WP473 (v1.0.1) 2016 年 4 月 11 日

如何将软件移植到 64 位 ARM 异构平台

Zynq UltraScale+ MPSoC 提供稳健可靠的平台，可供系统架构师开展创新，而且无需担心损失现有的软件基础架构投资。

摘要

Zynq® UltraScale+ ™ MPSoC 的核心 ARM®v8 架构使系统设计人员只需极少量修改就可以快速启用并运行现有的 ARMv7 代码。这种架构兼容性使设计人员可以提高生产力，加速产品上市进程，同时减少开发成本和工程设计投资。

简介

软件设计人员运用基于新一代 ARMv8 架构的赛灵思 Zynq UltraScale+ MPSoC，不仅可以充分利用其先进性能，同时还能保持现有软件投资。

在美国计算机学会杂志《美国计算机协会通讯全集》的一篇专栏中，Steve Furber 指出，32 位 ARMv7 SoC（例如 Zynq UltraScale+ MPSoC）是最丰富和最受欢迎的 SoC 产品。[参考资料 1] 另一层隐含的意义是：ARMv7 SoC 的用户群很大，意味着软件——从简单的库、工具一直到完整操作系统——的用户群更为庞大。最近，随着向 64 位平台的移植持续加速，系统设计人员面临的问题在于：是花大力气将现有软件移植到新架构，还是简单地在新平台上重新开始开发软件。[参考资料 2]

任何一个具有一定规模的软件项目都需要投入大量的人力和软件架构。损失这些投资无论从短期还是长期看对项目都有毁灭性影响。因此，SoC 平台之间的软件移植会带来很大不确定性。移植不当很容易导致功能和性能不理想。在这个需要缩减设计预算和加速产品上市进程的时代，不允许有太多的不确定性。

赛灵思 Zynq 产品组合的最新成员 Zynq UltraScale+ MPSoC 采用新一代 ARMv8 架构，其精湛的设计可解决上述种种问题。ARMv8 采用设计人员熟悉的 CPU 架构和软件开发流程，使他们能够立即着手进行项目移植，且可将现有软件基础架构投资损失降至最低。

ARMv8 通过设计简化软件移植

Zynq UltraScale+ MPSoC 中的 ARMv8 架构与前代 ARMv7 架构兼容，同时可扩展支持最新特性与功能，因此能够从根本上简化软件移植。这样可创建熟悉的开发环境，让软件开发人员在移植过程伊始就能专注于代码本身。ARMv8 架构通过两种基本方式实现这种深度兼容：

- ARMv8 架构支持两种不同执行状态：
  - AArch32，本机 32 位状态
  - AArch64，本机 64 位状态
- AArch64 状态中的 64 位 ARMv8 指令集是 ARMv7 中指令集的自然延伸；AArch32 状态中的 32 位指令集直接兼容于 ARMv7 中的指令集。

AArch64 执行状态是本机 ARMv8 执行环境。它支持 ARMv8 架构的完整特性集与功能。相反，AArch32 执行状态提供与 ARMv7 环境的本机向后兼容。从软件执行角度来看，该状态为本机 ARMv7 环境。开发人员可选择让代码在编译时间内在哪种执行状态下运行。

正常运行时间内，CPU 可在 AArch64 和 AArch32 这两种执行方式之间即时切换，以提供软件所要求的执行环境。这些上下文变化发生在异常边界，无需用户或软件设计人员进行输入；32 位代码和 64 位代码高效并行执行。

当在 AArch64 状态下运行时，提供完整的 64 位 ARMv8 指令集。很大程度上，64 位 ARMv8 指令集是 ARMv7 中指令集的延伸。只需对指令集在语法和行为上稍加修改，就能支持 32 位指令以及 32 位与 64 位混合寄存器。当在 AArch32 状态下运行时，处理器使用标准 32 位指令以及 32 位寄存器；软件设计人员处理指令集的方式与使用其他 ARMv7 处理器时相同。

使用代码库

低层裸机和 RTOS 代码开发人员是受低层处理器架构差别影响最多的人。当开始移植软件时，最佳方式是启用编译器中所有警告和错误消息，使用 ARMv8 编译器重新简单编译软件，不做修改。对这种方式下生成的警告或错误信息进行分析，这样软件开发团队可以确定哪些错误可以忽略，以及哪些需要在移植过程中加以注意。

ARMv8 编译器本机支持标准高级代码，例如 C 和 C++；当有合适的 ARMv8 板支持包 (BSP) 时，可编译和运行这类代码。相比之下，汇编代码时则需要特别注意如何使用代码。尽管 ARMv8 中依然存在诸多 ARMv7 汇编指令，但是它们的语法或行为存在微秒的变化。有些编码结构的编译或行为与 ARMv7 不同，其中包括：硬编码存储器位置 (适用任何软件移植项目)、对 ARMv7 协处理器 (例如 CP15) 和寄存器名称的访问，以及数据对齐。由 ARM 发布的 *ARM® Cortex ™ -A 程序员 ARMv8-A 使用指南* (DEN0024A) 对移植问题进行了详细的分析。

我们不妨看看容易移植的代码实例以及会产生问题的代码，这样有助于我们从概念上讨论架构之间的区别。

有的指令只需很少的操作就可从 ARMv7 转换为 ARMv8 语法，ADD 指令就是个很好的例子。ARMv7 中的寄存器的命名方案为 **R*n***，其中 ***n*** 是寄存器的个数。这样命名直接了当，因为 ARMv7 中所有寄存器都是 32 位。在 ARMv8 中，寄存器既可以是 32 位也可以是 64 位。32 位寄存器的命名方案为 **W*n***，而 64 位寄存器的命名方案则为 **X*n***。这样软件设计人员必须根据正在考虑的操作对象的尺寸，对现有的代码进行评估并适当地修改。见表 1。

*表 1：*ADD **各种环境中指令命名方案**

ARMv7	ARMv8 A32	ARMv8 A64 (32 位)	ARMv8 A64 (64 位)
ADD Rd, R*n*	ADD Rd, R*n*	ADD Wd, W*n*	ADD Xd, X*n*

移植 ADD 这样的指令不仅需要评估代码的最终用途，还要相应地改变寄存器名称。根据整个代码库的复杂程度，甚至可以自动执行移植。

其他情况下，例如访问 ARMv7 协处理器，可能需要更全面的代码分析。这种分析被认为是最佳方法，因为协处理器控制很多针对整个处理器的系统级属性和功能，而且控制协处理器是针对 ARMv7 的常见操作。

当在 AArch64 状态下运行时，无法直接访问 CP15 寄存器，因为它们不在 ARMv8 架构中。当在 AArch32 状态下运行时，则可通过概念协处理器访问它们。这是由 AArch32 执行状态提供的结构，用于为原有的 32 位代码提供一致的执行环境。

例如，假设一个程序需要 Main ID Register——协处理器 CP15 中 c0 寄存器的一部分——的内容。使用 ARMv7 的专用 MRC 指令和 ARMv8 的 AArch32 执行状态访问这个值：

```
MRC p15, 0, r1, c0, c0, 0
```

直接在 AArch32 状态运行的原有代码无需修改；但移植到 AArch64 状态的代码需要根据新架构的要求进行调整。

在本机 AArch64 执行状态下，Main ID Register 值位于名为 MIDR_El*n* 的新专用寄存器中。该寄存器可方便地通过系统寄存器访问指令来访问。

MRS：

```
MRS x1, MIDR_EL1
```

因此，开发人员应该知道这种专用操作和调用，以便相应地更改汇编指令。同样，这是单行修改，不会影响周围软件代码的整体结构。此外，还要注意源寄存器和目标寄存器都是本机 64 位寄存器，与 64 位架构的其余部分一致。

保持现有软件完整

对基于 Linux 操作系统的软件而言，移植条件通常比 RTOS 或裸机代码更直接。底层的 Linux 操作系统可通过抽象消除很多底层硬件中的差异。大多数情况下，只需使用 64 位 Linux 编译器进行重新编译就可以让代码运行在 64 位 Linux 上。赛灵思 PetaLinux 工具可针对 64 位 Linux 环境在本机中整合和重新编译代码，从而自动执行该过程。

其他情况下，开发人员可能需要使代码保持为 32 位二进制数。软件可能需要与只有 32 位形式的外部库链接；或者包含需要花时间转换的 32 位汇编代码。移植这类软件时，可利用很多现代 Linux 版本中都有

的 *multiarch* 或 *multib* 功能实现显著加速。尽管不是完全一样，但这些经常可以互换使用。

Multiarch 是一种系统和文件系统布局的配置方法，以便运行 32 和 64 位动态链接程序。原来的 32 位 ARMv7 Linux 程序无需修改，可直接导入，并与新的 64 位程序共存。该功能可用来快速建立并运行系统，同时移植特定的 32 位应用程序并针对底层 64 位架构进行微调。

支持 Multiarch 的版本将 32 位和 64 位库纳入相同 Linux 文件系统中，以实现这种兼容性。当执行 64 位应用时，使用本机 64 位库；32 位应用则使用 32 位库。

创建和维护 multiarch 操作系统需要投入不少时间。幸好，开源项目，例如 Yocto Project、Canonical 和 Linaro，提供易于集成的内容。Yocto Project 提供完整的端对端构建系统；Canonical 和 Linaro 提供现成可用的参考文件系统，使开发人员快速取得进展。赛灵思答复 66636 详细介绍了如何在 ZCU102 评估平台上集成 Ubuntu Core 14.04 文件系统。

默认情况下，这些 multiarch 文件系统只使用 64 位应用程序。由于它们基于 dpkg 管理工具，因此只需简单的命令行接口即可添加更多架构：

```
# dpkg -add-architecture armhf
```

配置 dpkg 工具以支持 32 位架构 (例如 armhf)，然后，使用更高级的封装管理工具，例如 apt-get，就像在 Linux 的 Intel x86 兼容版本上一样。

对于有些系统来说，通过 multiarch 在 64 位主机上运行现有的 32 位应用程序就足够了，无需其他工作。对于其他系统，multiarch 可起到节省时间的作用，同时还要对软件协议栈的其他部分进行微调，并针对底层 64 位架构和操作系统来优化它们。

总结

ARMv8 为软件开发人员提供很多性能、安全和架构方面的新功能，但是，如果软件开发人员因担心丢掉现有的软件架构而不愿意使用它，那么这些新功能也就无用武之地。ARMv8 可从硬件和软件接口实现处理器架构的自然演进，从而消除这种顾虑。将开发人员熟悉的指令集实现自然演进，以此简化低级代码的移植。此外，还可以回退到 AArch32 状态，这样开发人员就能够快速运行现有的 32 位代码，以最少的时间和精力在新硬件上评估代码。最后，multiarch Linux 允许将一些代码不经任何修改直接运行，这样就可以显著减少整体系统集成。ARMv8 处理器，例如 Zynq UltraScale+ MPSoC 允许开发人员利用新一代功能并保持原有软件投资。

如需了解有关 Zynq UltraScale+ MPSoC 的更多信息，敬请访问：china.xilinx.com. 也可以查看 ARM 的官方移植指南，网址：https://community.arm.com/docs/DOC-8453

参考资料

Jason Fitzpatrick, “Steve Furber 访谈”《美国计算机协会通讯全集》第 54 卷，编号：5：
http://cacm.acm.org/magazines/2011/5/107684-an-interview-with-steve-furber/fulltext

David Brash, “ARMv8-A 架构演进” *ARM Connected Community,* 2016 年 1 月 5 日：
https://community.arm.com/groups/processors/blog/2016/01/05/armv8-a-architecture-evolution

修订历史

下表列出了本文档的修订历史：

日期	版本	修订描述
2016 年 04 月 11 日	1.0.1	更新标题
2016 年 03 月 31 日	1.0	赛灵思初始版本

免责声明

关于与汽车相关用途的免责声明