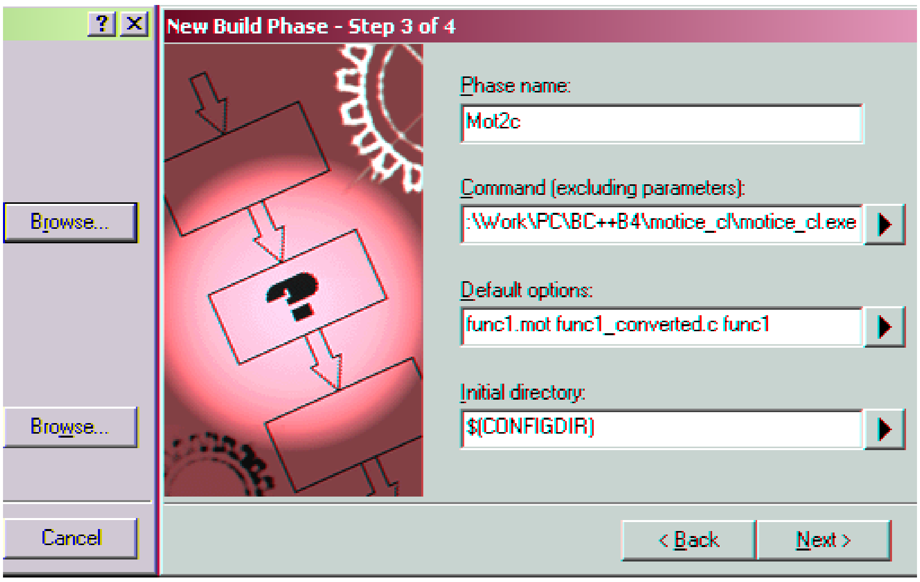
3. Create a custom build phase by clicking ‘Next’.



4. Create a single build phase. This means that the phase will only execute once per build. The ‘Motice_cl’ utility is only required to run once when it processes the s-record file output from the linker.

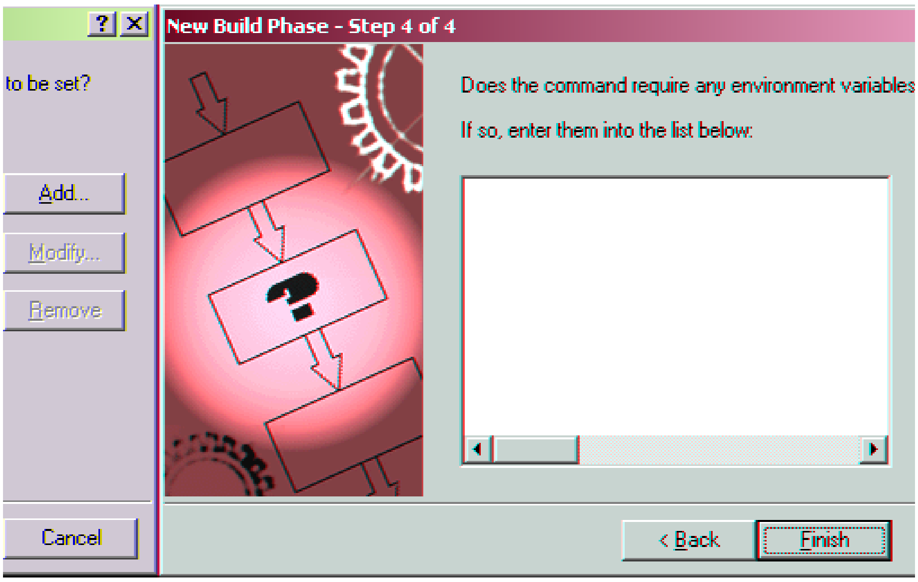


5. Configure the build phase to call ‘Motice_cl’ with the correct command line parameters.

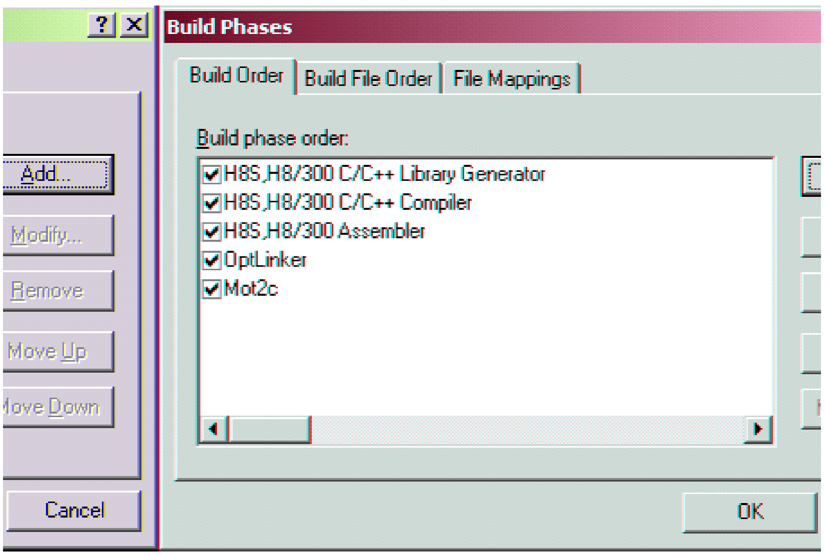


Here the phase has been called ‘Mot2c’, the full path for the ‘Motice_cl’ executable has been entered as have the default parameters for the command. In this case the default parameters are the command line parameters for the utility.

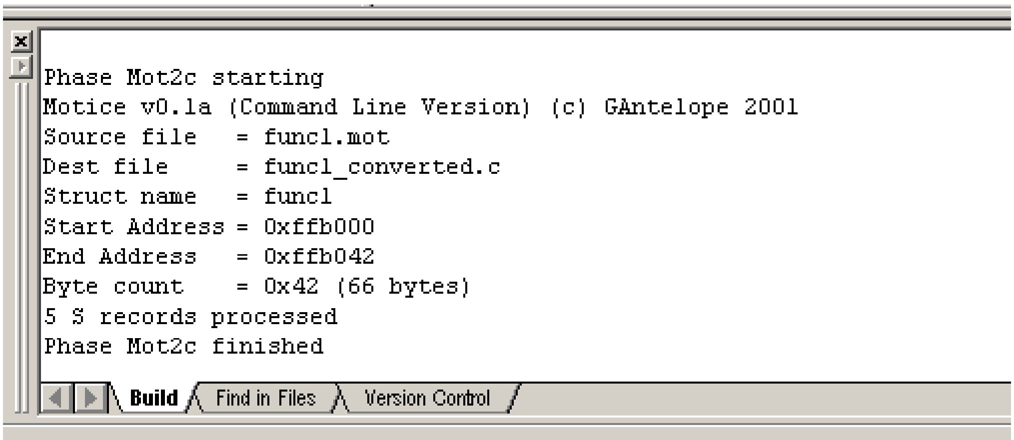
6. No environment variables are required for the utility so leave this blank and hit ‘Finish’.



7 The ‘Build Phases’ form should now show the newly created ‘Mot2c’ build phase. Ensure that it is ticked so that the phase is executed. The order the build phases are executed can be changed. This is not necessary here as the ‘Motc’ phase must run after the linker otherwise it will have no s-record file to process!

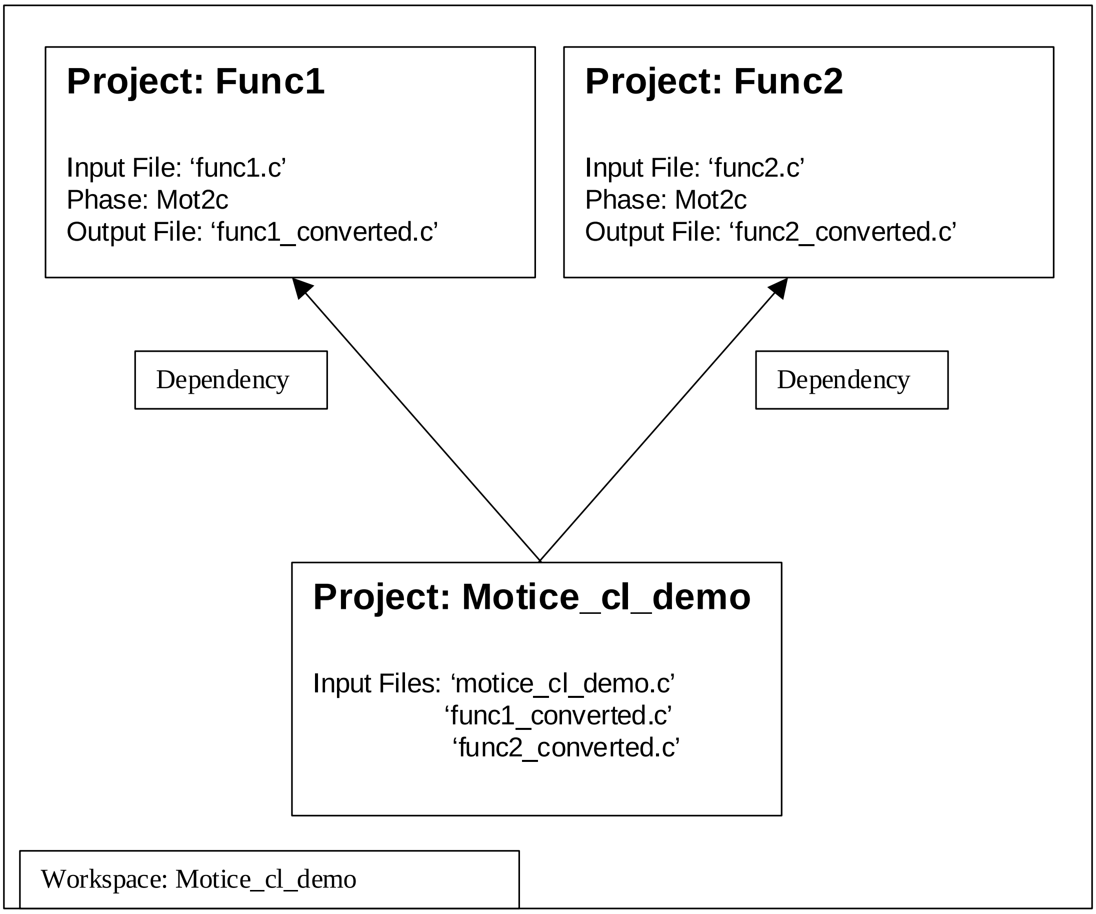


8 Click ‘OK’. Now when the project is built the ‘Mot2c’ phase will execute and convert the s-record file to a C structure. The screen shot below shows the output from the phase in the HEW build window.



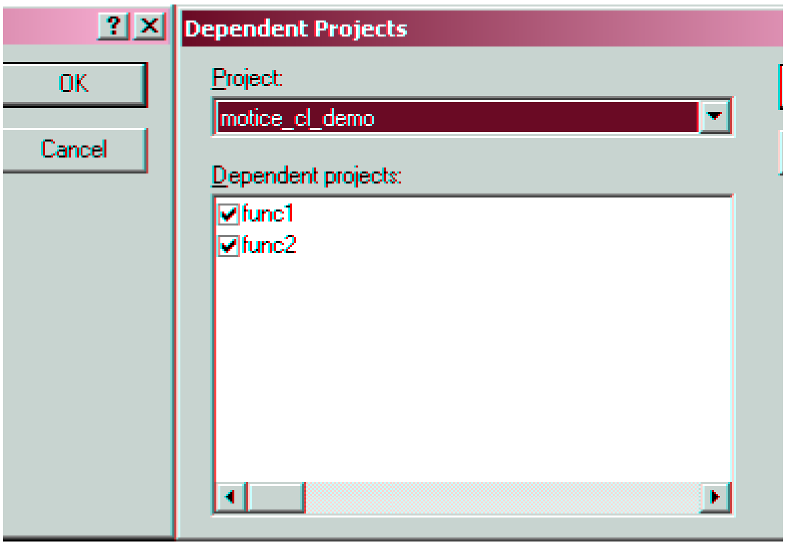
For simplicity it is recommended that each function to be processed be placed in its own project within a HEW workspace. Each function’s project should be created as an ‘Empty Project’ using the HEW project generator with the linker sections located at the required runtime addresses. Each project should have its own custom ‘Mot2c’ build phase. A master project can be created that compiles and links the C files output from the individual projects.

To aid clarification a HEW workspace's hierarchy is shown by the following diagram.



Here there is a HEW workspace called 'Motice_cl_demo' which contains the projects 'motice_cl_demo', 'func1' and 'func2'. Both the 'func1' and 'func2' projects consist of a single file which is compiled and linked to an s-record file and then processed by 'Mot2c' build phases to produce the files 'func1_converted.c' and 'func2_converted.c' respectively. To ensure that any changes to the code in the 'func1' and 'func2' projects is picked up by the main 'motice_cl_demo' project then the 'motice_cl_demo' project is made dependent on both the 'func1' and 'func2' projects.

Project dependencies are set up in HEW via the 'Projects/Dependent Projects' menu. The screenshot below shows the project dependencies for 'motice_cl_demo.



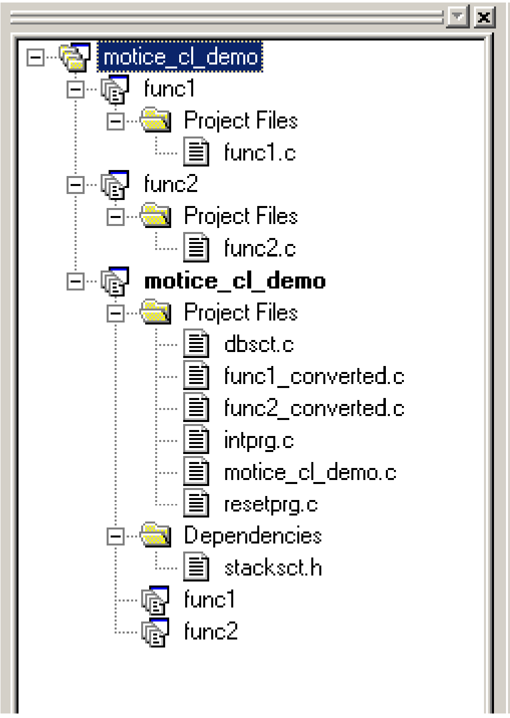
Now, whenever the 'motice_cl_demo' project is built both the 'func1' and 'func2' projects will be built first and so ensuring that very latest converted files are made available to 'motice_cl_demo.

Appendix 1 of this apps note shows the C source code for the files in the projects. In particular the file 'motice_cl_demo.c' may be of interest as it shows how to copy the program code from Flash to RAM using 'memcpy' and then call the copied functions via function pointers.

Summary

The utility 'motice_cl' can easily be integrated into HEW projects to provide the ability to relocate RAM based functions into Flash at build time for relocation and execution at runtime.

APPENDIX A : C SOURCE FILES FOR THE 'MOTICE_CL_DEMO' HEW WORKSPACE



```
Func1.c

// func1.c

unsigned long func1 (unsigned long ul)
{
   return ++ul;
}
```

```
Func2.c

// func2.c

unsigned long func2 (unsigned long ul)
{
   return ++ul;
}
```

```
Func1_converted.c

//
// This file has been generated by 'motice_cl' (v0.1a)
//
// FILE: func1_converted.c
// DATE: 24/7/2001
// TIME: 13:50:53


struct rom_data {
   unsigned long start_address;
   unsigned long data_length;
   unsigned char data[4];
};

const struct rom_data func1 = {
   0x00ffb000,
   0x00000004,
   0x0b, 0x70, 0x54, 0x70
};
```

```
Func2_converted.c

//
// This file has been generated by 'motice_cl' (v0.1a)
//
// FILE: func2_converted.c
// DATE: 24/7/2001
// TIME: 13:50:53


struct rom_data {
   unsigned long start_address;
   unsigned long data_length;
   unsigned char data[4];
};

const struct rom_data func2 = {
   0x00ffb100,
   0x00000004,
   0x0b, 0x70, 0x54, 0x70
};
```

```
Motice_cl_demo.c

/*********************************************************************/
/*                                                                   */
/*  FILE        :motice_cl_demo.c                                    */
/*  DATE        :Tue, Jul 24, 2001                                   */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE    :H8S/2633                                            */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/


#include <string.h>

struct rom_data {
   unsigned long start_address;
   unsigned long data_length;
   unsigned char data[0xffff];  // this value (0xffff) does not have to
        // reflect the true size of the data
        // but is required to satisfy the compiler
};

// externals
extern const struct rom_data func1;
extern const struct rom_data func2;

// function pointers
unsigned long (*ptr1) (unsigned long);
unsigned long (*ptr2) (unsigned long);


volatile unsigned long ul;

void main(void)
{
   // initialise the variable to be incremented
   ul = 0;

   // initialise function pointers
   ptr1 = (void *) func1.start_address;
   ptr2 = (void *) func2.start_address;

   // copy relocated functions to RAM
   memcpy ((void *) func1.start_address, func1.data, func1.data_length);
   memcpy ((void *) func2.start_address, func2.data, func2.data_length);

   while (1)
   {
    ul = ptr1 (ul);
    ul = ptr2 (ul);
   }
}
```

Website and Support

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

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