Assignment

Day 7

Monday, 11 September 2023

By: Ibrahim Tarek

SV G2 / Intake #3

Table of Contents

1. Executive Summary

This report discusses each of the following topics:

1- The main features of nested vector interrupt controller.
2- The interrupt latency, NVIC role in interrupt latency control.
3- The NVIC interrupt tail-chaining.
4- Context switching in ARM cortex M4.

2. Nested Vector Interrupt Controller.

Nested vector interrupt control (NVIC) is a method of prioritizing interrupts, improving the MCU's performance and reducing interrupt latency. NVIC also provides implementation schemes for handling interrupts that occur when other interrupts are being executed or when the CPU is in the process of restoring its previous state and resuming its suspended process.

2.1. Main Features

1- Suspends the exception being processed.
2- Starts high-priority exception processing.
3- Completes high priority exception processing.
4- Resumes interrupted exception processing.

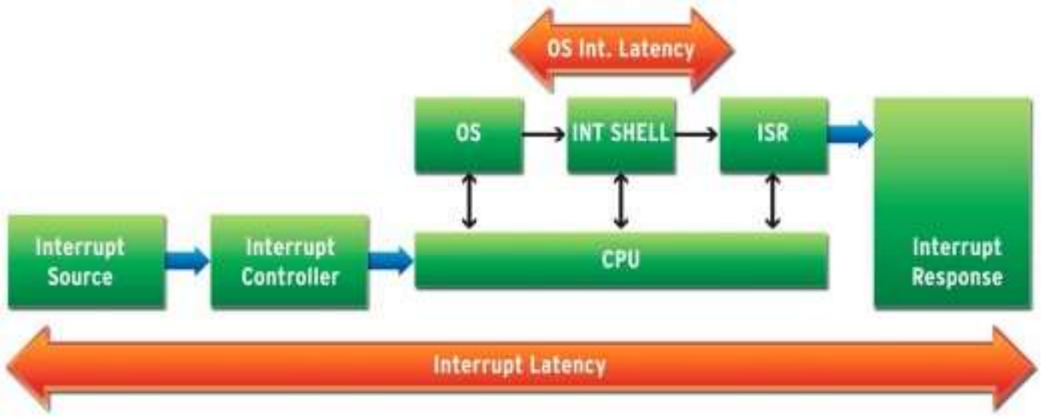
3. Interrupt Latency

Interrupt latency is a measure of the time it takes for a computer system to respond to an external event, such as a hardware interrupt or software exception. This metric is important in determining the performance and responsiveness of a system and is a key consideration in the design and optimization of real-time and embedded systems. In this article, we will discuss the concept of interrupt latency, its importance, and some of the factors that can affect interrupt latency.

3.1. Definition

Interrupt latency is the time that elapses between the occurrence of an interrupt and the execution of the first instruction of the interrupt service routine (ISR) that handles the interrupt.

1- It is a measure of the system's ability to respond to external events in a timely manner.
2- The shorter the interrupt latency, the more responsive the system will be.
3- Interrupt latency is expressed in core clock cycles.



3.2. Interrupt Latency Importance

Interrupt latency is an important consideration in the design and optimization of real-time and embedded systems. These systems often have hard real-time constraints, meaning that they must respond to external events within a specified time period.

1- A high interrupt latency can result in the system missing a deadline, which can cause the system to fail or produce incorrect results.
2- If interrupts are not handled in a timely manner then the system will exhibit slow response times.

3.3. NVIC role in interrupt latency control

The ARM Cortex-M microcontroller series has a low interrupt latency and is frequently used in embedded systems. The nested vectored interrupt controller (NVIC) on Cortex-M processors enables effective interrupt handling. Between the time the interrupt occurs and the time the processor starts executing the interrupt service routine, there are only a few instructions in the NVIC (ISR).

The following factors can still have an impact on the Cortex-M processors' interrupt latency:

1- **Priority of interruptions:** The NVIC prioritizes interruptions, so an interruption with a higher priority will take precedence over an interruption with a lower priority.

2- **Interrupt nesting:** Cortex-M processors are capable of interrupting nesting, which allows one interrupt to be interrupted by a different interrupt. If the nested interrupt has a higher priority, this may result in an increase in latency.

3- **Interrupt response time:** Reduced interrupt response time is a built-in feature of some Cortex-M processors, such as the "tail-chaining" feature of the Cortex-M4.

4. NVIC interrupt tail-chaining

One other concept that NVIC supports is tail chaining of interrupt. This is another name for nesting of interrupts, and it helps in executing the interrupts back to back without the problem of context switching. Without nested interrupt controller the coming interrupt goes in the pending state if an interrupt is already being executed unless that running interrupt completes its service routine and go back to the program and do complete context switch. In nested interrupts, however, we do not have to do this, and the next interrupts got executed within the first interrupt before giving the control back to the calling program.

When a context-switch occurs from a low priority task to a higher priority task, but the execution of interrupt handler of a low priority task is still pending. In this process, low priority interrupts chained and gets to execute only when all high priority tasks finished their execution.

5. Context Switching in ARM Cortex-M4

Due to the hardware specifics of the Cortex-M architecture, there is actually a lot of common in how function calls and exceptions are handled. The good thing for us as developers is that all exception handlers can be written as regular C functions. How that happens and what actions are performed by the software and by the hardware is the focus of this article.

5.1. Exceptions and Context Switching

Exceptions are events that disrupt the normal execution flow of the program. When an exception occurs the processor handles it by executing a dedicated piece of code called an exception handler (a.k.a interrupt service routine ISR). Once this exception handler code is executed, the CPU must return to the regular program that was being executed at the time the exception occurred. This switching between the regular program code and the exception handling code requires the implementation of context switching.

Context switching is the process of saving the state of a processor or a task (e.g in real-time operating systems) with the intention for it to be restored at a later point in time. The context switching is between the main program and the exception handler code that has to be executed when an exception occurs (if nested exceptions are allowed we can have further context switching between exception handlers). The state of the processor includes all relevant resources used by the interrupted program, that can be used also by the exception handler routine.

5.2. ARM Cortex-M Specifics

The context switching when an exception occurs in the ARM Cortex-M CPU is handled using a hardware and software component. The hardware is the CPU and the NVIC that automatically saves some resources when an exception occurs. The software handling is done by the compiler, complying with the requirement for callee-save registers described in Procedure Call Standard for the ARM® Architecture. that we covered in our article Function Calls on ARM Cortex-M microprocessors.

For understanding the exception handling sequence, some specifics of the ARM Cortex-M microprocessors should be mentioned.

The processor has the following operating modes:

1- Thread mode – Used to execute application code. It is the processor's default operating mode after reset.
2- Handler mode – Used to handle exceptions. After the exceptions are finished, the processor returns to Thread mode.

6. References

6.1. https://toshiba.semicon-storage.com/eu/semiconductor/knowledge/e-learning/tx03-series-microcontrollers/chapter2/nested-vectored-interrupt-controller.html

6.2. https://www.geeksforgeeks.org/what-is-interrupt-latency/

6.3. https://microcontrollerslab.com/nested-vectored-interrupt-controller-nvic-arm-cortex-m/

6.4. https://open4tech.com/exception-context-switching-on-arm-cortex-m/