Application Note

在 Sitara MPU 上启用 Matter

摘要

本应用手册探讨了如何在 Sitara 处理器器件上实现和使用 Matter 连接协议。以下各节概述了 Matter 的实现和演示，提供了从 SK-AM62B 收集的示例数据。

内容

插图清单

商标

1 引言

Matter 是一种开源应用层连接协议，专门用于创建与物联网设备交互的统一方法。它构建在 IP 基础之上，这使其原生适用于多种网络标准，例如 WiFi (802.11)、以太网 (802.3) 和 Thread (802.15.4)。

2 当前设计

此协议常见的实现方式是 connectedhomeip 项目的 chip-tool 中的参考设计，网址为：https://github.com/project-chip/connectedhomeip。该存储库包含：

- Matter 服务器的实现方式
- 消息传递接口的定义
- 广播和侦听广播事件所需的所有网络实用程序，包括：
  - mDNS 服务器
  - DNS 解析器
- 用于启用蓝牙配置的工具
- 每种可能的端点群集类型的定义
- 每个端点群集的示例
- 控制器/管理员应用的示例

对于一个简单的演示来说，只有两个方面很重要：管理员和端点。因此，我们将重点介绍 chip-tool 和锁定应用示例。从 chip-tool 开始，此示例应用有一个命令行界面 (CLI)，它充当管理员，能够链接到端点并根据该端点启用的集群发出命令或获取状态。lock-app 是端点的一个例子，通常用于控制电子锁存器。此应用会注册一些命令，例如：

- Lock
- Unlock
- Unbolt
- GetUser
- SetUser
- GetDoorState
- SetDoorState
- SetCredential
- GetCredential

其中每个命令都在 chiptool 中注册，并附带日志和状态更改消息，这些消息在调用时会广播。

3 支持

我们的演示使用了 SK-AM62B，有关该器件的更多信息，请参阅以下链接：https://www.ti.com.cn/tool/cn/SK-AM62B。关于软件，可以按照以下步骤使用 Yocto 编译演示：

1. 对于您的 Ubuntu 主机，下载先决程序：https://software-dl.ti.com/processor-sdk-linux/esd/AM62X/09_00_00_03/exports/docs/linux/Overview_Building_the_SDK.html#prerequisites-one-time-setup。
2. git clone https://git.ti.com/git/arago-project/oe-layersetup.git tisdk
3. cd tisdk
4. ./oe-layertool-setup.sh -f configs/processor-sdk/processor-sdk-09.00.00-config.txt
5. cd sources
6. git clone -b kirkstone https://github.com/kraj/meta-clang.git
7. cd meta-arago/meta-arago-demos/recipes-apps
8. mkdir matter && cd matter
9. 创建一个名为 matter_git.bb 的文件并添加以下内容：

```
SUMMARY = "Matter IoT connectivity on TI boards"
DESCRIPTION = "This recipe primes the matter environment"
LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
Apache-2.0;md5=89aea4e17d99a7cacdbeed46a0096b10"
```

```
BRANCH = "master"
SRC_URI = "gitsm://github.com/project-chip/connectedhomeip.git;protocol=https;branch=$
{BRANCH};lfs=1"

SRCREV = "a98bc64856aa161197e7dc7c1ffbdcc43323eda3"

do_matter_bootstrap[network] = "1"
do_compile[network] = "1"

TARGET_CC_ARCH += "${LDFLAGS}"
DEPENDS += " glib-2.0 gn-native ninja-native avahi dbus-glib-native pkgconfig-native python3-
native boost zap-native openssl-native ca-certificates-native clang-native"
RDEPENDS_${PN} += " libavahi-client openssl "
FILES:${PN} += "usr/share"
INSANE_SKIP:${PN} += "dev-so debug-deps strip"

PACKAGECONFIG ?= ""
PACKAGECONFIG[debug] = "is_debug=true,is_debug=false"

GN_TARGET_ARCH_NAME:aarch64 = "arm64"
GN_TARGET_ARCH_NAME:arm = "arm"
GN_TARGET_ARCH_NAME:x86 = "x86"
GN_TARGET_ARCH_NAME:x86-64 = "x64"

def gn_target_arch_name(d):
    """Returns a GN architecture name corresponding to the target machine's
    architecture."""
    name = d.getVar("GN_TARGET_ARCH_NAME")
    if name is None:
        bb.fatal('Unsupported target architecture. A valid override for the '
            'GN_TARGET_ARCH_NAME variable could not be found.')
    return name

# this variable must use spaces and double quotes for parameter strings because
# *gn* is evil
GN_ARGS = " \
    ${PACKAGECONFIG_CONFARGS} \
    target_cpu="${@gn_target_arch_name(d)}" \
    target_arch="${TUNE_FEATURES}" \
    target_os="linux" \
    treat_warnings_as_errors=false \
    enable_rtti=true \
    enable_exceptions=true \
"

# Make sure pkg-config, when used with the host's toolchain to build the
# binaries we need to run on the host, uses the right pkg-config to avoid
# passing include directories belonging to the target.
GN_ARGS += 'host_pkg_config="pkg-config-native"'

S = "${WORKDIR}/git"

common_configure() {
    # this block must use spaces and double quotes for strings because *gn* is
    # evil
    PKG_CONFIG_SYSROOT_DIR=${PKG_CONFIG_SYSROOT_DIR} \
    PKG_CONFIG_LIBDIR=${PKG_CONFIG_PATH} \
    gn gen out/ --args='
        ${GN_ARGS}
        import("//build_overrides/build.gni")
        target_cflags=[
            "-DCHIP_DEVICE_CONFIG_WIFI_STATION_IF_NAME=\"wlan0\"",
            "-DCHIP_DEVICE_CONFIG_LINUX_DHCPC_CMD=\"udhcpc -b -i %s \"",
        ]
        custom_toolchain="${build_root}/toolchain/custom"
        target_cc="${CC}"
        target_cxx="${CXX}"
        target_ar="${AR}"
    '
}

export https_proxy
export http_proxy
export ftp_proxy
export no_proxy

do_matter_bootstrap() {
```

```
    . ${S}/scripts/bootstrap.sh
}

do_configure() {
    . scripts/activate.sh
    pip install click

    cd ${S}/examples/chip-tool
    common_configure

    cd ${S}/examples/lock-app/linux
    common_configure

    cd ${S}/examples/thermostat/linux
    common_configure

    cd ${S}/examples/lighting-app/linux
    common_configure
}

do_compile() {
    . scripts/activate.sh

    cd ${S}/examples/chip-tool
    ninja -C out/

    cd ${S}/examples/lock-app/linux
    ninja -C out/

    cd ${S}/examples/thermostat/linux
    ninja -C out/

    cd ${S}/examples/lighting-app/linux
    ninja -C out/
}

do_install() {
    install -d -m 755 ${D}${bindir}

    # Install chip-tool
    install ${S}/examples/chip-tool/out/chip-tool ${D}${bindir}

    # lock-app
    install ${S}/examples/lock-app/linux/out/chip-lock-app ${D}${bindir}
    install ${S}/examples/thermostat/linux/out/thermostat-app ${D}${bindir}
    install ${S}/examples/lighting-app/linux/out/chip-lighting-app ${D}${bindir}
}

addtask matter_bootstrap after do_unpack before do_configure

INSANE_SKIP_${PN} = "ldflags"
```

10. 创建一个名为 zap_git.bb 的文件并添加以下内容：

```
PN = "zap-native"
SUMMARY = "ZAP prebuilt tools"
DESCRIPTION = "ZAP prebuilt binaries"
LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
Apache-2.0;md5=89aea4e17d99a7cacdbeed46a0096b10"

PACKAGES = "${PN}"

PV = "v2023.08.04-nightly"
SRC_URI = "https://github.com/project-chip/zap/releases/download/${PV}/zap-linux-
x64.zip;unpack=yes"
SRC_URI[sha256sum] = "b254a0c066ef6b1fe7c2bdd1ab5b137ca80413f0952dfe6e64f4b0fdc4479b55"

S = "${WORKDIR}"

#INSANE_SKIP:${PN} = " already-stripped arch file-rdeps "
BBCLASSEXTEND = "native"
INHIBIT_PACKAGE_STRIP = "1"
INHIBIT_SYSROOT_STRIP = "1"
INHIBIT_PACKAGE_DEBUG_SPLIT = "1"
INHIBIT_FILE_RDEPS = "1"
INHIBIT_PACKAGE_DEBUG_SPLIT_CHECK = "1"
```

```
INHIBIT_PACKAGE_DEPMODE_CHECK = "1"
INHIBIT_PACKAGE_RELOCATE = "1"
INHIBIT_PACKAGE_UNPACK = "1"

INSANE_SKIP:${PN} += "dev-so"
inherit native

do_install() {
    install -d -m 0755 ${D}${bindir}/
    cp -ar zap* ${D}${bindir}/
    # This is a workaround to bypass the issue that zap-cli modified by build system
    chmod 444 ${D}${bindir}/zap-cli
}

do_package_qa[noexec] = "1"
EXCLUDE_FROM_SHLIBS = "1"

# This is a workaround to bypass the issue that zap-cli modified by build system
do_deploy() {
chmod 755 ${D}${bindir}/zap-cli
}

do_populate_sdk:append() {
chmod 755 ${D}${bindir}/zap-cli
}

addtask deploy after do_install do_populate_sysroot
addtask deploy before do_cleansstate
addtask deploy before do_clean
```

11. cd ../../../../../
12. cd sources/bitbake/lib/bb/fetch2/
13. 按如下方式修改 gitsm.py：

```
diff --git a/lib/bb/fetch2/gitsm.py b/lib/bb/fetch2/gitsm.py
index c5f7c03c..ee852224 100644
--- a/lib/bb/fetch2/gitsm.py
+++ b/lib/bb/fetch2/gitsm.py
@@ -122,6 +122,7 @@ class GitSM(Git):
     url += ';protocol=%s' % proto
     url += ";name=%s" % module
     url += ";subpath=%s" % module
+    url += ";lfs=1"

     ld = d.createCopy()
     # Not necessary to set SRC_URI, since we're passing the URI to
@@ -238,7 +239,7 @@ class GitSM(Git):
     # All submodules should already be downloaded and configured in the tree. This simply sets
     # up the configuration and checks out the files. The main project config should remain
# unmodified, and no download from the internet should occur.
-    runfetchcmd("%s submodule update --recursive --no-fetch" % (ud.basecmd), d, quiet=True,
workdir=ud.destdir)
+    runfetchcmd("GIT_LFS_SKIP_SMUDGE=1 %s submodule update --recursive --no-fetch" %
(ud.basecmd), d, quiet=True, workdir=ud.destdir)

def implicit_urldata(self, ud, d):
    import shutil, subprocess, tempfile
```

14. cd ../../../../../build/
15. 打开文件 conf/local.conf 并在文件底部添加以下内容：IMAGE_INSTALL:append = "matter"
16. . conf/setenv
17. MACHINE=am62xx-evm bitbake-layers add-layer ../sources/meta-clang/
18. MACHINE=am62xx-evm bitbake tisdk-default-image
19. 使用以下目录中生成的 WIC 映像烧录 SD 卡：./arago-tmp-default-glibc/deploy/images/am62xx-evm/tisdk-default-image-am62xx-evm.wic.xz

生成 wic 映像后，请参阅以下使用 SD 卡启动 EVM 的说明：https://dev.ti.com/tirex/content/tirex-product-tree/am62x-devtools/docs/am62x_skevm_quick_start_guide.html

4 演示

图 4-1 显示了两个使用芯片工具和锁应用的 AM62x 器件通过以太网相互连接。

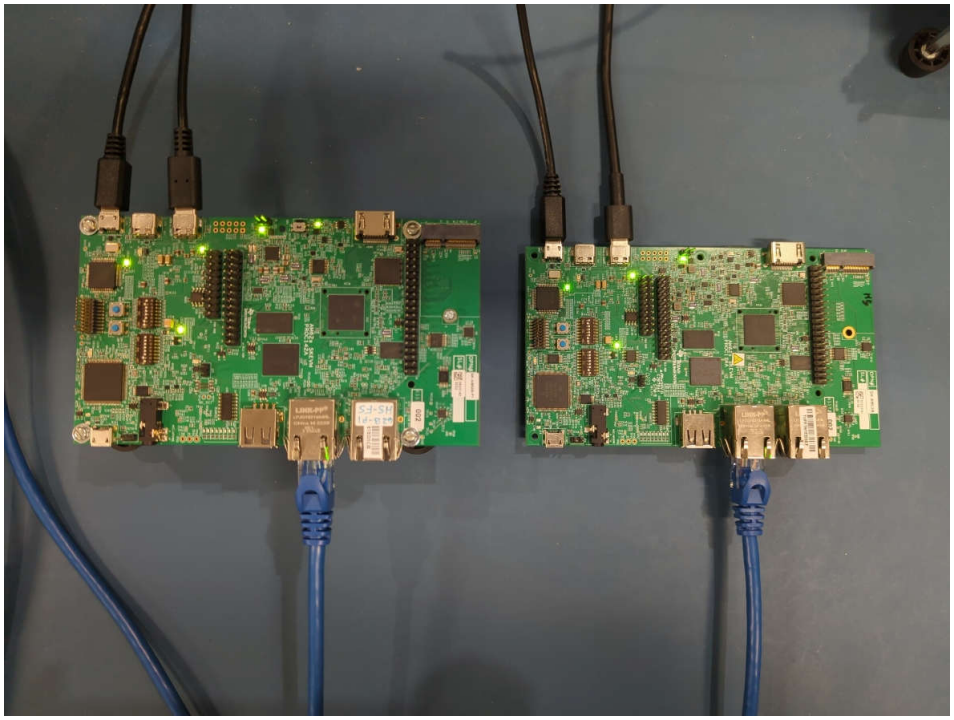

图 4-1. 硬件设置

图 4-2 显示了如何使用锁定应用将 AM62x 器件设置为端点。

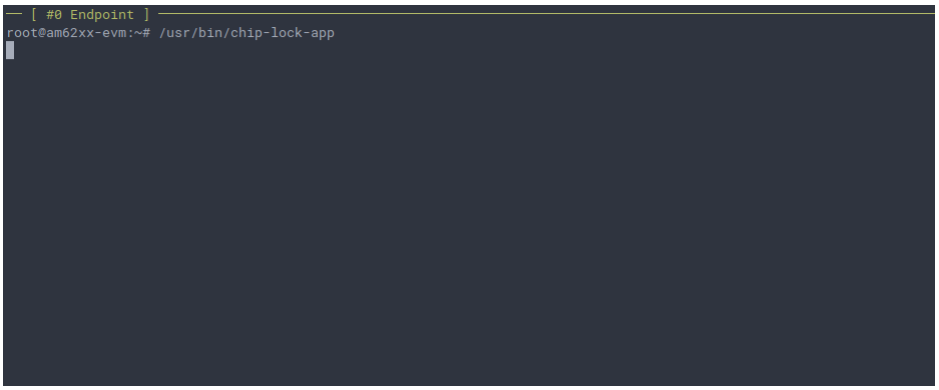

图 4-2. 创建端点

图 4-3 显示了预期的端点日志。请注意器件配置信息。

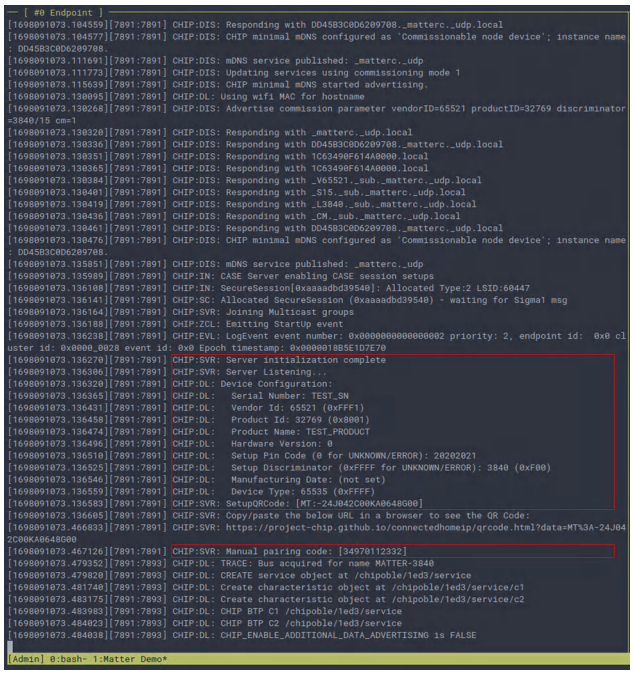

图 4-3. 预期端点日志

图 4-4 显示了管理员如何使用芯片工具与端点配对。

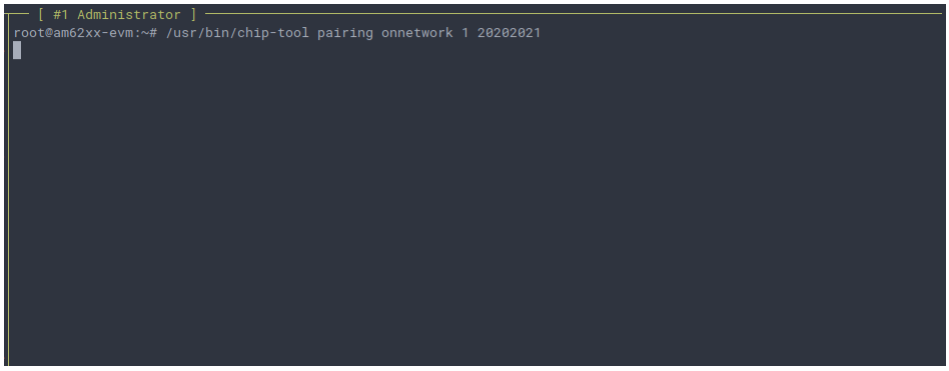

图 4-4. 与端点器件配对

图 4-5 显示预期的成功配对尝试日志。注意日志中的 CommissioningComplete 响应。

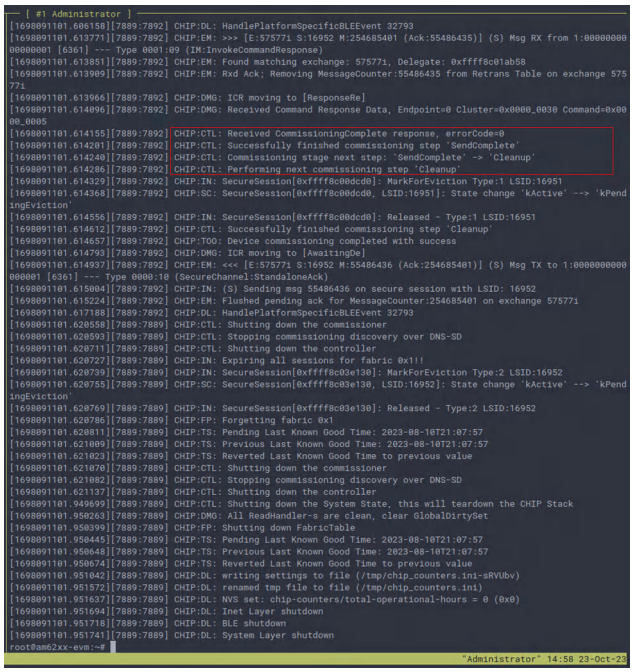

图 4-5. 成功配对

图 4-6 显示端点的状态设置为锁定。

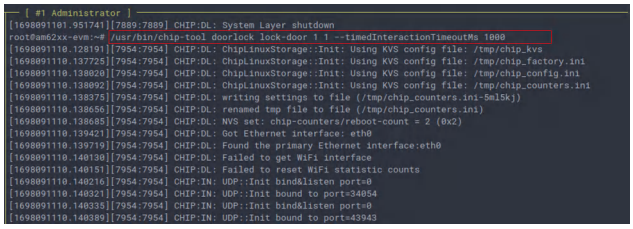

图 4-6. 将锁定状态设置为已锁定

图 4-7 显示了门锁请求之后在端点上报告的状态。

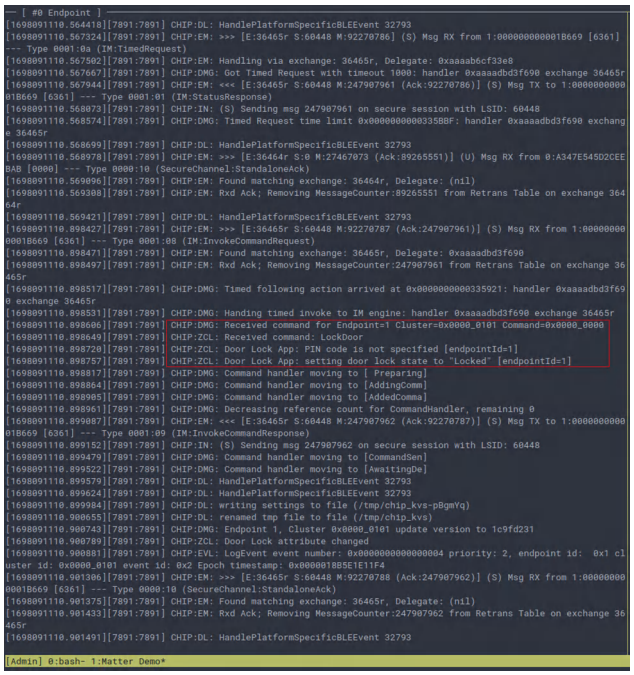

图 4-7. 端点日志中的锁定状态

要查看上述内容的录制演示以及同步更新的完整端点和管理员日志，请参阅以下内容：https://asciinema.org/a/620956。

5 总结

本应用手册的主要目标是演示如何从 connectedhomeip 工程编译 matter 的参考设计，并运行简单的锁定/解锁演示。尽管使用的是 AM62x 器件，上述说明适用于任何 ARM 32 位和 ARM 64 位 TI 处理器。

6 参考资料

- 德州仪器 (TI)，*AM625* 产品文件夹。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265