life.augmented

TFM NS NOTIF

M JAOUEN

December 2025

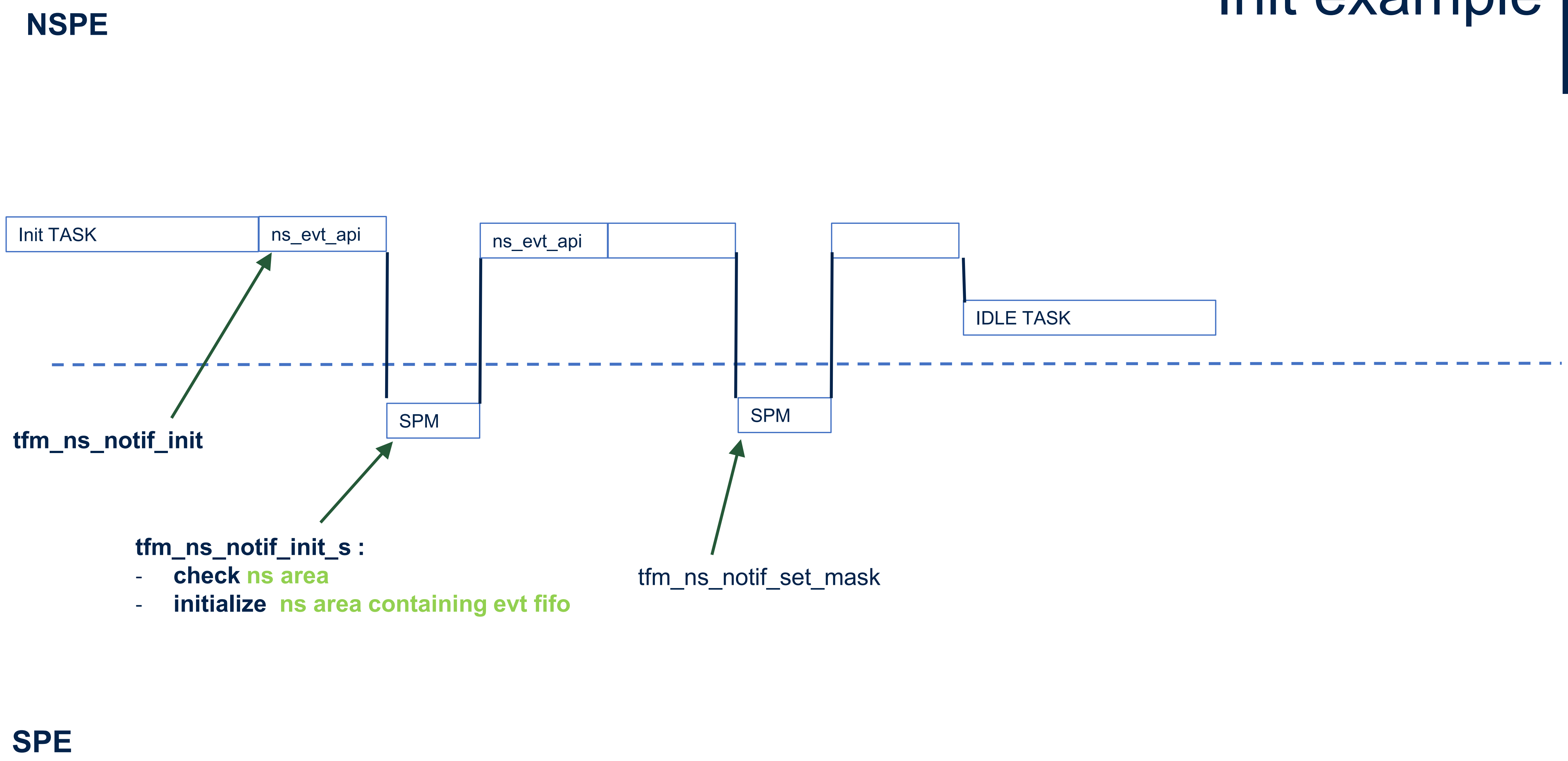
Constraint use case with TF-M :

-A- an asynchronous secure interrupt requires an execution context but the non secure application has a time constraint process
-B- secure service requires the usage of function implemented in non secure
(i.e flash driver for protected storage, non secure requires flash access)

NS Notif :
Provide an api to send an asynchronous 32 bits event mask to non Secure from Secure

life.augmented

Init example

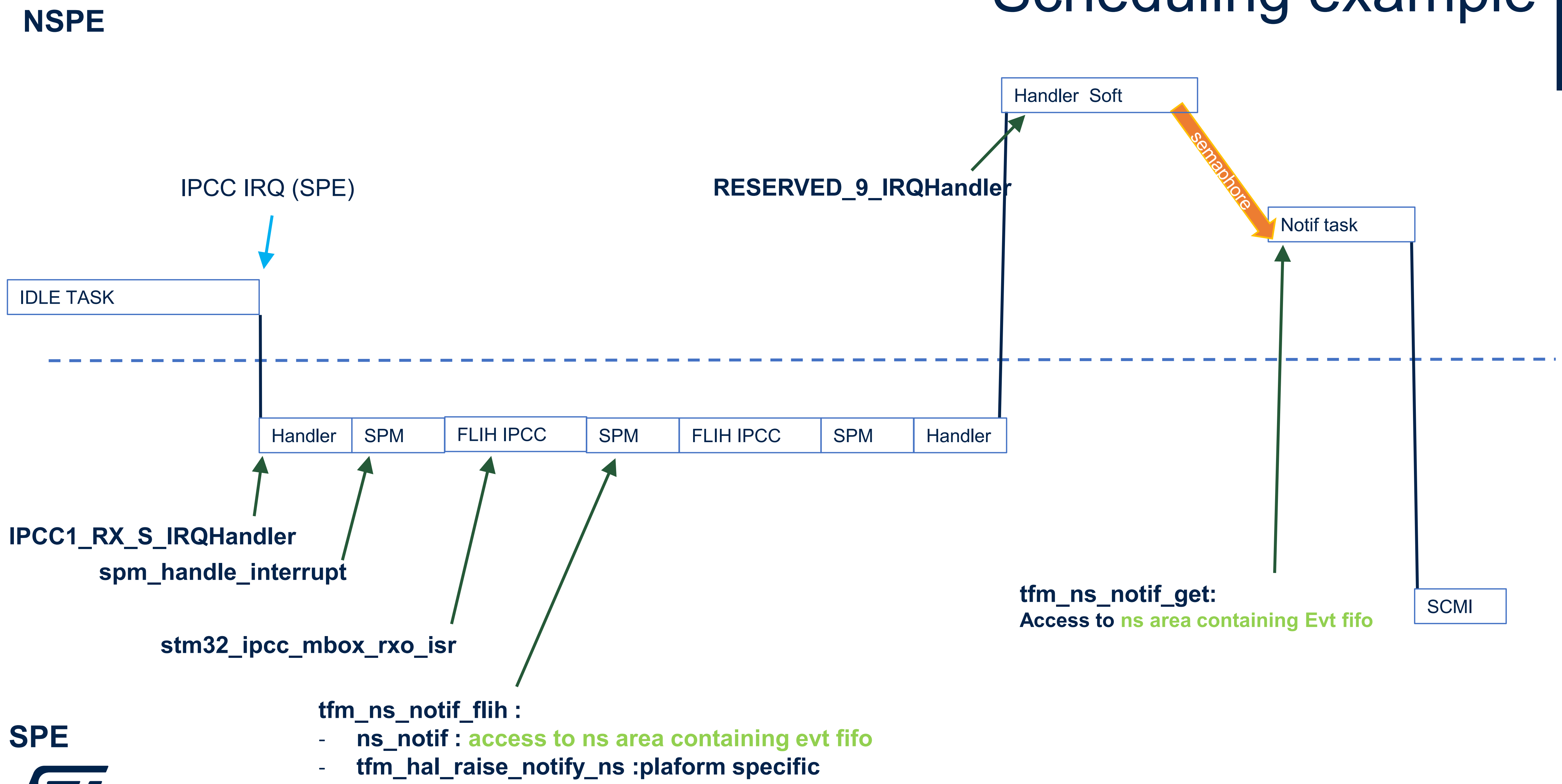


tfm_ns_notif_init_s :

- **check** ns area
- **initialize** ns area containing evt fifo

life.augmented

Scheduling example



tfm_ns_notif_flih :
- **ns_notif : access to ns area containing evt fifo**
- **tfm_hal_raise_notify_ns :plaform specific**

life.augmented

NS API

int32_t tfm_ns_notif_init(void *area, size_t area_size)

=>Define the Non Secure event memory FIFO

Int32_t tfm_ns_notif_set_mask(uint32_t event_mask)

=> Define the event Handled by Non Secure

int32_t tfm_ns_notif_get(uint32_t *event)

=> Get an event from Non Secure event memory FIFO

Use by Non secure to unqueue an event mask from FIFO see NS integration

NS Integration

An Interrupt Handler is used in non secure (in stm32mp2 handler is RESERVED_9_IRQHandler)

On integration with RTOS , this handler posts a semaphore.

The task waiting for the semaphore call api tfm_ns_notif_get and then call the service according

To received event.

File: platform/ext/target/stm/common/stm32mp2xx/tfm_ns_plat_init.c

Secure API

psa_status_t tfm_ns_notif_flih(uint32_t event) => post an event from an FILH Handler

psa_status_t tfm_ns_notif(uint32_t event)=> post an event from a partition

The 32 bits events are associated to a partition and only the event owned by partition can be post

By a partition or to by the FILH handler from the IRQ handle by a partition.

=> config is done in .yaml from a partition see next slide

An file ns_evt.h is generated to associate the name of the event to a bit in 32 bits mask event

Ns_evt.h is exported to non secure.

Secure API

tfm_ipcc.yaml

```
{
  "psa_framework_version": 1.1,
  "name": "TFM_SP_IPCC",
  "type": "PSA-ROT",
  "priority": "NORMAL",
  "model": "SFN",
  "entry_init": "tfm_ipcc_entry",
  "stack_size": "IPCC_STACK_SIZE",
  "irqs": [
   {
     "source": "IPCC_IRQ",
     "name": "IPCC",
     "handling": "FLIH"
   }
],
```

```
"ns_evts": [
{
  "name": "RSE"
 },
{
  "name": "SCMI_CA35"
},
{
  "name": "SCMI_CA35_BL31"
}
]
}
```

ns_evt.h

```
#ifndef __NS_EVT_H__
#define __NS_EVT_H__
#ifdef __cplusplus
extern "C" {
#endif
#define TFM_SP_IPCC_RSE_NS_EVT                   (0x80000000)
#define TFM_SP_IPCC_SCMI_CA35_NS_EVT             (0x40000000)
#define TFM_SP_IPCC_SCMI_CA35_BL31_NS_EVT        (0x20000000)

#ifdef __cplusplus
}
#endif

#endif /* __NS_EVT_H__ */
```

Secure Integration for platform

```
set(PLATFORM_HAS_NS_NOTIF          ON               CACHE BOOL   "Plaform implements hal for enabling the ns notifcation from the secure")
```

```
uint32_t tfm_hal_notify_ns_init(void);

uint32_t tfm_hal_raise_notify_ns(void);
```

Questions / Answers

life.augmented

ANNEX :NS weak API

void tfm_ns_notif_listener(uint32_t event);

=>call in function tfm_ns_notif_get , populates a variable containing all event beeing received

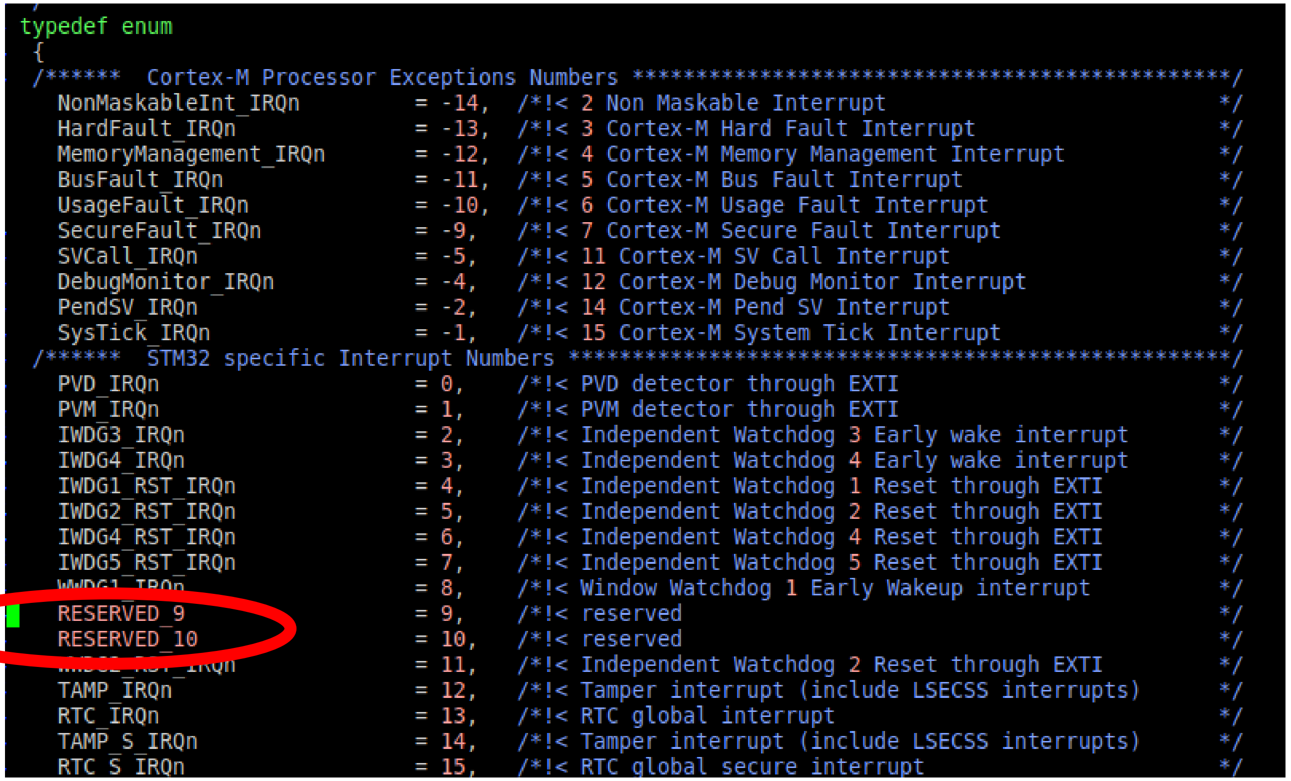
int32_t tfm_ns_notif_get_pending(uint32_t val)

=>call by a function in non secure to know if an event has been received

When event is present, event is returned and event is cleared variable containing all event beeing received.

life.augmented

ANNEX : NVIC usage



```
typedef enum
 {
 /******  Cortex-M Processor Exceptions Numbers ****************************************************/
   NonMaskableInt_IRQn         = -14,   /*!< 2 Non Maskable Interrupt                                  */
   HardFault_IRQn              = -13,   /*!< 3 Cortex-M Hard Fault Interrupt                           */
   MemoryManagement_IRQn       = -12,   /*!< 4 Cortex-M Memory Management Interrupt                    */
   BusFault_IRQn               = -11,   /*!< 5 Cortex-M Bus Fault Interrupt                            */
   UsageFault_IRQn             = -10,   /*!< 6 Cortex-M Usage Fault Interrupt                          */
   SecureFault_IRQn            = -9,    /*!< 7 Cortex-M Secure Fault Interrupt                         */
   SVCall_IRQn                 = -5,    /*!< 11 Cortex-M SV Call Interrupt                             */
   DebugMonitor_IRQn           = -4,    /*!< 12 Cortex-M Debug Monitor Interrupt                       */
   PendSV_IRQn                 = -2,    /*!< 14 Cortex-M Pend SV Interrupt                             */
   SysTick_IRQn                = -1,    /*!< 15 Cortex-M System Tick Interrupt                         */
 /******  STM32 specific Interrupt Numbers *********************************************************/
   PVD_IRQn                    = 0,     /*!< PVD detector through EXTI                                 */
   PVM_IRQn                    = 1,     /*!< PVM detector through EXTI                                 */
   IWDG3_IRQn                  = 2,     /*!< Independent Watchdog 3 Early wake interrupt               */
   IWDG4_IRQn                  = 3,     /*!< Independent Watchdog 4 Early wake interrupt               */
   IWDG1_RST_IRQn              = 4,     /*!< Independent Watchdog 1 Reset through EXTI                 */
   IWDG2_RST_IRQn              = 5,     /*!< Independent Watchdog 2 Reset through EXTI                 */
   IWDG4_RST_IRQn              = 6,     /*!< Independent Watchdog 4 Reset through EXTI                 */
   IWDG5_RST_IRQn              = 7,     /*!< Independent Watchdog 5 Reset through EXTI                 */
   WWDG1_IRQn                  = 8,     /*!< Window Watchdog 1 Early Wakeup interrupt                  */
   RESERVED_9                  = 9,     /*!< reserved                                                  */
   RESERVED_10                 = 10,    /*!< reserved                                                  */
   WWDG2_RST_IRQn              = 11,    /*!< Independent Watchdog 2 Reset through EXTI                 */
   TAMP_IRQn                   = 12,    /*!< Tamper interrupt (include LSECSS interrupts)              */
   RTC_IRQn                    = 13,    /*!< RTC global interrupt                                      */
   TAMP_S_IRQn                 = 14,    /*!< Tamper interrupt (include LSECSS interrupts)              */
   RTC_S_IRQn                  = 15,    /*!< RTC global secure interrupt                               */
```

Unused interrupt from vector can be selected to implement soft IRQ