Trusted Firmware – M Interrupt Handling
Content

- Secure Interrupt Handling in library model
- Secure Interrupt Handling in IPC model
- Interactions
- IDLE processing

This topic is the prologue of incoming designs. Interrupt has to be related with scheduling.
Secure Interrupt Handling in library model

Time Line & Priority Level

0
SVC

X < 255
ISR → SVC

Priority = X
SP ISR

256
SP

SECURE HANDLER
SECURE THREAD

S-ISR
SVC to Create SP ISR ctx
To SPISR
EOI
To S-ISR
To SP
Secure Interrupt Handling in IPC model

NOTE:
1. In further discussion the SCHEDULER in PendSV is not listed individually.
2. Assume the non-secure scheduler ISR/SCHEDULER is running under Handler mode.
1. Secure Handler is not preemptable to avoid secure execution stalling.
2. Preempt a non-secure Handler and do scheduling may cause NS stalls.
3. MSP_NS keeps increasing due to un-released execution.
Interactions – Preempts S/NS Thread Execution

1. May lack of NS context containers – need to **allocate** one context container if SPE don’t know which non-secure thread is running.

2. Potential different NS preemption context MAY cause Faults.
Interactions – Scenarios while Secure IDLE is ongoing

1. RTOS Kernel preempts the IDLE thread and regard the context as NS Thread #2
2. SPM preempts NS Thread #2 and regard the context as IDLE veneer
3. Scenario C described in the ‘Interactions’ page, do not schedule in SPE.
Do not do scheduling while NSPE is executing.

```c
set_irq_signal(...)
{
    if ((EXC_RETURN & SECURE_BIT) == SECURE_BIT) {
        PendSV = 1;
    }
}
```
IDLE Processing – Option 1: `wfi` in SPE

If there is IDLE thread(s)...

![Diagram showing the process of IDLE thread handling with `wfi` in SPE]

**Pros:**
No extra modification in NS RTOS sources.

**Cons:**
Execution needs to be activated by interrupt.

Re-use veneer context as IDLE Thread to save context

![Diagram showing the process of re-using veneer context as IDLE Thread]

**Pros:**
No extra modification in NS RTOS sources.

**Cons:**
Execution needs to be activated by interrupt.
IDLE Processing – Option 2: Return IDLE to Non-secure

```c
do {
  ret = psa_api();
  if (ret != S_IDLE) {
    break;
  }
  yield();
}
```

Pros:
Non-secure manages secure IDLE.

Cons:
Extra design get involved in non-secure code.
IDLE Processing – Option 3: Use IRQ/Event

If NS IRQ get triggered at SP...

Pros:
Non-secure thread software did not aware of the code change (change in kernel part).

Cons:
Extra design get involved in non-secure scheduler mode.

Here preempts directly from SP since priority is the lowest. This indicates the secure execution is IDLE, non-secure can go IDLE if it also has nothing to do.

SPM can’t do anything since execution belongs to non-secure now.

Have to wait non-secure scheduler back to preempted SP.
IDLE Processing – Option3: Use IRQ/Event

If NS IRQ get triggered at SPM...

Before Triggering IRQ, IDLE thread is the background context:

```c
if (next_thread->r0 == S_IDLE) {
    trigger_ns_idle_irq();
}
```

The preempted context is the IDLE veneer.

S_ISR can’t scheduling. Tail-chain to NS Handler.