Strengthening Interrupt Controls in Embedded Systems
by Cooperation between Windows CE and REMON

Shigeki Nankaku\(^1\), Hisao Koizumi\(^2\), and Akira Fukuda\(^3\)

\(^1\)Computer Science, Osaka Electro-Communication University, Shijyonawate, Osaka, Japan
\(^2\)Science and Engineering, Tokyo Denki University, Ishisaka, Hatoyama, Hiki, Saitama, Saitama
\(^3\)Information Science and Electrical Engineering, Kyushu University, Nishi-ku, Fukuoka, Japan

Abstract - Many recent embedded system products have sophisticated display functions. Microsoft Windows Embedded CE (hereafter referred to as ‘Windows CE’) is a widely used embedded OS with a simple GUI design. However, Windows CE has threaded interrupt processes, and therefore it has problems in handling processes for which a strict interrupt response time is requested. We have developed Real-Time Embedded Monitor (REMON) for controlling Interrupt Service Routine (ISR) processes. When using REMON, it is possible to improve the real-time characteristics of the interrupt processes. This paper proposes a system that combines Windows CE and REMON and utilizes the advantages of both to create an embedded system having both sophisticated display functionality and excellent responsiveness to interrupts.

Keywords: Embedded Systems, Interrupt, Interrupt Service Routine, HMI, Windows

1 Introduction

It is vital for embedded systems to be able to send a response to changes in an external environment within a set period of time, such as in the case of mobile phones where it is necessary to respond to an incoming call while creating an email. Changes in the external environment are detected by a wide variety of sensors and are communicated to the CPU using interrupts.

Interrupts are functions of the CPU, and the mechanism used by the CPU hardware is to place interrupt signals in the interrupt signal lines and call the Interrupt Service Routine (ISR). An ISR is software that is used to respond to changes in an environment. Conceptually, interrupts can be considered as a method by which hardware calls software. In other words, it is possible for hardware to process responses to changes in the embedded system environment by calling the ISR and returning the results.

There are various types of environmental changes, and there is also a wide variation in the times at which these changes occur. Multiple changes can occur simultaneously. The priority of a response depends on the type of change involved. As a result, concurrency is sought in ISRs in order to permit multiple interrupts with priorities attached.

Because ISRs directly handle hardware, such as when prohibiting/permitting hardware-level interrupts to attain exclusive control, knowledge of time restrictions for processes and hardware is required when designing ISR systems. Furthermore, as the ISR directly processes hardware, it has a major influence on the system as a whole [1]-[4].

Normally, hardware is encapsulated and virtualized using a real-time operating system (RTOS). This eliminates the need for most of software that makes up the embedded system to directly handle hardware. Furthermore, the ISR is encapsulated in the same way using the RTOS.

In an RTOS environment, processes are executed using tasks and threads (hereafter referred to as ‘threads’). The RTOS provides a variety of functions to threads, such as exclusive control and communication, known as system calls. Threads are able to process interrupts in concurrent using functions also provided by the RTOS [5]-[7].

Many recent embedded systems such as car navigation systems have sophisticated display devices. Microsoft Windows Embedded CE (hereafter referred to as ‘Windows CE’) is a widely used embedded OS with a simple GUI design.

However, Windows CE has threaded interrupt processes, and therefore it has problems handling processes for which a strict interrupt response time is requested. Although it is possible to directly embed interrupt processing into the Windows CE kernel, since the processing is performed in a state where interrupts are disabled by the kernel, problems such as lower interrupt response times and difficulty in predicting the interrupt response time may arise.

ISR controls are vital in embedded systems in order to enable them to respond to changes in the external environment. We have researched the interrupt scheduler...
Real-Time Embedded Monitor (REMON) as a means of controlling interrupts in embedded systems [8]-[10].

As REMON provides the same functionality as an RTOS semaphore for each ISR, it is possible for the ISR to execute exclusive control without using disable interrupt/enable interrupt (DI/EI). The result is that, by shortening the interval in which interrupts are prohibited, the interrupt responsiveness of the embedded system is enhanced, i.e. using REMON improves the real-time characteristics of the embedded system.

REMON provides an independent execution environment for each ISR in which each ISR has a state. Where execution is paused, the state is referred to as a 'wait state'. REMON uses the fact that ISR has a ‘wait state’ and that ISR can use a semaphore.

REMON is highly versatile and can also be applied to RISC-type CPUs, which do not have hardware-interrupt priorities. Furthermore, it records the interrupts that occur and their frequency so its drop rate for interrupts will be low even when they are occurring at a high frequency. REMON also has ISR control functions such as ISR stack overflow detection. In addition, there is little fluctuation in the processing time for ISR execution, making real-time design simple.

However, the objective of REMON is to control ISRs, and it does not have the sophisticated display functions and human interface (HMI) functionality integrated into Windows CE.

In this paper, we propose, through the link-up of Windows CE and REMON, a method to improve the interrupt response of embedded systems with sophisticated display functionality.

With the proposed system, both REMON and Windows CE are simultaneously loaded on one CPU. High-priority interrupts are processed by REMON, and low-priority interrupts are processed by Windows CE.

Currently, REMON prohibits low-priority interrupts, whereas Windows CE always permits high-priority interrupts. As a result, switching from Windows CE to REMON is always possible.

The proposed system makes it possible to handle processes for which a strict response time is requested and those which Windows CE has traditionally been unable to handle. Furthermore, in the proposed system, it is possible to use sophisticated display features using the functionality of Windows CE.

2 Interrupt Processing by Windows CE and REMON

2.1 What are interrupts?

In this paper, an interrupt is defined as a function that uses changes in a specific terminal within the CPU as a trigger for the CPU to suspend its current activities and to start the execution of a program specified in advance, i.e. the ISR.

Interrupts are functions contained by all CPU hardware. Using an interrupt, it is possible to switch from the executing program to a different program.

The computer system switches control from the application program to the OS using periodic interrupts from a timer device.

In an embedded system, changes in the external environment are detected by various sensors which notify the CPU by issuing an interrupt. The CPU can use this interrupt to execute a process that responds to the change.

Figure 1 show an example of a connection where the sending and receiving of packets is communicated by the network controller to the CPU via an interrupt signal pin.

2.2 Interrupt Processing by Windows CE and Related Issues

Windows CE is a 32-bit RTOS for embedded devices. It is compatible with multiple CPU architectures such as ARM, MIPS, SuperH and x86. Furthermore, as the supported application programming interface (API) is a subset of the Windows API, it has high software productivity and is used by a wide variety of devices such as portable AV players, point-of-sale registers, car navigation systems, video projectors and thin client terminals.

In Windows CE, when a device driver is loaded, a thread that processes interrupts, known as the interrupt service thread (IST) starts (Figure 2). The IST has a higher priority than normal threads. When the IST starts, a system call known as WaitForSingleObject, which waits for the generation of an event provided by Windows CE, is issued straight away and the IST goes into a wait state. When an interrupt is generated, the ISR searches the interrupt number for that interrupt (Figure 2). After notification of the interrupt number from the ISR, the Windows CE kernel generates an event responding to that interrupt number and releases the wait state of the IST. When this happens, IST will process the interrupt.

In Windows CE, an important issue in interrupt processing is that latency may occur because the interrupt is executed by a thread. Therefore, it is difficult to predict the interrupt.
response time and to handle processes for which a strict interrupt response time is requested.

In order to improve the response to the interrupt in Windows CE, it is possible to process the interrupt within the ISR (Figure 3). However, this approach poses a problem, as interrupt processing occurs in an interrupt-prohibited state, because other high-priority interrupts may be delayed. It also does not resolve the issue of predicting the interrupt response time.

2.3 Interrupt processing by REMON

REMON, by virtue of having a separate execution environment and state for each individual ISR, can provide each ISR with the same functionality as an RTOS semaphore [8]-[10]. By applying an independent execution environment to an ISR, REMON can associate each ISR with an interrupt control block (ICB, Figure 4). When pausing the execution of an ISR, the execution environment, including the CPU register data, is stored in the ICB, and when restarting the ISR, this data is retrieved.

A stack is allocated to each ISR for use as the local data area for the ISR.

In REMON, each ISR has an independent execution environment, and it is therefore possible for each ISR to restart execution in an arbitrary order. By using REMON, the ISR can be executed with minimal delay. In addition, it is possible to attain exclusive control without using DI/EI. Furthermore, through the use of semaphore provided by REMON for synchronization, it is possible to coordinate the operation of multiple ISRs.

2.4 Issues in the use of semaphore by ISR and the use of semaphore by REMON

Unrelated processing is not delayed in mutual exclusion through semaphores that are used in embedded systems with a RTOS. If an ISR can also use semaphores, the previously described issue does not occur. However, an ISR cannot use semaphores if REMON is not used.

If an ISR requests the acquisition of a semaphore at a time the semaphore is locked by another ISR, the ISR stops executing and saves the context data, which refer to data required for restarting the execution. The restart sequence is not related to the sequence in which the ISRs were stopped, as the restart of a stopped ISR is performed through the release of the semaphore by another ISR. Because ISRs use semaphores, an ISR must be stopped and restarted in a free sequence.

When REMON is not used, ISRs share one stack, where the context is saved. When an ISR is pre-empted, the context data are saved in the stack. As data are restored in the reverse order in which they have been saved in the stack, ISRs are only restarted in the reverse order in which they have been pre-empted.

REMON assigns each ISR an individual storage place for its context, thus enabling the use of ISR semaphores.
3 Cooperation between Windows CE and REMON

Here we propose a new embedded system that cooperate Windows CE and REMON. We hope that, by combining the sophisticated display capabilities of Windows CE and the interrupt-control functionality of REMON, the new system can be effective as an embedded system that has advanced display functionality and can process interrupts within strict response times.

There are several methods of combination of REMON and Windows CE and each is described briefly below.

3.1 Method involving replacement of the Windows CE interrupt handler by REMON

With this method (Figure 5), the Windows CE interrupt process can be freely started from REMON. However, the structure in which the interrupt is processed by IST does not change, and this does not promise much improvement in the interrupt response.

3.2 Method that calls Windows CE from REMON

With this method (Figure 6), all interrupts from the hardware are received by REMON and high-priority time. Furthermore, it would involve large-scale changes to Windows CE, making implementation difficult. Intermits are processed within REMON. For lower-priority interrupts, it calls the Windows CE process. The REMON scheduler (Figure 4) first searches the ICB database to locate an ISR that can be executed, i.e. the array order and priority match. Windows CE handles the lowest priority ISRs received by REMON. It is only when it is unable to execute all of the ISR that Windows CE is implemented.

With this method, it is also possible to monitor Windows CE. This method also allows REMON and Windows CE to be developed separately.

However, this method of linking REMON and Windows CE has a disadvantage in that it further complicates the already complicated Windows CE interrupt sequence. In addition, the Windows CE interrupts are also delayed.
3.3 Method to separate Windows CE and REMON interrupts using interrupt priority

With this method (Figure 7), the interrupts to be processed by REMON and Windows CE are separated. While REMON is executing high-priority interrupts, low priority interrupts are prohibited. As a result, the interrupts processed by Windows CE are prevented from hindering the execution of the high-priority interrupts processed by REMON. Furthermore, since Windows CE never prohibits interrupts and always allows high-priority interrupts, it is always possible to switch to REMON for any high-priority interrupts that occur while Windows CE is executing.

Figure 8 shows the operation of ISR and IST, using CPU interrupt priority, when the interrupts processed by REMON and Windows CE are separated. CPU interrupt priority is a function included in the CPU hardware that can set the priority of interrupts. It is also possible to prohibit/allow interrupts from the software on the basis of priority. Figure 9 shows the transitions in Windows CE and REMON when using a method that separates interrupts on the basis of priority.

Figure 10 shows the sequence of processing interrupts when high-priority interrupts occur in a system that uses interrupt priority to separate interrupts. The figure shows that REMON is called when a high-priority interrupt occurs. If other high-priority interrupts occur, REMON is called, but REMON is not called for low-priority interrupts, Windows CE is called.

Figure 11 shows the interrupt operating sequence when a low-priority interrupt is generated in a system that separates interrupts on the basis of priority.

These are processed by Windows CE, with the same interrupt process operation as that previously used by Windows CE.

When a low-priority interrupt is generated, Windows CE is called. Further, when high-priority interrupts that are processed by REMON are generated, REMON is called.
This method (Method 3.3) of creating a combined Windows CE–REMON system, using interrupt priority to separate Windows CE and REMON interrupts, best meets our objectives. Therefore, we have adopted this method to provide a link-up between Windows CE and REMON.

## 4 Testing and measurement of results when Windows CE and REMON work together

### 4.1 Testing environment

In order to test the combination of REMON and Windows CE, we have created an embedded system on the MINI2440 (Table 1), using the Samsung S3C2440 ARM architecture CPU, the ARM CPU most widely used by Windows CE (Figure 12). The Windows CE version used is Windows Embedded CE6.

Figure 13 shows the interrupt model in ARM and the ARM interrupt control register. In ARM, two levels of interrupts, known as IRQ and FIQ, are present. As FIQ processes at a faster speed than IRQ, ARM uses a banked register in which a part of the register can be switched. As FIQ has a banked register, it can process at faster speeds than IRQ.

FIQ is not used by Windows CE and is used only as an interrupt executed by REMON.

### 4.2 High-speed ISR switchover

FIQ is not used by the Windows CE kernel. By using REMON for FIQ interrupts, the efficiency of interrupt processing can be increased.

When the REMON ISR is initiated by temporarily disabling IRQ interrupts, the embedded system can process interrupts at high speeds without the interrupt overhead of Windows CE.
When processing interrupts with a strict interrupt response time, it is necessary to switch to ISR at high speeds when an interrupt occurs. Furthermore, if the interrupt that occurs has a low priority, it must return processing promptly. For this reason, the embedded system is constructed in such a way that switchover uses the FIQ banked register and can switch with the minimum amount of processing.

4.3 Measurement results
We used a logic analyser to measure the time from when the interrupt was generated until the time processing started for Windows CE alone and for the combined embedded REMON–Windows CE system. As the logic analyser conducted sampling using 800 MHz signals, the minimum measured unit was 0.25 ns. Measurements showed the mean value for each and every 100 calculations. The results are shown in Table 2.

While processing interrupts with Windows CE had a response time of 32.09 μs, this was reduced to 4.58 μs when processing interrupts with the combined embedded REMON–Windows CE system. Thus, we were able to attain a sufficiently practicable interrupt response time.

5 Conclusions
By combining Windows CE and REMON, it has become possible to handle and process strict interrupt response times that could not be processed with Windows CE alone. Furthermore, we believe that realizing exclusive control of interrupt processing has led to an improved level of reliability in regard to interrupts.

Issues to be examined in the future include the reinforcement of the interrupt control functionality of Windows CE through sharing the interrupts of Windows CE and REMON. We also plan to apply this to other real-time operating systems.

6 References


