Application Note

Enabling Matter on Sitara MPU

Krunal Bhargav and Randolph Sapp

ABSTRACT

This application note explores the implementation and usage of the Matter connectivity protocol on Sitara processor devices. The following sections outline the enablement and demonstration of Matter, including example data collected from SK-AM62B.

Table of Contents

List of Figures

Trademarks

1 Introduction

Matter is an open-source application-layer connectivity protocol that specializes in creating a uniform method of interacting with IoT devices. It's built on top of IP allowing it to work natively over multiple network standards, such as WiFi (802.11), Ethernet (802.3), and Thread (802.15.4).

2 Current Implementation

The most common implementation of this protocol is the reference implementation present in the chip-tool in the connectedhomeip project at: https://github.com/project-chip/connectedhomeip. This repository contains:

- An implementation of the Matter server
- A definition of the messaging interface
- All the required networking utils for broadcasting and listening for broadcast events, including:
  - A mDNS server
  - A DNS resolver
- Tools for enabling bluetooth provisioning
- A definition of every possible endpoint cluster type
- An example for every endpoint cluster
- An example of a Controller / Administrator application

There are only two things that are importatant for a simple demo: an Administrator and an Endpoint. As such, the focus will be on the chip-tool and lock-app examples. Starting with chip-tool, this example application has a Command Line Interface (CLI) that acts as an Administrator capable of linking to endpoints and issuing commands or fetching status based on the clusters enabled by that endpoint. The lock-app is an example of an endpoint that would normally be controlling an electronic latch. This application registers a handful of commands like:

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

Where each of these commands are registered with chiptool and have accompanying log and state change messages that are broadcast when called.

3 Enablement

For our demo, the SK-AM62B and for more information about the device are used, see the following link: https://www.ti.com/tool/SK-AM62B. With regards to software, the following steps may be used to compile the demo using Yocto:

1. For your Ubuntu host machine, download the prerequisites: 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. Create a file called matter_git.bb and add the following content:

```
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. Create a file called zap_git.bb and add the following content:

```
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. Modify gitsm.py like the following:

```
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. Open file conf/local.conf and add the following at the bottom of file: 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. Burn SD card using the wic image generated in the following directory: ./arago-tmp-default-glibc/deploy/images/am62xx-evm/tisdk-default-image-am62xx-evm.wic.xz

After generating the wic image, see the following instructions for booting the EVM with SD card: https://dev.ti.com/tirex/content/tirex-product-tree/am62x-devtools/docs/am62x_skevm_quick_start_guide.html

4 Demonstration

Figure 4-1 shows two AM62x devices using the chip-tool and lock-app interfacing with each other over Ethernet.

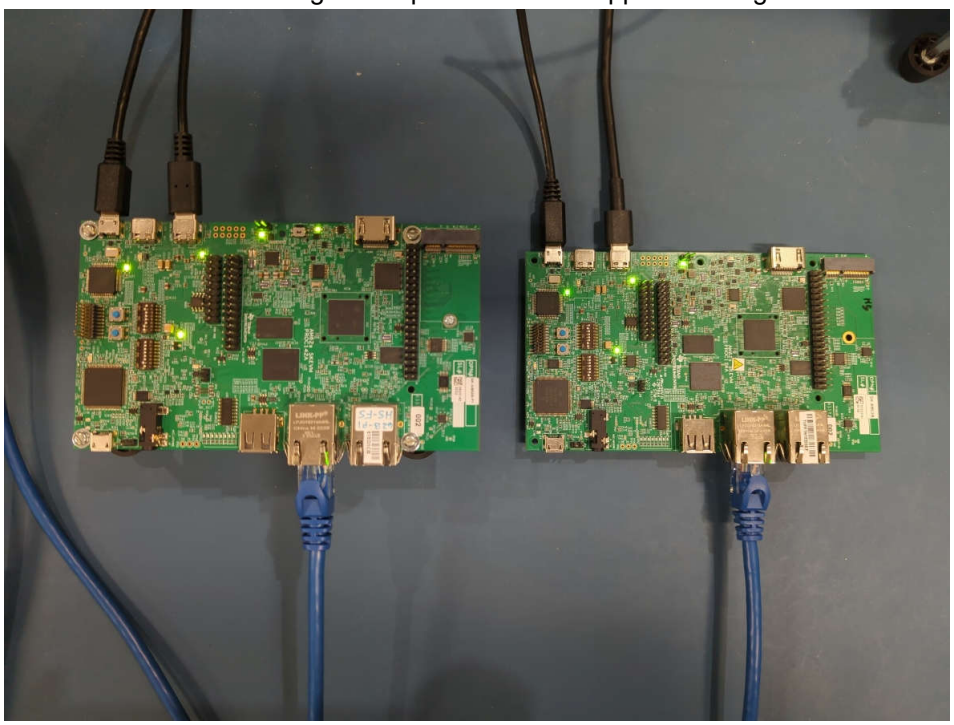

Figure 4-1. Hardware Setup

Figure 4-2 shows how to setup the AM62x device as an endpoint using the lock app.

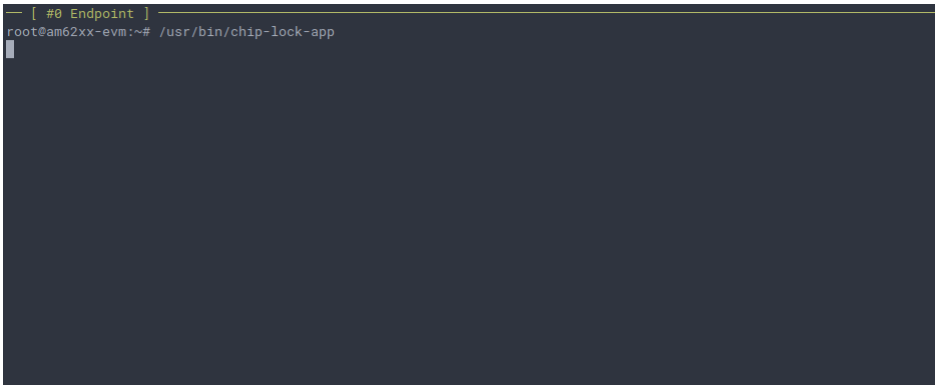

Figure 4-2. Creating an Endpoint

Figure 4-3 shows what an expected endpoint log should look like. Note the device configuration information.

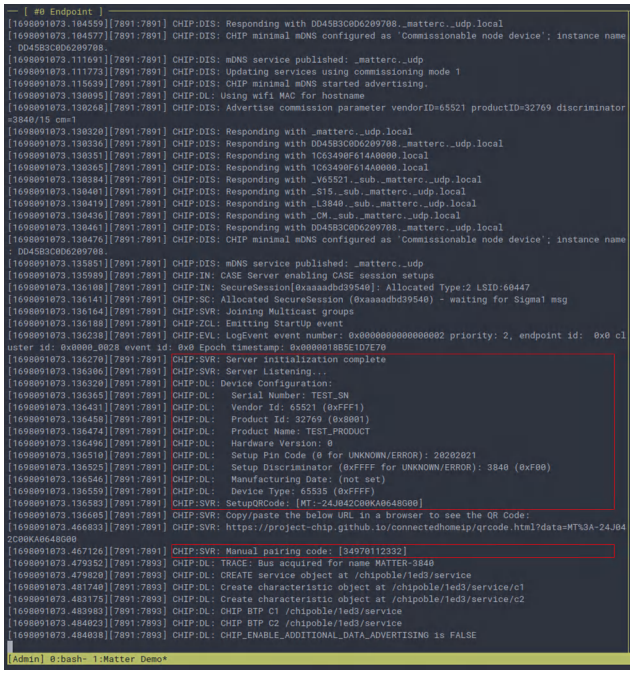

Figure 4-3. Expected Endpoint Log

Figure 4-4 shows how an administrator would pair with the endpoint using the chip-tool.

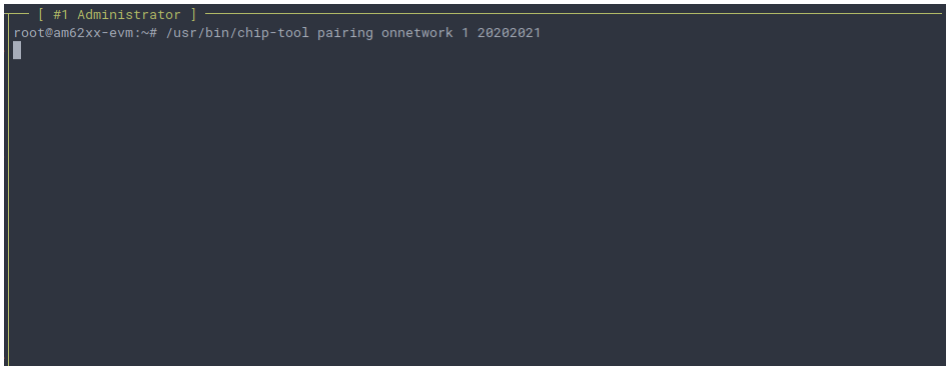

Figure 4-4. Pairing With Endpoint Device

Figure 4-5 shows the expect log of a successful pairing attempt. Note the CommissioningComplete response in the log.

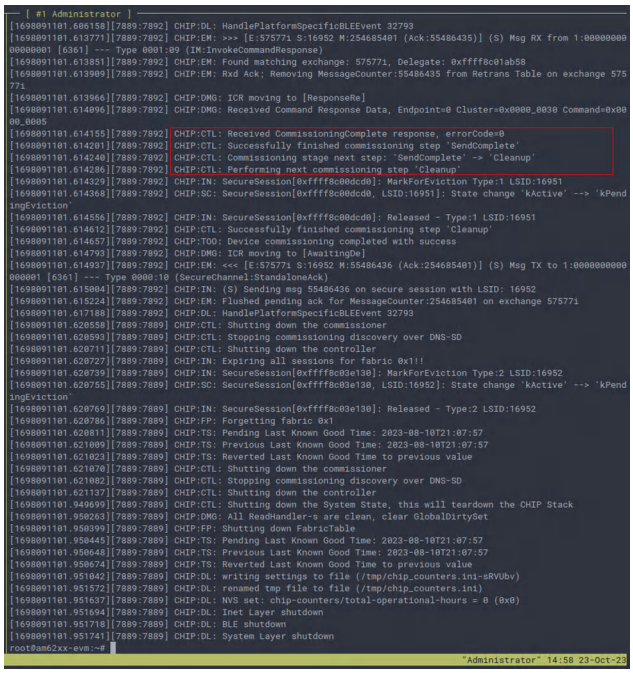

Figure 4-5. Successful Pairing

Figure 4-6 shows that the status of the endpoint is set to be locked.

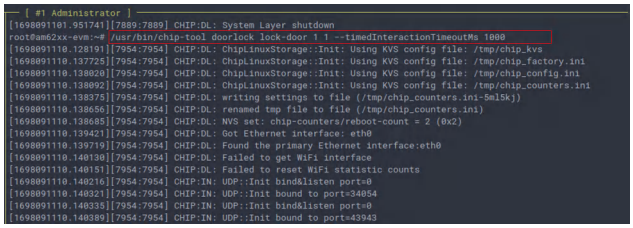

Figure 4-6. Setting Lock Status to Locked

Figure 4-7 shows the status reported on the endpoint following the lock-door request.

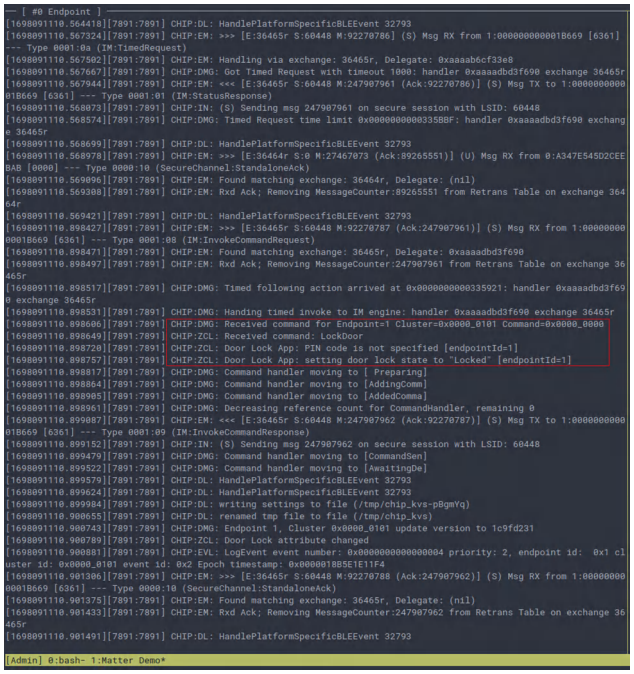

Figure 4-7. Lock Status in Endpoint Log

To see a recorded demonstration of the above with full endpoint and administrator logs being updated in sync, see the following:https://asciinema.org/a/620956.

5 Summary

The main goal of this application note is to demonstrate how to compile a reference implementation of matter from the connectedhomeip project and run a simple lock/unlock demo. Even though a AM62x device is used, the above instructions are applicable to any ARM32 bit and ARM64-bit TI processors.

6 References

- Texas Instruments, *AM625*, product folder.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265