# IoTSecJP

## ~JTAG ハッキング入門~

著:黒林檎
Twitter:r00tapple
Facebook:村島 正浩
問い合わせ:packr@packr.org

■はじめに

ハードウェアハッキングを勉強しようと考えた方でであれば一度は JTAG を用いたハッキングの記事を読んだことがあると思います。

その時に、簡単だと感じた方やや難しいと感じた方がおられると思います。

では、JTAG とは身近にあるものなのでしょうか?

答えでいうと、皆さんがご家庭でご使用されている無線 LAN ルーターなどにも、JTAG がある可能性が高いです。

また、開発者も開発に使用しているチップに JTAG が存在することを知らずに開発してしまい、ハッカーのよってハッキングの穴に使われるという事もしばしばあります。

しかし、JTAG(Joint Test Action Group)は、ハッキングのために用意された魔法のようなものではありません

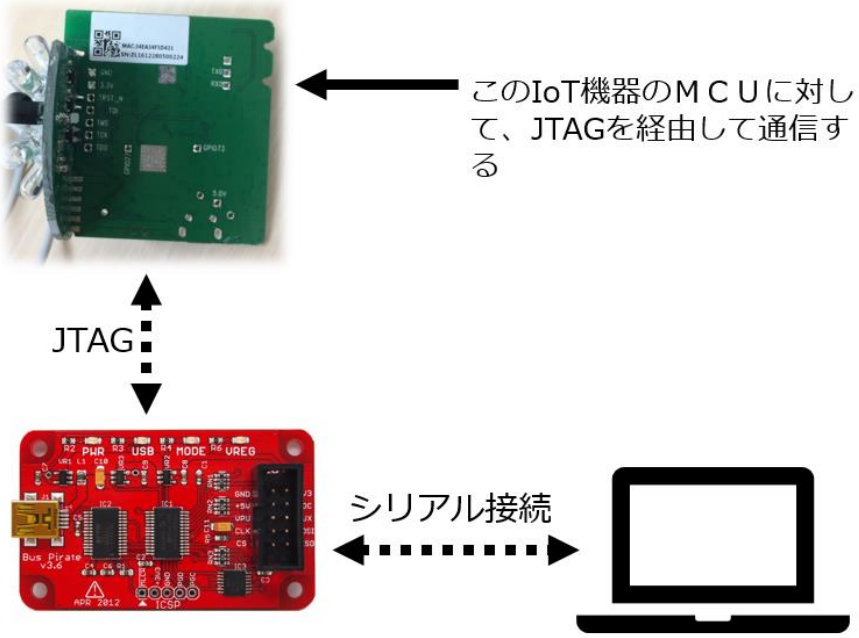
JTAG は、集積回路や基板の検査、デバッグなどに使える、バウンダリスキャンテストやテストアクセスポートの標準 IEEE 1149.1 の通称です。

Wikipedia の概要を確認しても下記のように記述されている。

| |
|---|
| 半導体技術の進歩により集積回路チップのピン間隔も狭くなりプローブを立てての検査が困難になってきている。表面実装の BGA などのパッケージに至っては、物理的に不可能である。そのため検査時に、チップ内部の回路を数珠繋ぎにし、内部状態を順番に読み出すしくみ（バウンダリスキャン（英語版））が考え出された。これをバウンダリスキャンテスト（boundary scan test）といい、それを規格化したのが JTAG である。1990 年に IEEE 1149.1 として標準化されている。<br>近年、検査目的だけではなく、組み込みシステムのソフトウェアのデバッグなどの目的で、ICE の一種として、CPU や FPGA にアクセスする手段として JTAG が用いられるようになってきている。その場合、本来の ICE の機能であるリアルタイム性の観測は、再現できない場合もある。 |

引用 ：https://ja.wikipedia.org/wiki/JTAG

JTAG のハッキングは難しいですが、少しずつ学んでいきましょう。そのためには、JTAG へのハッキングをイメージする必要があります。




(図)JTAG とシリアル接続概要

それでは、ハードウェアをハッキングするために JTAG を使用するために知る必要がある基本的な情報について学んでいきましょう。

■JTAG のピン
ハッキングに必要な JTAG ピンを見ていきましょう。
下記の画像は、筆者の手元にあった IoT 機器にあった JTAG で、各ピンの説明が基板に書かれていました。

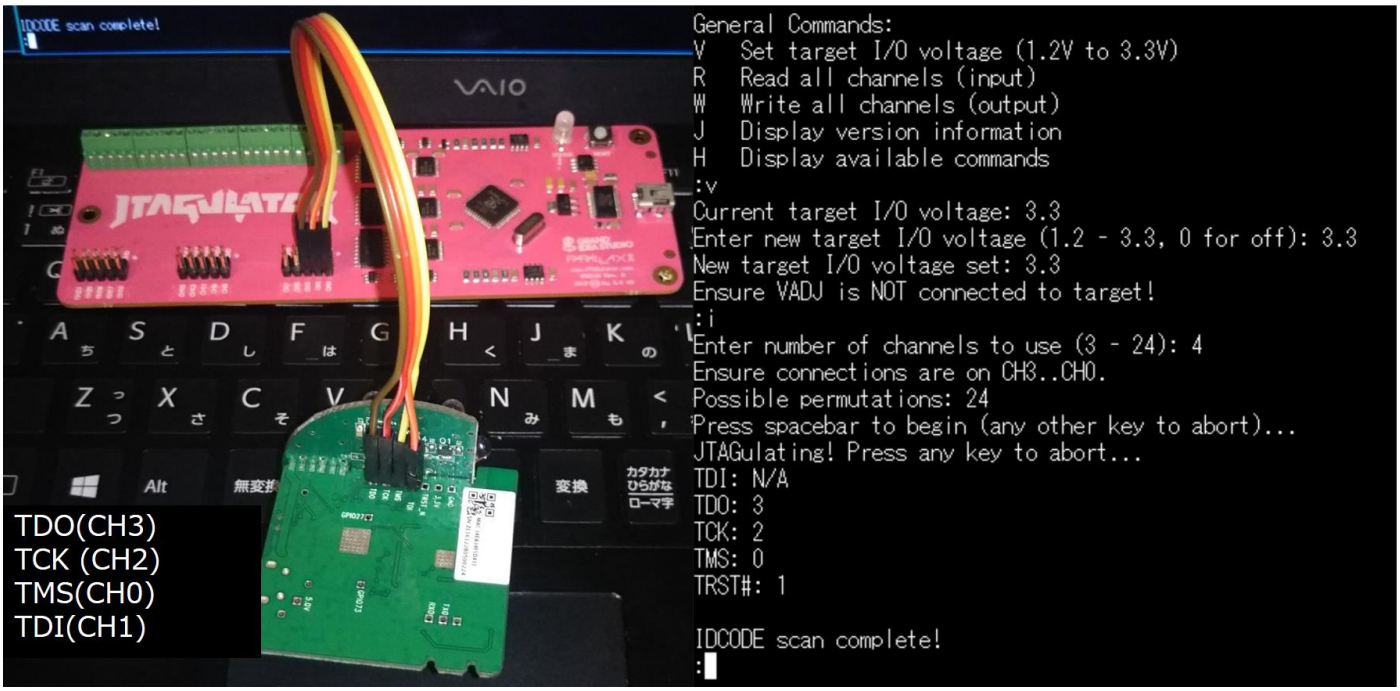
TDI (Test Data In　データ入力)※接続必須
TDO (Test Data Out　データ出力)※接続必須
TCK (Test Clock　クロック)※接続必須
TMS (Test Mode Select　状態制御)※接続必須
TRST (Test Reset リセット信号)※JTAGハックでは必要ない

逆に、ピン配列が基板に書かれておらず推測しなければならない状況があります。
オシロスコープなどで波長を確認し、確認するという手法がよく用いられますが、電子回路などに関する深い知識が必要になります。
そこで今回は、JTAGlator や JTAGEnum といった JTAG のピンを推測するツールを利用してピン配列を推測をしてみたいと思います。
今回筆者は JTAGlator を用いりますが、その手のハッキングツールを利用する際に、本当に使い物になるか動作を確認する必要があります
筆者の場合は、先ほど紹介した IoT 機器を利用して JTAGlator の動作を確認したいと思います。
理由として、PIN 配列が記述されているので JTAGlator の結果と比較することができるためです。。
それでは、TRST はハッキングにはあまり必要が無いので、以下の 4 つのピンを JTAGlator を使用してピンを推測してみたいと思います。
・TDI (Test Data In　データ入力)※接続必須
AP コントローラのスピードを決めるドラマー、またはメトロノーム。このピンの電圧は、単純にリズミカルで安定したビートで上下にパルスします。時計のあらゆる「鼓動」において、TAP コントローラは単一の動作をとる。実際のクロック速度は JTAG 規格では規定されていません。TAP コントローラは、JTAG を制御する外部デバイスから速度を受け取ります。

・TDO (Test Data Out　データ出力)※接続必須
モード選択ピンの電圧は JTAG の動作を制御します。このピンの電圧を操作することによって、JTAG にあなたが何をしたいかを指示します。

・TCK (Test Clock　クロック)※接続必須
チップにデータを供給するピン。JTAG 規格では、このピンを介した通信プロトコルは定義されていません。それは製造業者に委ねられている。JTAG に関する限り、このピンは 1 と 0 がチップに入るための単純な進入方法です。チップとの関係は JTAG とは無関係です。

・TMS (Test Mode Select　状態制御)※接続必須
このオプション信号は、JTAG を既知の正常な状態にリセットするために使用されます。これについては、いくつかの段落でなぜオプションであるのか説明します。

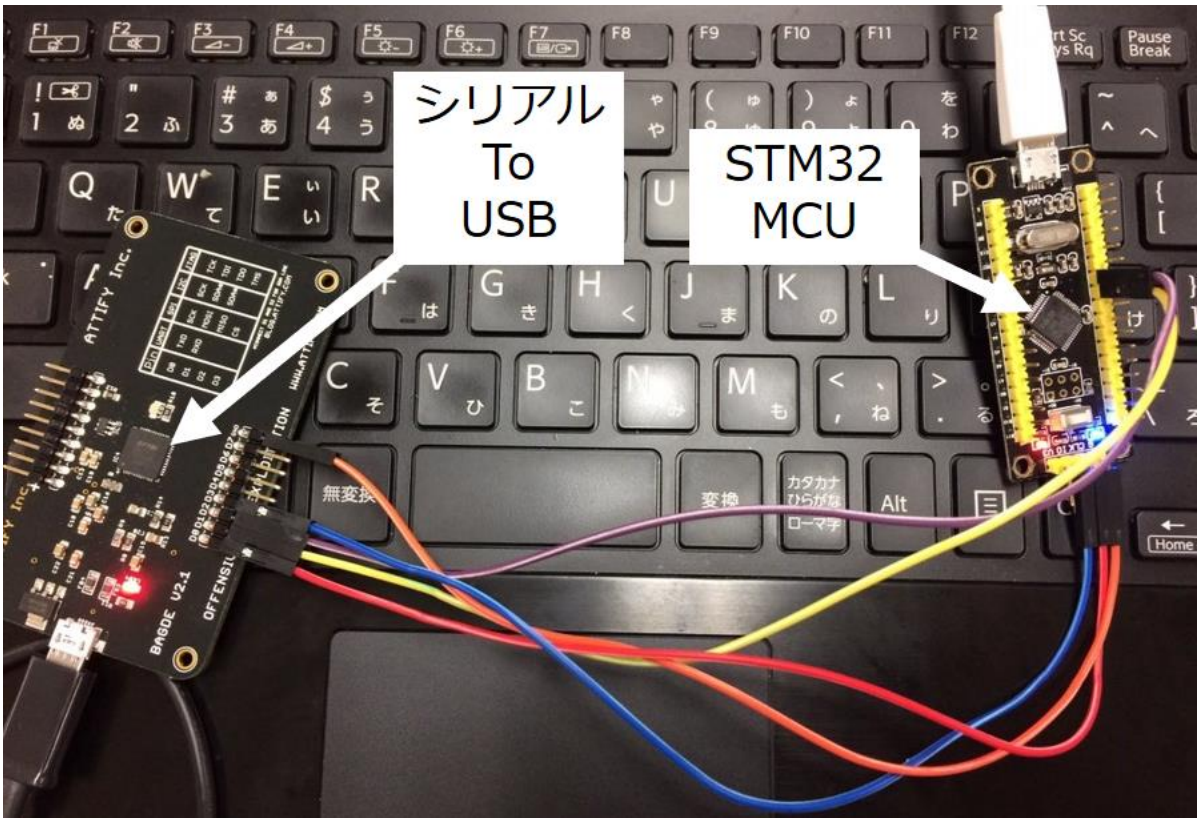



(図) JTAGlator を用いたピンの推測

TMS が N/A になってしまい、代わりに TRST と表示されています。
TRST も接続し JTAGlator を実行したところ下記のようなりました。
##########################################################################################
JTAGulating! Press any key to abort....
TDI: N/A
TDO: 3
TCK: 2
TMS: 0
TRST#: 1
TRST#: 4
##########################################################################################
TRST が 2 個表示されました。
「TRST:1」は基板に記述されていた TDI ピンで、「TRST:4」は今回の検証で追加し基板に記述されていた TRST ピンになります。
この結果から、JTAGlator が嘘をついていることを考え、JTAGlator の結果をふまえて自分で JTAG ピンを推測し接続したほうがよさそうです。

逆に、こういったツールを用いて JTAG ピンの推測をするハッカーに対する対策として、基板上の情報で嘘をつくというのも一種あるのかもしれません。
正確に SPI でもそうですが、JTAG でも正確にピンを推測するにはオシロスコープなどで波長を確認することが一番無難なのかもしれません。

■JTAG へのハッキング
JTAG インターフェースでチップは、コマンド文字列に反応する方法を知りません。
JTAG を単独で使用してデバイスをハッキングしようとすることは技術的には可能といわれていますが、チップの内部動作とアーキテクチャを深く理解する必要があり現実的なアプローチではありません。
JTAG を効率よくハッキングするためには、JTAG の TAP コントローラに入出力する人間が可読できるコードと低レイヤの命令を翻訳できるものが必要です。
そのために、OpenOCD のようなデバッグソフトウェアを使用する事で有効的に使用することができます。
OpenOCD は、さまざまなチップとインタフェースのための唯一のオープンソースリポジトリで、
対象デバイス上の JTAG の TAP コントローラを操作してチップに有効なコマンドとして解釈されるビットを送信することができます。
それでは、OpenOCD でうまく JTAG に接続するために、対象デバイスを配線していきましょう。
今回は、STM32 ARM Cortex 32bit マイクロコントローラを搭載した STM32F103C8T6 に対して JTAG アクセスの練習をしていきたいと思います。



(図)STM32 MCU が搭載された機器とデバッグツールとの配線

「IoT(STM32)」と 「デバッグツール」間が以下のように接続できていれば、配線は問題がない。
TCK <===> TCK
TDI <===> TDI

TDO <===> TDO
TMS <===> TMS
GND <===> GND(GND は、電源電圧を 2 点間で同じにする必要があるので接続する。)

■OPEN OCD を用いたファームウェアダンプ実践
先ほど簡単に OpenOCD について説明しましたが、少し簡単な使い方を学んでいきましょう。
これから皆さんがハードウェアハッキングをしていくことで、たくさんの壁がでてくると思いますが、そのたびに基本に返り、「このパターンであればできた」と思えることが重要です。
BUS pirate や Attify badge または Shikra といったデバッグツールたちを使用して、Linux 上で動作するデバッグソフトウェアである OpenOCD を使うために、cfg ファイルに関する説明をしたいと思います。
一般的な OpenOCD のコマンド文は下記のようになっています。

$sudo openocd -c "telnet _port 4444" --f interface.cfg target.cfg
この debug.cfg は、BUS pirate などのデバッグツール用の CFG ファイルです。
次にある、target.cfg ファイルは、対象プロセッサごとの cfg ファイルになります。
基本的な cfg ファイルは OpenOCD が下記のように提供してくれています。

(図)OpenOCD で有効な各種デバイス一覧表

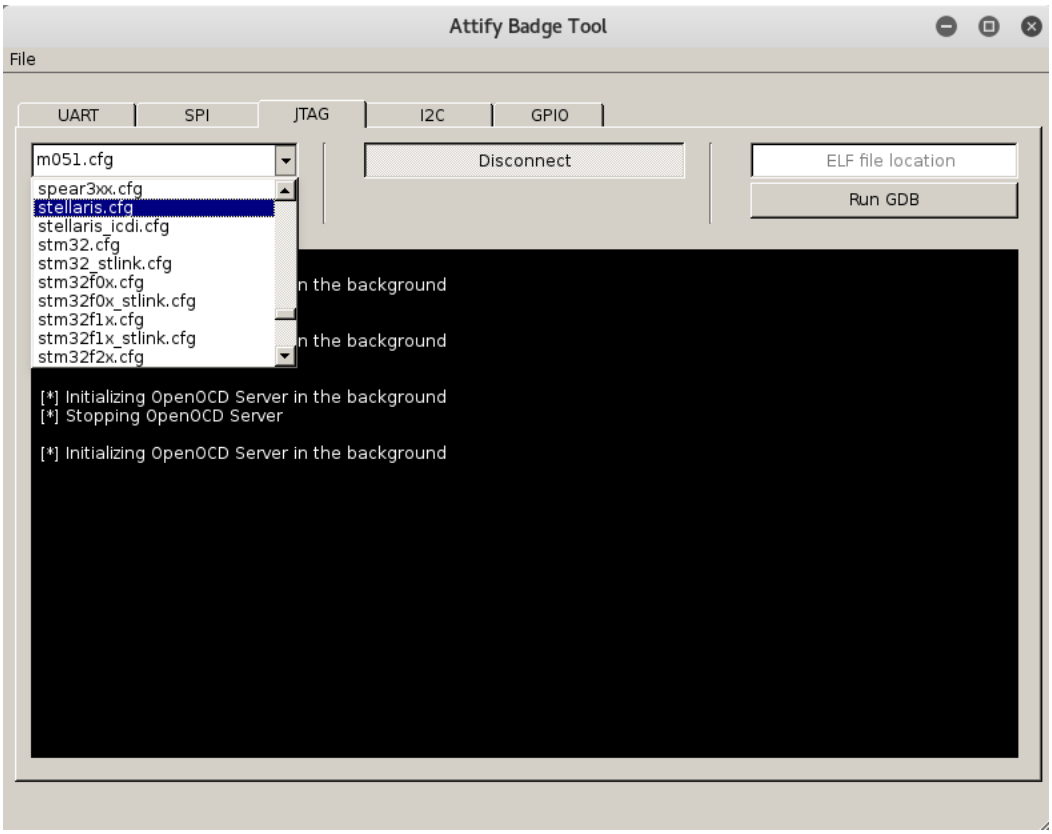
OpenOCD: https://github.com/arduino/OpenOCD
interface: https://github.com/arduino/OpenOCD/tree/master/tcl/interface
target: https://github.com/arduino/OpenOCD/tree/master/tcl/target

SPI フラッシュダンプの時と似ていますね。簡易的に JTAG をハッキングする一種の指標として、対象のプロセ

ッサがこの cfg ファイルの一覧にあるか確認すれば JTAG 攻略する時間の節約になりそうです。
それでは、実際に JTAG にアクセスしていきましょう。
今回は、OpenOCD に target ディレクトリにある、「stm32.cfg」を用いて stm32 マイコンを対象に JTAG ハッキングの練習を行おうと思います。
違うマイクロプロセッサを対象にする際は、都度 cfg ファイルを変更する必要があるので、もし手元の IoT 機器に対して JTAG のハッキングを行いたい場合は OpenOCD で対象機器のプロセッサーの cfg ファイルが提供されているか確認してみてください。

それでは、attify badge tool を起動し、ディレクトリ構成を確認してみましょう。
root@kali:~/libmpsse/src/examples/attify-badge# ls
cfg firmware.bin libftdi1-1.2 main.py src UI
CREDITS install.sh LICENSE README.md troubleshooter
$sudo python main.py
※attify badge tool は、管理者権限で実行しなければ正常に動作しません。

無事に、attify badge tool が起動すると、対象のマイクロプロセッサ(STM32)の cfg ファイルを選択します。

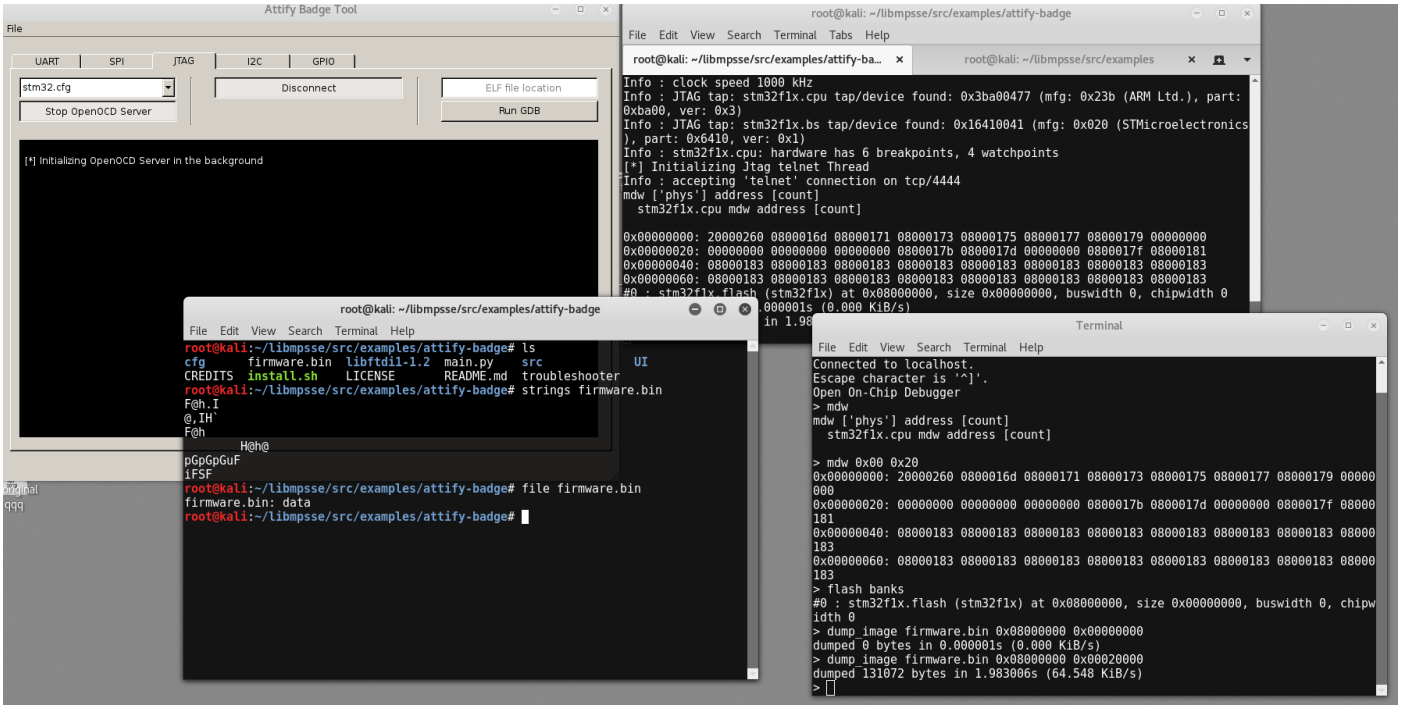



(図)attify badge tool の起動と cfg ファイルの選択

選択後は、[Start OpenOCD Server] -> [Connect to OpenOCD Server]の流れで JTAG への接続を行います。
無事に、接続が出来た場合は、下記のコマンドが無事実行でき、特にエラーは表示されないので以下のコマンドを実行する。

mdw 0x00 0x20(セクションメモリの読み取り)

flash banks (フラッシュメモリについて)
dump_image firmware.bin [at___] [size] (ファームウェアダンプ)




(図)attify badge tool を用いた firmware ダンプ

■JTAG 接続が上手くいっていない場合
telnet で接続後以下のコマンドを入力すると思いますが、どれを実行しても返答がない場合があります。
mdw 0x00 0x20(セクションメモリの読み取り)
Target not examined yet
flash banks (フラッシュメモリについて)
Target not examined yet
dump_image firmware.bin [at___] [size] (ファームウェアダンプ)
Target not examined yet
その際に、下記のような応答が帰ってくると思います。
※なお、成功した場合は、メモリの状態表示やファームウェアのダンプが成功します。

##########################################################################################
libusb_handle_events() failed with LIBUSB_ERROR_NO_DEVICE
unable to purge ftdi rx buffers: LIBUSB_ERROR_NO_DEVICE
error while flushing MPSSE queue: -4
Polling target lh79532.cpu failed, trying to reexamine
libusb_handle_events() failed with LIBUSB_ERROR_NO_DEVICE
unable to purge ftdi rx buffers: LIBUSB_ERROR_NO_DEVICE
error while flushing MPSSE queue: -4
Examination failed, GDB will be halted. Polling again in 6300ms
##########################################################################################

[OpenOCD のコマンド]
■サーバーコマンド
OpenOCD サーバを閉じて、すべてのクライアント（GDB、telnet、other）を切断
$shutdown
現在の Telnet セッションを終了
$exit
すべてのコマンドのヘルプテキストが表示(パラメータ未指定の場合)
$help [string]
再開する前に少なくとも msec ミリ秒待つ命令
$sleep msec [busy]
ユーザーの優先度でメッセージを記録し、メッセージを stdout に出力します
$echo [-n] message
ファイル/スクリプト検索パスにディレクトリを追加します。
$add_script_search_dir [directory]
待ち受ける TCP/IP 接続を待機するアドレスを名前で指定します。
$bindto [name]

■ターゲット状態の処理
番号または名前で 1 つのレジスタにアクセス
$reg [(number|name) [(value|'force')]]
停止
$halt [ms]
ターゲットが停止して（デバッグモードに入る)、パラメータがない場合は最大 ms ミリ秒、または 5 秒待機
$wait_halt [ms]
ターゲットを現在のコード位置で再開します。
$resume [アドレス]
ターゲットを現在のコード位置でシングルステップするか、オプションのアドレスが提供されている場合はそのアドレスをシングルステップ
$step [address
リセット
$reset
ターゲットを実行させる
$reset run
ターゲットを直ちに停止する
$reset halt
ターゲットをただちに停止し、reset-init スクリプトを実行します
$reset init
※http://openocd.org/doc/html/Reset-Configuration.html#Reset-Configuration
ターゲットの停止を要求し、ソフトリセットを実行
$soft_reset_halt

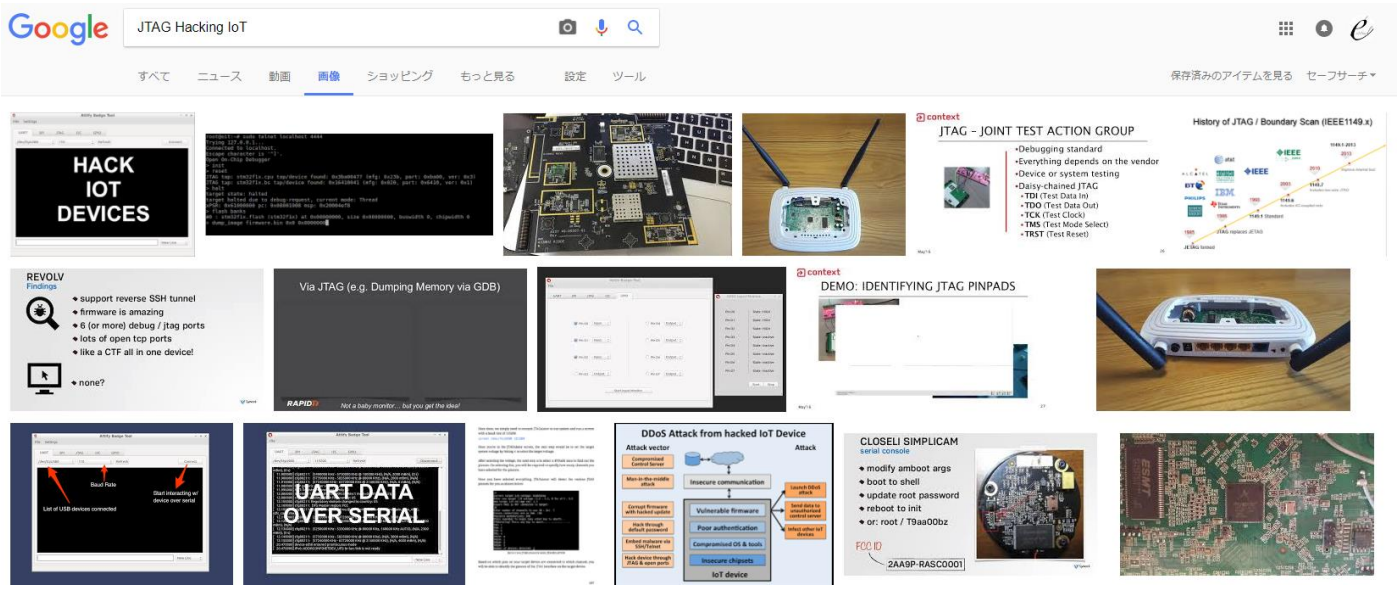
■メモリアクセスコマンド
アドレス内容を 32 ビットワードで表示
$mdw [phys] addr [count]
アドレス内容を 16 ビットワードで表示
$mdh [phys] addr [count]
アドレス内容を 8 ビットワードで表示
$mdb [phys] addr [count]

アドレスに指定された 32 ビットの値を書き込む
$mww [phys] addr word
アドレスに指定された 16 ビットの値を書き込む
$mwh [phys] addr halfword
アドレスに指定された 8 ビットの値を書き込む
$mwb [phys] addr byte

[イメージの読み込みコマンド]
バイナリファイルのアドレスから始まるターゲットメモリのダンプ
$dump_image filename address size

■それら以外のコマンド
http://openocd.org/doc/html/General-Commands.html

■JTAG の cfg ファイルについて
簡単に JTAG のハッキングアプローチについて解説しました。
JTAG は IoT ハッキングで一般的に用いられているアプローチなのでしょうか?
私の考えでは「NO」です。Google で「JTAG Hacking IoT」で検索してみた結果です。



(図)JTAG hacking IoT の検索結果

検索に出たものは、JTAG をハッキングするためのデバッグツールである、「attify badge」や「BUS Pirate」と無線 LAN ルーターだけです。
しかし、前章でご紹介した通り IoT 機器には、JTAG のポートがありますが、多くの場合その JTAG にアクセスするための CFG ファイルが提供されていません。
例えば、以下のサイトは JTAG の exploit について良くまとめられています。
参考 : http://blog.senr.io/blog/jtag-explained
なぜ、こんなにも手軽に JTAG を掌握できるのでしょうか。
答えは、チップセットにあります。上記のブログで使われているチップセットは、ATHEROS の ar9331 と呼ばれる Wi-Fi システムオンチップ（WiSoC）です。
実は、この ar934x 系のチップセットに JTAG でアクセスするための cfg ファイルが公開されています。

17 lines (13 sloc) | 356 Bytes　Raw　Blame　History

```
if { [info exists CHIPNAME] } {
        set _CHIPNAME $_CHIPNAME
} else {
        set _CHIPNAME ar9331
}

if { [info exists CPUTAPID] } {
        set _CPUTAPID $CPUTAPID
} else {
        set _CPUTAPID 0x00000001
}

jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id $_CPUTAPID

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME mips_m4k -endian big -chain-position $_TARGETNAME
```

(図)チップセットから該当する cfg ファイルを探し出した図

この cfg ファイルを使用して、今まで習った方法で JTAG にアクセスすれば、その機器であればハッキング可能です。
しかし、それでは応用が利かなすぎるので cfg ファイルに関して terget ファイルの説明を最後にします。
ターゲットコンフィグファイル(以下 target.cfg)のポイントは、ボードコンフィグファイルが知っておく必要のあるチップに関するすべてをパッケージ化することです。
つまり、ターゲットファイルには、多くの場合に、以下の設定などが記述されていなければなりません。
・デフォルトを設定する
・スキャンチェーンに TAP を追加する
・CPU ターゲットを追加する（GDB サポートを含む）
・CPU /チップ/ CPU コア特有の機能
・オンチップフラッシュ

■ar9331.cfg を解析して、JTAG の設定ファイルを理解する
それでは、先ほど挙げていた ar9331.cfg を解析していきましょう。
cfg ファイルにコメントを交えてみたものを、先にご覧ください。

```
##########################################################################################
#ボードはチップ名をオーバーライドする可能性があります。
#デフォルトはベンダが使用するものと一致する必要があります
#情報が存在する CPUNAME
if { [info exists CHIPNAME] } {
	set _CHIPNAME $_CHIPNAME
} else {
	set _CHIPNAME ar9331
}

#情報が存在する CPUTAPID
if { [info exists CPUTAPID] } {
	set _CPUTAPID $CPUTAPID
} else {
	set _CPUTAPID 0x00000001
}

#各チップの TAP を JTAG スキャンチェーンに追加
jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id $_CPUTAPID

#CPU に TAP を追加したら、GDB や他のコマンドがそれを使用できるように設定する
set _TARGETNAME $_CHIPNAME.cpu
#形式 target create target_name type configparams ...
target create $_TARGETNAME mips_m4k -endian big -chain-position $_TARGETNAME
##########################################################################################
```

それでは、区切りながら見ていきましょう。
最初は、チップ名を設定するところです。今回は ar9331 のチップセットを対象にしています。
if 文で記述されており、CHPNAME に関する情報が OpenOCD にあれば自動的に認識してくれますし、されない場合は ar9331 を CHIPNAME という変数に格納します。
ここで設定した値が後に重要な処理で使用されます。

```
if { [info exists CHIPNAME] } {
	set _CHIPNAME $_CHIPNAME
} else {
	set _CHIPNAME ar9331
}
```

次に CPUTAPID です。
TAP という文字列が含まれています。TAP（テストアクセスポート）は、JTAG の中核にあたり大変重要なものでした。
この設定値も自動で識別してくれると設定不要ですが、識別しない場合は、自分で CPUTAPID を設定する必要があります。
説明しておくと、CPUTAPID は、IDCODE のことで、JTAG チェーン内のパーツタイプを一意に識別する 32 ビットの番号です。今回の ar9331 の IDCODE は「0x00000001」です。

**JTAG** [edit]

- irlen 5
- IDCODE: 0x00000001
- impcode: 0x60414000
  - EJTAG: Version 3.1 Detected
  - EJTAG: features: R4k ASID_8 MIPS16 noDMA MIPS32

引用: https://wikidevi.com/wiki/Atheros_AR9331

それでは残りはすべて続けて説明していきます。
jtag newtap は TAP を JTAG チェーンに追加する処理です。
jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id $_CPUTAPID
このコマンドの構文は、以下の通りです。
「jtag newtap chipname tapname configparams」
chipname は、チップのシンボル名です。
tapname は、その TAP の役割を反映して、この規則に従ってください。

##########################################################################################
bs - これが別の TAP である場合、バウンダリスキャンの場合。
cpu-チップのメイン CPU、代わり arm と dspARM と DSP の CPU の両方を有するチップ上、 arm1 および

arm2 等々二つのアームとを有するチップ上。
etb - 組み込みトレースバッファ（例：ARM ETB11）の場合。
flash - チップに str912 のようなフラッシュ TAP がある場合、
jrc - JTAG ルートコントローラ（例：Beagleboards 上の OMAP3530 のような多くの Texas Instruments チップ上の ICEPick モジュール)。
tap - 単一の TAP を持つ FPGA または CPLD のようなデバイスでのみ使用する必要があります。
unknownN - TAP が何であるか分からないなら（N は数字です）；
##########################################################################################

irlen は、命令レジスタのビット長（4 ビットまたは 5 ビットなど）で、今回はデータシートに基づき 5 を指定しています。
最後に、「configparams」の設定についてです。
-expected-id は、ゼロ以外の数値は、スキャンチェーンの検証時に検出されると予想される 32 ビットの IDCODE を表します。
これらのコードは、すべての JTAG デバイスで必須ではありません。
それら以外にも幾つかの configparams の値があります。

##########################################################################################
-disable
TRST または JTAG ステートマシンの RESET ステートを使用してリセットした後にスキャンチェーンにリンクされていない TAP にフラグを立てます。
-ignore-version
オプションの JTAG バージョンフィールドを無視します。
-ircapture NUMBER
エントリ時に、TAP によって JTAG シフトレジスタにロードされるビットパターン（0x01 など)。
-irmask NUMBER
命令のスキャンが正しく機能するかどうかを確認するために使用されるマスク。

それら以外の設定値については、公式ドキュメント参照
http://openocd.org/doc/html/TAP-Declaration.html#TAP-Declaration
##########################################################################################

「set _TARGETNAME」と「target create $_TARGETNAME」で基本的な cfg ファイルの説明は最後になります。
以下の 2 つはセットで、CPU に TAP を追加したあとに、デバッグなどの目的で GDB などのコマンドを扱えるように設定しなければなりません。
そのための設定になります。

```
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME mips_m4k -endian big -chain-position $_TARGETNAME
```

set _TARGETNAME $_CHIPNAME.cpu は、次の構文のための定義です。
target create $_TARGETNAME mips_m4k -endian big -chain-position $_TARGETNAME
が重要な処理になります。
「target create」の構文は、以下の通りです。
「target create target_name type configparams 」

target_name
デバッグターゲットの名前です。これ は、このターゲットに関連する TAP の dotted.name と同じでなければなりません。これは configparam を使ってここで指定する必要があります

type
ターゲットタイプを指定します
サポートされている、CPU タイプの一部をご紹介します。
##########################################################################################
arm11 - これは ARMv6 コアの世代です
arm720t - これは MMU を持つ ARMv4 コアです
arm7tdmi - これは ARMv4 コアです
arm920t - これは MMU を持つ ARMv4 コアです
arm926ejs - これは MMU を持つ ARMv5 コアです
arm966e - これは ARMv5 コアです
arm9tdmi - これは ARMv4 コアです
avrAtmel の 8 ビット AVR 命令セットを実装しています。(これは暫定的かつ不完全なサポートです。)
cortex_a - これは MMU を持つ ARMv7 コアです
cortex_m - これはコンパクトな Thumb2 命令セットのみをサポートする ARMv7 コアです。
aarch64 - これは MMU を持つ ARMv8-A コアです
dragonite - arm966e に似ています
dsp563xx フリースケールの 24 ビット DSP を実装しています。(これに対するサポートはまだ不完全です。)
fa526 - arm920 と似ている (Thumb なし)
feroceon - arm926 に似ています
mips_m4k - MIPS コア

それ以外は、以下のサイトの target typw を参照
http://openocd.org/doc/html/CPU-Configuration.html
##########################################################################################

今回の cfg ファイルでは、「mips_m4k」として、記述されており、確かに対象内です。
もし、読者の方が jtag の cfg ファイルを書くことになった際に 1 つの目安になると思います。

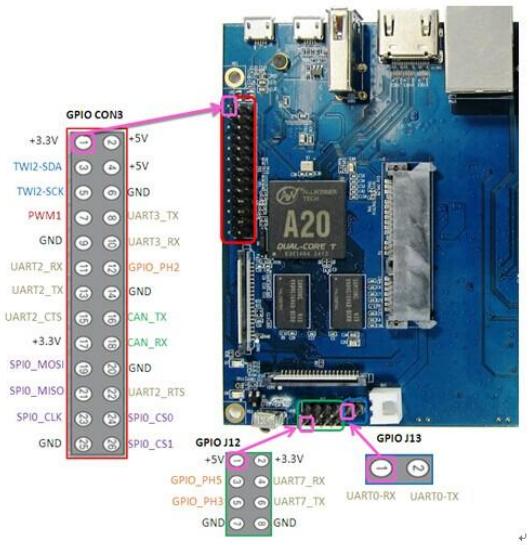
configparams
パラメータとして指定することもできます
例えば、今回のようにターゲットがビッグエンディアンの場合は、ここで「-endian big」を指定します。
configparams の設定の一部を紹介します。

##########################################################################################
-chain-position dotted.name - このターゲットにアクセスするために使用される TAP を指定します。
-endian （大きい|少し） - CPU が大小のエンディアン規則を使用するかどうかを指定します
-work-area-backup （0|1） - 作業領域がバックアップされるかどうかを示します。
-defer-examine - 最初の JTAG チェインスキャンおよびリセット後のターゲット検査をスキップします。

それら以外の設定値については、公式ドキュメント参照
http://openocd.org/doc/html/CPU-Configuration.html#CPU-Configuration
##########################################################################################

ここまでが、ar9331.cfg に関する説明になります。
他の jtag の target.cfg などを見て OpenOCD の公式ページと比較すると大変勉強になると思います。
最後にもう、cfg ファイルの作成に関して 1 つ違うアプローチをご紹介します。

■GPIO ピンを対象にした JTAG 設定ファイルの作成
開発ボードなどで OpenOCD を使いたいことがあります。その際は、GPIO ピンを指定すると良いでしょう。
GPIO ピンとは、集積回路やコンピュータボード上の一般的なピンのことです。その動作は、実行時にユーザによって制御可能であり、マイクロコントローラなどは、外界と接続するための一つまたは複数の GPIO インタフェースを持っています。




引用: https://ja.wikipedia.org/wiki/GPIO

GPIO ピンの番号 trst が 6 であり、 TRST / 7 - TDI / 8 - TDO / 18 - TMS / 19 - TCK / 20 - SRST
##########################################################################################
ar934X.cfg
##########################################################################################
telnet_port 4444

interface sysfsgpio

```
#                                tck tms tdi tdo
sysfsgpio_jtag_nums 19 18 7 8

sysfsgpio_trst_num 6
# sysfsgpio_srst_num 6

adapter_nsrst_delay 100
jtag_ntrst_delay 100

reset_config trst_and_srst

set CHIPNAME ar9341

jtag newtap $CHIPNAME cpu -irlen 5 -ircapture 0x1 -irmask 0x1f -expected-id 1

set TARGETNAME $CHIPNAME.cpu
target create $TARGETNAME mips_m4k -endian big -chain-position $TARGETNAME

$TARGETNAME configure -work-area-phys 0xa0600000 -work-area-size 0x20000
```

telnet_port は、OpenOCD に telnet で接続する際のポート番号を表します。
interface は、JTAG をシミュレートするためのインターフェースで、今回が sysfsgpio を指定しています。
sysfsgpio_jtag_nums は、「TCK/TMS/TDI/TDO」の 4 つのピンを sysfsgpio_jtag_nums 19 18 7 8 として順次に定義し、対応する GPIO を指定しています。
sysfsgpio_trst_num 6 は、TRST に対応する GPIO 番号を指定して、同じく、sysfsgpio_srst_num 6 で SRST に対応する GPIO 番号を指定しています。
sysfsgpio_trst_num と sysfsgpio_srst_num は、両方かどちらかが必要になります。
jtag 接続時に、以下のようなエラーになった場合は、JTAG の配線を最初に疑ってください。
もし、配線が正しければ、起動したターゲットボードでブートローダが JTAG を無効にしている可能性があります。

```
Error: JTAG scan chain interrogation failed: all ones
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: ar934x.m4: IR capture error; saw 0x00 not 0x01
```

以上で、JTAG の CFG ファイルに関する説明を終了します。
JTAG は大変難しい技術で、今回勉強したことを実際の機器に対して行うと色々な困難があると思いますが。挫けずに頑張ってください。