R8C Family, M16C Family

Simple Flash API for R8C, M16C, M32C, R32C

Introduction

A simple Application Program Interface (API) has been created to allow users of Flash based R8C, M16C, M32C and R32C devices to easily integrate reprogramming abilities into their applications using CPU Rewrite. CPU Rewrite is the term used to describe a Renesas MCU's ability to reprogram its own internal Flash memory while running in its normal operational mode.

Some devices in the M16C and M32C and family, as well as the entire R8C and R32C family, use a flash technology referred to as HND. This Flash technology allows a user to program the internal flash memory of the MCU. For M16C and M32C devices, data is programmed a WORD (16 bits) or a DOUBLE WORD (32-bits) at a time. For R8C devices, data is programmed a BYTE (8-bits) at a time. For R32C devices, data is programmed in QUAD WORD (64-bit) at a time. Since the flash technologies between many of the HND MCUs are so similar, a single API has been created that will accommodate multiple Renesas MCUs.

The API source files comply with Renesas' NC30 compilers only.

Target Devices

The following is a list of devices able to use this API:

R8C 1x Series: R8C/10, /11, /12, /13, /14, /15, /16, /17, /18, /19, /1A /1B

R8C 2x Series: R8C/20, /21, /22, /23, /24, /25, /26, /27, /28,
/29, /2A, /2B, /2C, /2D, /2E, /2F, /2G, /2H, /2J, /2K, /2L

R8C 3x Series: R8C/35A

M16C/10 Series: M16C/1N2

M16C/20 Series: M16C/26, /26A, /28, /29

M16C/60 Series: M16C/6N4, /6N5, /62P, /64

M32C/80 Series: M32C/83, /84, /85, /86, /87, /87A, /87B, /88

R32C/100 Series: R32C/111, /116, /117, /118

Contents

1. API Files 2

2. Configuring the API 2

3. API Functions 6

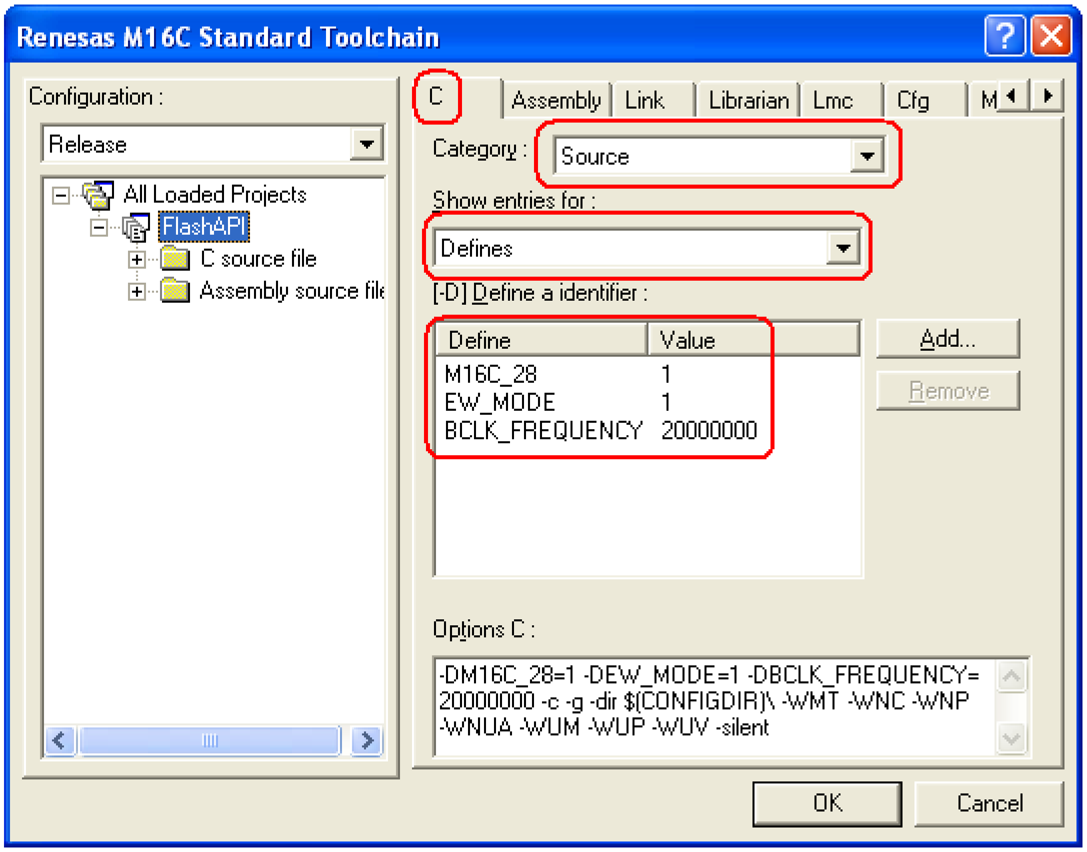
4. Example Code 8

1. API Files

The API files have been separated according to MCU Family. For example, for flash reprogramming of an M16C device, you would only need to add files “Flash_API_M16C.c” and “Flash_API_M16C.h” to your project.

2. Configuring the API

Before using the API, you must first configure the code so that it compiles for your specific device. All general settings can be done by modifying the Flash_API_xxx.h file. Alternatively, since the settings are controlled via #define directives, you may also set the configuration parameters via the command line when calling the NC30 compiler. This is done using the –D command line switch. You may also use the Hew environment to set up these parameters (see image below).



Configuration : Release

Category : Source

Show entries for : Defines

[-D] Define a identifier :

Define	Value
M16C_28	1
EW_MODE	1
BCLK_FREQUENCY	20000000

Options C :

```
-DM16C_28=1 -DEW_MODE=1 -DBCLK_FREQUENCY=
20000000 -c -g -dir $(CONFIGDIR)\ -WMT -WNC -WNP
-WNUA -WUM -WUP -WUV -silent
```

Figure 1. Setting Flash API Options within HEW

2.1 Selecting your intended device

A list of supported devices are located near the top of the Flash_API_xxx.h file. Please be sure you find your device on this list. Devices are listed by their Renesas “Group” name. For example, a M30626FHPGP device is part of the M16C/62P Group. Therefore, you would remove the comment markers (//) from the line defining “M16C_62P” to be “1”. This will produce source code compatible with all the devices within the M16C/62P Group.

For example, line….

```
  /*******************
   * SELECT YOUR MCU *
   *******************/
//#define M16C_62P 1
```

…would become…

```
  /*******************
   * SELECT YOUR MCU *
   *******************/
#define M16C_62P 1
```

...and now all devices in the M16C/62P Group will be supported.

2.2 Specifying your system clock speed

The HND Flash memory controller for these devices can only be access using an internal bus clock speed lower than its maximum rated operating frequency (usually 6.25MHz or 5MHz). Therefore, the API must know your system's running bus clock speed (also referred to in the spec as "BCLK") in order to modify the speed of the MCU while communicating with the flash controller. If your system clock settings need to be modified for flash programming, they will be saved and resorted automatically by the API functions.

The value of BCLK_FREQUENCY needs to be specified in Hertz. Therefore, you would remove the comment markers (//) from the beginning of the line defining "BCLK_FREQUENCY" and set the number after it to your system's running speed. For example, if your system ran at 24MHz, you would specify…

```
  #define BCLK_FREQUENCY 24000000
```

For the R32C family only, value of BCLK_FREQUENCY specifies the 'Base Clock' frequency in MHz.

NOTE: For devices with an external memory address bus, the API automatically puts the device into Single Chip Mode if using EW1 Flash Mode as required by the device spec. The API functions automatically restore the previous processor mode settings if they were needed to be changed.

NOTE: If using an R8C device (excluding R8C/10, /11), then you do not need to specify the system clock frequency when using EW1 Mode. This is because the newer

NOTE: For the R32C family, additional care to the bus control registers in the 'AdjustFlashBusRegister' function may be needed.

2.3 Specifying Flash Mode Type

You may select to use Erase-Write Mode 0 (EW0) or Erase-Write Mode 1 (EW1) for sending commands to the flash sequencer. Note that EW0 mode executes the flash commands to RAM and waits while the MCU polls for the flash operation to complete. EW1 mode sends commands while executing out of flash, but the MCU automatically goes into a HOLD state (does not execute any code) until the flash operation is complete. Therefore, EW1 mode requires less RAM and ROM to execute. On the other hand, if your system has a watchdog timer, you *may* (depending on the device you select) need to use EW0 mode in order to continually "kick" the watch dogs while waiting for the flash operation to complete. Therefore, do not use EW1 mode with a watchdog timer on the R8C Family of devices.

To specify which mode to use, remove the comments markers "//" from the front of one of the lines below.

```
  /*********************************
   * SELECT EW0 or EW1 Flash MODE *
   *********************************/
  //#define EW_MODE 0
  //#define EW_MODE 1
```

2.4 Specifying EW0 Mode Options

This option only applies when using EW0 Mode programming.

If you choose to use EW0 Mode Flash programming, there are two other settings. The first is used to specify how much RAM memory needs to be allocated to hold the executable code that will send commands to the flash controller. This value is defined as RAM_CODE_SIZE and is expressed in bytes. This number is used for both the Erase and Write routines. Please note that if you would like to use the lowest possible RAM space, base your RAM usage on the FlashWrite routine because it uses the most RAM.

```
#define RAM_CODE_SIZE 120
```

The other option specifies where the RAM code will be allocated and how. This is expressed using the option STATIC_RAM_CODE.

```
#define STATIC_RAM_CODE 1
```

If STATIC_RAM_CODE is defined as a ‘1’, the array that is allocated for the flash functions will always be resident in RAM. If the value of STATIC_RAM_CODE is ‘0’, the RAM space for the code will be allocated on the stack when an API function is called.

NOTE: If not enough RAM space is allocated to hold the code (especially if you’ve added some of your own code to the routine), the API functions will return with an error. At that point, you can look at the global variable “reflash_code_size” to determine how much to increase you “RAM_CODE_SIZE” setting.

2.5 R32C/11x Clock settings

The R32C Flash code is currently set for a maximum Base Clock frequency of 50MHz and a peripheral clock divide of 2. Running a Base Clock frequency slower than 50MHz will still enable the code to work, but the Flash operations may be slower than could be achieved if the Flash registers settings in the *AdjustFlashBusRegister* function in the Flash_API_R32C.c file. Also, if the Peripheral Divide Clock settings are set to anything other than divide-by-2, please change the setting in the *AdjustFlashBusRegister* function in the Flash_API_R32C.c file.

If you are interested in how to set the ‘Flash Memory Rewrite Bus Control Register’ (FEBC) for the R32C/11x series, the following instructions explain the process.

When looking at the Electrical Characteristics chapter for the correct timings for Rewrite Mode Flash accesses, the ‘Read cycle time’ and ‘Write cycle time’ values are not your only concern when setting up the FEBC. You also need to concern yourself with the setup and hold time values as well as the read/write pulse width values.

While the settings for read accesses are using bits FWR0-4 in the FEBC register, and the settings for write accesses are set using bits FSUW0/1 and FWW0/1 in the FEBC register, both read and write settings share a common bit setting of MPY0/1 also in the FEBC register. Therefore when looking up the correct values in the Rewrite Bus Timing charts (read/write), please note that your MPY0/1 settings will have to be the same for reads/writes. The MPY0/1 setting specifies by how much you want to divide (either ÷2 or ÷3) down the current peripheral bus clock in order to use for your rewrite bus clock. Remember, the rewrite bus is always a division of the peripheral clock (which in itself is always a division of the Base clock).

The times in the Electrical Characteristics are in time (ns), but the values in the chart for register setting the FEBC register is in Base Clock cycle counts. Therefore you first need to convert the units from time to cycles.

For example, if the ‘*Read cycle time*’ (t_{CR}) value in the ‘Flash Memory CPU Rewrite Mode Timing‘ table in the Electrical Characteristics chapter is 200ns, and your Base Clock is set to run at 50MHz, then your *read cycle time* value that you need for your FEBC register setting is 50MHz * 0.2µs =10. (NOTE we converted 200ns to 0.2µs to match powers of $10E^6$ to $10E^{-6}$). So this result means it will take 10 Base Clock cycles in order to satisfy the minimum read cycles time needed for Flash Read accesses.

We then take that value of 10 and look it up in our ‘Read Cycle and Bit Settings’ chart for the FEBC register. We look in the ‘t_{CR}’ column under either ‘*mpy* = 3’ or ‘*mpy* = 4’ (keeping in minded that we will have to use the same *mpy* value for the ‘Write Cycle and Bit Settings’ chart) and find a row that will has a ‘t_{CR}’ value of 10 or higher. We must then repeat the same process for the ‘*Chip-select setup time for read’* ($t_{su(S-R)}$) value, the ‘*Address setup time for read*’ ($t_{su(A-R)}$) value, the ‘*Read pulse width*’ ($t_{w(R)}$) value, the ‘*Chip-select hold time after read*’ ($t_{h(R-S)}$) and the ‘Address hold time after read’ ($t_{h(R-S)}$) value.

After you have all these values converted to Base Clock cycle times, find a row that satisfies all minimum values. When looking for a compatible row, keep in mind that the lower in the chart you go, the slower the flash access and overall operation time you will have. Therefore it’s desired to find the highest row in the chart. Once you have your row, you will then have all you bit settings for the FEBC register concerning Read Access timing.

Now that your read access timing is done, do the same procedures for the write access timing using the ‘Write Cycle and Bit Settings’ chart for the FEBC register. Keep in mind that your *mpy* bit setting has to be the same for both Read and Write accesses.

3. API Functions

3.1 FlashErase

Format

```
unsigned char FlashErase( unsigned char block );
```

Parameters

block

Specifies the block to erase. This value is defined in the Flash_API_xxx.h file. The blocks are labeled in the same fashion as they are in the device specifications. For example, the highest address block is labeled '"BLOCK_0" and a Data Block A is define as "BLOCK_A".

Return Values

Returns the outcome of the erase operation:

0: Erase successful
1: Erase error reported by flash control
2: Command Sequence Error (locked block, incorrect command, etc)
3: Not enough RAM space allocated for re-write code (EW0 Mode only)

Properties

Prototyped in file "Flash_API_xxx.h"
Implemented in file "Flash_API_xxx.c"

Description

Erases a single block of Flash memory. The block may be either "Data Flash" block or "User Flash" block.

NOTE: Do not attempt to erase a flash block that you are currently executing out of.

NOTE: If you are using a watchdog timer, beware that erase operations may take longer than the watchdog timeout. You may need to use EW0 mode on some devices so you can "kick" the watchdog while you are waiting for the Flash operation to complete. Consult your device's specific hardware manual for more information on how the watchdog timer is handled during Flash operations for your particular device.

3.2 FlashWrite

This function allows data to be written into Flash.

Format

```
   unsigned char FlashWrite( FLASH_PTR_TYPE  flash_addr,
                             BUF_PTR_TYPE    buffer_addr,
                             unsigned int    bytes);
```

Parameters

flash_addr

This is a pointer to the Flash area to write. The value of FLASH_PTR_TYPE is defined in Flash_API_xxx.h because it varies depended on what device you have selected. See *Description* below for important restrictions regarding this parameter.

buffer_addr

This is a pointer to the buffer containing the data to write to Flash. The value of BUF_PTR_TYPE is defined in Flash_API_xxx.h because it varies depended on what device you have selected. This address can be either a location in RAM or Flash.

bytes

The number of bytes contained in the *buffer_addr* buffer. See *Description* below for important restrictions regarding this parameter.

Return Values

Returns the outcome of the write operation:

0 = Operation Successful
1 = Write Error reported by flash control register
2 = Invalid parameter passed
3 = Command Sequence Error (locked block, incorrect command, etc)
4 = Not enough RAM space allocated for re-write code (EW0 Mode only)

Properties

Prototyped in file "Flash_API_xxx.h"
Implemented in file "Flash_API_xxx.c"

Description

Writes data to flash memory. The destination location may be either in "Data Flash" or "User Flash".

If the device is an **M16C or M32C**, the number of bytes to write MUST be an even number because the flash controller has to write a WORD (16 bits) at a time. The flash destination address MUST also be an even number as well because the flash controller needs to write WORDS to even addresses only.

If the device is an **M16C/64**, then the *bytes* must be a multiple of 4 because it has to write a DWORD (32 bits) at a time. Also, the flash destination address MUST be on a 4-byte address boundary. This mean you can only write to address that end in 0, 4, 8 or C. Example: 0xF000, 0xF004, 0xF0008, 0xF00C.

If the device is an **R32C**, then the *bytes* must be a multiple of 8 because it has to write a QWORD (64 bits) at a time. Also, the flash destination address MUST be on a 8-byte address boundary. This mean you can only write to address that end in 0 or 8. Example: 0xF000, 0xF008, 0xF0010, 0xF018.

If the device is a **R8C**, then both the destination flash address and number of bytes may be either even or odd.

4. Example Code

The following is an example of using the API in order to erase and program a Data Flash block A in a M16C device.

```
#include "Flash_API_M16C.h"

char write_buffer[12] = "Hello World";
void main(void)
{
    unsigned char success;

    /* Erase Data Block A */
    success = FlashErase( BLOCK_A );

    /* Write our data buffer into Flash.
       Note that we cast the hard coded flash address and buffer address
       pointer to make the compiler happy. */
    success = FlashWrite( (FLASH_PTR_TYPE) 0xF000,
                          (BUF_PTR_TYPE)   write_buffer,
                                           12);

    while(1);    /* END OF DEMO */
}
```

Website and Support

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

Renesas Technology America Website
http://america.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com (Global Support)
TechSupport.rta@renesas.com (United States / Canada / Mexico only)

Revision Record

Rev.	Date	Description	
		Page	**Summary**
1.00	Apr.01.04	—	First edition issued
2.00	May.14.04	1 to 6	Second edition issued
2.10	Dec.03.07	1 to 9	Changed to latest app note template.
2.11	Dec.03.08	1,3,6	Added support for R8C/35A and R32C/11x and added new section 2.5.
2.12	May.15.09	1	Changed app note title
2.13	Nov.20.09	-	Modification of source file 'Flash_API_R8C.c' only