Application Note

Sitara MPU 上で Matter を有効化

Krunal Bhargav, Randolph Sapp, Divyansh Mittal

概要

このアプリケーションノートは、Sitara プロセッサ デバイス上での Matter コネクティビティ プロトコルの実装と使用方法について説明します。以下のセクションでは、SK-AM6X デバイスから収集したサンプルデータを含め、Matter の実現とデモの概要を示します。

目次

図の一覧

商標

1 はじめに

Matter は、オープンソースのアプリケーション層コネクティビティ プロトコルであり、IoT デバイスとの対話型操作を行うための一様な方法を製作することに特化しています。IP 上に構築され、WiFi (802.11)、イーサネット (802.3)、Thread (802.15.4) など、複数のネットワーク規格上でネイティブに動作します。

2 電流の実装

このプロトコルの最も一般的な実装は、https://github.com/project-chip/connectedhomeip の connectedhomeip プロジェクトのチップツールに存在するリファレンス実装です。このリポジトリの内容:

- Matter サーバーの実装
- メッセージング インターフェイスの定義
- ブロードキャスト イベントのブロードキャストとリッスンに必要なすべてのネットワーク ユーティリティで、以下のものが含まれます。
  - mDNS サーバー
  - DNS リゾルバ
- Bluetooth プロビジョニングを有効にするための各種ツール
- 考えられるすべてのエンドポイント クラスタ タイプの定義
- すべてのエンドポイント クラスタの例
- コントローラ / 管理者アプリケーションの例

シンプルなデモで重要なのは、管理者とエンドポイントの 2 つだけです。そのため、チップツールとロックアプリのサンプルに焦点を当てています。チップツール以降、このサンプル アプリケーションには、エンドポイントにリンクしてそのエンドポイントで有効になっているクラスタに基づいてコマンドを発行したり、ステータスを取得したりする管理機能として使用できるコマンドライン インタフェース (CLI) があります。ロックアプリは、通常電子ラッチを制御するエンドポイントの一例です。このアプリケーションは、以下のような一連のコマンドを登録します。

- ロック
- ロック解除
- ボルト除去
- GetUser
- SetUser
- GetDoorState
- SetDoorState
- SetCredential
- GetCredential

これらのコマンドはそれぞれチップツールに登録され、ログと状態の変更メッセージが呼び出されたときにブロードキャストされます。

3 イネーブル

デモでは、AM62P と AM62L を使用しました。Linux と連携して動作し、ネットワークに接続できる任意の SoC を一般的に使用できます。詳細については、AM62P および AM62L を参照してください。ソフトウェアに関しては、ホスト PC 上でクロスコンパイルを行うために次の手順を使用できます。

1. Ubuntu ホストマシンに、以下のようなカーネル バージョンを使用する SDK をダウンロードして、インストールします。AM62L 11.00 SDK インストーラー。
2. 次のコマンドを使用して、SDK 内の rootfs を解凍します。

```
cd <SDK Install Path>/filesystem/<device>
tar -xf tisdk-default-image-am62lxx-evm.rootfs.tar.xz -C temp/
```

抽出されたこのディレクトリのパスを手元に保管してください。

3. 次の手順を使用して、Matter リポジトリのクローンを作成して更新します。

```
git clone --recurse-submodules git@github.com:project-chip/connectedhomeip.git
cd connectedhomeip
git pull
git submodule update --init
```

リポジトリのサイズが大きいため、クローン作成に時間がかかります。

4. build_matter_example.sh スクリプトをダウンロードし、Matter のルートディレクトリ内に配置します。

```
#!/bin/bash
set -e

# ================================================================================
# Matter aarch64 Cross-Compilation Build Script (Unified)
#
# This script handles all necessary fixes and configurations:
# - Bluezoo dependency fix for Python 3.10
# - TI SDK toolchain wrapper creation
# - Complete Matter build process
# ================================================================================

if [[ $# -ne 1 ]]; then
  echo "Error: Please enter exactly one example-name as argument"
  echo "Usage: $0 <your_argument>"
  exit 1
fi

EXAMPLE_NAME="$1"

# ================================================================================
# CONFIGURATION - MODIFY THESE VARIABLES FOR YOUR SETUP
# ================================================================================

# SDK Path
SDK_PATH="/home/<user>/ti-processor-sdk-linux-am62lxx-evm-11.00.15.05"

# Path to your aarch64 sysroot
SYSROOT_AARCH64="$SDK_PATH/filesystem/am62lxx-evm/temp"     # TI SDK sysroot

# Toolchain binary prefix (TI SDK uses aarch64-oe-linux)
TOOLCHAIN_TARGET="aarch64-oe-linux"       # TI SDK compatible target

# Path to your aarch64 cross-compilation toolchain (using TI SDK native toolchain)
TOOLCHAIN_PREFIX="$SDK_PATH/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/
$TOOLCHAIN_TARGET"  # TI SDK toolchain

# Path to connectedhomeip repository (relative to script location)
REPO_PATH="."                                # CHANGE THIS if different

echo "=== Matter aarch64 Cross-Compilation Build Script (Unified) ==="
echo "Toolchain: $TOOLCHAIN_TARGET"
echo "Sysroot: $SYSROOT_AARCH64"
echo "Example: $EXAMPLE_NAME"
echo ""

echo "================================================================================"
echo "STEP 1: FIX BLUEZOO DEPENDENCY ISSUE"
echo "================================================================================"

REQUIREMENTS_FILE="$REPO_PATH/scripts/tests/requirements.txt"
if [ -f "$REQUIREMENTS_FILE" ]; then
    # Check if bluezoo is already commented out
    if grep -q "^bluezoo" "$REQUIREMENTS_FILE"; then
        echo "Commenting out bluezoo dependency (requires Python 3.11+)..."
        sed -i 's/^bluezoo/#bluezoo/' "$REQUIREMENTS_FILE"
        echo "✓ Bluezoo dependency commented out"
    else
        echo "✓ Bluezoo dependency already fixed"
    fi
else
    echo "Warning: Requirements file not found at $REQUIREMENTS_FILE"
fi

echo "================================================================================"
echo "STEP 2: VERIFY PATHS AND TOOLCHAIN"
```

```
echo "================================================================================"

# Construct toolchain binary paths
export CC_AARCH64="$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-gcc"
export CXX_AARCH64="$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-g++"
export AR_AARCH64="$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-ar"
export STRIP_AARCH64="$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-strip"
export LD_AARCH64="$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-ld"

# Add toolchain to PATH for any remaining usage
export PATH="$TOOLCHAIN_PREFIX:$PATH"

if [ ! -f "$CC_AARCH64" ]; then
    echo "Error: Compiler not found at $CC_AARCH64"
    echo "Please check your TOOLCHAIN_PREFIX and TOOLCHAIN_TARGET variables"
    exit 1
fi

if [ ! -d "$SYSROOT_AARCH64" ]; then
    echo "Error: Sysroot not found at $SYSROOT_AARCH64"
    echo "Please check your SYSROOT_AARCH64 path"
    exit 1
fi

if [ ! -d "$REPO_PATH" ]; then
    echo "Error: Repository not found at $REPO_PATH"
    echo "Please check your REPO_PATH variable"
    exit 1
fi

echo "✓ Toolchain: $($CC_AARCH64 --version | head -1)"
echo "✓ Sysroot: $SYSROOT_AARCH64"
echo "✓ Repository: $REPO_PATH"

echo "================================================================================"
echo "STEP 3: CREATE TOOLCHAIN WRAPPER"
echo "================================================================================"

cd "$REPO_PATH"
#
# # Create toolchain wrapper directory
WRAPPER_DIR="$PWD/toolchain-wrapper-bin"
mkdir -p "$WRAPPER_DIR"

# Create symbolic links with the names GN expects
echo "Creating symbolic links for GN compatibility..."
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-gcc" "$WRAPPER_DIR/aarch64-linux-gnu-gcc"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-g++" "$WRAPPER_DIR/aarch64-linux-gnu-g++"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-ar" "$WRAPPER_DIR/aarch64-linux-gnu-ar"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-strip" "$WRAPPER_DIR/aarch64-linux-gnu-strip"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-ld" "$WRAPPER_DIR/aarch64-linux-gnu-ld"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-objdump" "$WRAPPER_DIR/aarch64-linux-gnu-objdump"
ln -sf "$TOOLCHAIN_PREFIX/${TOOLCHAIN_TARGET}-nm" "$WRAPPER_DIR/aarch64-linux-gnu-nm"

echo "✓ Created toolchain wrapper directory: $WRAPPER_DIR"
echo "Contents:"
ls -la "$WRAPPER_DIR/"

# Add wrapper to PATH
export PATH="$PWD/toolchain-wrapper-bin:$PATH"

echo "================================================================================"
echo "STEP 4: SETUP BUILD ENVIRONMENT"
echo "================================================================================"

source scripts/activate.sh

echo "Testing cross-compilation..."
echo 'int main(){return 0;}' > test.c
$CC_AARCH64 --sysroot="$SYSROOT_AARCH64" -o test test.c
file test
rm test test.c
echo "✓ Cross-compilation test passed"

echo "================================================================================"
echo "STEP 5: CONFIGURE AND BUILD"
echo "================================================================================"

#Check if the example exists
```

```
if [ ! -d "examples/${EXAMPLE_NAME}" ]; then
    echo -e "No such '$EXAMPLE_NAME' exists in examples!! \nExiting !!"
    exit 1
#Check if example does not need specific platform like linux to build
elif [[ $(find ./examples/ -maxdepth 2 -type f -name args.gni | grep -c "$EXAMPLE_NAME") -gt
0 ]]; then
    ROOT_PATH="examples/$EXAMPLE_NAME"
#Check if example needs specific platform to build and linux platform is available
elif [[ $(find ./examples/ -type f -name "args.gni" -path "*/linux/*" | grep -c
"$EXAMPLE_NAME") -gt 0 ]]; then
    ROOT_PATH="examples/$EXAMPLE_NAME/linux"
#Check if example needs specific platform to build but linux platform is NOT available
else
    echo -e "'$EXAMPLE_NAME' is not supported on Linux!! \nExiting !!"
    exit 1
fi

gn gen "out/${EXAMPLE_NAME}-arm64" --root="$ROOT_PATH" --args="
  target_cpu=\"arm64\"
  target_os=\"linux\"
  sysroot=\"$SYSROOT_AARCH64\"
  is_clang=false
  treat_warnings_as_errors=false
  target_cflags = [
    \"-D_GNU_SOURCE\",
    \"-D__USE_GNU\",
    \"-pthread\",
    \"-DCHIP_DEVICE_CONFIG_WIFI_STATION_IF_NAME=\\\"wlan0\\\"\",
    \"-DCHIP_DEVICE_CONFIG_LINUX_DHCPC_CMD=\\\"udhcpc -b -i %s \\\"\",
  ]
  target_ldflags=[\"-pthread\"]
"

echo "Building $EXAMPLE_NAME..."
ninja -C "out/${EXAMPLE_NAME}-arm64"

echo "================================================================================"
echo "STEP 6: VERIFY BUILD RESULTS"
echo "================================================================================"
EXECUTABLE_NAME=$(awk -F'"' '/executable\("/ {print $2}' $ROOT_PATH/BUILD.gn)
echo "Expected executable name: $EXECUTABLE_NAME"
BINARY_PATH="out/${EXAMPLE_NAME}-arm64/${EXECUTABLE_NAME}"
if [ -f "$BINARY_PATH" ]; then
    file "$BINARY_PATH"
    echo "✓ Build complete! Binary located at: $BINARY_PATH"
else
    echo "Error: Build failed, binary not found at $BINARY_PATH"
    exit 1
fi

echo "================================================================================"
echo "BUILD SUMMARY"
echo "================================================================================"
echo "Target: aarch64 (ARM64)"
echo "Toolchain: $TOOLCHAIN_TARGET"
echo "Example: $EXAMPLE_NAME"
echo "Output: out/${EXAMPLE_NAME}-arm64/${EXECUTABLE_NAME}"
echo ""
echo "All fixes applied:"
echo "✓ Bluezoo dependency commented out for Python 3.10 compatibility"
echo "✓ TI SDK native toolchain configured for compatibility"
echo "✓ Toolchain wrapper created for GN naming conventions"
echo "✓ Matter build completed successfully"
echo ""
echo "To modify configuration, edit the variables at the top of this script:"
echo "- TOOLCHAIN_PREFIX: $TOOLCHAIN_PREFIX"
echo "- SYSROOT_AARCH64: $SYSROOT_AARCH64"
echo "- TOOLCHAIN_TARGET: $TOOLCHAIN_TARGET"
echo "- EXAMPLE_NAME: $EXAMPLE_NAME"
echo ""
echo "To build a different example, run the script with other example's name as argument."
```

5. build_matter_example.sh を変更して、次の変数を更新します。
   a. SDK_PATH -> インストールされているプロセッサ SDK のパス。
   b. SYSROOT_AARCH64 -> プロセッサ SDK で抽出されたファイルシステムのパス。

6. 次の方法でスクリプトを実行します。

```
# sudo ./build_matter_example.sh <example-name>
# For example:
sudo ./build_matter_example.sh chip-tool
sudo ./build_matter_example.sh lock-app
```

7. **注**:前述のスクリプトを実行している場合、「ステップ手順 4:ビルド環境のセットアップ」が停止した場合、または使用している環境が最新でない場合は、スクリプトを終了して次のコマンドを実行します。

```
sudo -E bash scripts/bootstrap.sh
```

これにより、環境が最初から再作成されるため、時間がかかります。build_matter_example.sh をもう一度実行します。

8. ビルドが成功すると、スクリプトは次の出力バイナリのパスを記述します。<Matter root>/out/<example_directory>/<example_executable>。
9. 実行可能ファイルをターゲットにコピーします。この例では、AM62L に「chip-lock-app」(チップロックアプリ) を、AM62P に「chip-tool」(チップツール) をコピーし、両方のデバイスが同じネットワークに接続されています。

このアプリケーション ノートの実験は、前回のリビジョンの次のバージョンの SDK/ リポジトリでテストされています。

プロセッサ SDK	11.00.15.05
Matter リポジトリ	コミット:e156205783 (マスターブランチ)

4 デモ

図 4-1 に、ロックアプリを使用する AM62L と、チップツールを使用して Ethernet 経由で相互にインターフェイスになって通信する AM62P を示します。

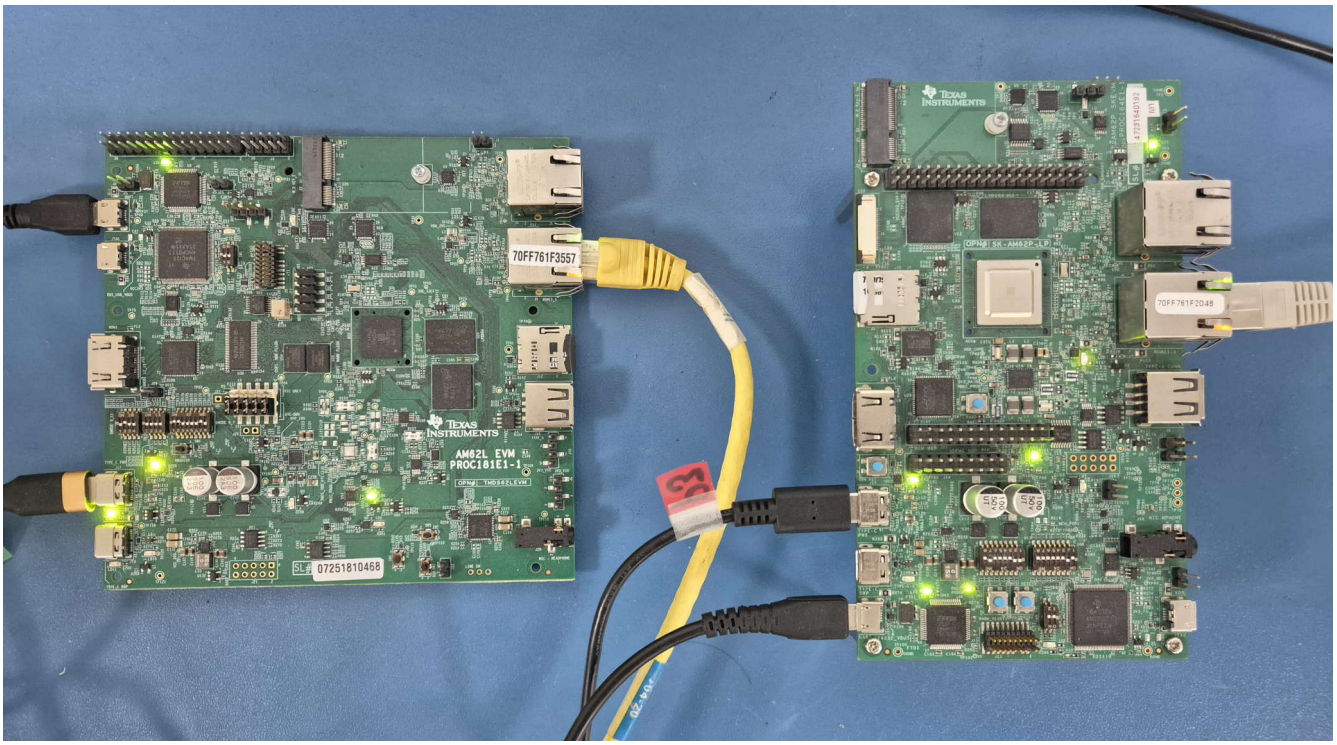

図 4-1. ハードウェア設定

図 4-2 に、ロックアプリを使用して AM62L デバイスをエンドポイントとして設定する方法を示します。

図 4-2. エンドポイントの作成

図 4-3 に、予想されるエンドポイント ログの内容を示します。デバイス構成情報を書き留めます。

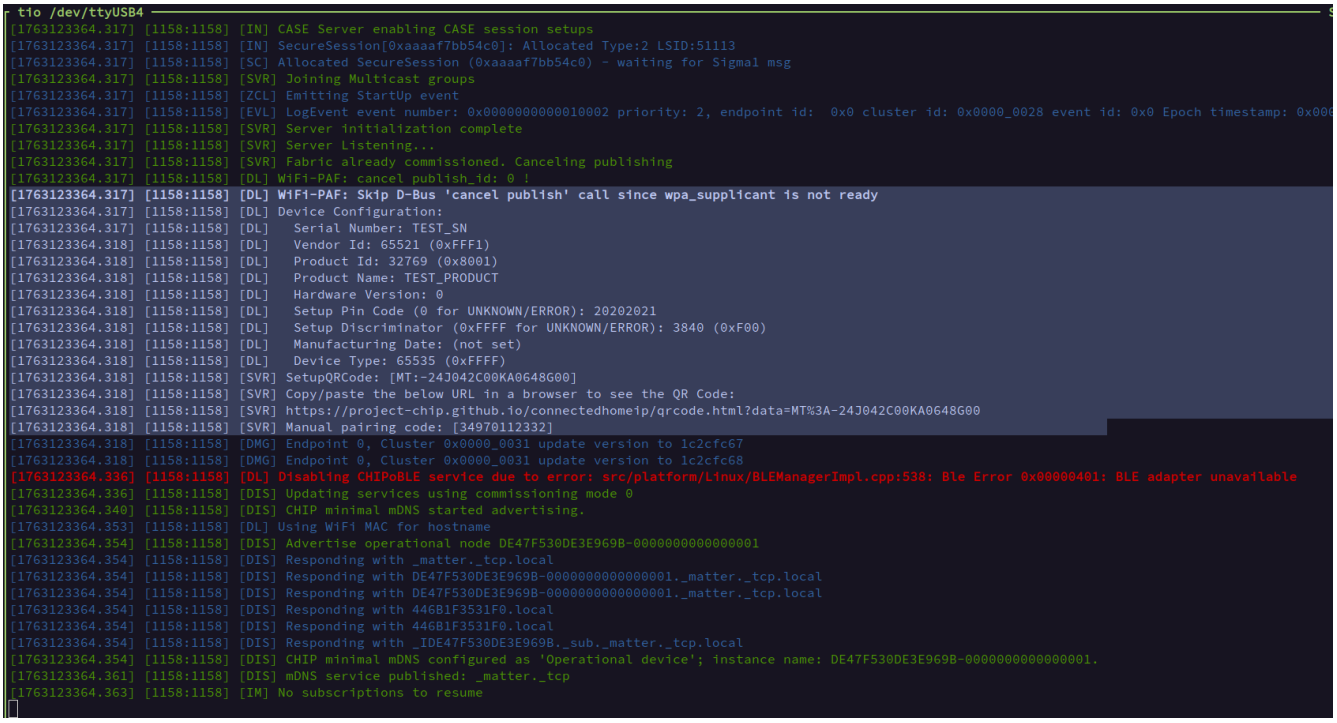

図 4-3. 予期されるエンドポイント ログ

図 4-4 に、管理者がチップツールを使用してエンドポイントとペアリングする方法を示します。

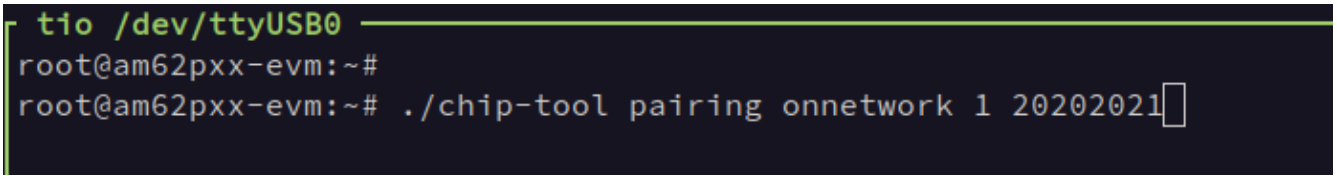

図 4-4. エンドポイント デバイスとのペアリング

図 4-5 に、ペアリングが正常に試行されたことを示す予測ログを示します。CommissioningComplete 応答をログに書き留めます。

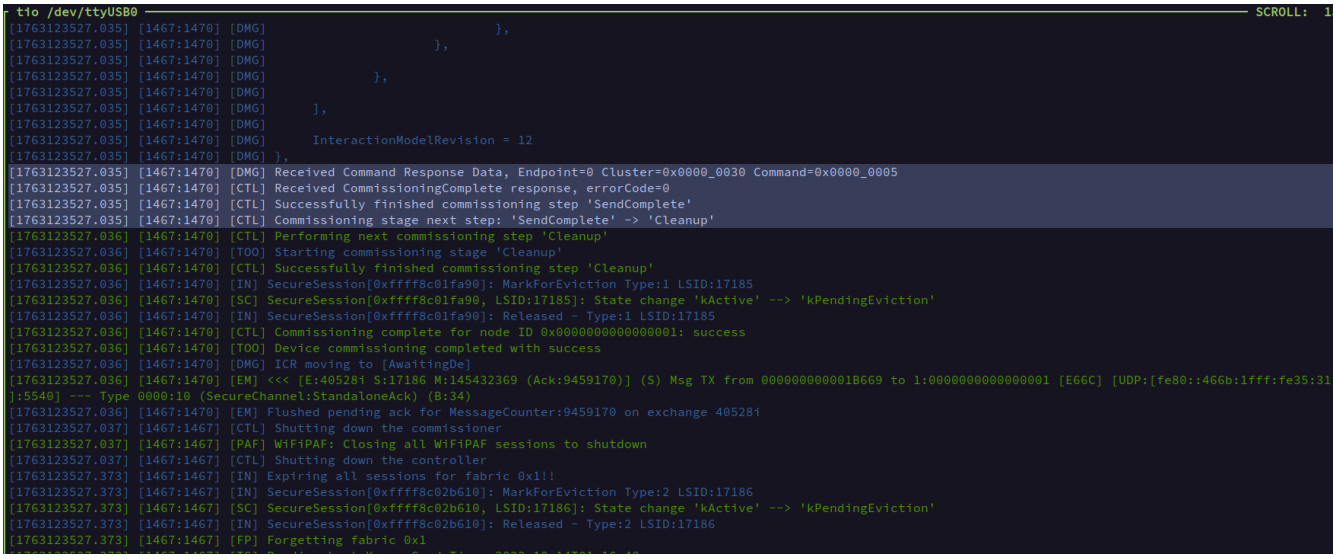

図 4-5. ペアリング成功

図 4-6 に、エンドポイントのステータスがロックされるよう設定されていることを示します。

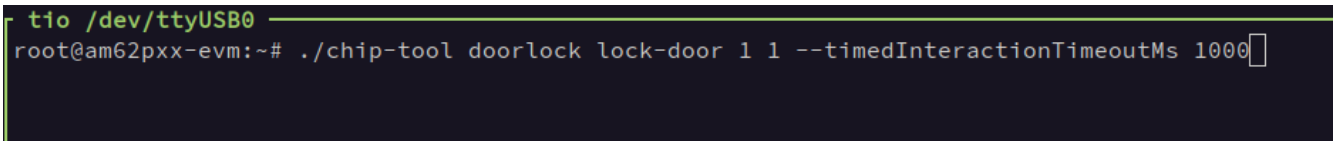

図 4-6. ロック ステータスをロック済みに設定

図 4-7 に、ロックドア要求に続いてエンドポイントで報告されるステータスを示します。

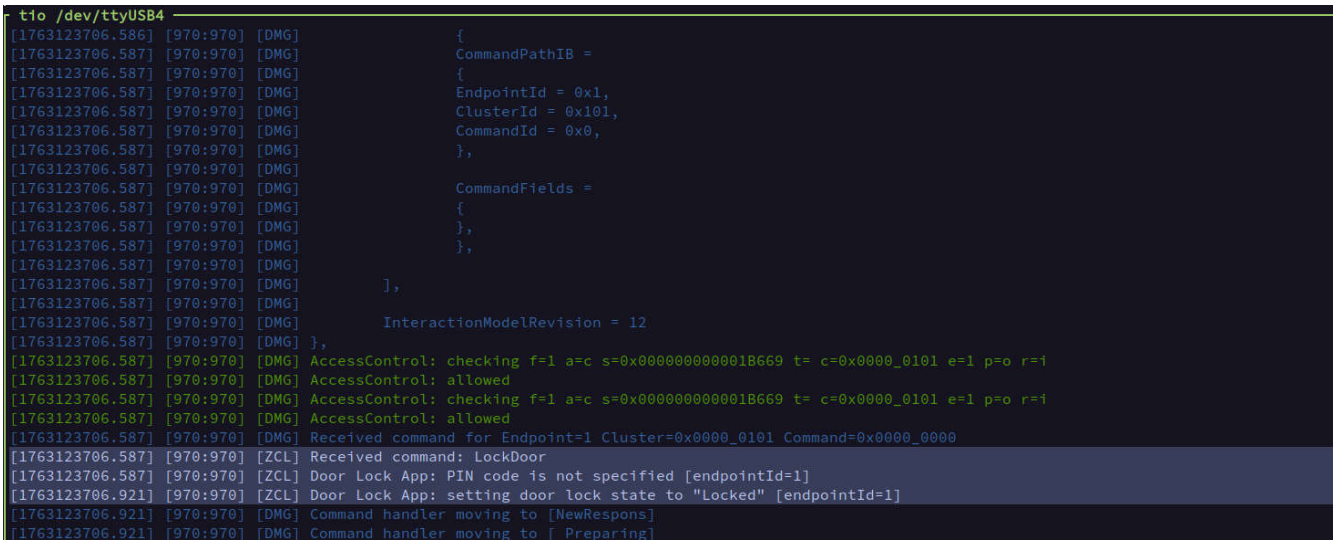

図 4-7. エンドポイント ログのステータスをロック

エンドポイント ログと管理者ログの完全な同期が行われた前述のデモの記録を確認するには、https://asciinema.org/a/755835 を参照してください。

5 まとめ

このアプリケーションノートの主な目標は、connectedhomeip プロジェクトから Matter のリファレンス実装をコンパイルし、シンプルなロック / ロック解除デモを実行する方法を示すことです。AM62L および AM62P デバイスを使用していますが、上記の命令は、ARM32 ビットおよび ARM64 ビットの TI プロセッサにも適用できます。

6 参考資料

- テキサス・インスツルメンツ、*AM62P* 製品フォルダ。
- テキサス・インスツルメンツ、*AM62L* 製品フォルダ。

7 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

Changes from JANUARY 30, 2024 to NOVEMBER 21, 2025 (from Revision * (January 2024) to Revision A (November 2025)) **Page**

- Yocto ベースのアプローチからクロスコンパイル ベースのアプローチへ Matter の対応を更新。 2
- 最新の Matter リビジョンで結果を反映するよう画像とコンソール出力を変更。 6

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月