International Journal of Hybrid Information Technology
Vol. 9, No.11 (2016), pp. 447-460
http://dx.doi.org/10.14257/ijhit.2016.9.11.38

Research on the Fast Interrupt Mechanism of MQX Lite

Chen Zhi [1]*, Jiang Jianwu[12] and Yang Xuanyuan[1]

[1] Department of Electronic and Information Engineering,Taizhou Polytechnic College,Taizhou 225300, China; [2] School of Computer Science and Technology, Soochow University, Suzhou 215006,China
tzchenzhi@sina.com

Abstract

Interrupt response is an important index to measure the embedded operating system, and the high efficiency interrupt handling mechanism can greatly improve the system's interrupt response. In this paper, we analyze interrupt processing flow of MQX Lite, and propose a fast interrupt mechanism ,in which Emergency handling is embedded in the kernel ISR. The interrupt response time of three interrupt handling points including kernel interrupt, user terminal and user task in the fast interrupt mechanism is detected, and the experimental data show that the mechanism can significantly improve the system's interrupt response, and can meet the high risk of high voltage, explosion-proof, and other high risk applications.

Keywords*: MQX Lite,Fast Interrupt Mechanism,Interrupt Response*

1. Introduction

Real-time response to outside signals is an important performance index to evaluate the embedded application systems and it reflects the ability of application systems to handle outside emergencies. On some special ocassions and enviroments (high pressure, high temperaure, explosive gas), abnormal real-time responses could result in catastrophic consequences. The reponses of application systems to outside signals are usualslly done by controlling interrupt processing of chips. Hence, the outcome of systems' real-time response degree is decided directly by the performance of chip's interrupt processing.

During applicative designing without RTOS（Real Time Operations System）, the interrupt realization is quite intuitive. It's easy for designers to intervene details in interrupt realization . Practical controls and processing are quickly done to improve real-time responses of special applications[1,2]. However,due to complex interrupt response mechanisms in RTOS, it will bring huge hidden troubles to systems' stability to amend interrupt realization codes without knowing specific realizing mechanisms in applicative designing processes with RTOS. To optimize interrupt procssing, a lot of researchers have studied interrupt mechanisms of various RTOS. CENA did studies on real-time responses of wifi broadcasting under the linux platform[3]. Songlihua analyzed interrupt processing mechanisms based on message queues underμC/OS-Ⅱsystems[4]. Keum did some explorations on realization mechanisms of interrupt mechanisms of Android sytem in rea-time despatching[5].

There are quite a few ducuments which focus on interrupt processing researches of embedded operation systems including Linux and Android. Our research group put efforts on studies of interrupt mechanisms of MQX operation systems. Zhu Shilang analyzed the relationship between task priority level and interruption[6]. Shijing outlined interrup mechanism of MQX[7]. Jiangjianwu put forward some deep analysis of detailed processing of interrupt realization[8]. This article presents a fast interrupt mechanism by inserting emergency interrupt processing in Kernel ISR based on above studies.

ISSN: 1738-9968 IJHIT

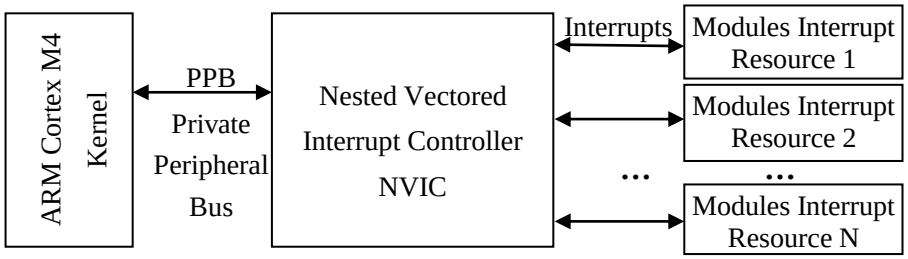


Figure 1. Interupt Struct of ARM CortexM4

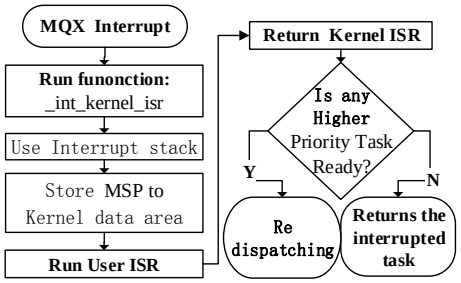


Figure 2. MQX Interrupt Execution

MQX Lite interrupts response mechanism

MQX Lite interrupt response mechanism is composed of hardware interrupt response mechanism and software response mechanism. Hareware interrupt response mechanism is related to kernel interrupt processing mechanism and the figuration can be optimized by softwares. The software interrupt response processing is realized through RTOS kernel processing procedures and it also can complete specific response activities processing of interrupt source signals.

1.1. Arm Cotex-M4 Interrupt Response Mechanisms and Principles

The ARM Cortex-M4 (hererinafter referred to as CM4) structure is shown as in Figure 1: NVIC(Nested Vectored Interrupt Controller) connects modules interrupt resource and CM4 kernel. Moudles interrupt resource receives outside interrupt.

Signals send interrupt request to NVIC and then NVIC decides which interrupt will be sent to CM4 kernel though private peripheral mainlines according to interrupt number and interrupt optimal configuration *etc*. When many interrupt source signals arrive at the same time, priority level is the key standard for arbitrament . As for interrupt source signals with the same priority level, the small number will be carried out first while other interrupt source signals will be suspended[9].

1.2. Analysis of MQX Lite Interrupt Response Mechanism

MQX lite interrupt mechanism includes two parts: Kernel ISR and User ISR.Kernel ISR is related to processor's Kernel runned by the operation system to realize corelation between modules interrupt source and User ISR. To Accelerate MQX's prompt response to modules' interrupt source, programming language is used and programming codes varies due to different processors. Except for reset interrupt, Supervisor call(SVC) and pendable supervisor(PendSV), all interrupt vector quantities point to Kernel ISR[10].User ISR is not a task but a small quick routine which could respond promptly to hardware

interrupt. Its main functions are to reset equipments, obtain equipment data and send signals to relevant tasks through non-blocking MQX functions[11]. When an interrupt transaction comes, MQX implements processes shown in figure 2. When interrupt signals arbitrated by NVIC hardware are sent to kernals, kernels will put the current context register (xPSR, PC, LR, R12, R3, R2, R1, R0) into the stack, stop other interrupt to enter KernelISR except above 3 special interrupts and perform _int_kernel_isr processing functions. Detailed performing processes are shown in Figure 3 which has been stated in " deep analysis of MQX interrupt mechanism based on ARM Cortex-M4. We will not repeat here.

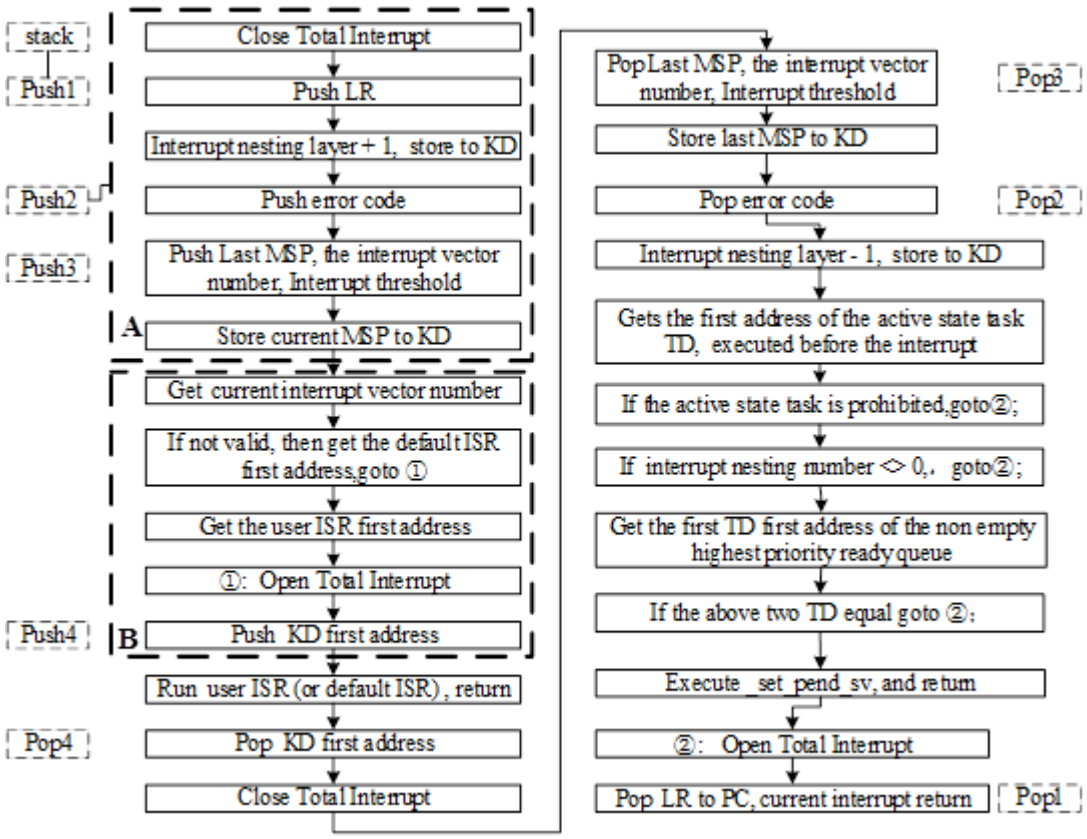


Figure 3. MQX Kernel Interrupt Mechanism

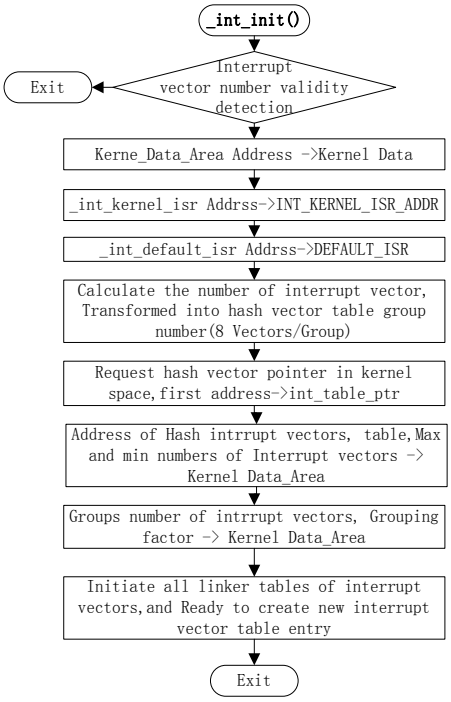


Figure 4. Initiation of Hash Interrupt Vectors Table Structure

2. Analysis of MQX Lite Interrupt Starting Process

2.1. Initialization Process of MQX Lite Interrupt Environment

Initialization of Lite MQX interrupt environment includes static interrupt vector table initialization, interrupt stack initialization and sparse interrupt vector table initialization. The initialization process of Lite MQX interrupt environment is shown in Figure 4. The process can be summarized as follows:

（1）To apply for a static interrupt vector address space in Flash, initialization ISR acts as the kernel ISR, and this step is specified in the link file.

（2）Initialization system interrupt stack is to open up the system space for the interrupt stack, and the function _mqx is to perform the process of initialization of the core data area to start the support for the interruption.

（3）To apply the sparse interrupt vector table space in memory RAM, initialization of sparse will interrupt the vector table, and the function of _mqx is to execute the process of initialization of peripherals, which is the interruption of the system initialization.

If any error occurs during the MQX initialization, MQX calls the function of _mqx_exit to exit the MQX, and the so-called exit MQX is actually executing a section of “perpetual cycle”. It seems to be the “system crash” status to the outside user.

2.2. Set-up of System Interrupt Stack

The system interrupt stack is created when the MQX startup the initialized function of _mqx. Main code snippet is shown as the below, and the system uses malloc for the size of interrupting stacks, then it assigns the first pile of stack address to the address field of “INTERRUPT_STACK_PTR” in the kernel data, thus in this way, the interrupt stack was ready in the system [9].

2.3. Setting up of Sparse Interrupt Vector Table

2.3.1. The Structure of Sparse Interrupt Vector Table

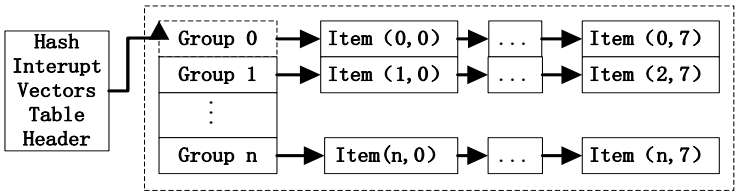


Figure 5. MQXLite RTOS Sparse Interrupt Vectors Table Structure

Table 1. Interrupt_Sparse_Rec_Struct

Members	Describe
VEC_NUM	Interrupt Vector Number（0~255）
(_CODE_PTR_APP_ISR)(pointer)	
APP_ISR_EXCEPTION_HANDLER	
APP_ISR_DATA	Pointer of ISR parameter
* NEXT	Next pointer of item in Interrupt Vector Table

The structure of sparse interrupt vector table is set up according to the hash structure, as shown in Figure 5. The sparse interrupt vector table entry is managed in a group of 8 (by default). When the system is initialized according to a given interrupt vector number divided by 8 to determine the number of dynamic interrupt vector table. For example, 251 interruptions can be divided into 32 groups as the unit of 251/8 (method of entry for getting the group number). Since MQX system calls the function of malloc for the 32 group heads (pointer array, and each pointer is 4 bytes, not 20 bytes) for a continuous address space during the initialization, the group heads between nodes can locate the address by offsetting a team of head node-size space. What's more, each table node address is not continuous in the group, and each list item node is an independent dynamic application. Through the "next" field in the linked-list, the pointers string into a linked-list. The content of each group header node is initially set to NULL when the MQX is initialized. The vector table entry content calls the function of _int_install_isr to fill in by the user in the installation of the interrupt service routine.

2.3.2. Creation of Sparse Interrupt Vector Table Group Header Pointer Array （_int_init）

Array pointer of sparse interrupt vector table is actually one-dimensional, the function of _mqx has made a simple introduction of the interrupt system initialization in the process of initializing the peripheral.Through the program code analysis, the essay will further elaborate the creation process of sparse interrupt vector array pointer. Sparse interrupt vector chain header node creates a function to call the specific location: firstly, the function of _mqx () calls the function of _bsp_pre_init() to initialize the timer and interruption. Secondly, in the initialization of a chip peripherals, it calls the function of _psp_int_init() to interrupt initialization function, and once more, it calls function of _int_init() in interrupting initialization function to create a sparse interrupt vector header node and to process the initialization.The execution process is shown in Figure 4.

2.3.3. Data Structure of Sparse Interrupt Vector Table （interrupt_sparse_rec_struct）

The management of sparse interrupt vector table is in the form of HASH linked-list, and in the list, the elements is the table item element of the interrupt vector table. The structure of sparse interrupt vector table (INTERRUPT_SPARSE_REC_STRUCT) is shown in Table 1, accounting for 20 bytes, with 4 bytes in each field. The field “VEC_NUM” is the interrupt vector number. Field "APP_ISR" is the ISR function address, and the type of interrupt service function is void (*) (void*), which means a return value is null, and the parameter is function pointer of null. Field “APP_ISR_EXCEPTION_HANDLER” performs an abnormal processing function pointer when an exception occurs. Field “APP_ISR_DATA” is the parameter address transferred to ISR, and the user can put all the parameters into a structure type, in which the address of the structure can be transferred to the interrupt service routine as the calling of a parameter. Field “NEXT” is the structure type of sparse interrupt vector table, and the “NEXT” field is a pointer to the next interrupt vector table entry element for a group of 8.

2.3.4. Creation of Table Entry and Installation of Interrupt Service (Routine_Int_Install_Isr)

User ISR is written by MQX users, and can be dynamically installed in the system of the MQX interrupt processing system during the procedure. After the external interrupt event occurs, MQX will postpone it to the relative safe time point for execution.

MQX interrupt processing system will preserve the user ISR installed by the user in a software implementation of the sparse interrupt vector table, and the user ISR can use a pointer to pass the input parameters. When users install a user-defined ISR through the installation of API—_int_install_isr— via an interrupt handling system, the user ISR function is filled to the user ISR vector table. When a reponse is needed to an interrupt event, it will turn to the implementation of the corresponding user ISR via the _int_kernel_isr. Figure 6 is the illustration for the User ISR running mechanism.

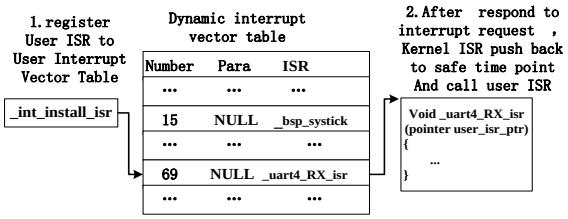


Figure 6. Execution Mechanism of User ISR

When setting up the group of header nodes in sparse interrupt vector table during the period of MQX system initialization, the group of nodes will be created by calling the function of _int_install_isr to make a dynamic application as users increase the interrupt service routine. The function _int_install_isr calculates the location of the interrupt vector table entry based on the introduction of the interrupt vector number in order to create sparse interrupt vector linked-list nodes, and complete the installation of the user interrupt service routine. Detailed procedures are described in the following code and notes.

Repeating the installation of an interrupt user ISR will not make the number of entries in a group surpass the number of 8, because the interruption of the installation function is to replace the old operation, in which the maximum number of entries in a group are 8 units at most.

3. MQX Quick Interrupt Mechanism Building

3.1. MQX Lite Quick Response Interrupt Mechanism Principle

According to analysis from the article " the MQX interrupt mechanism and interrupt program design framework based on ARM Cortex-M4 " [7], MQX interrupt responsiveness is mainly affected by four factors, namely the maximum time of C- kernel screening interrupt ; D- time bewteen interrupt source sending an interrupt signal and the hardware responding interrupt; L- default time of kernel checking dynamic interrupt vector quantity; R- time of ISR performing and restoring . Interrupt latency (interrupt latency time, ILT) Lite is calculated as follows: ILT = C + D + L + $\sum_{i=1}^{N}(D + L + R)$, N is a number of higher priority after concerned interrupt not a non-mask interrupt.When the interrupt transactiont to be implemented is the most urgent one,the interrupt priority level can be set as the highest and assigned to a non-mask interrupt pin, the equation C = 0, N = 0, the above formula is simplified as ILT = D + L where D is fixed at 12 cycles, while the L time starts from entering Kernel ISR and ends to implementing user ISR,. In other words, the time(as shown in Figure 3) is for interrupt implementing preprocessing (dashed box a) plus finding and obtaining the first UserISR address (dotted box B).According to souce codes of ".. \ mqx \ psp \ dispatch.S" , AB implementation time is as shown in table 2. Since the implementation process D and A can not be changed, we need to put processing of emergent interrupt transactionss before implementing processing B to improve interrupt responsiveness.

3.2. MQX Lite Fast Interrupt Mechanism

MQX Lite fast interrupt mechanism aims to interrupt the operation of the intervention as soon as possible。Throughout the entire interrupt implementing process, once interrupt vector number is determined the interrupt source is located where all kinds of emergencies can be processed. Meanwhile to reduce time using Kernel ISR , other routine processing can be done in later User ISR . Since the kernel Kernel ISR is programmed by assembly language so does the inserted code. During the implementation of Kernel ISR, every general-purpose registers are stored in data with specific meaning. Therefore Regesters being used should be protected before and after programming fast interrupt mechanism codes. The implementation process is shown in Figure 7.

Table 2. Response Time From Kerelisr To Userisr

Execute Phase	Execute Time （ Instruction cycle ）
A	36
B	44+M*5

notes：M is the position of the interrupt source table entry in the corresponding group of the sparse interrupt hash table（0-7）

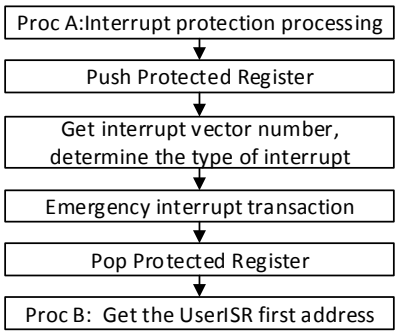


Figure 7. Fast Interrupt Mechanism Execution

4. Anaysis of Fast Interrupt Mechanism for MQX Lite

Time of interrupt response is a primary index to evaluate functions of interrupt mechanism implementation.

Though comparing interrupt response time we can see response effects of interrupt response processing in different locations of interrupt mechanism.

4.1. MQX Lite –RTOS Establishing the engineering frame

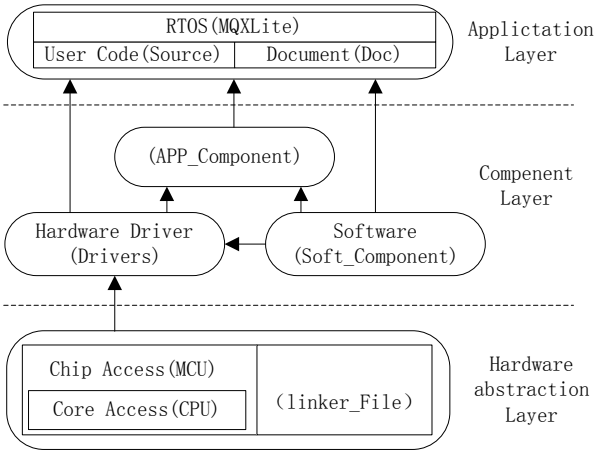


Figure 8. MQXLite Framework Layer Structure

For the purpose of using MQX Lite embedded operating system more quickly and efficiently in the development of application system, with the software-component and software-engineering idea, a structural-clear, reasonable-setting, component-complete embedded engineering framework MQKLite is constructed. Engineering framework(as shown in Figure 8) adopts three layers of logic structure, that is, hardware abstraction layer, component layer and application layer..The hardware abstraction layer is divided into three parts: the kernel level access layer, the chip level access layer and the link file., which includes related files needed for the start ing when the chips get reset. The kernel level access layer mainly defines the special register, the interrupt nested control register, and the access interface of the debugging subsystem [7]. The chip level access layer mainly defines the hardware register address, the interrupt and the abnormal number of the device. In order to ensure the security of the application program, the hardware abstraction layer can only be called by the underlying driver component. In terms of the application layer, the hardware abstraction layer is completely shielded. The component

layer is divided into three parts: the bottom drive component, the software component and the application component. User codes include interrupt service routines and user's main program. When the application engineering requires no embedded operating system, it does not need to include the MQX related contents.

The idea of software engineering requires that the engineering framework meet a clear structure, a reasonable arrangement for the contents of the file, and a feature of portability, easy-modifying[12]. According to ideas of engineering framework level and architecture design, the 8 basic folder such as NOS engineering framework is constructed. When the project needs to use the MQX operating system, a MQX optional folder is added, which constitutes the unified engineering framework AMQXFW by MQX-RTOS and NOS, and numbers each folder in accordance with the order of the bottom up in the framework of the project. The contents are shown as Table 3[7].

Table 3. Framework Documents Struct

MQXLite-Frameworks	Project root
Includes	Includes files
01_Doc	Document
readme.txt	Project description Guide
02_CPU	CPU Core
03_MCU	MCU files
boot.S	System Boot File
MKL26Z4.h	Chip Head File
startup.c	Start File
sysinit.c	Initialization File
sysinit.h	Initialization Head File
vectors.c	Interrupt Vectors File
vectors.h	Interrupt Vectors Head File
04_Linker_File	Linker Files
ProcessorExpert.ld	Linker Files
05_Driver	Hardware component Files
06_App_Component	Application component Files
07_Soft_Component	Software component Files
08_Source	Source Code Files
includes.h	Source Code Includes file
isr.c	Interrupt handler Code
isr.h	Interrupt handler Head File
main.c	Main Code
09_MQXLite	MQXLite RTOS Files

4.1.1. Engineering Root Folder And Engineering Documents Doc

In order to facilitate the management of the engineering framework, all the documents are stored in the project root directory folder, whose name can be modified according to the actual project contents. According to the principle of software engineering document priority, a detailed functional document description is needed in front of the software code . Therefore, in the framework of the project a special folder document Doc is opened for the storage of the system documentation, including the text Readme file, which is general description document of the whole project, and it records the project name, version number, modification time, and other basic information.

4.1.2. Kernel Folder CPU

The selection of processor kernel must be taken into account before the development of project. The mode of ARM kernel management and operation are followed by kernel manufacturers after the rise, and kernel manufacturers only engage in designing and maintaining the kernel architecture without direct production of chip [13], thus the kernel related source code is also under the maintenance and release of the kernel manufacturers. Hence in the framework of the project design, kernel related documents will also be departed from the chip files and stored independently to CPU directory so as to reduce code revision during a kernel upgrading, and is only responsible for modifying the name and content of the CPU folder file.

4.1.3. Chip Folder MCU:

The starting process of the embedded system is related to the specific chip, and once the chip is determined, the chip header files, interrupt vector table and start codes are relatively fixed, and the unified storage of the file in the MCU folder is to improve the start-up code re-usability. Except that the name of chip header file can be changed in MCU folder, other files' names are relatively fixed. The usage of different MCU or chip to upgrade in engineering is only related to the adjustment of the header files and some codes while the chip related documents can be obtained from the chip manufacturers.

4.1.4 Link File Linker_Files

Due to different methods when the environment link is developed, the file name connected with each other and their extensions are not the same. Therefore using different development environment in projects to modify the contents of link-file and corresponding files can be modified by the link files generated by developing environment engineering template.

4.1.5. Component Folder

In the framework of the AMQXFW project, it contains the underlying driver component folder Driver, software component folder Soft_Compenent and application component folder App_Compenent.

Driver component is directly related to the function of the chip module and driver package, including GPIO as well as UART, and this part is the common function for general MCU. Thus during the general design of the file name, function interface name and related parameters, they cannot be modified and its detailed realization method is to adjust the relevant code based on different chip registers.

Software components are general software components which is irrelevant to CPU and MCU, including the queue and list *etc.*, so names and contents of the relevant components in this folder are not allowed to change.

Application components complete a specific functional design by calling the underlying driver components and software components, such as LED, LCD, motor and other hardware drivers, so file names and contents of this folder can not be changed after being packed. When different chips are used, the contents of the underlying driver can be modified, and the code of the application component does not need any change.

4.1.6. Source Program Folder Source Component Folder

Application layer code is the part which developers tend to modify the most. When the development environment is fairly built and other folders in the content are relatively fixed, all later function debugging is concentrated in the application layer, therefore this part of contents should be stored in the source folder source, containing the project total header files "include.h", main program "main. C" and interrupt handling of header files

"isr.h" and source files "isr.c". These file names are fixed, and the file contents should be revised according to the content of engineering project. When using the MQX operating system, the main. C file should be remained, but the task it attempts to complete is to start MQX system scheduling and to transfer the control of the system to the MQX operating system. The related application layer code is also transferred to the MQX folder in the function of app App_inc.h and task in main. C. Since there is no difference to the interrupt processing process whether the operating system exists, in these two cases, the interrupt processing actions can be completed in the Source folder isr.h and isr.c. [12]

4.2 Structure Design of Testing Models Based On MQX Lite Fast Interrupt Mechanism

Starting from the interrupt source generating an interrupt to completing interrupt transaction, the implementation process is shown in Figure 9 .It contains three interrupt transaction insertion point (hereinafter referred to as IP), which are located in the kernal ISR, the user ISR and interrupt tasks

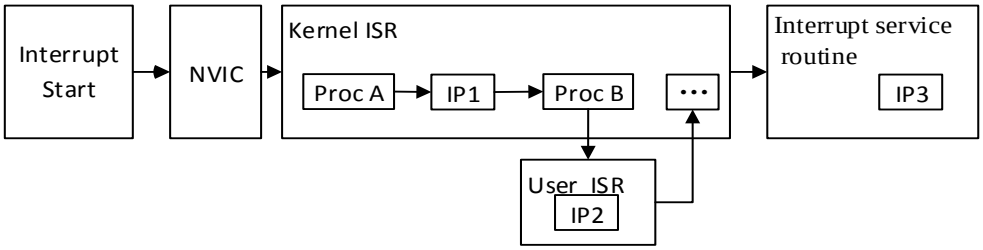


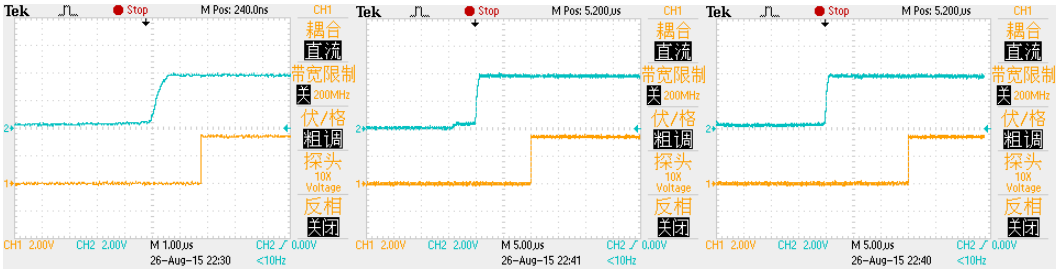
Figure 9. MQX Lite Interrupt Mechanism Flow Chart

In this article,the concerned fast interrupt test models are designed on MKW01Z128 chip in Freescale Arm Cortex M0 + kernel core. Under KDS3.0.0 IDE development environment all software designs are done based on KW01-Zigbee engineering framework in MQX Lite kernal core in Suzhou University. based on the use of Soochow University. Waveform measurements are completed by TEK TDS2024B oscilloscope .

Test model utilizes PTA2 as an external interrupt port input, tests equipments' short circuit pulse,and uses PTC1 as a general output port to turn off 9KV high voltage output. It could make sure that when a short circuit pulse is generated within the system,high voltage output function is disabled. Hence equipments will not be damaged and safety of personnel is guaranteed., In the test model we could insert operations of turning off high voltage into 3 different testing points of IP1 ~ IP3. We can capture waveforms by an oscilloscope in three cases, and observe the response time

4.3. Analysis of MQX Lite Fast Interrupt Machanism's Testing Data

PTA2 is set as rising edge interrupt in Software design.The oscilloscope's channel 1 is connected with PTA2 and channel 2 is connected with PTC1. When handling high-voltage output shutting in three different insertion points respectively,interrupt response waveform is shown in Figure 10. Throughout the interrupt processing, the more forward interrupt transaction point is, the higher the response degree is and the faster the processing speed will be. The response time is shown in Table 4.



（a）IP1　　（b）IP2　　（c）IP3

Figure 10. MQX Lite Interrupt Response Waveforms of differenttransaction Processing Points

Table 4. Fast Interrupt Response Time

Transaction Point	Langua ge	Response Time	Response Degree
Kernel ISR	ASM	1.8µs	High
User ISR	C	10µs	Middle
Interrupt task	C	15µs	Low

We can find the highest response degree when dealing with emergent transactions in ISR kernel. But it requires that the designer can clearly understand the kenel ISR processing and locate the processing pinpoint correctly. Meanwhile, the designer must have capabilities of language programming, with brief refining interrupt handling procedures, short running-time and fast execution, otherwise it will affect other interrupt execution time to reduce the overall performance of the system. Thus demands for programmers are very high. It's reletively easy to deal with transactions in User ISR and interrupt tasks, and the interrupt corresponding degree is able to meet requirements of most projects processing.

Ending

It's our ultimate goal to improve interrupt response degree for optimizing interrupt mechanism in embedded operating systems. Completing actual transations as soon as possible is as effective method to improve interrupt responseness in whole interrupt process. MQX Lite fast interrupt mechanism can shorten response time significently if we upgrade emergency services in application systems to MQX Lite ISR core processing. It can also improve the response capacity of the system and have certain reference values to other similar embedded operation systems.

References

[1] S. Nankaku, H. Kawakami and H. Koizumi, "A Dynamic Control Mechanism of Interrupt Stack Overflow on Real-Time Embedded Monitor (REMON)", Electronics and Communications In Japan, vol. 98, no. 3, **(2015)**, pp. 57-69.
[2] G. Gracioli, A. A. Froehlich and R. Pellizzoni, "Implementation and evaluation of global and partitioned scheduling in a real-time OS", Real-Time Systems, vol. 49, no. 6, **(2013)**, pp. 669-714.
[3] G. Cena, S. Scanzio and A. Valenzano, "Implementation and Evaluation of the Reference Broadcast Infrastructure Synchronization Protocol", IEEE Transactions On Industrial Informatics, vol. 11, no. 3, **(2015)**, pp. 801-811.
[4] L.-H. Song and Q.-J. Zhang, "Design and Realiztion of Double Message Queue in Interrupt mamagement System for µC/OS-Ⅱ System", Computer Engineering and Design, vol. 34, no. 7, **(2013)**, pp. 2377-2383.

[5] K. Keum, S.-J. Kim, J. M. Kim, "Real-Time Scheduling Method to assign Virtual CPU in the Multocore Mobile Virtualization System", Journal of Digital Convergence, vol. 12, no. 3, **(2014)**, pp. 227-235.
[6] S.-L. Zhu, Y.-H. Wang and D.-W. Feng, "Research on MQX task Priority Setting and Its Impact on Interrupt", Computer Applications and Software, vol. 35, no. 7, **(2014)**, pp. 2375-2379.
[7] J. Shi, Y.-H. Wang, Y. Su and C. Shen, "Analysis of MQX interrupt Mechanismand Designof Interrupt Program Frame Based on Arm Coretex-M4", Computer Science, vol. 40, no. 6, **(2013)**, pp. 41-44.
[8] J.-W. Jiang and Y.-H. Wang, "Depth Profile of Interupt Mechanism in MQX Based on ARM Cortex-M4", Electronic Technology & Software Engineering, vol. 3, no. 22, **(2014)**, pp. 214-217+233
[9] Y.-H. Wang, S.-L. Zhu and W.-S. Yao, "Application Development Technology Based on Embedded RTOS MQX", Public House of Electronics Industry, BeiJing, **(2014)**.
[10] Freescale. Freescale MQX RTOS 4.1.0[CP/OL]. http://www.freescale.com/mqx, **(2013)**.
[11] Freescale.Freescale MQX RTOS User's Guide Rev.6 [EB/OL]. **.**http://www.freescale.com/mqx, **(2013)**.
[12] G. Giovani and S. Fischmeister[Canada], "Tracing and recording interrupts in embedded software", Journal of Systems Architecture, vol. 58, no. 9, **(2012)**, pp. 372–385.
[13] Y.-H. Wang, J. Wu and Y.-Z. Jiang, "The Principle and Practice of Embedded System Based on ARM Cortex-m4 kinetis Micro Controller Unit", Public House of Electronics Industry, BeiJing, **(2012)**.

Authors

Chen Zhi, she was born in 1975,Shi is an Associate Professor with the master degree in computer science. Her current research interests include computer application technology and network technology. She has published more than 10 papers.

