 傻孩子(GorgonMeducer) · 2020 年 06 月 11 日

簡單粗暴解讀 Cortex-M23/33(上)

Cortex-M
作者：GorgonMeducer 傻孩子
首發：裸機思維

猝不及防的，ARM 在最新的 TechComm 上發布了兩款 ARMv8-M 架構的新處理器—自從年初發布新架構以來，靴子總算落地了。

> 內心飄過彈幕：
> 誰 TM 告訴我，這個 M23 和 M33 是什麼鬼?
> 從個位數一下蹦到兩位數了喂!
> 前面十幾位兄弟怎麼了?喂!
> 別說跟 M3 有啥關系，這以後下第 n 代是不是就該叫 2333333 了?

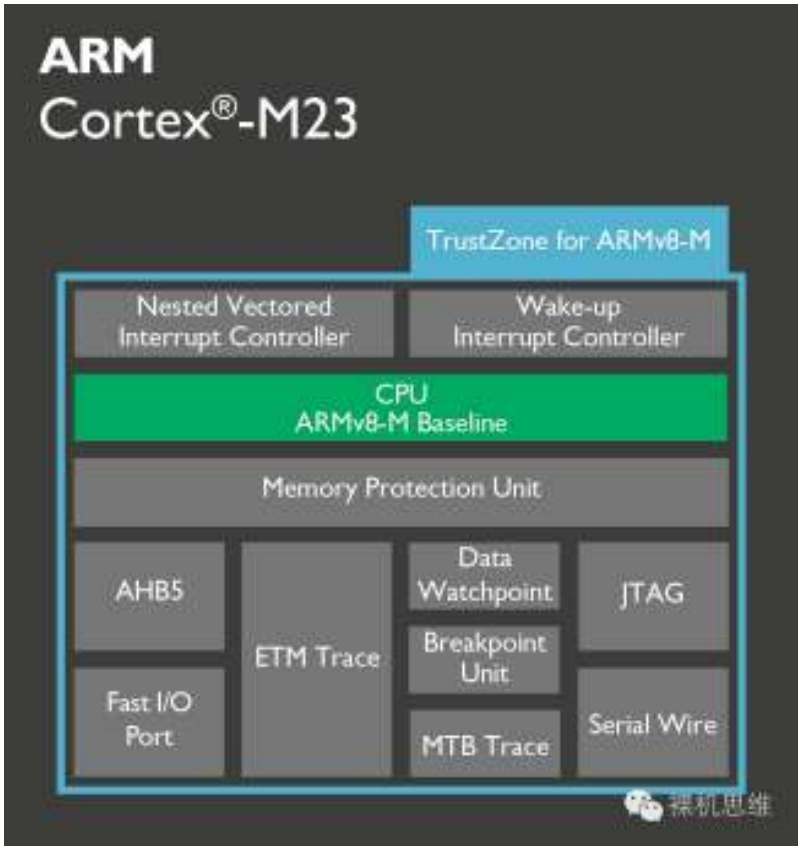
該來的總會來，那麼如何簡單粗暴的理解這兩個全新的處理器呢?以下是傻孩子獨家特別提供的的無責任囫圇吞棗公式：

> Cortex-M23 =
> Cortex-M0/M0 + 硬件除法器 + 性能提升 +
> 指令集不可忽略的小動作 +
> 安全擴展(TrustZone for ARMv8-M) +
> “哎媽呀我錯了，我改還不行麼?”
> Cortex-M33 =
> Cortex-M3/M4 + 性能提升 +
> 指令集可以忽略的小動作 +
> 安全擴展(TrustZone for ARMv8-M)+
>
> “哎媽呀我錯了，我改還不行麼?”

再簡單點說就是無敵增強版的”M0/M0+，M3/M4”加”安全擴展”。有人說，ARMv8-M 的主要功能就是為 Cortex-M 家族引入 TrustZone，這麼看來也是不無道理的。

1. 增強版的 Cortex-M0/M0+

根據官方的說法，Cortex-M23 實現的是 ARMv8-M 架構的 Baseline 子架構，我們不妨理解為手機裡面的”入門級”產品。



註：圖片來自 ARM 官網

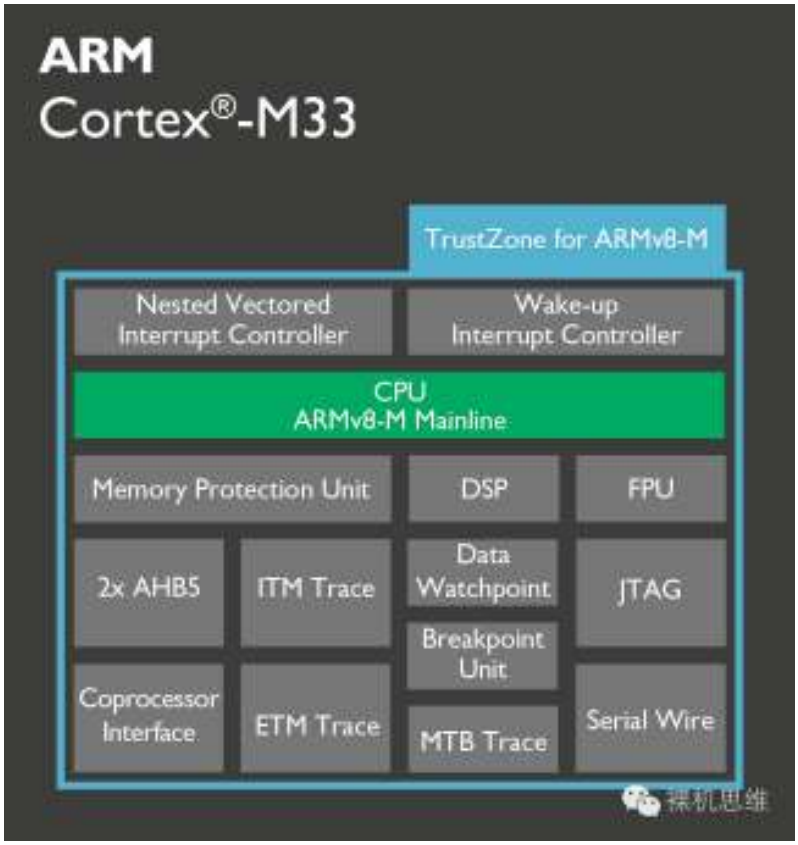
M23 從定位上也非常直接，就是給 M0/M0+增加個安全擴展。因此，實際上所有為 M0/M0+編譯生成的二進制代碼基本上都可以”無修”的在 Cortex-M23/M33 上執行—除非你原本的代碼使用了 MPU。此外 M23 居然配備了硬件除法器，這無疑在原本 M0 和 M0+主打的 8 位/16 位市場上把”基本配置”又提升了一個檔次。當然，相應的，原本就不大的內核上，除法器面積的增加在比率上可是”禿子頭上的蝨子”—你懂的—關我們消費者屁事!

指令集上，M23 師承 ARMv6-M，除了支持”安全擴展”所必須的一系列指令之外，這款入門級產品還做了一個”不可忽略的小動作”—也就是說，除了 M33 以外，M23 也可以通過很小的代價支持”暗代碼”。什麼是暗帶麼呢?和”暗物質”只能理論上知道它存在卻很難探測到類似—”暗代碼”是一類只能執行(取指令)卻根本無法讀取代碼本身 OPCODE 的機器碼—也就是人們常說的 XO(Execute-Only)代碼。”暗代碼”並不是依靠內核來實現的，但卻需要編譯器和內核共同努力才能支持。這是因為 XO 的特性是靠芯片設計廠商提供提供一個特殊的存儲器來實現的，這個存儲器最特殊的特性就在於，只能”取指”，不能進行普通的數據訪問。這就要求”暗代碼”裡不能直接保存任何常數，他們必須編碼到指令裡面—成為指令的一部分，以單純 OPCODE 的立即數存在。ARMv6-M 的指令集大部分都是 16 位的 Thumb 指令，16 位的 OPCODE 可以編碼的立即數長度可想而知—少得可憐。ARMv7-M 由於引入了 32 位的 Thumb2 指令集，從而極大增強了 OPCODE 攜帶立即數的能力。為了將這一能力引入 ARMv8-M 的 Baseline 指令集，MOVT 和 MOVW 這兩個可以分別攜帶 32 位立即數高低 16 位的指令就被特別加入到 M23 的指令集中。ARMv8-M 強調的是安全，”暗指令”有多大的份量，可想而知。

Cortex-M23—這個 M0 不簡單。

2. 增強版的 Cortex-M3/M4

相對 Cortex-M3/M4 來說，M33 在性能上有了提升並不是什麼意料之外的事情，不提也罷。值得說明的是，從城裡來的 Cortex-M7 在性能上仍然可以甩其他 Cortex-M 土包子幾條街—6 級流水線和 3 級流水線的差別可是三缸夏利和六缸寶馬之間的差距所不能比擬的!(認真臉)。



註：圖片來自 ARM 官網

3. ARMv8-M 是個知錯就改的好少年

我不知道有多少人真正用過 ARMv7-M，也就是 M3/M4 的 MPU—簡單說就是個以 Region 為單位來修改 Memory 屬性的系統級外設。原本設計的時候想法很簡單，一個 Region，給個大小(Size)給個基地址(Base Address)，再給個屬性(Memory Attribute)，一使能，就工作了，很簡單，很 Happy。然而，出於優(pi)化(gu)內(jue)核(ding)面(nao)積(dai)的原因，Region 地址范圍的設定被人為加入了一個限定：

基地址(Base Address)必須對齊(Aligned with)到它的尺寸(Size)，而且尺寸必須是 2 的整數次方(還必須大於 4 次方)。

舉個例子：一個 Region 大小為 512K，那麼基地址必須是 512K 的整數倍……如果你還不能理解這個問題蛋疼的點在哪裡，設想一個任意大小的 Region 該怎麼設定，比如，一個 234K 大小的 Memory 該咋辦?—還能咋辦，用多個 Region 組合出來唄。

正是這個蛋疼的限制，導致幾乎沒有什麼 RTOS 可以很好的使用 MPU，也罕有身邊的項目把 MPU 這麼骨感的現實應用的如理想般美好。

那麼 ARMv8-M 做了什麼呢?他更正了這一蛋疼的設定，Region 的設置由”基地址+尺寸”進化為”起始地址+終止地址”，除了這兩個地址都必須是 32 字節的整倍數的要求外，再也沒有變態的關於”基地址必須是 Region 大小的整倍數”這樣的限定。是不是突然覺得眼前一亮，是不是突然發現了一個寶藏?MPU 頓時好玩起來。

ARMv8-M 的 MPU 是個好同志，士別三日當刮目相看

4. 安全擴展(Trust Zone for ARMv8-M)是什麼?

請聽下回分解。

簡單粗暴解讀 Cortex-M23/33(下)

Cortex-M
作者：GorgonMeducer 傻孩子
首發：裸機思維

上篇文章，我們揭秘了 Cortex-M 家族的新成員、ARMv8-M 架構的兩位先驅—傳承自 Cortex-M0/M0+ 的 Cortex-M23 和傳承自 Cortex-M3/M4 的 Cortex-M33—指令集、流水線、外設的改變我們都懂，那麼作為 ARMv8-M 重頭戲的**安全擴展(Security Extension)**或者說 **TrustZone for ARMv8-M** 又是何方神聖呢?

- **TrustZone for ARMv8-M 和 TrustZone 是什麼關系**

首先 “TrustZone for ARMv8-M” 是一個 專有名詞，它和 Cortex-A 系列上引入的 “TrustZone” 具有以下的共同特點：

- 都是銷售用語
- 都高舉 TrustZone 大旗
- 僅在純理論層面共享一些抽象的模型，用於理解和設計嵌入式信息安全
- 安全效果基本相同

它們至少在以下幾個方面存在差異：

- 架構定義完全不同
- 技術實現完全不同
- 執行效率完全不同
- 各類成本完全不同
- 使用方法完全不同
- ……

(其實，我**個人覺得 TrustZone for ARMv8-M 比 TrustZone 要先進**。這當然不僅僅因為”我是 Cortex-M 陣營的”，更因為我覺得”用腳趾頭想都知道，TrustZone for ARMv8-M 是後來者，當然有充分的理由比 TrustZone 先進啦。”)

- **功能安全 (*Safety*)*和 信息安全*(*Security*)**

打個比方，你買了一個智能燈泡，那麼對於這個產品來說：

- 用於保護燈泡不會因為電壓過高、過低或者電流過大而損壞的保護電路，實現的就是**功能安全**，用英文單詞 **Safety** 表示；

- 用於保護你家燈泡不被隔壁老王控制，或者保護你家燈泡上的攝像頭(如果有的話)以及麥克風(如果有的話)不被個比老王竊聽的設計，實現的就是**信息安全**，用英文單詞 **Security** 表示。

再進一步總結來說，你可以簡單粗暴的認為：

Safety 保證的是系統在各種不同(通常是極端)的環境下，都擁有正常的工作邏輯；或者說所提供的功能和服務都是正常的；如果環境太極端，就進入某種保護狀態，以避免為用戶提供錯誤或者危險的服務。**—Safety 對抗的是來自環境的挑戰**。

Security 保證的是在人為破壞的情況下，系統能有效地檢測到攻擊行為、確保有效信息不會被洩露、系統不會被未經授權的用戶所控制**— Security 對抗的是隔壁老王以及各類隱藏在網絡上的雲老王**。

實際上 Security 必然是通過硬件和軟件實現的，只有它的功能、邏輯得到充分的保護，才能有效地對抗攻擊者。因此，”老王”們通常利用攻擊系統的 Safety，也就是功能安全，來試圖破構建其上的 信息安全**— Security 是建立在 Safety 之上的，談論 Security 的時候必然離不開 Safety—**這也是大家常常混淆著兩個概念的原因。同時，我們不能因為它們有單方向的依存關系(Security 依賴於 Safety，反過來卻不一定)，就認為我們可以不分場合的將它們混用。

- **為什麼人們突然這麼重視 Security**

過去，大多數微控制器的項目，1)本地團隊自己就可以完成了，2)往往不用跟第三方合作，3)也不需要大規模的連接到網絡上，4)模塊化的目的單純為了快速開發，因而過去的系統在信息安全上的問題並不是非常突出，基本上就停留在克隆抄襲這個層面上。

然而，除了克隆抄襲這個永恆的原因以外，1)IoT 的到來使得更多的嵌入式設備無法孤立的存在，因此通信安全變得突出；2)生態系統和平台的概念深入人心，單一的本地團隊越來越無法獨立的完成整個項目，因而與第三方合作成為常態，這就必然導致第三方黑盒子模塊的引入，運行時的系統信息安全變得突出；3)商業模式的建立鼓勵多方合作，模塊化只會幫助 IP 更高效的被使用而並不天然保護知識產權，更進一步說，單純的模塊化技術並不能保證廠商從最終產品的收益中獲得持續穩定的收益。

基於以上目的，簡單說就是：因為要與老王合作賣煎餅，我提供設備，老王提供服務，我即擔心隔壁老王”你懂的”原因窺探你的財產，同時又希望老王不至於偷了我設備的圖紙、把我一腳踢開自己賺錢，所以 **Security 在 IoT 時代是必須的**。

- **一句話抓住安全技術的精髓**

    - **Security** 技術實現的核心是 隔離**(Isolation)**。
    - **Isolation** 在時間上的實現就是把處理器時間按照不同的安全級別進行分配**—**建立所謂的安全的運行**(Secure)**和非安全**(Non-secure)**的運行、或者是不同安全級別的運行模式。

- **Isolation** 在空間上的實現就是各類對存儲器以及外設訪問的權限控制(**Access Attribution Management**)。

值得特別說明的是，對訪問權限的控制是一個通用的工具(Tool)，你既可以用它來實現各類資源的分配，比如操作系統中的資源管理；也可以用它來實現信息安全。這並不是說，資源管理是信息安全的一部分，也不能說信息安全就是通過資源管理來實現的—這種說法最要命的地方就是似是而非，迷惑了很多。如果有人跟你討(爭)論這個，我的建議是：你跟他打賭吵架都沒用，自己心裡明白就可以了—”對對對，諸葛孔明是兩個人”。

- **TrustZone for ARMv8-M 之 程序員不得不知的技術**

既然我們就是要簡單粗暴，那麼就不用扯那麼多犢子。ARMv8-M Security Extension 的本質仍然是實現 Isolation。那麼要達到怎樣的效果呢?

- CPU 在時間上被劃分成了兩個運行狀態：**Secure state** 和 **Non-Secure state**
- 空間上，4GB 的地址空間被劃分為兩個陣營：**Secure Memory** 和 **Non-Secure Memory**
- 保存在 Secure Memory 上的代碼就是 **Secure Code**，它必須在 Secure State 下運行；保存在 Non-Secure Memory 上的代碼就是 **Non-Secure Code**，它必須在 Non-Secure State 下運行—簡單說就是”你是你、我是我”。
- Secure Code 可以訪問**所有**的數據。

Non-Secure Memory: 你瞅啥?

Secure Code: 瞅你咋地?

Non-Secure Memory: 不……咋地……你……你瞅我我也不知道……

- Non-Secure Code **只能訪問** Non-Secure Memory 上的數據；

Secure Memory: 你瞅啥?

Non-secure Code: 瞅你咋地?

Secure Memory: 我老大你認識不?

Secure Fault: 是誰在我地盤上橫啊?

Non-secure Code: 哥，這……這誤會阿……哥

Secure Fault: 誤會?Secure Memory 是你來得地 er 麼?你!這!就!是!搞!事!來啊，拖走!

Non-secure Code: 哥……哥，消消氣，你看啊，這地盤也不是你劃分的，咱還不得找管事 er 的人說理不是?

- Secure Memory 和 Non-Secure Memory 是由 **Secure Attribution Unit** 和 **Implementation Defined Attribution Unit** 共同決定的。

  你可以把他們理解為一對夫妻：男人 **IDAU** 主外(由芯片廠商定義)，女主內(**SAU**，是 Secure Code 在運行時刻通過寄存器來配置的)。對 Cortex-M23/33 的每一個總線訪問，SAU 和 IDAU 都會根據目標地址對比自己所掌握的信息進行投票，仲裁的優先順序依次是：**Non-Secure, Non-Secure-Callable** 和 **Secure**。誰更接近 Secure 誰說了算。

SAU: 呦~ Fault 哥，又抓到人啦?

Secure Fault: 可不，Secure Memoy 說這小子是 Non-Secure 的。

Secure Memory: 姐，你可要幫我做主，這小子居然瞅了我一眼!

Non-secure Code: 我哪知道……

SAU/IDAU : 小子，說其他沒用，地址多少?

Non-secure Code: x 鐵的

SAU: 呦，Non-Secure 的人啊，你知道它是 Secure 的麼?你就瞅人家?你也不照照鏡子，Secure Memory 是你能瞅的麼?

Non-secure Code：我哪知道它是 Secure Memory 啊?

SAU/IDAU : 我說是就是!

Non-secure Code：哦……哦……(低頭不敢看)

SAU/IDAU：Fault 哥，去查查老祖宗立下的規矩，該匯報就匯報，該 RESET 就 RESET，按程序辦事。

Secure Fault: 得令，走你!

- Secure Code 會通過一些專用的 API 來為 Non-Secure Code 提供服務，這些專用的 API 被稱為 **Secure Entry**。Secure Entry 必須放在 **Non-Secure-Callable** 屬性的 Memory 內。Non-Secure-Callable 本身其實是 Secure Memory，但是它特殊的地方就在於可以存放 Secure Entry。**—你可以把 NSC 理解為銀行的營業大廳，而用防彈玻璃隔開，僅開一個個櫃台小洞就是 Secure Entry**。
- **TrustZone for ARMv8-M 所追求的是，Non-Secure Code 和 Secure Code 都 以為自己獨佔整個系統**。對 Cortex-M23 來說，Non-Secure Code 以為自己運行在一個 Cortex-M0/M0+上；而對 Cortex-M33 來說，Non-Secure Code 來說，它已為自己獨佔的是 Cortex-M3/M4。

我們知道，Non-Secure Code 的獨佔是”錯覺”，因為它並不知道 Secure Code 的存在，任何出格的訪問(對站在它角度來說未知空間的訪問)都會被截獲，當作 Secure Fault。Secure Code 的獨佔是貨真價實的，因為它不僅知道 Non-Secure 的存在，也可以隨時訪問他們。

為了構建這種錯覺，代價是巨大的。對於一些核心的資源，比如 **NVIC，SysTick，MPU**，Cortex-M23/33 都貨真價實的為 Non-Secure Code **提供了額外的一份**；對於另外一些昂貴的核心資源，比如**流水線，通用寄存器頁、Debug 邏輯、浮點運算單元**，Secure Code 就只能屈尊和 Non-Secure Code 分時復用(**共享**)了。

- **Non-Secure Code 與 Secure Code 通過 Secure Entry 進行信息交換。** 當 Secure Code 調用 Secure Entry 時，如果 Secure Entry 是有效的，並且存放於 NSC Memory 裡，那麼 CPU 就會從 Non-Secure state 切換到 Secure State，並運行 NSC 裡面的代碼(NSC 是 Secure Memory，所以裡面的代碼是 Secure Code)，一般來說，Secure Entry 會立即馬不停蹄的跳轉到其它單純的 Secure Memory 中去執行。

  Secure Code 借助特殊的函數指針以回調(Callback)的方式調用 Non-Secure Memory 中的代碼(並暫時性的切換回 Non-Secure state)。至於這裡面眾多的細節，還請自己閱讀公開的各類文檔。

- **Secure Code 和 Non-Secure Code 都可以擁有自己的異常處理程序，這裡面涉及到的 Secure 和 Non-Secure 的切換都是硬件自動處理好的，程序員不用操心**。值得說明的是，復位以後，整個系統都是 Secure 狀態，所有的異常都屬於 Secure Code，這個時候，只有 Secure Code 可以大發慈悲的分配一些中斷給 Non-Secure Code 使用。而且，我們有一個專門的寄存器位可以讓所有 Non-Secure Exception 的優先級比任何 Secure Exception 都”低人一等”。

總結來說，TrustZone for ARMv8-M 創造了兩個世界，**Secure domain** 和 **Non-Secure domain**。SAU/IDAU 共同將 4G 地址空間拆分為 Secure，Non-Secure 和 Non-Secure-Callable。無論是 Secure Domain 還是 Secure Domain 都可以把自己看作是一個普通的 Cortex-M0 或者 Cortex-M3 處理器來開發。大家都有自己獨立的 NVIC，Systick 甚至是獨立的 MPU—Secure 可以打 Non-Secure，Non-Secure 不能還手，因為**所有的資源理論上都是首先屬於 Secure domain 的**。

- **結語**

Cortex-M23/33 所引入的安全擴展為整個 ARM 嵌入式系統的信息安全提供了基礎(Fundation)或者說根基(The Root of the Security)。然而，單單依靠”基礎”不足以構建起堅固的防禦體系—芯片廠商、OEM 廠商，軟件 IP 廠商、工具鏈、系統軟件及應用設計，任何一環都需要引入必要的信息安全技術和措施。我們可以說：

沒有 TrustZone for ARMv8-M，建立在 Cortex-M 系統之上的安全將是空中樓閣；而單單依靠 TrustZone for ARMv8-M 來保護信息安全，更是掩耳盜鈴。用戶不僅要知道自己要保護什麼，如何保護，更要知道：構建一個基礎堅實的安全設計所要做的 比貼一張"TrustZone Inside"的標簽紙 到自己的產品上 要多得多的多。

更多關微控制器嵌入式信息安全設計的內容，將在後續的文章中(如果有的話)慢慢為您展開，如果您對嵌入式信息安全有什麼想說的，歡迎直接在公眾號留言。