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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

H8/300L SLP Series

Fast MSB and LSB Rotate Method for 8 bit Data

Introduction

The serial communication interface present on some of the SLP devices is capable of both synchronous and asynchronous communication. When receiving byte information, it is sometimes necessary to rotate the received data from MSB first to LSB first and vice versa. This application note investigates methods of speeding up an LSB to MSB rotate.

Contents

Aim

The aim of this application note is to demonstrate a method for reducing the time taken by a H8 CPU to perform an 8 bit rotate from LSB first to MSB first. The code included in this note contains two methods for completing an 8 bit rotate. The first method utilizes a look up table; the second physically rotates the data from MSB first to LSB first.

1. 8 Bit Rotate Theory

This section gives a brief overview of each method used to perform the 8 bit rotate

Traditional 8 bit rotate

This method copies the received value into an 8 bit location. Bit seven of the received value is then placed in bit 0 of the copied value, then bit six into bit 1 etc. until finally bit 0 of the received value is placed in bit 7 of the copied value. A copied value is required as otherwise the bits of the received value would be overwritten by each other.

Figure shows a diagrammatic explanation of this method.

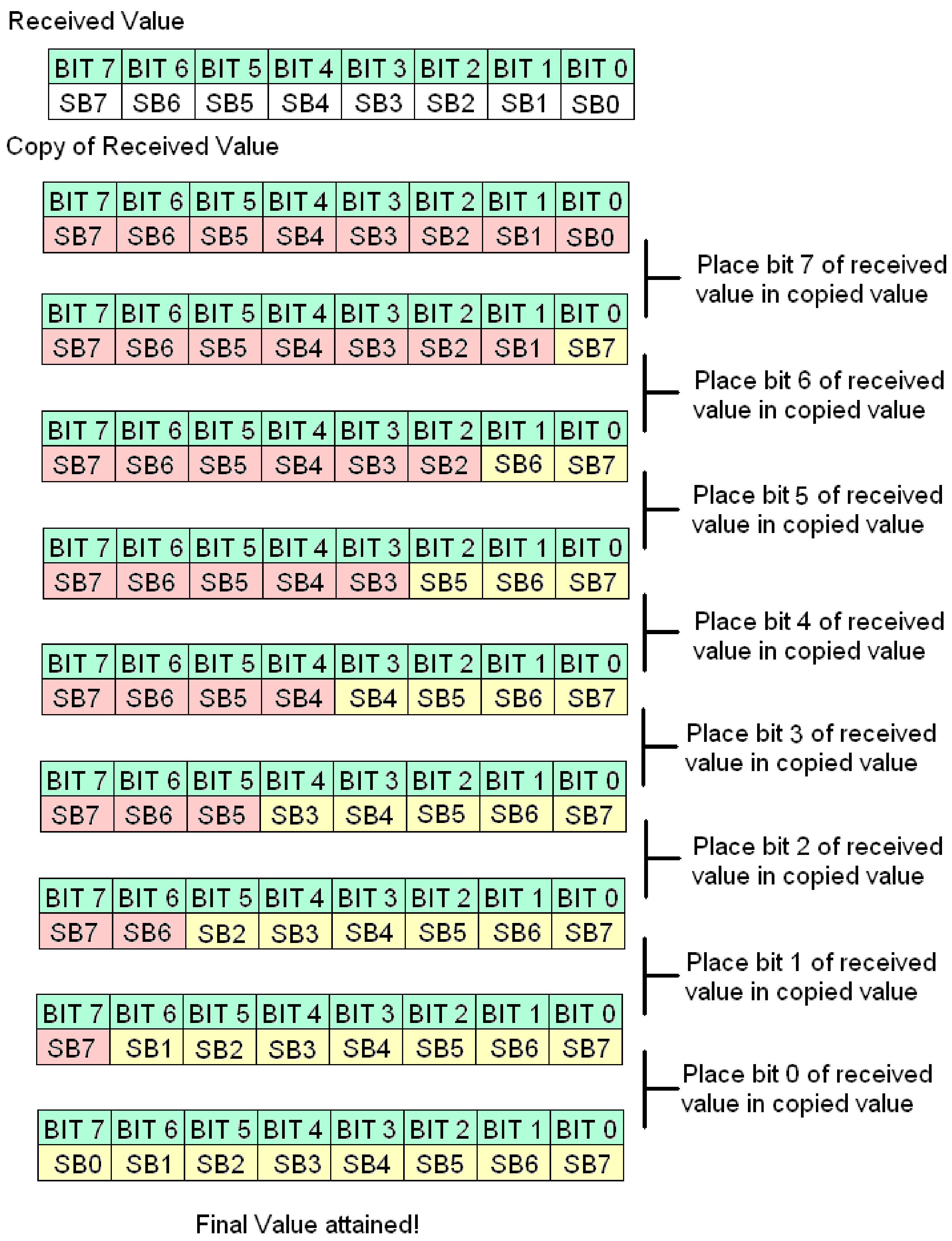


Figure 1: MSB and LSB rotate via traditional method

Look Up Table

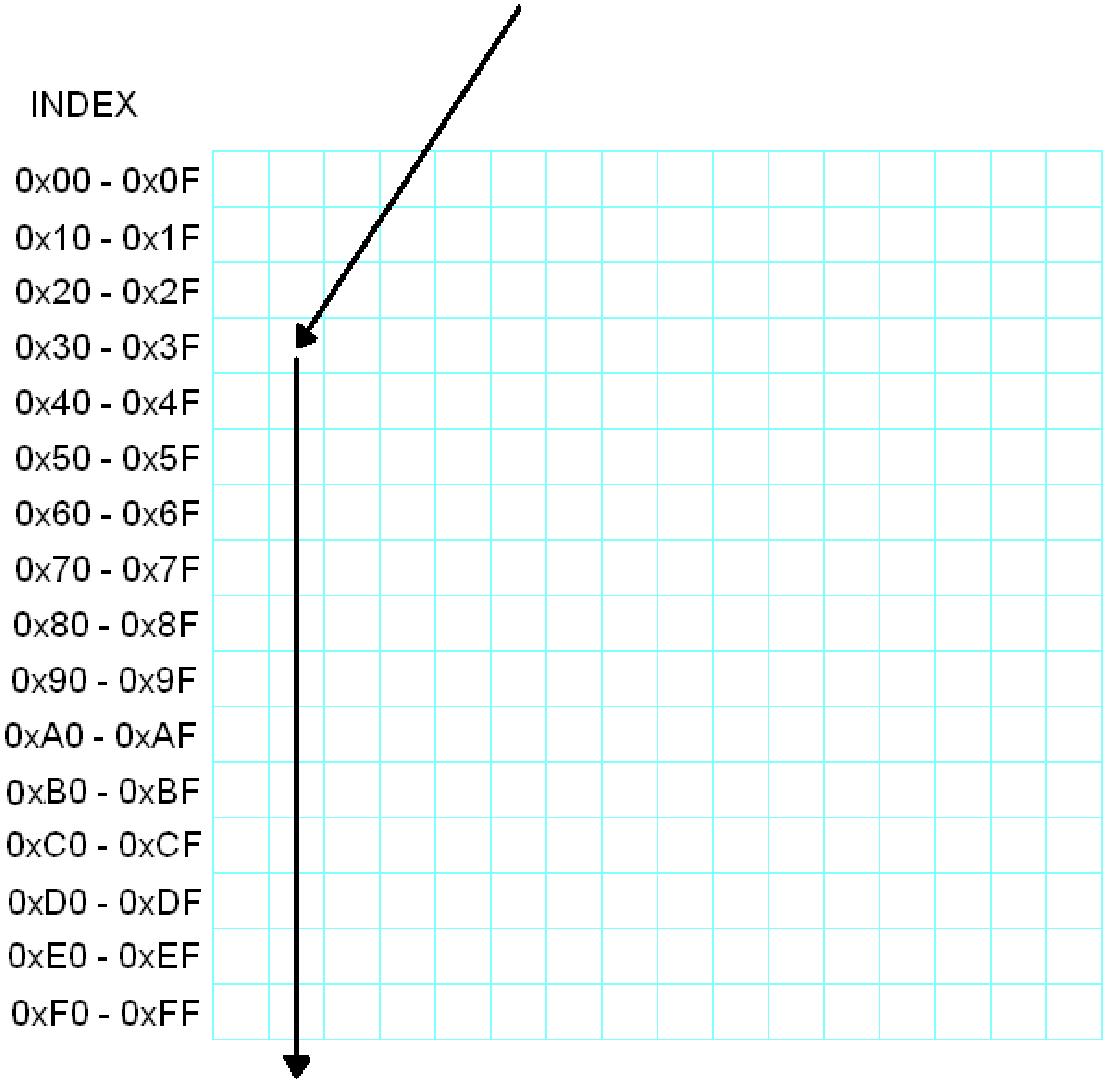
The look up table method involves a table of 258 values. The data within the table reflects pre-rotated MSB to LSB values. The first value in the table is the MSB to LSB rotated value of 0, the second is the rotated value of 1, the third the rotated value of 2, etc. This means that if the received 8 bit value is used as the index value in the look up table, then the value in the look up table at this location will give the received MSB to LSB rotated value. Figure 2 shows a graphical explanation of the look up table method.

INDEX	Table contains MSB and LSB rotated values of the index															
0x00 - 0x0F	0															F0
0x10 - 0x1F		88													78	
0x20 - 0x2F			44											B4		
0x30 - 0x3F				CC									3C			
0x40 - 0x4F					22							D2				
0x50 - 0x5F						AA					5A					
0x60 - 0x6F							66			96						
0x70 - 0x7F								EE	1E							
0x80 - 0x8F								E1	11							
0x90 - 0x9F							69			99						
0xA0 - 0xAF						A5					55					
0xB0 - 0xBF					2D							DD				
0xC0 - 0xCF				C3									33			
0xD0 - 0xDF			4B											BB		
0xE0 - 0xEF		87													77	
0xF0 - 0xFF	0F															FF

Figure 2: The look up table with example values

Received Value

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	= index
SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0	= 0x31
0	0	1	1	0	0	0	1	



BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SB0	SB1	SB2	SB3	SB4	SB5	SB6	SB7
1	0	0	0	1	1	0	0

Figure 3: The look up table method: An explanation

2. Workspace

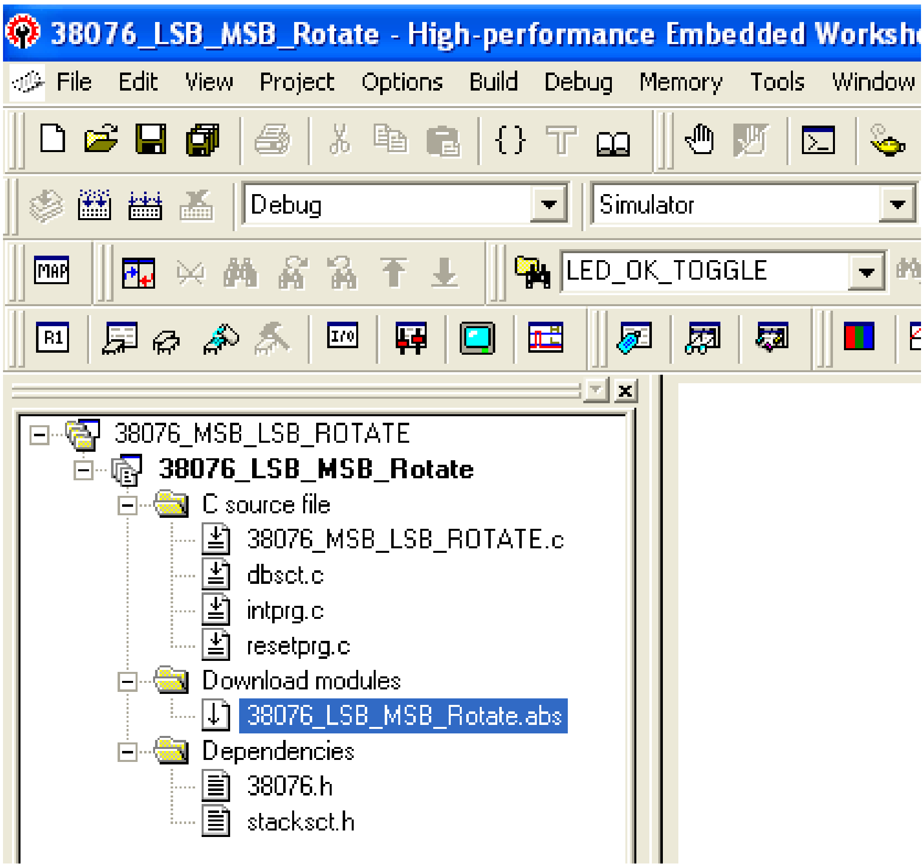


Figure 4: 38076_MSB_LSB rotate workspace view

There are four source files in the workspace; these being “38076_MSB_LSB_ROTATE.c”, “dbsct.c”, “intprg.c” and “resetprg.c”.

The first file contains the main function. dbsct.c is used to set up memory sections, intprg.c contains the interrupt vector table and resetprg.c calls the section initializing function “INITSCT()” as well as main().

The figure above also shows the build configuration “Debug” and the debug setting “Simulator”. The "simulator" session sets up the 300H simulator, which replicates the code running on a 300H CPU.

3. Code Description

“dbsct.c”: This sets up the sections of code in memory.

“intprg.c”: This sets up an interrupt vector table

“resetprg.c”: This contains code for the CPU to execute after a reset.

“38076_LSB_MSB_ROTATE.c”: This contains the main function and the two functions for performing the MSB and LSB rotate.

ResetPRG.c

```
#pragma section ResetPRG

__entry(vect=0) void PowerON_Reset(void)
{
     set_imask_ccr(1);
    _INITSCT();

//  _CALL_INIT();

//  _INIT_IOLIB();

//  errno=0;
//  srand(1);
//  _s1ptr=NULL;

//  HardwareSetup();
    set_imask_ccr(0);

    main();

//  _CLOSEALL();

//  _CALL_END();

    sleep();
}
```

The power on reset vector is defined. As soon as the device resets, the interrupt mask in the CCR register will be masked, and the INITSCT function is called. The interrupt mask bit is then cleared before the main function is called.

Main.c

```
 void main(void)
 {
  Swap_MSB_LSB();
  Look_Up_Table_Method();
     while(1);
 }
```

The main function contains two functions. The first Swap_MSB_LSB uses an 8 bit structure to shift the bits round one bit at a time (this is the technique described in the figure). The second function Look_Up_Table_Method uses a look up table method as described in the figure.

An infinite loop is then set up to prevent the device falling out of main function and executing non-code.

Swap_MSB_LSB Function

```
void Swap_MSB_LSB(void)
{
  volatile unsigned char Received_Value_SWAP = 0;
  volatile unsigned char Swapped_Value_SWAP = 0;
    for (Received Value SWAP = 0; Received_Value_SWAP
    < 0xFF; Received_Value_SWAP++)
    {
       RECEIVED.BYTE = Received_Value_SWAP;
       SWAPPED.BIT.b0 = RECEIVED.BIT.b7;
       SWAPPED.BIT.b1 = RECEIVED.BIT.b6;
       SWAPPED.BIT.b2 = RECEIVED.BIT.b5;
       SWAPPED.BIT.b3 = RECEIVED.BIT.b4;
       SWAPPED.BIT.b4 = RECEIVED.BIT.b3;
       SWAPPED.BIT.b5 = RECEIVED.BIT.b2;
       SWAPPED.BIT.b6 = RECEIVED.BIT.b1;
       SWAPPED.BIT.b7 = RECEIVED.BIT.b0;
       Swapped_Value_SWAP = SWAPPED.BYTE;
    }
}
```

Firstly two volatile variables are declared. The function then enters a loop which provides the loop variable "Received_Value_SWAP" with all possible values from 0 up to 0xFF. This means this variable at some point has had every possible value from 0x00 up to 0xFF. Within the loop, the Received_Value_SWAP variable is copied into an 8-bit structure.

```
union {
       unsigned char BYTE;
       struct {
                unsigned char b7:1; /* bit7 */
                unsigned char b6:1; /* bit6 */
                unsigned char b5:1; /* bit5 */
                unsigned char b4:1; /* bit4 */
                unsigned char b3:1; /* bit3 */
                unsigned char b2:1; /* bit2 */
                unsigned char b1:1; /* bit1 */
                unsigned char b0:1; /* bit0 */
               } BIT;
              }RECEIVED;
union {
       unsigned char BYTE;
       struct {
                unsigned char b7:1; /* bit7 */
                unsigned char b6:1; /* bit6 */
                unsigned char b5:1; /* bit5 */
                unsigned char b4:1; /* bit4 */
                unsigned char b3:1; /* bit3 */
                unsigned char b2:1; /* bit2 */
                unsigned char b1:1; /* bit1 */
                unsigned char b0:1; /* bit0 */
               } BIT;
              }SWAPPED;
```

The global variables SWAPPED and RECEIVED are 8 bit structures which are declared at the top of the 38076_MSB_LSB_ROTATE.c file. In the loop shown in the Swap_MSB_LSB function above, the last bit of the RECEIVED variable is placed in the first bit of the SWAPPED variable and so on until all the bits have been rotated. The variable SWAPPED then holds the swapped value of the RECEIVED variable.

Look_Up_Table_Method Function

This function uses the method described in figure. The look up table used for this method is shown below.

```
unsigned char Look_Up_Table[256] =
 {0x00, 0x80, 0x40, 0xC0, 0x20, 0xA0, 0x60, 0xE0, 0x10, 0x90, 0x50, 0xD0, 0x30, 0xB0, 0x70, 0xF0,
  0x08, 0x88, 0x48, 0xC8, 0x28, 0xA8, 0x68, 0xE8, 0x18, 0x98, 0x58, 0xD8, 0x38, 0xB8, 0x78, 0xF8,
  0x04, 0x84, 0x44, 0xC4, 0x24, 0xA4, 0x64, 0xE4, 0x14, 0x94, 0x54, 0xD4, 0x34, 0xB4, 0x74, 0xF4,
  0x0C, 0x8C, 0x4C, 0xCC, 0x2C, 0xAC, 0x6C, 0xEC, 0x1C, 0x9C, 0x5C, 0xDC, 0x3C, 0xBC, 0x7C, 0xFC,
  0x02, 0x82, 0x42, 0xC2, 0x22, 0xA2, 0x62, 0xE2, 0x12, 0x92, 0x52, 0xD2, 0x32, 0xB2, 0x72, 0xF2,
  0x0A, 0x8A, 0x4A, 0xCA, 0x2A, 0xAA, 0x6A, 0xEA, 0x1A, 0x9A, 0x5A, 0xDA, 0x3A, 0xBA, 0x7A, 0xFA,
  0x06, 0x86, 0x46, 0xC6, 0x26, 0xA6, 0x66, 0xE6, 0x16, 0x96, 0x56, 0xD6, 0x36, 0xB6, 0x76, 0xF6,
  0x0E, 0x8E, 0x4E, 0xCE, 0x2E, 0xAE, 0x6E, 0xEE, 0x1E, 0x9E, 0x5E, 0xDE, 0x3E, 0xBE, 0x7E, 0xFE,
  0x01, 0x81, 0x41, 0xC1, 0x21, 0xA1, 0x61, 0xE1, 0x11, 0x91, 0x51, 0xD1, 0x31, 0xB1, 0x71, 0xF1,
  0x09, 0x89, 0x49, 0xC9, 0x29, 0xA9, 0x69, 0xE9, 0x19, 0x99, 0x59, 0xD9, 0x39, 0xB9, 0x79, 0xF9,
  0x05, 0x85, 0x45, 0xC5, 0x25, 0xA5, 0x65, 0xE5, 0x15, 0x95, 0x55, 0xD5, 0x35, 0xB5, 0x75, 0xF5,
  0x0D, 0x8D, 0x4D, 0xCD, 0x2D, 0xAD, 0x6D, 0xED, 0x1D, 0x9D, 0x5D, 0xDD, 0x3D, 0xBD, 0x7D, 0xFD,
  0x03, 0x83, 0x43, 0xC3, 0x23, 0xA3, 0x63, 0xE3, 0x13, 0x93, 0x53, 0xD3, 0x33, 0xB3, 0x73, 0xF3,
  0x0B, 0x8B, 0x4B, 0xCB, 0x2B, 0xAB, 0x6B, 0xEB, 0x1B, 0x9B, 0x5B, 0xDB, 0x3B, 0xBB, 0x7B, 0xFB,
  0x07, 0x87, 0x47, 0xC7, 0x27, 0xA7, 0x67, 0xE7, 0x17, 0x97, 0x57, 0xD7, 0x37, 0xB7, 0x77, 0xF7,
  0x0F, 0x8F, 0x4F, 0xCF, 0x2F, 0xAF, 0x6F, 0xEF, 0x1F, 0x9F, 0x5F, 0xDF, 0x3F, 0xBF, 0x7F, 0xFF
 };
```

This table contains all the bit rotated values for 0x00, 0x01, 0x02 etc. all the way up to 0xFF.

```
void Look_Up_Table_Method(void)
{
  volatile unsigned char Received_Value_LU = 0;
  volatile unsigned char Swapped_Value_LU = 0;

    for (Received Value LU = 0; Received_Value_LU
    < 0xFF; Received_Value_LU++)
    {
        Swapped_Value_LU = Look_Up_Table[Received_Value_LU];
    }
}
```

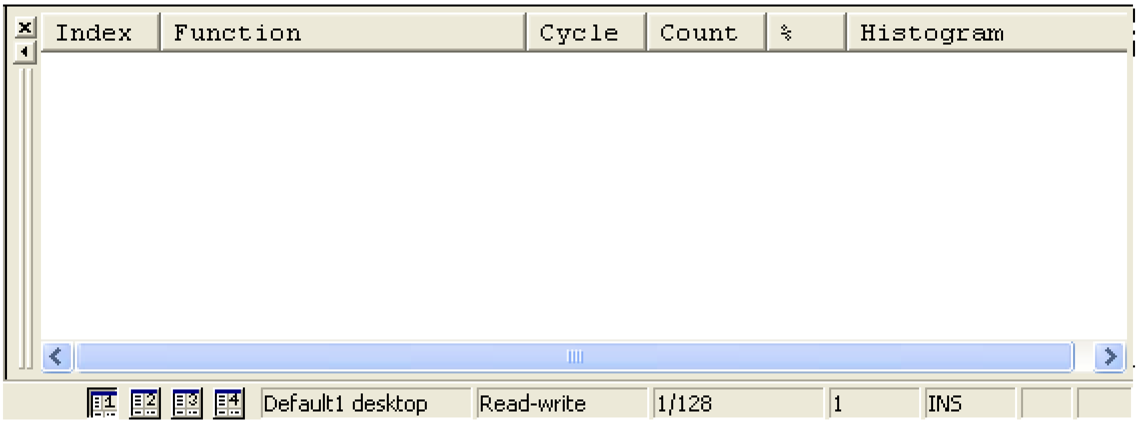
This again declares two volatile variables. One for declaring the value received and the other declaring a variable used to hold the complete result. Again a loop is set up which gives the Received value variable “Received_Value_LU” every value from 0x00 to 0xFF. The swapped value is attributed to that given by the look up table, when its index is the received value (see figure 3).

4. Performance Results

Speed

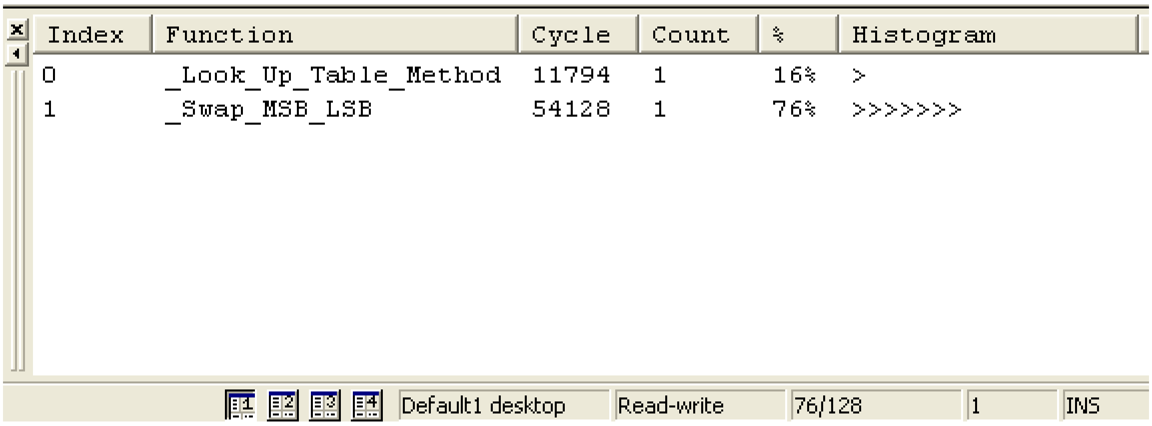
With the simulator session selected, the performance analyser was loaded by pressing the icon . The following window in figure will appear.

Figure 5: Performance Analysis Window



In the window, it is possible to right click on the mouse and select "add Range". The ranges Look_Up_Table_Method and Swap_MSB_LSB are added. The code is then run once. Figure 6 shows the performance results for running the two functions.

Figure 6: Performance Analysis window with results



Code Size

To determine the code size of both methods, the map file is used. This file is generated on a build and shows the code size in all the section types. To obtain the code size value for the traditional 8 bit rotate method, all code relating to the look up table method was commented out. The code size in the map file was then obtained. For the look up table method, all code related to the traditional rotate method was commented out, and the associated map file examined.

Table 1 shows the results

Code Segment	All rotate code commented out	Look up table code sizes	Traditional code sizes
PResetPRG	1A	1A	1A
PIntPRG	12C	12C	12C
P	5A	9C	112
C	0	100	0
C$DSEC	6	6	6
C$BSEC	4	4	4
D	0	0	0
B	0	0	2
R	0	0	0
S*	200	200	200
Total	3AA	4EC	464

* The S (STACK) size is defined by the user, and does not reflect either the actual stack size used, or the stack size required

Table 1: Code segment sizes for the rotate methods

The code size difference between the traditional rotate method and the look up table method is 0x88Bytes. This means the traditional method requires 89% of the extra space that the look up table method requires. The Look up table method requires less instruction code than the traditional LSB to MSB rotate method and hence it runs faster as it has less bytes in the P section. However, it requires an extra 0x100 bytes for the look up table in the constant data section.

Having no rotate code at all saves at least 0xBA Bytes of memory.

Conclusion

The Look up table method required 11794 cycles to execute for 256 values whereas the traditional swapping of the MSB and LSB method required 54128 for 256 values. The look up table method is therefore more than four times faster then the traditional method.

The look up table method requires 0x88Bytes of space more than the traditional rotate method. This is 12% of the extra space that the traditional method requires.

The 8 bit look up table method could be extended for 16 bit data by masking off the lower byte, using the upper byte in the look up table to establish the lower byte of the swapped word. The upper byte may then be masked, and the lower byte used in the look up table to find the upper byte value. The upper and lower bytes may then be added to give the final swapped value.

Website and Support

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

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