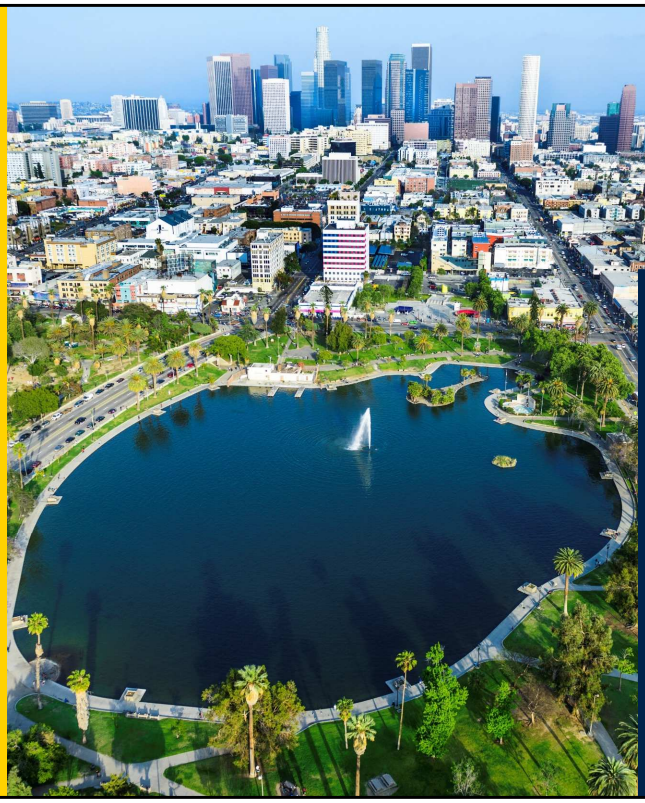


life.augmented

STM32U0 – NVIC

Nested Vectored Interrupt Controller

Hello, and welcome to this presentation of the STM32 nested vectored interrupt controller (NVIC). We will be presenting the features of this controller.

Overview

- The NVIC is integrated in the Cortex®-M0+ CPU:
  - 32 maskable interrupt channels
  - 4 programmable priority levels
  - Low-latency exception and interrupt handling
  - Power management control

Application benefits

- Supports prioritization levels with dynamic control
- Fast response to interrupt requests
- Relocatable vector table

life.augmented

2

The interrupt controller belongs to the Cortex®-M0+ CPU enabling a close coupling with the processor core.
The main features are:

- 32 interrupt sources
- 4 programmable priority levels
- low-latency exception and interrupt handling
- Automatic nesting
- Power management control

Applications can benefit from dynamic prioritization of the interrupt levels, fast response to the requests thanks to low latency responses and tail chaining as well as from vector table relocation.

Key features

- Fast response to interrupt requests
  - The number of interrupt request entries of the NVIC is 32
    - ARM Cortex®-M0+ limitation
    - However, the STM32U0 implements **more than 32 interrupt events** thanks to the STM32U0's SYSCFG module
      - Since the number of interrupt events exceeds 32 in the STM32U0, the SYSCFG unit is in charge of combining several interrupts onto the same interrupt line
    - User software can quickly determine the peripheral requesting the interrupt by reading ITLINEx registers in the SYSCFG module
- Dynamic reprioritization of interrupts
- Dynamic relocation of interrupt vector table

life.augmented

3

The NVIC provides a fast response to interrupt requests, allowing an application to quickly serve incoming events.
The ARM V6-M limits to 32 the number of interrupt request inputs of the NVIC.
However, the STM32U0 implements more interrupt events than 32.
A separate unit called SYSCFG is in charge of combining several interrupt events onto the same interrupt line.
By reading the ITLINE registers in this SYSCFG module, software can quickly determine the peripheral that has requested the interrupt.
The priority assigned to each interrupt request is programmable and can be dynamically changed.
The interrupt vector table can also be relocated, which allows the system designer to adapt the placement of interrupt service routines to the application's memory layout.
For instance, the vector table can be relocated in RAM.

Priority handling

- Regarding Cortex®-M CPUs exception management, the lower the value, the higher the priority

Exception source	Priority level	
Reset	-3	Fixed hardcoded priority
Non-Maskable Interrupt (NMI)	-2	
Hard Fault	-1	
Other exceptions including: ➢ Peripheral interrupts ➢ Software exceptions	Programmable level from 0 to 3	

life.augmented

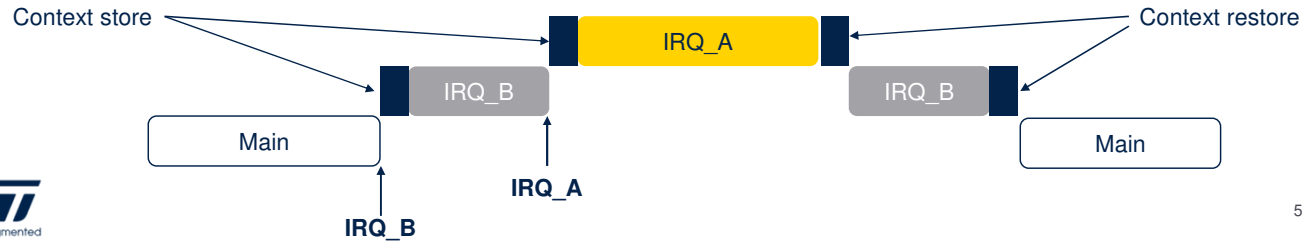
Software is in charge of assigning a priority level to each interrupt as well as to all exception sources not including reset, NMI and hard fault, which have a hardcoded priority. Whenever a peripheral interrupt is requested at the same time as a supervisor call instruction is executed, the relative priority of these hardware and software exceptions will dictate which one will be taken first.
Regarding the STM32U0, the NMI is caused by an SRAM parity error, a flash double ECC error or clock failure.
The priority of any of the 32 peripheral interrupt requests is programmable in a dedicated priority field located in Cortex®-M0+ NVIC registers.

Tail-chaining and nesting

- In order to explain the tail-chaining and nesting mechanism, let us consider the following peripheral interrupt sources:

Interrupt source	Priority level
IRQ_A	0
IRQ_B	1

- Preemption and interrupt nesting



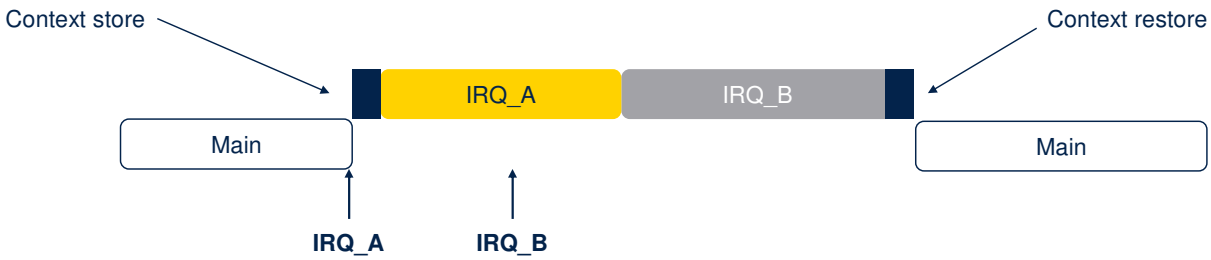
life.augmented

5

The NVIC provides several features for efficient handling of exceptions.
Let us consider two interrupt requests: A and B. IRQ A has a higher priority than IRQ B.
When an interrupt is served and a new request with higher priority arrives, the new exception can preempt the current one.
This is called nested exception handling.
In this example, IRQ A preempts the execution of the interrupt service routine related to IRQ B.
The previous exception handler resumes execution after the higher priority exception is handled.
A microcode present in the Cortex®-M0+ automatically pushes the context to the current stack and restores it upon interrupt return.

Exception entry and return

- Tail-chaining
  - When an interrupt is pending on the completion of an exception handler, the context store is skipped and the control is immediately transferred to the new exception handler when the previous handler is completed



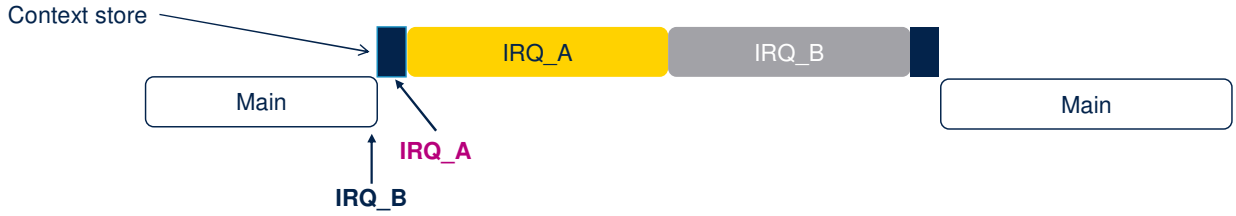
life.augmented

6

When an interrupt request with lower or equal priority is raised during execution of an Interrupt Service Routine (ISR), it becomes pending.
Once the current ISR is finished, the context saving and restoring process is skipped and control is transferred directly to the new exception handler to decrease interrupt latency.
So back-to-back interrupts with decreasing priorities (higher priority values) are chained with a very short latency of a few clock cycles.
In this example, IRQ A is taken first. Since its priority is higher (lower priority value) than IRQ B, execution of the ISR keeps running when IRQ B is requested.
Once the ISR of IRQ A Is completed, then tail-chaining occurs prior to executing the ISR or IRQ B.

Exception entry and return

- Late-arriving
  - When a higher-priority exception occurs during state-saving for a previous exception, the processor switches to handle the higher-priority exception immediately



- Return
  - When the exception handler is finished and no other exception is pending, the processor pops the stack and restores the program state before the interrupt occurred

life.augmented

When an interrupt arrives, the processor first saves the program context before executing the interrupt handler.
If the processor is performing this context-saving operation when an interrupt of higher priority arrives, the processor switches directly to handling the higher-priority interrupt when it is finished saving the program context.
Then tail-chaining will be used prior to executing the IRQ_B interrupt service routine.
When all of the exception handlers have been run and no other exception is pending, the processor restores the previous context from the stack and returns to normal application execution.

NVIC tricks and tips

- Ensure software uses correctly-aligned register accesses
- An interrupt can become pending even if disabled
  - Disabling an interrupt only prevents the processor from taking that interrupt
- Before relocating the vector table, ensure new entries are correctly set up for all enabled interrupts
  - This includes fault handlers and NMIs
  - Do this before programming the VTOR register to relocate the vector table

life.augmented

When accessing the NVIC registers, ensure that your code uses a correctly-aligned register access.
Unaligned access is not supported for NVIC registers as well as all memory-mapped registers located in the Cortex®-M0+.
An interrupt becomes pending when the source asks for service.
Disabling the interrupt only prevents the processor from taking that interrupt. Make sure the related interrupt pending flag is cleared before enabling the interrupt vector.
Before relocating the vector table using the VTOR register, ensure that fault handlers, NMI and all enabled interrupts are correctly set up on the new location.

Related peripherals

- Refer to the training material for the following peripherals linked to the timers:
  - SYSCFG
    - It is in charge of, among other things, combining interrupt sources generated by peripherals into interrupt request signals connected to the NVIC
  - Cortex®-M0+
    - The CPU implements an exception mechanism used to handle both software and hardware exceptions

life.augmented

The NVIC is linked with the SYSCFG module and the Cortex-M0+ CPU. Please refer to the related presentations.

References

- For more details, please refer to the following documents:
  - PM0223 Programming manual for Cortex®-M0+
  - STM32U0 reference manuals

life.augmented

For detailed information, please refer to the programming manual for the Cortex®-M0+ core and the reference manual of the STM32U0.

Thank you

life.augmented

Thanks for attending this presentation.