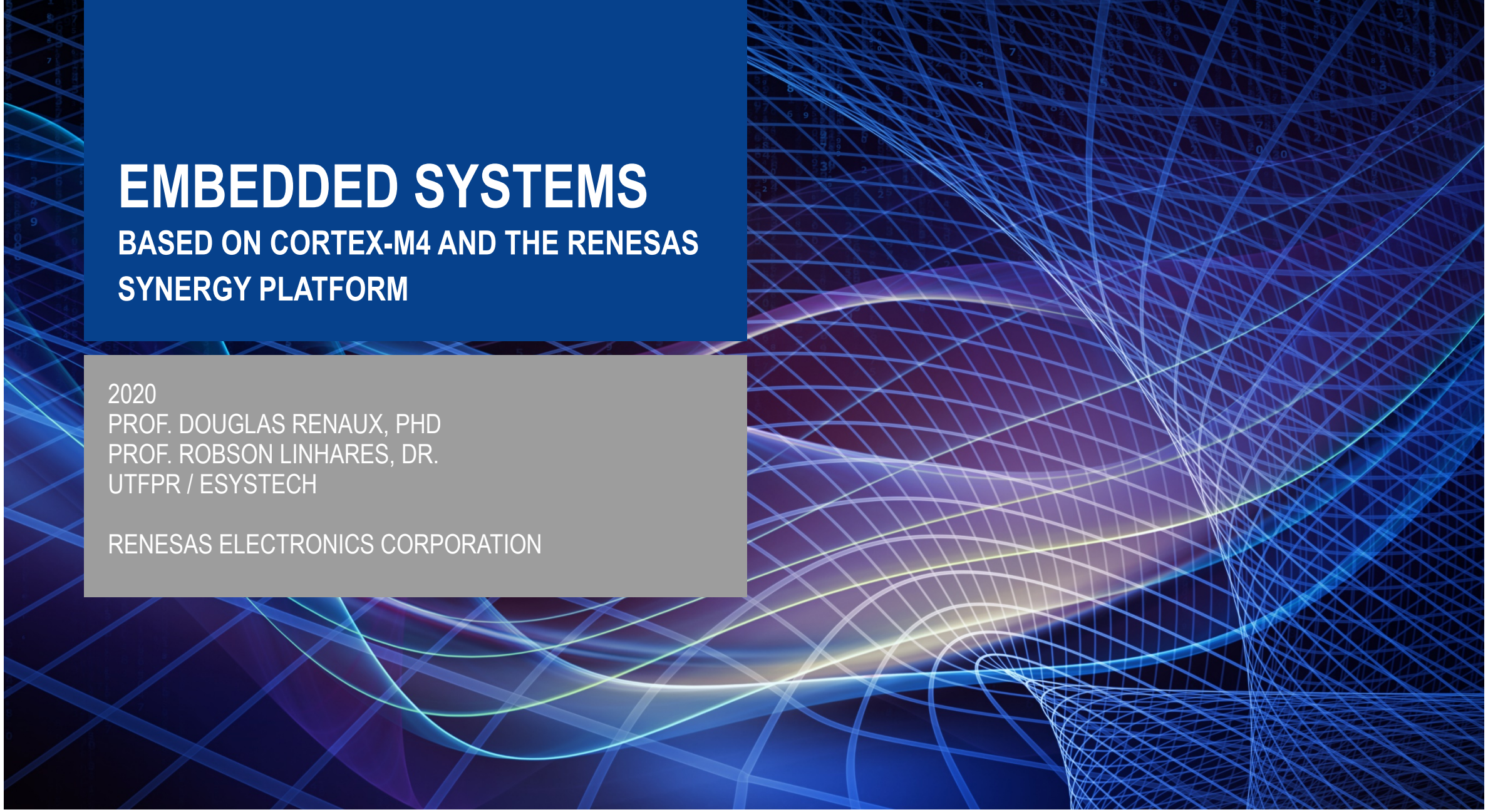


# EMBEDDED SYSTEMS

## BASED ON CORTEX-M4 AND THE RENESAS SYNERGY PLATFORM

2020
PROF. DOUGLAS RENAUX, PHD
PROF. ROBSON LINHARES, DR.
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

# 6 – INTERRUPT CONTROLLER

- NVIC
  - Structure
  - Registers
- CMSIS interface for NVIC operations

# NVIC

The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M4 architecture. Hence, all Cortex-M4 have the same interrupt controller.

Exception handling was examined in Section 3 (link)

Functionality of the NVIC:

- **Detect** interrupt requests (IRQ) at its inputs and **combine** them into a single interrupt request to the microprocessor core
- Capability to **mask** any given input
- Associate inputs to interrupt **priority** levels

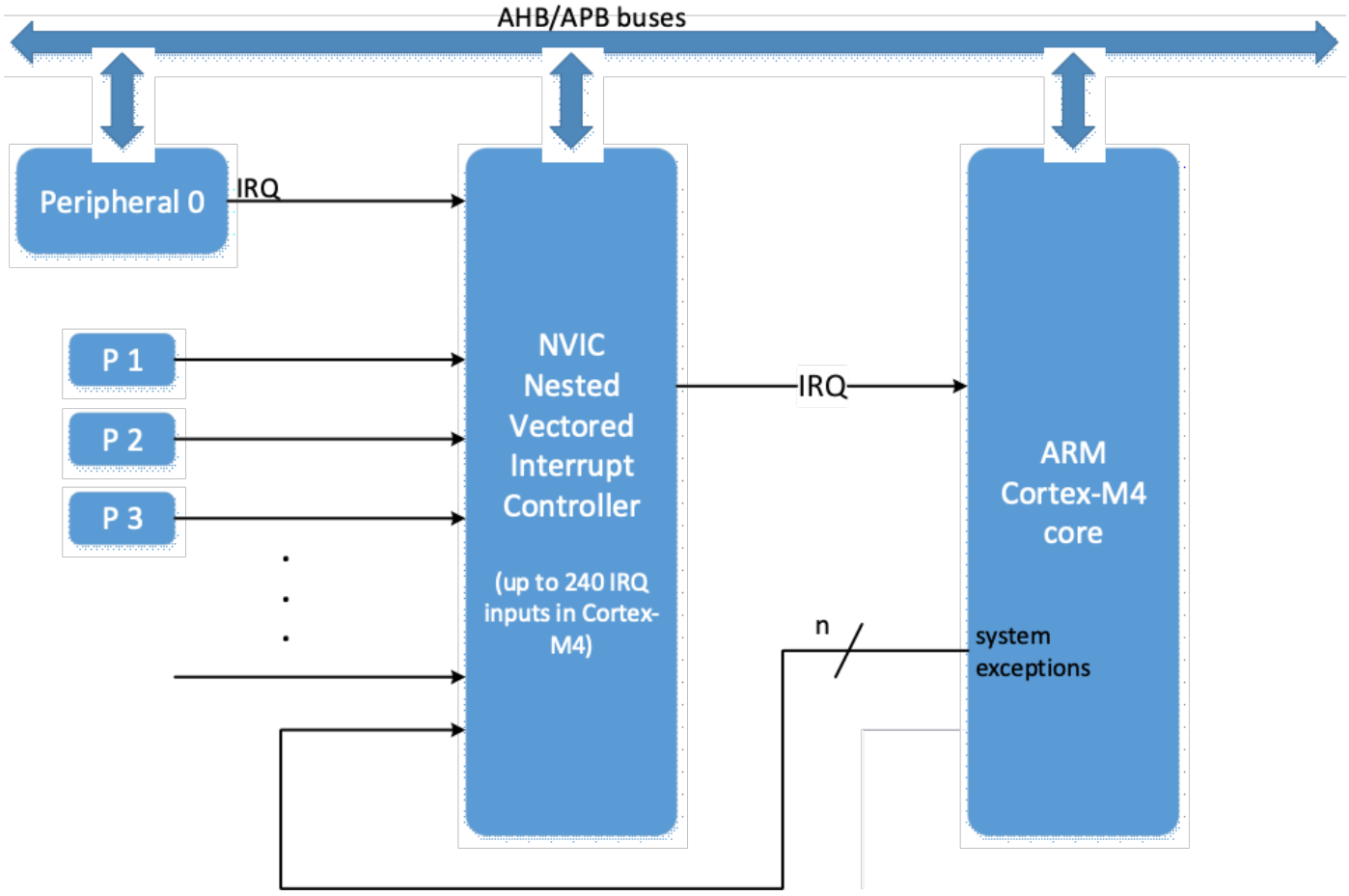
# NVIC

The Cortex-M4 NVIC may have up to 240 IRQ inputs.

Each input is an interrupt request coming from an integrated peripheral, an external pin, or even from the core itself to inform of a system exception.

The operation of the NVIC is configured by the core via a set of NVIC registers that provide the functionality of:

- input masking
- IRQ priority assignment
- check status of input
  (is there a pending IRQ?)




source: Authors

# NVIC CHARACTERISTICS

The Nested Vectored Interrupt Controller provides support for:

- **Nested exceptions** – each exception has an associated priority level. Nesting means that when an exception is being serviced, it can be interrupted by a higher priority exception that was activated. Once the higher priority exception has its service completed, the lower priority service resumes.
- **Vectored exceptions** – the starting addresses of the service routines (handlers) are stored in a table (vector table). When IRQi is detected the processor simply reads entry i of the table and starts servicing, avoiding delays to start the handler.
- **Interrupt masking** – each IRQi can be individually masked, i.e. ignored.

# VECTOR TABLE

By default, its start address is 0x0.

The table has a 31-bit address for each of the possible exceptions (number is implementation dependent).

The 32th bit (the LSb) is used to identify the instruction set (ARM or Thumb) and must be always set in a Cortex-M processor

The first entry in the table is the initial value of the SP.

The next 15 entries are for the system exceptions
(Reset, NMI, … SysTick).

Then follow up to 240 entries for IRQ0 to IRQ239.

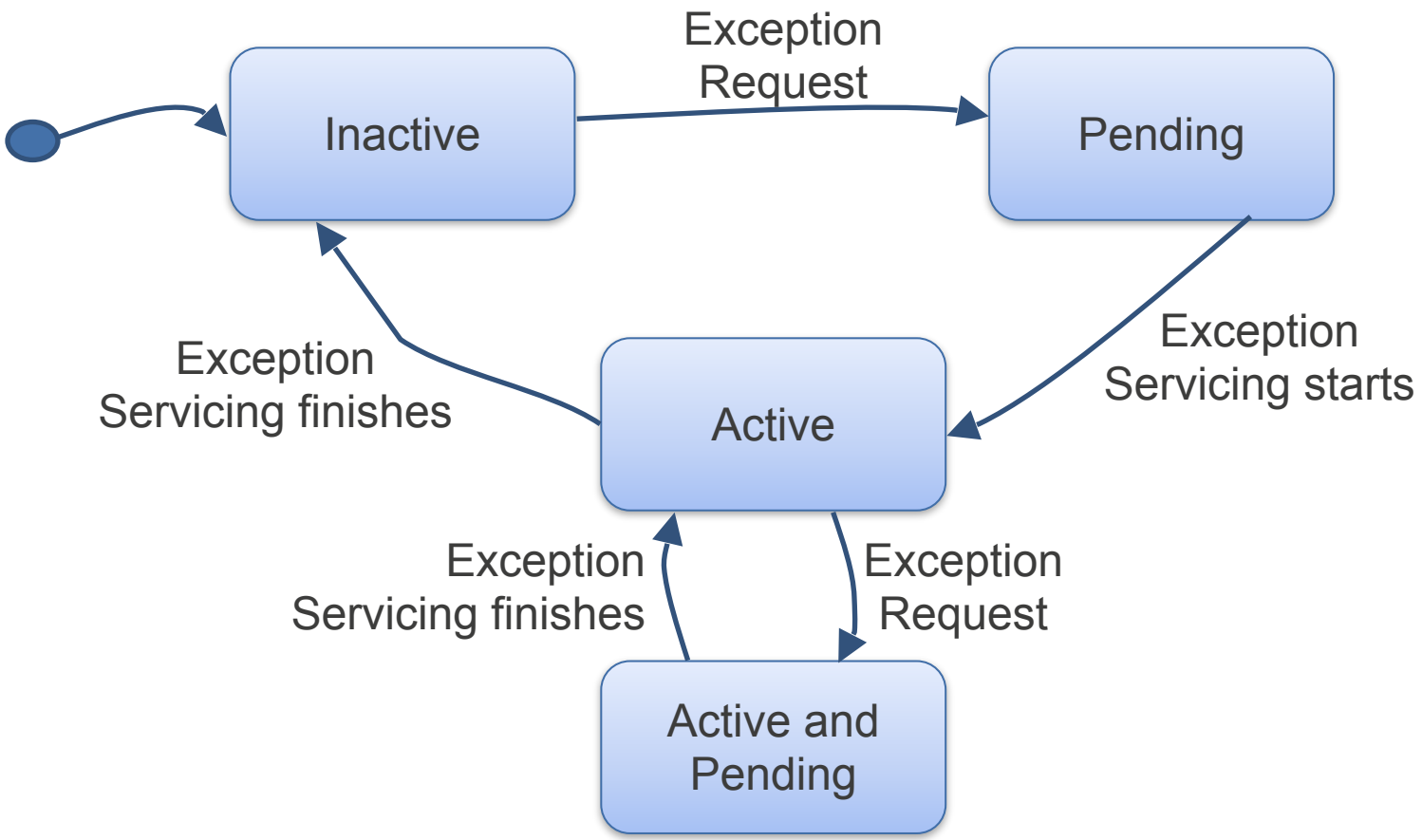
# VECTOR TABLE FOR THE RENESAS S7G2

On the S7G2 the interrupt sources are managed by the ICU – Interrupt Control Unit.
The ICU sits between the peripherals and the NVIC, preprocessing the interrupt requests before sending them to the NVIC.

**Table 14.3 Interrupt vector table (1/3)**

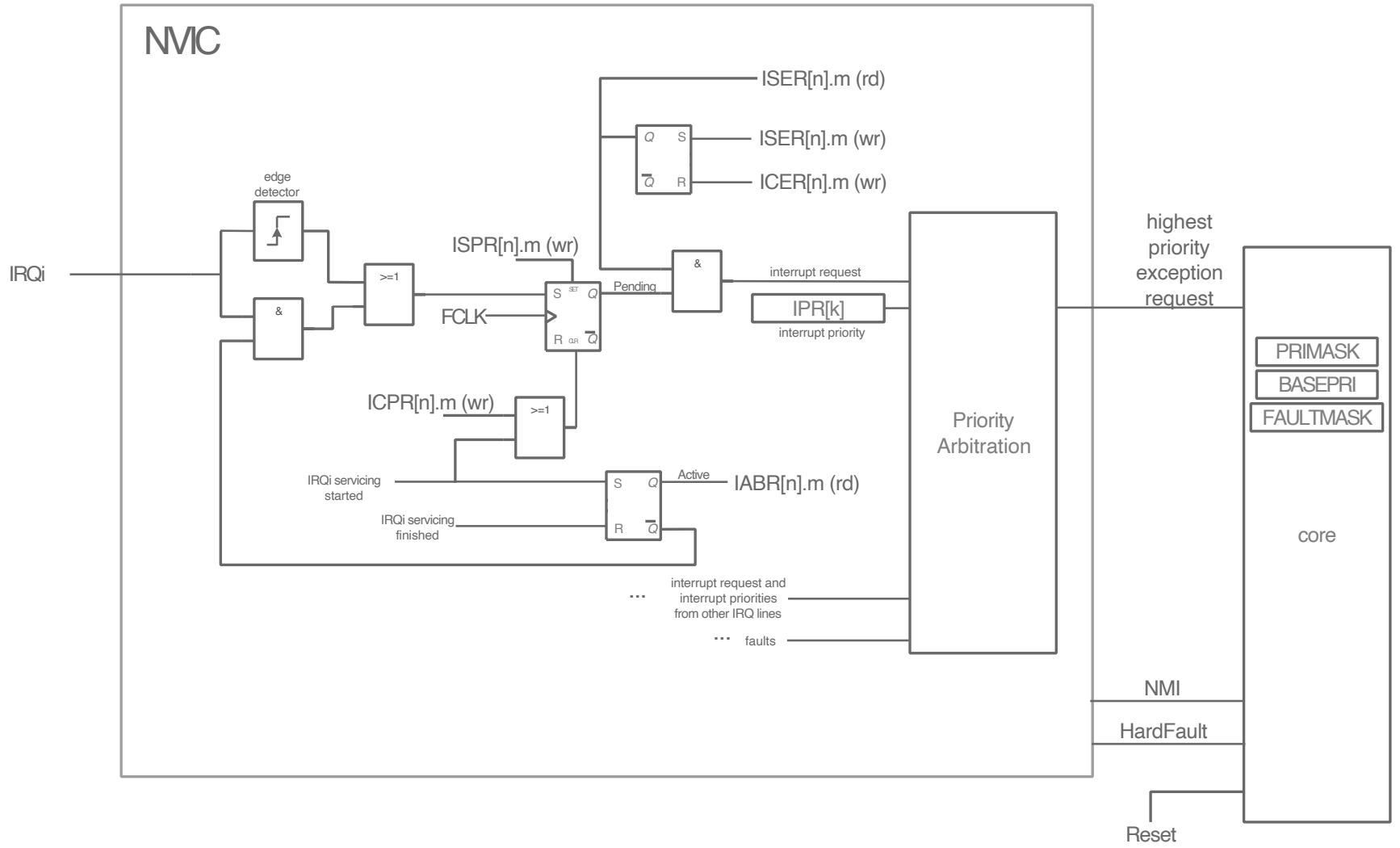
| Exception number | IRQ number | Vector offset | Source | Description |
|---|---|---|---|---|
| 0 | — | 000h | ARM | Initial stack pointer |
| 1 | — | 004h | ARM | Initial program counter (Reset Vector) |
| 2 | — | 008h | ARM | Non-maskable interrupt (NMI) |
| 3 | — | 00Ch | ARM | Hard fault |
| 4 | — | 010h | ARM | MemManage fault |
| 5 | — | 014h | ARM | Bus fault |
| 6 | — | 018h | ARM | Usage fault |
| 7 | — | 01Ch | ARM | Reserved |
| 8 | — | 020h | ARM | Reserved |
| 9 | — | 024h | ARM | Reserved |
| 10 | — | 028h | ARM | Reserved |
| 11 | — | 02Ch | ARM | Supervisor call (SVCall) |
| 12 | — | 030h | ARM | Debug Monitor |
| 13 | — | 034h | ARM | Reserved |
| 14 | — | 038h | ARM | Pendable request for system service (PendableSrvReq) |
| 15 | — | 03Ch | ARM | System tick timer (SysTick) |
| 16 | 0 | 040h | ICU.IELSR0 | Event selected in the ICU.IELSR0 register |
| 17 | 1 | 044h | ICU.IELSR1 | Event selected in the ICU.IELSR1 register |
| 18 | 2 | 048h | ICU.IELSR2 | Event selected in the ICU.IELSR2 register |
| 19 | 3 | 04Ch | ICU.IELSR3 | Event selected in the ICU.IELSR3 register |
| ▪ ▪ ▪ | | | | |
| 107 | 91 | 1ACh | ICU.IELSR91 | Event selected in the ICU.IELSR91 register |
| 108 | 92 | 1B0h | ICU.IELSR92 | Event selected in the ICU.IELSR92 register |
| 109 | 93 | 1B4h | ICU.IELSR93 | Event selected in the ICU.IELSR93 register |
| 110 | 94 | 1B8h | ICU.IELSR94 | Event selected in the ICU.IELSR94 register |
| 111 | 95 | 1BCh | ICU.IELSR95 | Event selected in the ICU.IELSR95 register |

# EXCEPTION HANDLING STATE MACHINE




source: Authors

# FUNCTIONAL MODEL OF THE NVIC




source: Authors

# NVIC – ARM DOCUMENTATION

- ARMv7-M Architecture Reference Manual (2014)
  ARM DDI 0403E.b
- ARM® Cortex® -M4 Processor Revision: r0p1 (2015)
  Technical Reference Manual
  ARM 100166_0001_00_en
- Cortex™-M4 Devices Generic User Guide (2011)
  ARM DUI 0553A

# NVIC REGISTERS

**Table 6-1 NVIC registers**

| Address | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0xE000E004 | ICTR | RO | - | Interrupt Controller Type Register, ICTR |
| 0xE000E100 - 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 - 0xE000E19C | NVIC_ICER0 - NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 - 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280- 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400- 0xE000E4EC | NVIC_IPR0 - NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

source: ARM® Cortex® -M4 Technical Reference Manual

# NVIC REGISTERS – ICTR

Since the number of input lines (IRQi) to the NVIC is implementation dependent, so is the number of registers to control the NVIC.
ICTR informs how many control registers are actually implemented in any given NVIC.

For the Renesas S7G2, ICTR.INTLINESNUM holds the value 2, meaning that there are 3 registers of each type (ISER, ICER, ISPR, ICPR, IABR, with indexes 0..2; and 24 registers IPR, with indexes 0..23). The number of input lines IRQi on this processor is limited to 96.

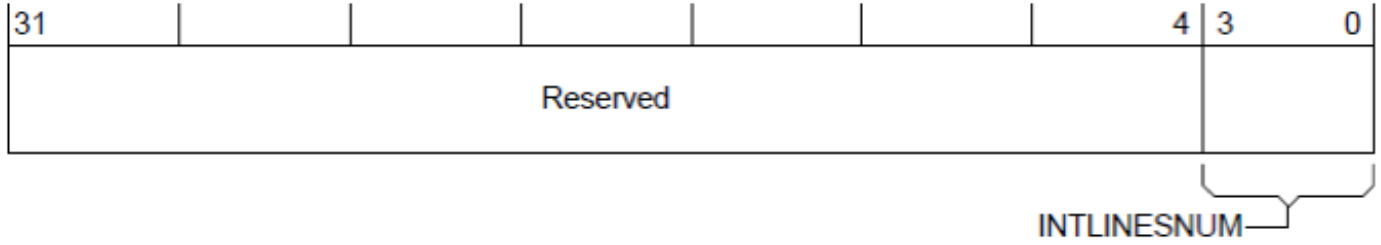
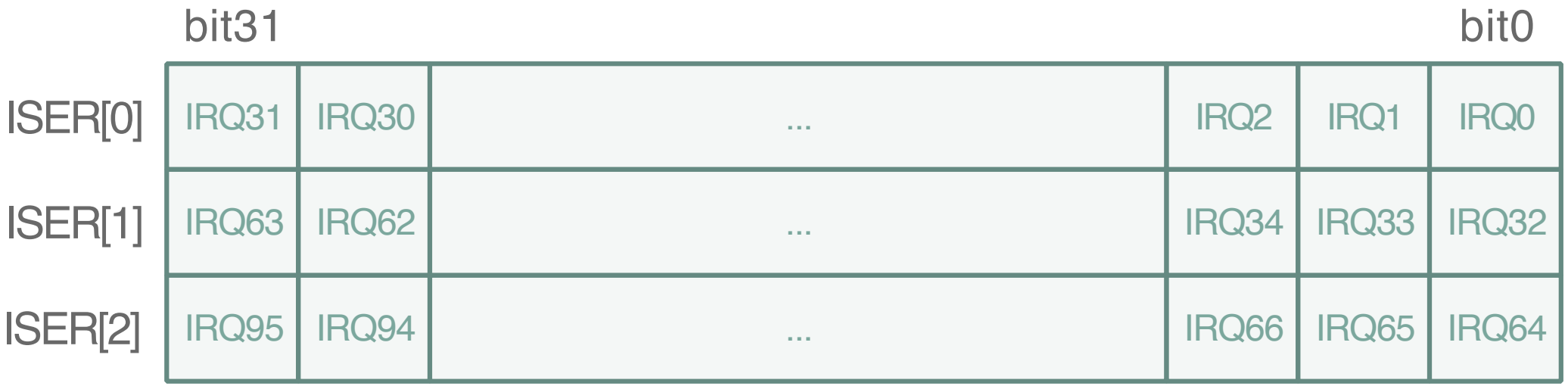



**Figure 6-1 ICTR bit assignments**

| | | |
|---|---|---|
| [3:0] | INTLINESNUM | Total number of interrupt lines in groups of 32:<br>0b0000 = 0...32<br>0b0001 = 33...64<br>0b0010 = 65...96<br>0b0011 = 97...128<br>0b0100 = 129...160<br>0b0101 = 161...192<br>0b0110 = 193...224<br>0b0111 = 225...256 |

source: ARM® Cortex® -M4 Technical Reference Manual

# NVIC REGISTERS – ISER – S7G2



| | bit31 | | | | | bit0 |
|---|---|---|---|---|---|---|
| ISER[0] | IRQ31 | IRQ30 | ... | IRQ2 | IRQ1 | IRQ0 |
| ISER[1] | IRQ63 | IRQ62 | ... | IRQ34 | IRQ33 | IRQ32 |
| ISER[2] | IRQ95 | IRQ94 | ... | IRQ66 | IRQ65 | IRQ64 |

In a Renesas S7G2, each bit of registers ISER[0] to ISER[2] corresponds to one of the 96 IRQi lines.

on write: 1 enables the interrupt line, 0 has no effect

on read: returns the current state of each IRQi line

0 = disabled, 1 = enabled

# NVIC REGISTERS – ICER/ISPR/ICPR/IABR – S7G2

Same bit assignment as for ISER.

ICER: on write: 1 **disables** the interrupt line, 0 has no effect

on read: report the current state of each IRQi line,  0 = disabled, 1 = enabled

ISPR: on write: 1 sets the interrupt line to **pending**, 0 has no effect

on read: report the current state of each IRQi line,  0 = not-pending, 1 = pending

ICPR: on write: 1 clears the pending state of the interrupt line, 0 has no effect

on read: report the current state of each IRQi line, 0 = not-pending, 1 = pending

IABR:  read only: report the current state of each IRQi line, 0 = not-active, 1 = active

# NVIC PRIORITIES

On the Cortex-M4, each IRQi line has a register that defines its priority level in relation to the other IRQi lines. This register may have up to 8 bits (implementation defined).

On the S7G2, 4-bit registers are implemented, allowing the definition of up to 16 priority levels. 0 is the highest priority and 15 is the lowest. Since these bits are left aligned, the actual values are 0, 0x10, 0x20, 0x30, … 0xF0.

**A higher priority interrupt preempts a lower priority interrupt whose handler is being executed.**

# NVIC PRIORITIES

The bits of the priority register are divided into:

- Priority Group
- Priority Subgroup

The bits in the Priority Group define the number of priority levels for the purpose of preemption.

The bits in the Priority Subgroup define the number of sub-levels for the purpose of selecting which IRQi will be serviced first if two IRQi are simultaneously pending when the core accepts an interrupt.

# NVIC PRIORITIES

The PRIGROUP bits of the AIRCR register are used to configure the division of bits among Priority Group and Subgroup.

AIRCR.PRIGROUP may be written with values in the range of 0 .. 7

On the S7G2, programming PRIGROUP with 0, 1, 2 or 3 has the effect of using the four bits for Priority Group and none for the Subgroup.

For PRIGROUP = 4 the division is 3.1 (3 for Group and 1 for Subgroup)

For PRIGROUP = 5 the division is 2.2, and so on…

# CMSIS-CORE INTERRUPT FUNCTIONS

These is a partial list of the functions available in
CMSIS-CORE that provide access to the NVIC as well
as to other interrupt functionality.

Next slides detail these functions

| CMSIS function | Description |
|---|---|
| `void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)` | Set priority grouping |
| `uint32_t NVIC_GetPriorityGrouping (void)` | Read the priority grouping |
| `void NVIC_EnableIRQ (IRQn_Type IRQn)` | Enable a device-specific interrupt |
| `uint32_t NVIC_GetEnableIRQ (IRQn_Type IRQn)` | Get a device-specific interrupt enable status. |
| `void NVIC_DisableIRQ (IRQn_Type IRQn)` | Disable a device-specific interrupt |
| `uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)` | Get the pending device-specific interrupt |
| `void NVIC_SetPendingIRQ (IRQn_Type IRQn)` | Set a device-specific interrupt to pending |
| `void NVIC_ClearPendingIRQ (IRQn_Type IRQn)` | Clear a device-specific interrupt from pending |
| `uint32_t NVIC_GetActive (IRQn_Type IRQn)` | Get the device-specific interrupt active |
| `void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)` | Set the priority for an interrupt |
| `uint32_t NVIC_GetPriority (IRQn_Type IRQn)` | Get the priority of an interrupt |

source: infocenter.arm.com

# CMSIS-CORE INTERRUPT FUNCTIONS

```
typedef enum {
/* ------------------  Cortex-M4 Processor Exceptions Numbers  ------------------- */
  Reset_IRQn             = -15,      /*!<   1  Reset Vector, invoked on Power up and warm reset                   */
  NonMaskableInt_IRQn    = -14,      /*!<   2  Non maskable Interrupt, cannot be stopped or preempted             */
  HardFault_IRQn         = -13,      /*!<   3  Hard Fault, all classes of Fault                                   */
  MemoryManagement_IRQn  = -12,      /*!<   4  Memory Management, MPU mismatch, including Access Violation
                                                and No Match                                                     */
  BusFault_IRQn          = -11,      /*!<   5  Bus Fault, Pre-Fetch-, Memory Access Fault, other address/memory
                                                related Fault                                                    */
  UsageFault_IRQn        = -10,      /*!<   6  Usage Fault, i.e. Undef Instruction, Illegal State Transition      */
  SVCall_IRQn            =  -5,      /*!<  11  System Service Call via SVC instruction                           */
  DebugMonitor_IRQn      =  -4,      /*!<  12  Debug Monitor                                                     */
  PendSV_IRQn            =  -2,      /*!<  14  Pendable request for system service                               */
  SysTick_IRQn           =  -1,      /*!<  15  System Tick Timer                                                 */
} IRQn_Type;
```

This is the IRQn_Type enumeration defined in file S7G2.h

# CMSIS-CORE INTERRUPT FUNCTIONS

`void __enable_irq(void)`

Resets PRIMASK in the core, allowing interrupts from NVIC to be serviced.

Example: `__enable_irq( );`

`void __disable_irq(void)`

Sets PRIMASK in the core, preventing exceptions with configurable priorities (exceptions 4 and up) to be serviced.

Example: `__disable_irq( );`

# CMSIS-CORE INTERRUPT FUNCTIONS

`void NVIC_EnableIRQ(IRQn_Type)`

Enables (**unmask**) a specific IRQi line

Example: `NVIC_Enable(SysTick_IRQn);`

`void NVIC_DisableIRQ(IRQn_Type)`

Disables (**mask**) a specific IRQi line

Example: `NVIC_Disable(SysTick_IRQn);`

# CMSIS-CORE INTERRUPT FUNCTIONS

`uint32_t NVIC_GetPendingIRQ(IRQn_Type)`

Reads the Pending status (P_FF) returning 0 (not pending) or 1 (pending)

Example: `uint32_t pend = NVIC_GetPendingIRQ(SysTick_IRQn);`

`void NVIC_SetPendingIRQ(IRQn_Type)`

Sets the Pending status (P-FF) of a specific IRQi line

Example: `NVIC_SetPendingIRQ(SysTick_IRQn);`

`void NVIC_ClearPendingIRQ(IRQn_Type)`

Clears (resets) the Pending status (P-FF) of a specific IRQi line

Example: `NVIC_ClearPendingIRQ(SysTick_IRQn);`

# CMSIS-CORE INTERRUPT FUNCTIONS

```
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
```

Sets the priority of a specific IRQi line

Example: `NVIC_SetPriority(SysTick_IRQn,4);`

```
uint32_t NVIC_GetPriority(IRQn_Type IRQn)
```

Get the priority level of a specific IRQi line

Example: `uint32_t prio = NVIC_GetPriority(SysTick_IRQn);`

Rem: The integer value of the priority must be in the range of 0 .. 2N-1, where

N is the number of priority bits implemented. For S7G2 the range is 0..15.

# CMSIS-CORE INTERRUPT FUNCTIONS

`uint32_t NVIC_GetActive(IRQn_Type)`

Reads the Active status (ACTIVE_FF) returning 0 (not active) or 1 (active)

Example: `uint32_t act = NVIC_ GetActive(SysTick_IRQn);`

Renesas.com