MPU in STM32 MCUs

MPU လို့ခေါ်တဲ့ Memory Protection Unit ကိုစတင်ဆွေးနွေးသွားမှာဖြစ်ပါတယ်။ အပိုင်းဆက်လေး တွေနဲ့ပေါ့။ ကျွန်တော်တို့တစ်တွေ Memory Protection ကိုသုံးရတဲ့အဓိကရည်ရွယ်ချက်ကတော့ Memory ပေါ်မှာမရှိတဲ့ Process တစ်ခုက Memory ကိုလှမ်းသုံးတာကိုကာကွယ်ဖို့ရန်အတွက်ဖြစ်ပါ တယ်။ အဲဒီ process က malware လည်းဖြစ်နိုင်တယ်။ bug လည်းဖြစ်နိုင်တယ်။ ဒီလို process မျိုး ကိုကာကွယ်ပေးလိုက်ခြင်းအားဖြင့် Memory ပေါ်မှာအလုပ်လုပ်နေတဲ့ အခြားသော အလုပ်တွေကို ထိခိုက်မှုမရှိစေမှာဖြစ်ပါတယ်။ Memory Protection ကိုလုပ်ဆောင်ဖို့ရန်အတွက် နည်းလမ်းတွေ အများကြီးရှိပါတယ်။

- Segmentation - Memory ကို အပိုင်းငယ်လေးတွေအဖြစ်ပိုင်းခြားပေးတဲ့လုပ်ငန်းစဉ် ဖြစ်ပါတယ်။ Memory Location ရဲ့ Reference မှာ အပိုင်းငယ်လေးတွေကိုခွဲခြားထားတဲ့ တန်းဖိုးတစ်ခုပါဝင်ပါတယ်။ နောက်တော့ Memory အပိုင်းငယ်လေးတွေရဲ့ ကြားမှာ offset တစ်ခုလည်းပါဝင်ပါတယ်။ 32 bit (x86) processors တွေမှာဆိုရင် Segment Register (4) ခုရှိပါတယ်။ CS(Code Segment), SS(Stack Segment), DS(Data Segment) နဲ့ ES(Extra Segment) တို့ဖြစ်ကြပါတယ်။
  နောက်ပိုင်းထပ်ထည့်လိုက်တဲ့ Segment တွေက တော့ FS နဲ့ GS တို့ဖြစ်ကြပါတယ်။

STM32 မှာ MPU ကို Manage လုပ်မယ့်အကြောင်းကို ဆက်လက်ဆွေးနွေးသွားပါ့မယ်။ အရင်ဆုံး MPU ကိုမသုံးခင် STM32 Microcontroller ရဲ့ MPU ကို enable လုပ်ထားရပါမယ်။ program ရေး ထားရမယ်ပေါ့ဗျာ၊ MPU ကို မဖြစ်မနေ enable လုပ်ထားခဲ့ရမှာပါ။ MPU ကို enable လုပ်မထားခဲ့ဘူး ဆိုရင် မိမိ manage လုပ်သမျှက Memory အပေါ်မှာ ဘာသက်ရောက်မှု မှရှိမှာမဟုတ်ပါဘူး။ MPU ကို Support ပေးတဲ့ STM32 တွေကတော့ ARM Cortex M0+/M3/M4 နဲ့ M7 တို့ဖြစ်ကြပါတယ်။ STM32F1 Series, STM32F2 Series, STM32F3 Series, STM32F7 Series, STM32L0 Series, STM32L1 Series, STM32L4 Series တွေနဲ့ STM32H7 Series တွေမှာ MPU ကို support ပေး ထားကိုတွေ့ရပါမယ်။ ပိုပြီးတော့ လုံခြုံ ပြည့်စုံတဲ့ Embedded System တစ်ခုကို MPU သုံးပြီးတော့ ဖန်တီးပြုလုပ်နိုင်ပါတယ်။ အဲဒီလိုမျိုးပိုကောင်းတဲ့ Embedded System တစ်ခုကိုတည်ဆောက်ဖို့ရန် အတွက်အောက်ပါလုပ်ငန်းစဉ်တွေကို ဆောင်ရွက်ပေးရမှာဖြစ်ပါတယ်။

- OS Kernal လိုမျိုးအရေးကြီးတဲ့ လုပ်ငန်းစဉ်တွေကနေသုံးမယ့် data တွေကိုဖျက်ဆီးပစ်မယ့် User Application တွေကိုတားမြစ်ထားရမှာဖြစ်ပါတယ်။

- Injection လုပ်တာကိုကာကွယ်နိုင်ဖို့အတွက် SRAM Memory ကို non-executable အဖြစ် သတ်မှတ်ထားရပါမယ်။ Code Injection အကြောင်းကို Java Program တစ်ခုနဲ့ ရှင်းပြပြီး တော့်စာရေးပါဦးမယ်။
- Memory Access Attributes တွေကိုပြောင်းလဲပေးရပါမယ်။

  Memory Region ၁၆ ခု အထက်ကို ကာကွယ်နိုင်ဖို့အတွက် MPU ကိုအသုံးပြုနိုင်ပါတယ်။ Memory Region ၁ ခုဟာ 256 bytes အနည်းဆုံးရှိမယ်ဆိုရင် အဲ့ဒီ Memory မှာ Sub-Region အနေနဲ့ စုစုပေါင်း ၈ ခုရှိပြန်ပါတယ်။ Sub-Regions တစ်ခုချင်းစီဟာဆိုရင် တစ်ခုနဲ့ တစ်ခု size တူကြပါတယ်။ ဘာလို့လဲဆိုတော့ အငယ်ဆုံး Memory Region Size ဟာ Cache Line ကနေထိန်းချုပ်လို့ဖြစ်ပါတယ်။ Cache Line ဟာ Length အနေနဲ့ 32 bytes ရှိ ပါတယ်။ ဒါဆိုရင် စောနက ကျွန်တော်ပြောခဲ့တဲ့ 32 bytes အရွယ်အစားရှိတဲ့ Sub-region စုစု ပေါင်းရှစ်ခုမှာဆိုရင် 256 bytes ရှိပါတယ်။ (8x32=256) ဖြစ်ပါတယ်။ Regions တွေဟာဆို ရင် 0 (သုည) ကနေ 15 (ဆယ့်ငါး) အထိနံပါတ်စဉ် အတိုင်းရှိနေပါတယ်။ Default Region ဆိုပြီးတော့လည်း ရှိပါသေးတယ်။ သူ့ရဲ့ ID ကတော့ (-1) ဖြစ်ပါတယ်။ 0-15 အထိ Memory Regions တွေဟာဆိုရင် Default Region ထက်ပိုပြီးတော့ Priority မြင့်ပါတယ်။

  Region တွေကို ထပ်ထားလို့ nested အနေနဲ့ တစ်ခုနဲ့တစ်ခုရောထားလို့လည်းရပါတယ်။ Region နံပါတ် 15 ဟာဆိုရင် Priority အမြင့်ဆုံးဖြစ်ပါတယ်။ Region နံပါတ် 0 ကတော့ Priority အနိမ့်ဆုံးပါခင်ဗျ။ Priority ဆိုတာဟာလည်း Region တွေ Overlap (ထပ်) တဲ့အခါ မှာလိုအပ်ပါတယ်။ Priority တွေဆိုတာ အသေသတ်မှတ်ထားတာဖြစ်ပြီးတော့ ပြောင်းလဲလို့ မရနိုင်ပါဘူး။ ပုံ (၁) မှာ Region တွေကို Overlap လုပ်ပြထားပါတယ်။ ကြည့်လိုက်ပါ။ ပုံ (၁) က Region (6) ခုနဲ့ဥပမာပြပေးထားတာပါ။ Region 4 ဟာ Region 0 နဲ့ 1 ကို Overlap လုပ်ထားတာကို မြင်ကြမှာပါ။ Region 3 ထဲမှာဆိုရင်တော့ Region 5 ကို Enclosed လုပ်ထားတာကို တွေ့ကြရပါမယ်။ Priority ကို Ascending Order (ကြီးရာမှငယ်ရာသို့) စီစဉ်ထားတဲ့အတွက်ကြောင့် လိမ္မော်ရောင်နဲ့ပြထားတဲ့ Region တွေက Priority ပိုမြင့်ပါ တယ်။ ဒါကြောင့်မို့လို့ Region 0 က Write လုပ်လို့ရပြီးတော့ Region 4 က Write လုပ်လို့မရခဲ့ရင် Region 0 နဲ့ 4 ရဲ့ Overlap ဖြစ်စဉ်ထဲမှာကြရောက်နေတဲ့ address တစ်ခုဟာလည်း write လုပ်လို့ရမှာမဟုတ်ပါဘူး။

MPU ဟာ unified ဖြစ်ပါတယ်။ unified ဖြစ်တယ်ဆိုတာဘာလည်းဆိုတော့ data အတွက် region နဲ့ instruction အတွက် region ဆိုပြီးတော့ regions တွေကိုခွဲခြားထားတာမဟုတ် လို့ပါ။ Cache လိုမျိုး Memory attributes တွေကိုသတ်မှတ်ဖို့ရန်အတွက်လည်း MPU ကို အသုံးပြုနိုင်ပါတယ်။ Cacheability ဟာ System Level Cache Unit ထဲကို ထည့်လို့ရပါ တယ်။ ဒါမှမဟုတ် Memory Controller ဆီကိုထည့်လို့လည်းရပါတယ်။ ARM Processor Architecture ရဲ့ Memory Attribute Setting တွေထဲမှာ Cache Level နှစ်ခုရှိပါတယ်။ အတွင်းပိုင်း cache (Inner Cache) နဲ့ အပြင်ပိုင်း cache (Outer Cache) ဆိုပြီးတော့ပေါ့နော် ။ သို့ပေသိ ခွင်းချက်အနေနဲ့ STM32F7 series နဲ့ STM32H7 Series တွေမှာ Cache Level တစ်ခုသာလျှင်ရှိပါတယ်။ (L1- Cache) Cache Control Register ကနေပြီးတော့ Cache တွေကို ထိန်းချုပ်ပါတယ်။ သို့ပေသိ Memory Region က cacheable ဖြစ်သလား မဖြစ်ဘူး လားဆိုတာကို MPU ကနေပြီးတော့ ခွဲခြားသိပါတယ်။ ဒါ့အပြင် MPU ဟာ Cache Policy ကို သတ်မှတ်နိုင်ပါသေးတယ်။ ခွင်းချက်အနေနဲ့ L1-Cache ရှိတဲ့ STM32F7 တွေနဲ့ STM32H7 တွေအတွက်တော့ Memory Region ကနေ L1-Cache အတွက် cache attributes တွေကို သတ်မှတ်ဖို့ရန်အတွက် MPU ကခွင့်ပေးပါတယ်။

STM32 products တွေရဲ့ processor မှာ default memory map ကအသေသတ်မှတ်ထား တာရှိပါတယ်ခင်ဗျာ။ သူ့ကိုတော့ ပုံ(၂) မှာပြသထားပါတယ်။

Memory types

Memory အမျိုးအစာ သုံးခုရှိပါတယ်။

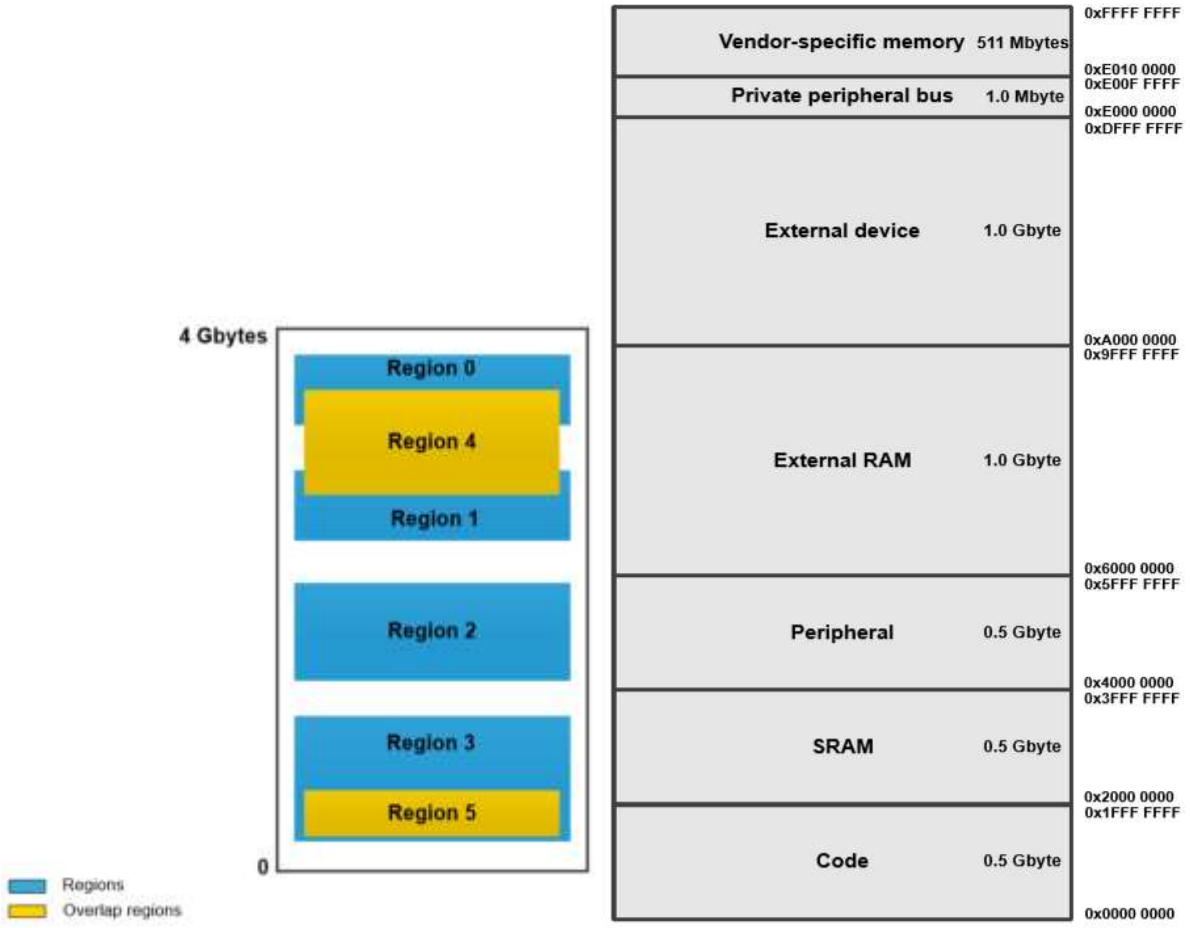
- Normal Memory

  Data bytes တွေကို သိုလှောင်ခြင်းနဲ့ load(ဆွဲတင်) လုပ်ငန်းစဉ်တွေကိုဆောင်ရွက်ပါတယ်။ Normal Memory မှာဆိုရင် သိုလှောင်ခြင်း (store) နဲ့ load(ဆွဲတင်ခြင်း) လုပ်ငန်းစဉ်တွေကို CPU ကဆောင်ရွက်စရာမလိုအပ်ပါဘူး။
- Device Memory

  Device Memory ဆိုရင်တော့ သိုလှောင်ခြင်း (store) နဲ့ load(ဆွဲတင်ခြင်း) လုပ်ငန်းစဉ်တွေ ကိုအစီအစဉ်အလိုက်အတိအကျ သတ်မှတ်ထားပါတယ်။ Registers တွေကိုတော့ သင့်လျော်တဲ့ အစီအစဉ်အလိုက်အတိအကျသတ်မှတ်ထားဖို့လိုအပ်ပါလိမ့်မယ်။

- Strongly Ordered Memory:

  Program ရေးထည့်ထားတဲ့ အစီအစဉ်အတိုင်းဘဲ အကုန်လုံးကိုဆောင်ရွက်ပါတယ်။ ဒီနေရာမှာဆိုရင်တော့ CPU ဟာ Program ထဲက နောက် instruction တစ်ခုကိုအလုပ်မလုပ်ခင် load/store လုပ်ငန်းစဉ်အလုပ်လုပ်တာပြီးဆုံးအောင်စောင့်ပေးပါတယ်။ ဒါဟာလည်း Performance ကို သက်ရောက်မှုရှိပါတယ်။



Interrupt Latency ဆိုတာ

Interrupt Latency ဆိုတာဟာ Interrupt Request တစ်ခုကို Processor ကနေ Response ပြန်နိုင်ဖို့ရန်အတွက် Processor ကနေယူသုံးလိုက်ရတဲ့ Clock Cycles အရေအတွက်ဖြစ်ပါတယ်။ Cortex-M Processor တွေမှာ ရှိတဲ့ Interrupt Latency အကြောင်းကိုဆက်လက်ဆွေးနွေးသွားပါ့မယ်။ Cortex-M processor တွေမှာ Interrupt Latency ကနည်းပါတယ်။ Interrupt Latency နည်းတော့ ကောင်းတာပေါ့။ ဘာလို့လည်းဆိုတော့ System ကို delay ဖြစ်စေတယ်လေ။ Interrupt Latency များလေလေ System ကို delay ဖြစ်လေလေဘဲ။ နည်းတာဘဲကောင်းတယ်။ အောက်ကဇယား မှာလည်း Cortex M ရဲ့ Interrupt

Latency ကို ကြည့်နိုင်ပါတယ်။ ဒီတော့ မေးခွန်းတစ်ခုရှိလာပါတယ်။ Interrupt Latency ကို ဘယ်လိုလျှော့ချကြမလဲပေါ့နော်။ Interrupt Latency ဟာအကြောင်းအမျိုးအမျိုးအပေါ် မှာမူတည်နေပါတယ်။

- Platform and Interrupt Controller
- CPU Clock Speed
- Timer Frequency
- Cache Configuration
- Application Program

သင့်တော်တဲ့ platform နဲ့ Processor ကိုရွေးချယ်အသုံးပြုခြင်းအားဖြင့် Interrupt Latency ကို လွယ်လွယ်ကူကူလျှော့ချနိုင်ပါတယ်။ ISR လို့ခေါ်တဲ့ Interrupt Service Routine ကိုနောက် post မှာဆွေးနွေးပေးပါ့မယ်။ Cortex-M ရဲ့ ISR ကတော့ Interrupt တွေကို စီမံခန့်ခွဲတဲ့ NVIC ဆိုတာကြီး ဖြစ်ပါတယ်။ NVIC အကြောင်းကို ဒီ Group ထဲမှာအရင်က တင်ဖူးပါတယ်။ ရှာဖတ်ကြည့်ပါ။