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.

R8C Family

How to Implement MR8C/4 in Embedded Applications

Introduction

MR8C/4 is an embedded real-time operating system (RTOS) that use preemptive priority-based task switching like most RTOS. This ability to run multiple tasks brings new complications. In addition, performance of an RTOS in an embedded application is highly dependent on the design considerations in its implementation. Switching from a “one-big-loop” style of programming to a multithreaded RTOS thus require proper planning and execution.

This document discuss on the steps and considerations to be taken in the process of incorporating MR8C/4 in an embedded application.

Target Device

Applicable MCU: R8C Family

Contents

1. Guide in using this Document

Implementation of a system using an RTOS requires calculation and planning. The designer needs to consider all the timing aspects of the system. In addition, designer will need to consider task partitioning, task prioritization, use of interrupts and hardware device capabilities in the implementation.

This document aims to explain how to implement a system using MR8C/4.

Table 1 Explanation of Document Topics

Topic	Objective	Pre-requisite
Design Considerations	Highlighting probable considerations to be made in the designing stage of incorporating an RTOS.	MR8C/4 and MR30/4
MR8C/4 Implementation	An explanation of the steps involved in the implementation of MR8C/4 in an embedded application	MR8C/4 and RTOS Scheduling Algorithms and Protocols
Example: Implementing MR8C/4 in “Watch_RSKR8C_Demo” Application	A step by step illustration of implementing MR8C/4 in the demo application	MR8C/4
Reference Documents	Listing of documents that equip users with knowledge in the pre-requisite requirements	None

2. Design Considerations

The main objective of incorporating an RTOS into an application is to improve the real-time response to external events by ensuring predictability without consuming much of the embedded systems' limited resources.

2.1 Preemptive performance

To ensure that external events trigger the corresponding operational responses of a device, it is crucial to assign the operations accurately to the designated task. To achieve an accurate real-time response, tasks partitioning and dispatching need to be correctly carried out.

MR8C/4 is an RTOS that employ preemptive priority based scheduling where tasks are executed in the order of their priority. Therefore, to ensure the preemptive performance of the embedded application, assignment of appropriate priority level to the corresponding tasks is of utmost importance.

Inter-task synchronization is another important factor contributing to the preemptive performance of the embedded application. Inappropriate inter-task synchronization management might result in priority inversion, race conditions, starvation or deadlock that will deter the responsiveness of the system.

Interrupt latency introduced by RTOS is another deterrent factor. RTOS might not be suitable for embedded applications that require the engagement of multiple high frequency interrupts.

2.2 Extra memory consumption

Extra ROM/RAM is required for the RTOS code and associated data structures (e.g. Data Queue module). Additionally, each task requires extra RAM for its own private stack space. In such situation, size of individual tasks plays a critical role in the total memory consumption. User may perform computation of the required RAM size of each task and allocate the appropriate RAM size to the respective tasks to reduce wastage.

2.3 Power consumption

For an embedded device, the hardware design and the RTOS play important roles in ensuring that power consumption is reduced. If the RTOS is well implemented, memory use can be minimized, and less power will be consumed. An efficient embedded application with RTOS can make a lower-specification CPU viable and reduce power even further.

3. MR8C/4 Implementation

The whole cycle of implementing MR8C/4 encompasses the feasibility analysis by understanding the RTOS and hardware device involved to ensuring an effective deployment through proper partitioning, prioritization of tasks, inter-task communications and interrupts mapping. Figure 1 depicts the entire process of implementing MR8C/4.

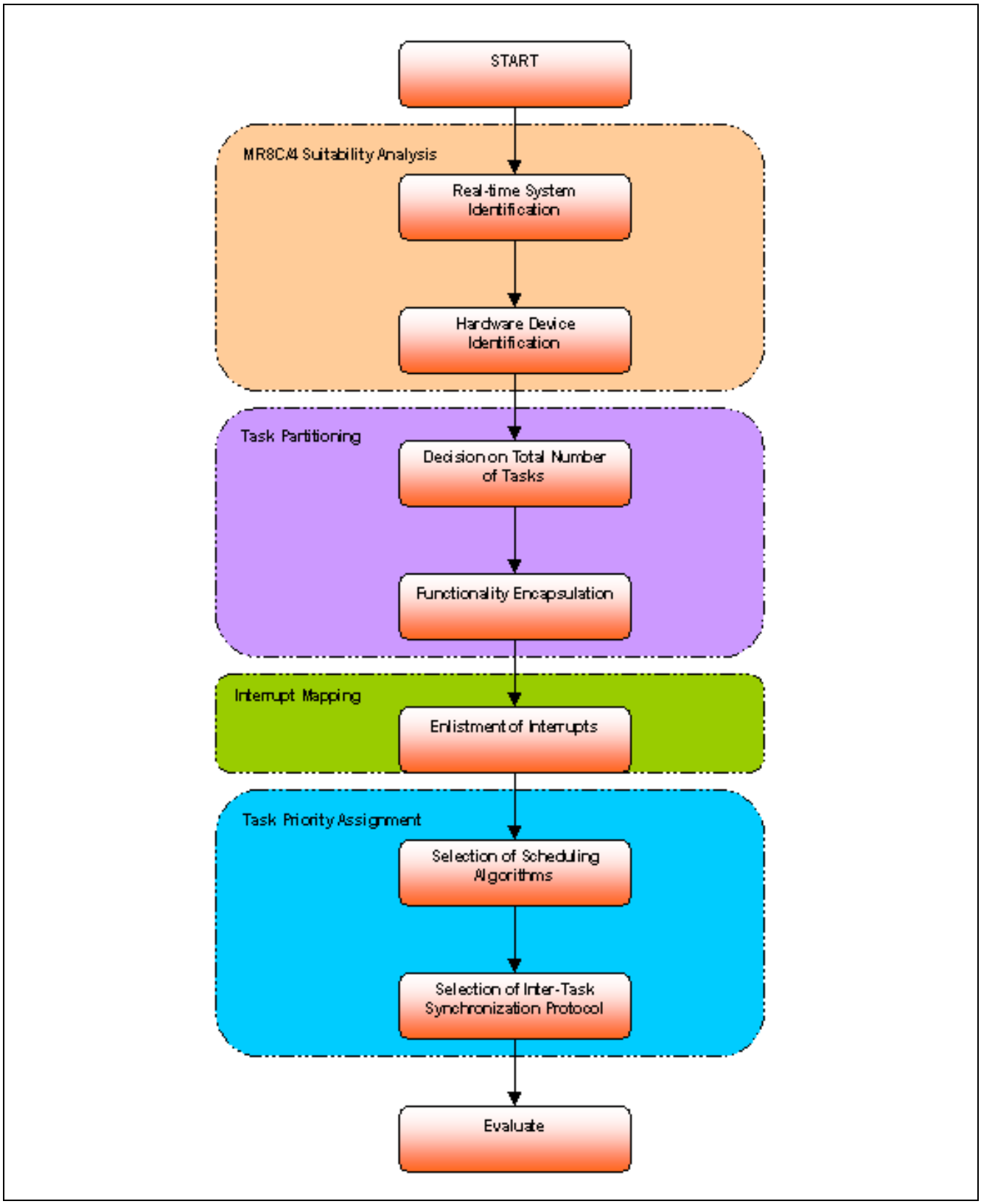

Figure 1 MR8C/4 Implementation Procedures

3.1 MR8C/4 Suitability Analysis

A clear understanding of the objective/s of an embedded system and the choice of hardware device to be utilized are important in determining the relevance and suitability of implementing it with MR8C/4. The key considerations are the available timer period, inter-task communication methods, contention resolution, memory protection and capabilities of handling the set of tasks.

3.1.1 Real-time System Identification

A real-time system is one, which must satisfy bounded response-time constraints or risk severe consequences, including failure. Real-time system may be classified as hard, firm or soft systems. And such identification is crucial in the designing of RTOS implementation in an embedded application involving processes such as tasks partitioning, task scheduling, inter-task synchronization methods and etc.

In hard real-time systems, failure to response to time constraint requirements leads to catastrophic results for the system.

Firm real-time systems entail an unacceptable quality reduction when a deadline is missed. Systems whose deadlines may be missed and can be recovered from with acceptable reduction in quality are called soft real-time systems.

MR8C/4 is an evolution from uITRON4.0 that supports hard-real time systems. As such, MR8C/4 can be implemented in firm or soft-real time systems as well.

3.1.2 Hardware Device Identification

Understanding the functional capabilities and peripherals available in a device is equally important in the decision, designing and implementation of an RTOS. An RTOS is limited by the hardware it runs on. MR8C/4, being specially designed and fully compatible with R8C family devices. Understanding the full features of the chosen R8C device will aid designer in the allocation of MR8C/4 resource to fulfill the application requirements of the system. At the same time, designer may review the feasibility of an implementation based on the available resources of MR8C/4 and the device. Figure 2 provides a non-exhaustive list of R8C family devices features.

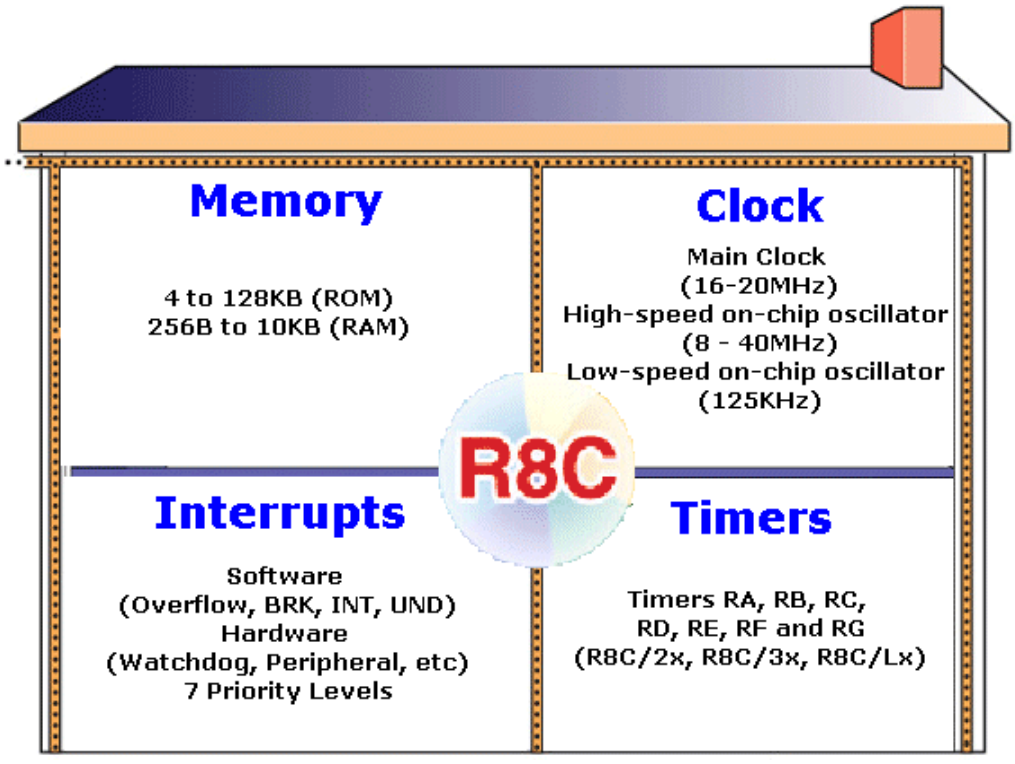

Figure 2 R8C Family Devices Features

3.2 Tasks Partitioning

This process involves decomposing an embedded application into small, schedulable, and sequential program units called *Tasks* to optimize the handling of inputs and outputs within set time constraints. A task is a unit of concurrency, but it is not the smallest unit of software architectural design. Most tasks contain a number of functions that are executed sequentially within the task.

In embedded systems, individual task typically provide the following services:

- Handling a single asynchronous input or output device
- Executing the software that must meet a single time deadline
- Performing a large calculation
- Maintaining a large data store
- Executing software that must run at a single point in time
- Executing software that must run periodically

This section discusses on the considerations when performing tasks partitioning.

3.2.1 Decision on total number of tasks

The number of tasks to create is highly dependent on the requirements of the system. MR8C/4 allows declaration of up to 255 tasks. However, it will be unwise to declare all 255 tasks unless required. The decision on the total number of tasks to be created therefore based on a leverage of what is needed versus the amount of trade-off in memory consumption that can be tolerated.

Table 1 explores the pros and cons of defining more tasks.

Table 1 Pros & Cons of defining more Tasks

No.	Pros	Cons
1	Better overall response time control	Require better data protection with more intertask communications
2	Modularity with individual task assigned with specific functionality	Long response time due to associated overheads
3	Better encapsulation of data and functionality within individual task	More memory required for increase in task stack size

3.2.2 Functionality encapsulation in tasks

Functionality encapsulation involves the classification of tasks based on functions of embedded system. The segregation approach is done by classifying functions into asynchronous, synchronous and repetitive tasks using desynchronous, synchronous and architectural methods respectively.

Desynchronous method identifies functions that operate asynchronously to one another, classify them into single tasks and prioritize them (refer to Section 3.4 for priotization guidelines). Few common asynchronization function variations include:

- User interface function
- Millisecond function
- CPU hogging function

Synchronous method involves identifying functions that performs in synchronization with other functions. Synchronous tasks usually offer services of shared resources to other asynchronous tasks. Few synchronous functions include:

- Error logging function
- Data control function

Architectural method identifies periodic and state machine functions and classifies them into single tasks. Periodic tasks operate on a regular interval basis. Whereas a state machine tasks operates in accordance to external events triggered by other tasks. Examples of periodic and state machine functions include:

- Monitoring function
- Display function

3.3 Interrupts Mapping

Interrupts latency will be introduced with implementation of RTOS. To reduce the amount of latency and avoid scheduling conflict, following guidelines may be exercised.

- Keep ISRs short and simple (to minimize interrupt response time, testing and debugging time)
- Disable preemption before calling interrupt service routine function (to enable completion of interrupt service routine)
- Assign only urgent operation to interrupts and use task to perform bulk of interrupt service (to minimize amount of overheads due to interrupts)
- Avoid disabling interrupts (to prevent delays in handling high priority interrupts by kernel
- Avoid implementing high-latency instructions in ISRs (mathematical computation like division and string manipulations may take many cycles to execute and thus increase latency)
- Avoid improper usage of RTOS APIs in ISRs (to prevent occurrence of possible nondeterministic nature of APIs that hog the ISRs).

3.4 Task Priority Assignment

Selection of scheduling algorithm and inter-task synchronization protocol defines the assignment of priority level to individual task.

3.4.1 Selection of Scheduling Algorithms

Scheduling algorithms define whether individual tasks of the system will be able to meet their deadlines without incurring priority inversion, deadlocks, race conditions and starvations.

There are generally two approaches namely, fixed/static priority scheduling and dynamic priority scheduling as shown in Figure 3.

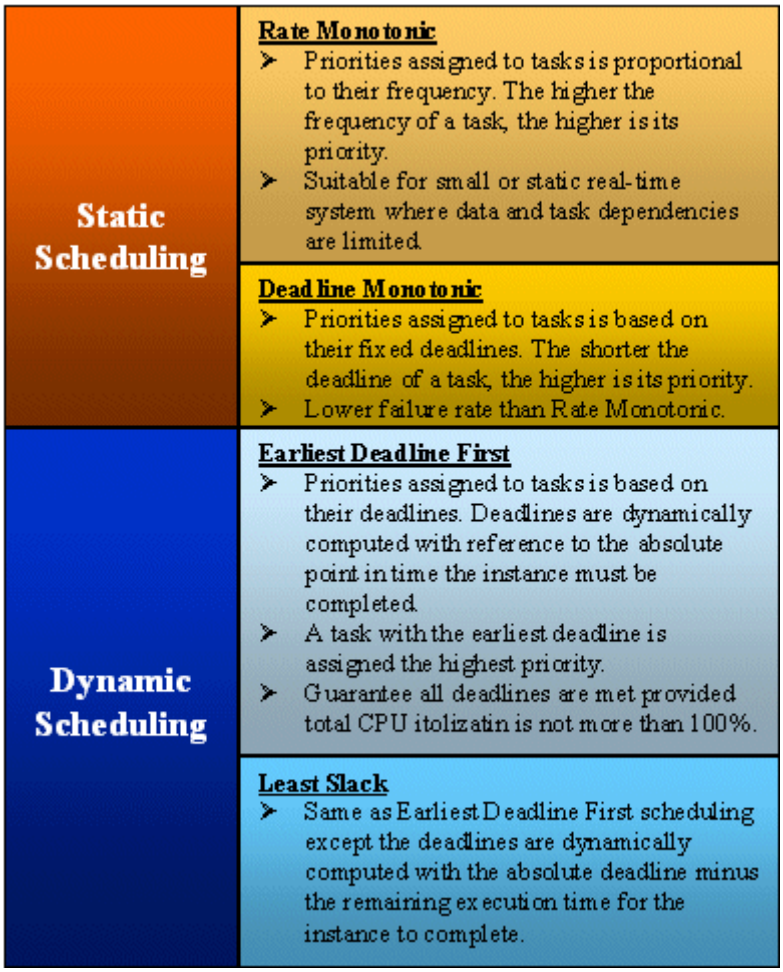

Figure 3 Static and Dynamic Scheduling Algorithms

In static scheduling algorithms, priority of a task will not be modified when it is running unless the task changes its own priority. This approach can be implemented more easily and scheduler is fast and more predictable. Dynamic scheduling algorithms allow a task priority to be modified when it's running. This approach is more dynamic but considerably more complicated.

3.4.2 Selection of Inter-task Synchronization Protocol

Both the static and dynamic scheduling algorithms might not be sufficient to ensure no occurrence of priority inversion and deadlocks in inter-task synchronization of resource. As such, additional scheduling protocols may be deployed. Few of the protocols include:

- Critical Section

Critical Section method directs the locking of the scheduler when a resource is accessed by making the current task the highest priority task in the system. This prevents another task from simultaneously accessing it.

The advantage of this method is its simplicity, partially in terms of understandability and implementation. The downside lies with the possibility of incurring tasks blockage when the resource access take too long.

This method is suitable for embedded application that do not require the simultaneous sharing of resources among tasks.

- Priority Ceiling

Priority Ceiling method impose a priority ceiling for each resources and in turn escalate the priority level of the task holding the resource to the same level as another task that has been blocked from accessing the same resource or resource with the same priority ceiling.

This method helps to resolve the issues of priority inversion and deadlock but is more complex for implementation and incur larger overhead.

This method is more suitable for more complex embedded applications that require better control in the elimination of deadlocks and inversion priority.

- Simultaneous Locking

Simultaneous Locking method is device to solely avoid deadlock avoidance by locking or unlocking all resources needed at any one instance.

This method prevents deadlock without resolving priority inversion. This method incurs large computational overhead that may become severe when there are many shared resources.

This method is more suitable for embedded applications that resulting in deadlock situation where some resources might be blocked while waiting for others to become available.

- Ordered Locking

Ordered Locking method eliminates deadlock by ordering resources and enforcing a policy which resources must be allocated only in a specific order.

This method effectively removes any possible deadlocks but impose significantly more complex algorithm and does not prevent priority inversion.

- Priority Inheritance

Priority Inheritance method introduces the manipulation of executing priorities of tasks that lock resources. This method aims to reduce priority inversion by limiting to; at most, a task will only be blocked by a single, lower-priority task owning a needed resource.

This method is highly effective in handling the problem of priority inversion when at most a single resource is locked at any given time. However, multiple tasks blockage termed *chain blocking* will occur when there are multiple resources that may be locked at any time. In addition, this method does not address deadlock and incur larger overhead.

- Highest Locker

Highest Locker method defines a priority ceiling for each resource where the task owning the resource runs at the highest priority ceiling of all the resources that it owns at the instance.

This method has better priority inversion bounding properties than Priority Inheritance method and avoids deadlock. The disadvantage of this method will incur larger overhead and its ability to bound priority inversion only to a single level.

4. Example: Implementing MR8C/4 in "Watch_RSKR8C_Demo" Application

4.1 Step 1: MR8C/4 Suitability Analysis

Objective of the project is to develop a digital watch that provides timing display, alarm setting, and stopwatch functions. Figure 4 provides a comparison of the feasibility of incorporating MR8C/4 in the project utilizing R8C/25 hardware device.

Items	Project Specifications	MR8C/4	R8C/25	Compatibility
Real-System Type	Soft	Soft,Firm & Hard	NA	Full
Response Time	1s clock update	19µs task switching (@20MHz)	20MHz high speed clock (50ns interval)	Full
	50ms inter-response interval for function selection (watch, alarm or stopwatch)			
	10ms update for stop watch counting			
Number of task	~10	255 (total)	NA	Full
Memory Size	~3489Bytes (CODE ROM)	1.5Kbytes to 6.5Kbytes (Kernel ROM)	32Kbytes (ROM)	Full
	~357Bytes (ROMDATA)			
	~1075Bytes (RAMDATA)	~400Bytes (STACK)	2Kbytes (RAM)	

Figure 4 Suitability Analysis of MR8C/4 Implementation in "Watch_RSKR8C_Demo"

From Figure 4, we may deduce it is suitable to implement MR8C/4 in the embedded application "Watch_RSKR8C_Demo".

4.2 Step 2: Task Partitioning

The next step is to list out the main functions and classify them respectively into single tasks. Figure 5 shows the listing of main functions in "Watch_RSKR8C_Demo" application program. The functions are analyzed and classified into the respective task types and groups.

Function Classifications				
No.	Function List	Purpose	Task Type	Task Group
1	main	To perform the startup of the digital watch	Startup	Asynchronous
2	determine_mode_function	To determine the mode (time display, alarm setting, stop watch or turn on the LEDs) user wish to switch to.	User Interface	Asynchronous
3	determine_watchmode_function	To determine whether user wish to set the time	User Interface	Asynchronous
4	determine_alarmmode_function	To determine whether user wish to perform alarm setting	User Interface	Asynchronous
5	determine_stopwatchmode_function	To determine whether user wish to start/stop the stopwatch counting	User Interface	Asynchronous
6	Update_Watch	To update the time display at 1s interval and/or allow user to set adjust the time setting	Display/ CPU Hogging	Architectural/ Asynchronous
7	Update_Alarm	To allow user to adjust the alarm setting	Display/ CPU Hogging	Architectural/ Asynchronous
8	Update_StopWatch	To update the stop watch counting at 10msec interval	Display/ Millisecond	Architectural/ Asynchronous
9	TurnOnOffLEDs	To turn on/off the LEDs display	Display	Architectural
10	IntInt0	To detect user depress switch "SW1"	User Interface	Asynchronous
11	IntInt1	To detect user depress switch "SW2"	User Interface	Asynchronous
12	IntKey	To detect user depress switch "SW3"	User Interface	Asynchronous

Figure 5 Classifications of Function Listings in "Watch_RSKR8C_Demo"

With the classifications, the main functions are further partitioned into single tasks. Figure 6 shows that six tasks and three interrupt service routines (ISRs) are defined.

TASK PARTITIONING									
No.	Function List	Tasks							
		ISRs	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7
1	main		x						
2	*determine_mode_function			x					
3	*determine_watchmode_function			x					
4	*determine_alarmmode_function			x					
5	*determine_stopwatchmode_function			x					
6	Update_Watch				x				
7	Update_Alarm					x			
8	Update_StopWatch						x		
9	TurnOnOffLEDs								
10	**StartAlarm_Task							x	
11	**StopAlarm_Task								x
12	IntInt0	x							
13	IntInt1	x							
14	IntKey	x							

* Functions jointly to form a task "ModeFunc_Task"
** Newly create functions for the activation/deactivation of alarm

Figure 6 Task Partitioning of "Watch_RSKR8C_Demo"

Out of the six tasks defined, “Update_Watch”, “Update_Alarm” and “Update_StopWatch” are three periodic tasks that occur at a regular interval (e.g. “Update_Watch” occur at one interval). Thus, three cyclic handlers will be defined to activate the tasks on a regular basis.

“StartAlarm_Task” and “StopAlarm_Task” are sporadic tasks that trigger at a user-defined interval upon activation. Two alarm handlers will be defined to activate the tasks respectively.

4.3 Step 3: Interrupt Mapping

After all the tasks have been defined, next step is to determine the interrupt routine and corresponding interrupt vector to be defined.

“Watch_RSKR8C_Demo” embedded application is required to response to three sporadic events respectively from three tactile switches, namely, “SW1”, “SW2” and “SW3”. To response to these three events, three separate interrupt routine will be required. With reference to Figure 7, “SW1”, “SW2” and “SW3” are tied to interrupt sources “INT0”, “INT1” and “Key Input” respectively.

Interrupt Source	Vector Addresses(1) Address (L) to Address (H)	Software Interrupt Number	Interrupt Control Register	Reference
BRK instruction(3)	+0 to +3 (0000h to 0003h)	0	–	R8C/Tiny Series Software Manual
(Reserved)		1 to 2	–	
(Reserved)		3 to 7	–	–
Timer RD (channel 0)	+32 to +35 (0020h to 0023h)	8	TRD0IC	14.3 Timer RD
Timer RD (channel 1)	+36 to +39 (0024h to 0027h)	9	TRD1IC	
Timer RE	+40 to +43 (0028h to 002Bh)	10	TREIC	14.4 Timer RE
(Reserved)		11 to 12		
Key input	+52 to +55 (0034h to 0037h)	13	KUPIC	12.3 Key Input Interrupt
A/D	+56 to +59 (0038h to 003Bh)	14	ADIC	18. A/D Converter
Clock synchronous serial I/O with chip select / I²C bus interface(2)	+60 to +63 (003Ch to 003Fh)	15	SSUIC/IICIC	16.2 Clock Synchronous Serial I/O with Chip Select (SSU), 16.3 I²C bus Interface
(Reserved)		16	–	–
UART0 transmit	+68 to +71 (0044h to 0047h)	17	S0TIC	15. Serial Interface
UART0 receive	+72 to +75 (0048h to 004Bh)	18	S0RIC	
UART1 transmit	+76 to +79 (004Ch to 004Fh)	19	S1TIC	
UART1 receive	+80 to +83 (0050h to 0053h)	20	S1RIC	
$\overline{INT2}$	+84 to +87 (0054h to 0057h)	21	INT2IC	12.2 $\overline{INT}$ Interrupt
Timer RA	+88 to +91 (0058h to 005Bh)	22	TRAIC	14.1 Timer RA
(Reserved)		23	–	–
Timer RB	+96 to +99 (0060h to 0063h)	24	TRBIC	14.2 Timer RB
$\overline{INT1}$	+100 to +103 (0064h to 0067h)	25	INT1IC	12.2 $\overline{INT}$ Interrupt
INT3	+104 to +107 (0068h to 006Bh)	26	INT3IC	
(Reserved)		27	–	–
(Reserved)		28	–	–
$\overline{INT0}$	+116 to +119 (0074h to 0077h)	29	INT0IC	12.2 $\overline{INT}$ Interrupt
(Reserved)		30	–	–
(Reserved)		31	–	–
Software interrupt(3)	+128 to +131 (0080h to 0083h) to +252 to +255 (00FCh to 00FFh)	32 to 63	–	R8C/Tiny Series Software Manual

Figure 7 Relocatable Vector Tables of R8C/25

It is mandatory to set a timer as the system clock for MR8C/4 kernel as five time-event handlers (three cyclic handlers and two alarm handlers) are used. Timer RA is chosen as the system clock.

The full outline of tasks, time event and interrupt handlers are shown in Figure 8.

No.	Name	Type	Purpose
1	ID_Task1_Main	Task 1	Startup Task
2	ID_Task2_ModeFunc	Task 2	To call "determine_mode_function", "determine_watchmode_function", "determine_alarmmode_function" and "determine_stopwatchmode_function"
3	ID_Task3_UpdateWatch	Task 3	To call "Update_Watch" function
4	ID_Task4_UpdateAlarm	Task 4	To call "Update_Alarm" function
5	ID_Task5_UpdateStop	Task 5	To call "Update_Stop" function
6	ID_Task6_StartAlarm	Task 6	To set alarm and start countdown
7	ID_Task7_StopAlarm	Task 7	To stop alarm
8	ID_Cyc1_WatchUpdate	Cyclic Handler 1	To activate Task 3
9	ID_Cyc2_AlarmUpdate	Cyclic Handler 2	To activate Task 4
10	ID_Cyc3_StopWatchCount	Cyclic Handler 3	To activate Task 5
11	ID_Cyc4_LEDFlicker	Cyclic Handler 4	To call "LEDFlicker_Cyc4" function
12	AlarmSetup_Alm1	Alarm Handler 1	To activate Task 6
13	AlarmFlash_Alm2	Alarm Handler 2	To activate Task 7
14	interrupt_vector[13]	ISR 13	To call "IntKey" function
15	interrupt_vector[22]	ISR 22	To call "_SYS_STMR_INH" function
16	interrupt_vector[25]	ISR 25	To call "IntInt1" function
17	interrupt_vector[29]	ISR 29	To call "IntInt0" function

Figure 8 Outlines of Tasks, Time Event and Interrupt Handlers for "Watch_RSKR8C_Demo"

4.4 Step 4: Task Priority Assignment

The final process is to define the priority of individual task and deduce the inter-task communication and synchronization mechanisms required.

In this example, Rate Monotonic Analysis (RMA) scheduling method is chosen due to:

- No resource sharing required
- Deterministic deadlines are exactly equal to periods
- Static scheduling is suitable due to simplicity of "Watch_RSKR8C_Demo" embedded application

In RMA, priority of each task is assigned according to its period. The shorter its period, the higher its priority. In the "Watch_RSKR8C_Demo" application, computation/execution time was measured. Its corresponding period was determined based on its specification (e.g. Task "ID_Task3_UpdateWatch" is required to be executed at an interval of 1second to refresh the display, thus its period is 1 second).

Task Name	Task Num ,i	Computation Time ,Ci (msec)	Period ,Ti (msec)	Utilisation (Ui=Ci/Ti)	Priority Assignment
ID_Task1_Main	1	NA			6
ID_Task2_ModeFunc	2	8	100	0.08	2
ID_Task3_UpdateWatch	3	9	1000	0.009	5
ID_Task4_UpdateAlarm	4	8	500	0.016	3
ID_Task5_UpdateStop	5	8	10	0.8	1
ID_Task6_StartAlarm	6	1	900	0.0011	4
ID_Task7_StopAlarm	7	1	900	0.0011	4

1) "*ID_Task1_Main*" is a one-time startup task, thus not within the computation.

2) "*ID_Task2_ModeFunc*" is a sporadic event dependent task (User selection of SW1, SW2 & SW3) and at worst case will occur at and interval of 100 msecs (human hand speed).

3) "*ID_Task6_StartAlarm*" and "*ID_Task7_StopAlarm*" are sporadic events dependent tasks (User selection of SW1, SW2 & SW3) and will need to be activated within the next second, thus it is set at 900msecs.

Figure 9 RMA Priorities Setting for "Watch_RSKR8C_Demo"

With reference to Figure 9, priority of the respective tasks is set based on its period. "*ID_Task5_UpdateStop*" has the smallest period so it will be assigned with the highest priority.

After defining the priority of individual task, schedulability test (based on theorem shown in Figure 10) will need to be conducted to ascertain the priority assignment.

$$W_i(n+1) = C_i + \sum_{j<i} \left\lceil \frac{W_i(n)}{P_j} \right\rceil C_j, \qquad W_i(0) = 0$$

Figure 10 RMA Schedulability Test Theorem

Based on the above theorem, analysis can be performed for all the tasks. Below illustrate the schedulability test for Task2 "*ID_Task2_ModeFunc*" in Figure 11.

$W_2(0) = 0$

$W_2(1) = C2 + [W_2(0)/P_1]*C1$
$= 8 + [0/10]*0.8$
$= 8$

$W_2(2) = C2 + [W_2(1)/P_1]*C1$
$= 8 + [8/10]*0.8$
$= 8.64$

$W_2(3) = C2 + [W_2(2)/P_1]*C1$
$= 8 + [8.64/10]*0.8$
$= 8.69$

$W_2(4) = C2 + [W_2(3)/P_1]*C1$
$= 8 + [8.69/10]*0.8$
$= 8.70$

$W_2(5) = C2 + [W_2(4)/P_1]*C1$
$= 8 + [8.70/10]*0.8$
$= 8.70$

Since $P_2 = 100 > W_2(5) = 8.70$, thus Task2 is **scheduable**

W_2	n+1	C2	$[W_2(n)/P_1]^* C1$
0.00	0	8	0.00
8.00	1	8	0.00
8.64	2	8	0.64
8.69	3	8	0.69
8.70	4	8	0.70
8.70	5	8	0.70

Figure 11 RMA Schedulability Test for Task2

Using the theorem computation, schedulability test can be conducted for the rest of the tasks in the order of their assigned priority.

$W_3(5) = 9.79$
Since $P_3 = 1000 > W_3(5) = 9.79$, thus Task3 is **scheduable**

$W_4(5) = 8.70$
Since $P_4 = 500 > W_4(5) = 8.70$, thus Task4 is **scheduable**

$W_5(5) = 8$
Since $P_5 = 10 > W_3(5) = 8$, thus Task5 is **scheduable**

$W_6(5) = 1.09$
Since $P_6 = 900 > W_6(5) = 1.09$, thus Task6 is **scheduable**

$W_7(5) = 1.09$
Since $P_7 = 900 > W_7(5) = 1.09$, thus Task7 is **scheduable**

Figure 12 RMA Schedulability Test for Task3, Task4, Task6 and Task7

As there is no exchanging of message/data among the tasks, no inter-task synchronization protocol will need to be included.

5. Reference Documents

User's Manual

- MR8C/4 V1.00 User's Manual

The latest version can be downloaded from the Renesas Technology website

Document

- Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems (Bruse Powel Douglass)

Website

- EMBEDDED.COM, http://www.embedded.com.

Website and Support

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

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

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Jan.01.10	—	First edition issued

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.