Hello, and welcome to this presentation about MPU usage in STM 32. With ARM Cortex M seven

The purpose of the presentation is to show usage and setting of MPU on STM32 with Cortex-M7 and mainly raise awareness of the issue with speculative access on Cortex-M7, which may cause speculative read lock, and the issue may be prevented by MPU.

Presentation covers also basic parameters of MPU and options we can set.

At the end you can find also few typical examples of MPU setting.

Presentation is not covering security aspect of MPU usage like setting application permissions to access only some part of memory.

Only purpose of MPU usage in this presentation is to make project running reliable and with best possible performance.

We start with the description of Cortex-M7 speculative read feature.

Speculative memory read may be performed by Cortex-M7 core on normal memory regions.

Purpose of speculative read is to increase performance of the microcontroller.

Speculative memory read may cause high latency or even system error when performed on external memories like SDRAM, or Quad-SPI.

External memories even don't need to be connected to microcontroller, but its memory range is accessible by speculative read because by default, its memory region is set as Normal.

ARM technical reference manual exactly lists situations when speculative access may be done by core.

Speculative access cannot be predicted.

It's possible to disable speculative access in core registers, but due to performance drop, this option is not recommended.

Speculative instruction fetches can be initiated to any normal executable memory address. This can occur regardless of whether the fetched instruction gets executed or in rare cases, whether the memory address contains any valid program instruction.

Speculative data reads can be initiated to any normal read, write or read only memory address. In some rare cases, this can occur regardless of whether there is any instruction that causes the data read.

Speculative cache linefill can be initiated to any cacheable memory address and in rare cases, regardless of whether there is any instruction that causes the cache linefill.

There are three types of memory regions for Cortex-M7 devices.
Memory type determines which operations are allowed on given memory region.

In Normal memory regions processor can perform speculative reads or re-order transactions for efficiency. Processor can also perform unaligned memory access, so [Normal] memory type is convenient for code execution.

Both Device memory type and Strongly Ordered memory type do load and store operations strictly in program order.
Difference is, that device memory type is bufferable - meaning that instruction execution may continue before memory right is done.
Memory right is then finished from a buffer.

For Strongly Ordered memory region CPU waits for the end of memory access instruction

Speculative access is never made to Strongly Ordered and Device memory areas.
Device memory type is used for microcontroller registers.
Not buffered Strongly Ordered type is used for memories where each writes need to be visible for devices - for example, for external NAND memories.

In ARM Cortex-M devices there are additionally two attributes to be set for each memory region:

Shareable shall be set for a region if multiple masters can access the region, and it is up to the memory system to provide data synchronization between multiple masters. Typical example of multiple masters accessing same memory in STM 2 is processor core and DMA. In this case, data cache may cause different data to be visible for core and for DMA.

As STM32 microcontrollers don't contain any hardware feature for keeping data coherent, setting a region has Shareable means that data cache is not used in the region. If region is not shareable, data cache can be used, but data coherency between bus masters need to be ensured by software.

Second attribute is Execute Never:

When Execute Never is set for a region, instructions cannot be executed from that region , and any attempt for that causes hard fault.

This attribute has more usage in security usage of MPU.

For this presentation it is more important that *speculative instruction fetch* cannot be done in Execute Never region.

Table on this slide shows complete address range of Cortex-M7 devices and default memory types in given region.

Memory access in the regions follows rules of its memory type default memory type setting for each region or its part can be changed using MPU.

For speculative access issue, it is important to note that external RAM region is by default Normal memory type with enabled code execution, but

system must ensure that all Executable and Normal memory type regions are safe to access.

If any inaccessible memory location is addressed by speculative access, processor cannot guarantee cancellation of such speculative read, which may lead to extensive delay or even to device lock.

In default mapping of Cortex-M7 devices, external RAM memory region is critical: it's type is Normal, without Executed Never attribute set.

So speculative access can be performed to this region, but external memories

- don’t have to be connected, or
- don't have size to cover complete region size, or
- range not covered by external memory,

shall get their memory type changed by MPU setting to prevent speculative read issue.

External memory may have Normal memory type or any other type convenient for such memory type.

Unlike the external RAM region, both Code and SRAM region are safe to access. Microcontroller memory driver handles this memory range.

How to recognize the issue:

Typically, if microcontroller got locked due to speculative read, program main loop is not executed anymore, but interrupts are still invoked, no hard fault is triggered.

If microcontroller was in Debug Mode, debug session fails and it's not possible to connect again not even using connect to running target devices not responding on reset. Power cycle is needed.

Occurrence of the issue is random. Even very small change in the code may hide or release the problem.

For example, one luck instruction can determine if the lock will occur, or not.

To prevent speculative access issue, all addresses not safe to access shall change memory type to

- Device or
- Strongly Ordered memory type AND set Execute Never attribute.

After that device cannot be locked by speculative memory access.

To make such setting of memory regions Cortex-M7 contains memory protection unit alias MPU.

On following slides we will introduce code for setting safe background region to set attributes for critical region.

We recommend to use MPU for handling critical external RAM region in each project which is using STM32 based on Cortex-M7.

Memory protection unit is part of microcontroller core.

Its task is to define memory attributes and access permissions.
MPU is then monitoring bus transaction and if any rule violation is detected, fault exception is triggered.

In Cortex-M7, users see memory management handle the trigger when MPU rules are violated. But for example in Cortex-M0+, memory management handler is not available and MPU fault triggers hard fault.

Depending on used device, also number of region, which can be defined in MPU, is changing.

We need to highlight also MPU behavior with overlapping regions which we will use further in this presentation.
If some memory area is covered by more MPU regions, region with the highest number is used for setting attributes on the memory address.
We set region 0 to prevent speculative read issue, but any other region will get priority over region 0 to propagate preferred rules on used memory.

By enabling MPU in a project based on Cube HAL libraries, you have additionally one parameter to pass into HAL MPU enable function.

There are four possible settings, which covers all combinations of settings to bits in MPU control registers:

- First bit sets if MPU is enabled during hard fault, NMI and fault mask handlers. If kept at zero, MPU is disabled during hard fault, NMI and fault mask handlers.
- Second bit: allow default memory map. If this bit is enabled, default memory map is used as a background region for privileged software accesses. In this case, background region acts as if it has region number -1. Any region that is defined and enabled has priority over this default map. If disabled, default memory map is disabled and any memory access to a location not covered by any enabled region causes a fault.

In examples which you will find later in this presentation, MPU_PRIVILEGED_DEFAULT parameter is used, meaning that MPU is disabled in fault and default memory map is enabled.

Few parameters need to be set also for each MPU region which will be used:

- its starting address of the region and
- its size,
- then TEX parameter, which together with other MPU region parameters determine region memory type and cache behavior.

Then you need to set parameters we already discussed on previous slides:
By enabling or disabling you can make region

- Cacheable,
- Bufferable and
- Shareable.

Not all combination of MPU regions parameters are allowed.
List of possible combinations is on slide 17. Not listed on this slide is

- parameter Execute Never.

This parameter can be set for any region without influence on use memory type.

Setting Execute Never on a region disables code execution from that region.

Last parameter in MPU region deserves more explanation. It's MPU sub-region setting.

For each region which size is 256 bytes or more, it is possible to divide region into 8 sub-regions with equal size. Excluding sub-region from region rules is set by writing one on corresponding position of 8-bit value.

Concrete usage is demonstrated on a picture.
We choose to have 8 kB region starting at address 0.
So each sub region size is 1 kB , and if sub region field is set to value 3A hex, the second, fourth, fifth and sixth sub-regions from start won't be included in MPU region.

Table on this slide lists all allowed MPU configuration in STM32 microcontrollers with memory protection unit.

Values from this table shall be followed when you design MPU for any region.

Settings in an MPU region also determines Cache Policy, when parameter Cacheable allowed in a region.

Cache Policy may have influence on performance based on memory kind and usage, different policy may be more efficient.

But it's not purpose of this material to cover more deeply Cache Policy setting.

For more information please find related materials at the end of this presentation.

Just would be good to warn you additionally, that in **STM 32F7 microcontrollers and some older STM32H7 microcontrollers, older revision of ARM Cortex-M7 is used, and this older version of Cortex-M7 has Errata for data cache usage when configured with write-through policy.**

Conditions to reproduce the issue are very specific. But to be safe, it's recommended to **use write back-policy instead**. For more details, please check product Errata sheet.

Default setting of shareability and Cash Policies is visible in table.

By default, all cacheable regions are non-shareable. Software needs to handle data coherency there.

Also be careful that write-through Cache Policy is used on code memory region and part of external RAM memory region. Again, if this default setting is not suitable for your usage, you may change it using memory protection unit.

Now we're leaving the theoretical part and moving to examples of MPU settings, starting with region 0 which we recommend to set as basic region.

The purpose of the [MPU] region is to set [address] regions of unused memory - which can cause speculative access issue - into safe configuration which is not allowing speculative access.

Region is set as strongly ordered with Execute Never parameter set using sub-region.

Range 60 000 000 hex to E0 000 000 hex is set in region 0.

Code example: how region 0 for preventing speculative memory read can be implemented in project (using sub-region usage address 60 000 000 hex to E0 000 000 hex is covered).

A few more typical use cases of MPU usage are covered in this presentation.

There is example for

- setting external Q-SPI flash memory,
- SDRAM on FMC,
- DMA usage with internal RAM,
- RAM buffers for Ethernet, and
- LCD configuration.

All examples count with usage of region 0 from previous slide to prevent speculative read issue.

As already covered on start of this presentation, there are some general recommendations which memory types shall be used for various kinds of memories.

For **code execution** it’s best option to use **Normal memory type,** which allows unaligned memory access.

Also for **RAM** memories is **Normal memory type** convenient as it's not having additional restriction and offers best performance.

For **MCU registers,** access is important to preserve program order of instruction. instruction can be written in a burst using buffer will attribute so **Device** memory type is best option.

Strongly Ordered memory type is used in **memories which need to have each write be a single transaction** - for example NAND memories or FPGAs.

First example is external Q-SPI flash memory, those days often used as memory for code execution or large data storage.

In STM32F7 family, Q-SPI or FMC need to rewrite setting of region 0 to allow access on address a 0 000 000 hex, where Q-SPI and FMC control registers are located.

in STM32H7 microcontrollers is for Q-SPI and FMC control register storage used different address, then only one MPU region for Q-SPI memory ranges enough-

Starts with Q-SPI memory depends on us bank usually its address 90 000 000 hex.

We recommend to set this region as Normal, Shareable with Cache write-back policy. If code won't be executed from the memory set also Execute Never attribute.

Code example setting Q-SPI and FMC control register access,

and initialization code for Q-SPI memory range.

Like for Q-SPI, FMC on STM32F7 also needs a region added to allow access to control registers starting on address a 0 000 000 hex.

Start of SDRAM depends on used bank. Usually SDRAM is mapped on bank 1 at address C0 000 000 hex.

We recommend to set as Normal memory type with write-back Cache Policy.
If only one bus master is accessing the memory area, set as NOT_SHAREABLE to use also data cache.

Code example setting Q-SPI and FMC control register access,

and SDRAM memory range setting.

DMA is very often used with microcontroller internal SRAM.
By default, RAM memory region is Shareable. That makes program responsible to keep data coherent when cache is used. **If you want to achieve optimal performance, it’s strongly recommended to enable both data and instruction cache.**

If you don't want to keep data coherency using software (which means flush complete cache before each DMA usage), best option is to **set just the part of RAM memory which is used by DMA as Shareable**.
That ensures data coherency of the memory and preserves data cache enabled for parts of RAM where application can safely use it, meaning DMA is not used there.

In an example here we choose two buffers in RAM, total size 1 kB.
In program, buffer address will be fixed to start from address 20 020 000 hex.

Code setting for MPU region:

This setting has multiple possibilities.

As instructions won't be saved in the region, it makes no difference if cache is completely disabled or cache is enabled when set as Shareable - then data cache is not used again.

Ethernet peripheral in STM32 uses DMA for data transfers between peripheral and buffers placed in RAM.

Like frame buffer used by DMA with cache usage discussed in previous part, **region where Ethernet RAM buffers are placed, need to be set as Shareable.**

Additionally, it's recommended to use different RAM for Ethernet buffer and application data to achieve better performance.

For example, on STM 32F746, use SRAM1 for application data storage and SRAM2 for Ethernet buffers.

Ethernet peripheral demands two pairs of buffer:

- one for data itself,
- second for DMA descriptor tables.

Transmit and receive buffer are RAM storage with DMA usage.

Set as **Normal**, non-cashable memory region.
As cache disabled, **Shareable don't need to be enabled**.

DMA descriptor tables shall be set as Shared device memory.

Last example of MPU setting in this presentation is for LCD display controlled by flexible memory controller (FMC).

In such scenario, internal memory in LCD display is typically used, and 8 or 16 bit bus width is used.

Depending on FMC register select pin usage, two different memory regions may be used. which may need to use two memory region with same settings.

That is also the case in theexample here.
Region 1 covers 32 bytes of address 60 000 000 hex, which is the starting address of bank 1 on STM 32F7.

32 Bytes is minimal MPU region size.

First region setting 32 bytes, **Strongly Ordered** memory starting from 60 000 000 hex.

Second region for toggling register select also 32 bytes, **Strongly Ordered** memory in our configuration starting from address 60 020 000 hex

Here you can find references to other material types of memory protection unit from ST Microelectronics:

- Programming Manual for Cortex-M7 microcontrollers,
- Application Note about level 1 cache,
- dedicated application note about memory protection unit,
- and for some users surprising the also application note about LTDC which contains quite detailed description of MPU usage and also contains mentioned about speculative read lock.

You can explore this material to get more complete information about MPU usage.

In the presentation, addresses of microcontroller peripherals are often mentioned. They need to be be changed from family to family and you can find peripheral addresses and complete memory mapping reference manual.

Thank you for watching this presentation and wish you lots of success with STM32.