非推奨：非静的メソッドNumbers_Words :: toWords（）は、**206**行目の**/u1/MediaWiki/LocalSettings.php**で静的に呼び出すべきではありません

WL18xx NTPによるクロック同期

From Texas Instruments Wiki
移動 ： ナビゲーション、 検索

目次

WiLink8をネットワークタイムプロトコルの基準クロックとして使用する

このページはhttp://processors.wiki.ti.com/index.php/WL18xx_Clock_Synchronisation_with_NTP_in_SDK3に置き換えられました

NTPの重要な変更は、ビルドが基準クロックの定義を変更してPRECISIONを-1ではなく-13に設定することです。これは、基準クロックが500ミリ秒ではなく122マイクロ秒に正確であることをNTPに通知するため、時間ではなく分単位ではるかに高速に収束します。

ネットワークタイムプロトコル（NTP）は、デバイスのシステム時刻を維持するための標準ツールです。通常、インターネット上の非常に正確なNTPサーバーにクエリを送信して、現在のリアルタイムを提供します。時間の経過とともに、デバイスのNTPデーモンは、NTPサーバーからの定期的な時間更新を使用して、ローカルデバイスのクロックを更新します。デーモンは、2つの方法でデバイス時間を管理します。

- サーバー時間と一致するようにデバイス時間にステップ変更を適用する
- デバイスの時間とサーバーの時間のずれを監視すると、それぞれのクリスタルの特性が少し異なります。多くのサンプルを使用して受信したタイムスタンプの統計を理解し、サーバーからデバイスへのパケットの送信に固有のジッタの影響を最小限に抑えます。速度よりも精度を優先して収束するため、ロックに数時間かかる場合があります。Linuxカーネルは、デバイスクリスタルとサーバークリスタル間の計算された「ドリフト」に基づいてデバイスのシステムクロックを更新するためのNTP用の特定のAPIを提供します。

NTPは、デバイスがインターネット経由でサーバーからではなくローカルの基準クロックから時刻を取得する方法も提供します。これらのリファレンスドライバーは通常、デバイスに接続されているラジオ受信機用で、GPSまたは国時計の放送から時刻情報を取得します。この作業により、WiLink8がアクセスポイント（AP）から受信できるタイミング情報を使用する新しい基準クロックドライバーが作成されます。各Wi-Fiビーコンには、ビーコンのタイムスタンプである時間同期機能（TSF）値が含まれています。これは、APがオンになってからの時間ですが、APに接続されているすべてのステーションに共通のタイムベースを提供するため、基準クロックの基準として使用できます。このシナリオでは、デバイスは正確なリアルタイムを他の世界と共有しません。それらが共有するのは、APのクロックである一般的なクロックリファレンスです。これには、ドリフト/エラーが含まれています。これは、APに接続されているすべてのWiLink8を同期できることを意味します。

ビーコンを使用してタイミングを導出することは、トラフィックがない場合でもデバイスを同期できることを意味します。これは、実際のトラフィックからタイミング制御情報を抽出するPrecision Time Protocol（PTP）/ IEEE1588などの他の手法とは対照的です。

状態

現在のバージョンは0.2です。これにより、マスターとスレーブの概念が追加されます。マスターは、最初にTSFと実際のgettimeofdayの間のオフセットを計算します。このオフセットは、すべてのスレーブで利用できるようになります。1つのマスターと1つのスレーブでテスト。

歴史

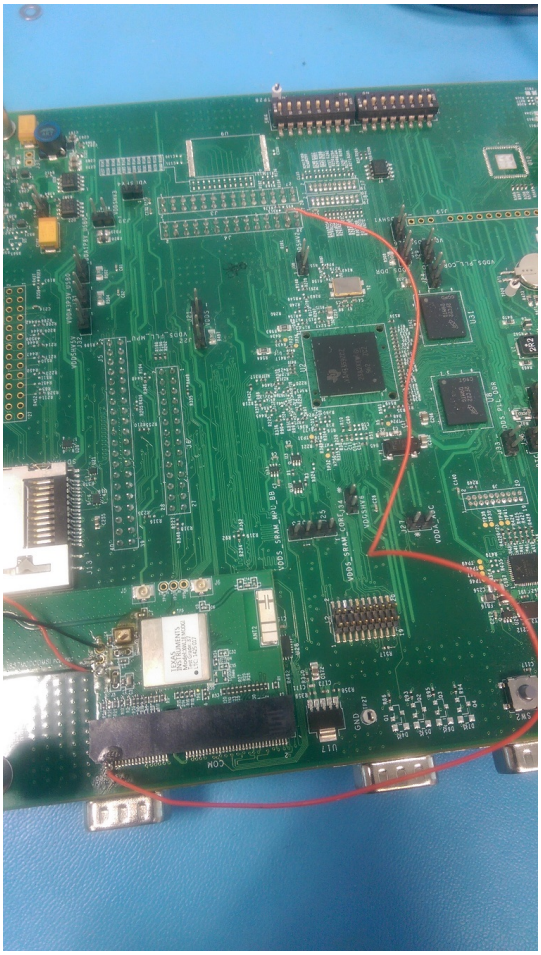
v0.1は、単一のボードをAPに同期する方法を示しています。

デモの構築

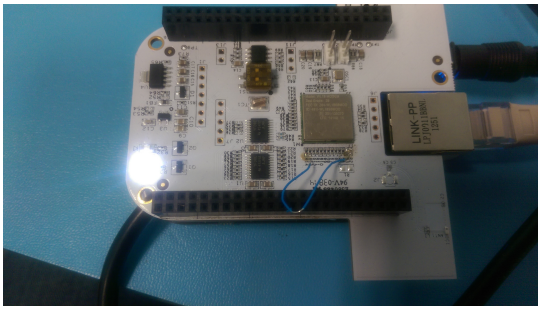
ハードウェアの変更

ビルドは、COM8スロットにWL183xMODを備えたAM335x EVM、またはWL1835ケープを備えたBeagleBoneBlackのSDK8に基づいています。いずれの場合も、WL1835MODのGPIO11をAM335x GPIOに接続するには、ワイヤーの変更が必要です。

- EVMでは、J11.96（COM8コネクタのGPIO11）からJ4.26（GPIO2_2）です。



- WL1835ケープでは、R14（GPIO11）からR2（Reserved2 / GPIO1_15）までです。



ソフトウェアを構築する

同期の操作の基本は、WiLink8ドライバーがGPIO信号がパルスされるLinux時間を認識しており、WiLink8がドライバーにTSFタイムベースでパルスを見た時間を報告することです。ビルドには次の主要な機能があります

- Linuxカーネルには次のパッチがあります。
  - WiLink8ドライバーバージョン8.6デバイスツリー
  - クロックが500PPMから2000PPMに補正する最大エラーの増加。
- WiLink8 R8.6ドライバーにはパッチがあります
  - 新しいdebugfsファイル/ sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_readには、NTP基準クロックドライバーが読み取る必要のあるデータが含まれています
  - この64ビットTSFをサポートするカスタムR8.6ファームウェアも含むAPによって報告されている64ビットTSFのサポート。
  - 受信した最初のパルスに基づいて、デバイスのLinux時間（ほぼ正しい）とTSF（APがオンになってからの時間）の間のデルタを保持するTSFOffsetが計算されます。このオフセットは、TSFを変換する後続のすべてのイベントで使用されますイベントの時刻から参照Linuxホストの時刻。
  - wl18xxドライバーは、APクロックとデバイスクロック間のドリフトの推定値を計算します。これは、ドライバーが起動してから3秒から4秒の間に1秒以上かかるため、fwがTSFカウントを安定させることができます。これは、NTPデーモンが開始されたときに読み取られ、通常はntpで数ミリ秒を超える初期エラーを回避するのに十分です。
  - TSFOffsetを計算してそれを使用可能にするマスターを定義し、スレーブデバイスが接続時にそれを使用できるようにします。
- NTPをダウンロードし、次にパッチを追加します。

- WiLink8をサポートする新しい参照クロックドライバー番号47を追加します。
- 処理する最大ドリフトを500PPMから2000PPMに増やします。
- NTPデーモンは、サーバーのローカルアドレス127.127.47.0を使用して、この新しいドライバー番号47にアクセスします。

- NetCatをダウンロードしてビルドします
  - このGNU Netcatにはリスニングサーバーがあり、マスターで使用されます。ARMデバイスにバグがあるため、スレーブでは使用できません
  - 既存のBusybox Netcatはスレーブに使用されますが、リスニングサーバーが実装されていないため、マスターには使用できません。
  - Netcatは、各スレーブがマスターにAP TSFオフセット値を要求できるようにするために使用されます。

ビルドを行うには、このパッチセットをクリーンなSDK8インストールのベースディレクトリに抽出し、。/ wl8-update-evm.shまたは./wl-update-bbb.shを実行します。EVMの完全なスクリプトを以下に示します。-bbbファイルとの唯一の違いは、時間同期gpioがwl18xxドライバーにハードコードされているため、両方のオプションをサポートするために2つの個別のビルドディレクトリが必要であることを処理することです。

```
＃！/ bin / bash
#Copyright（c）2015 Texas Instruments
＃
#Permissionは、このソフトウェアおよび関連するドキュメントファイル（以下「ソフトウェア」）のコピーを取得するすべての人に、使用、コピー、変更、次の条件に
＃
＃上記の著作権表示とこの許可通知は、ソフトウェアのすべてのコピーまたは実質的な部分に含まれるものとします。
＃
＃本ソフトウェアは「現状のまま」で提供され、商品性、特定の目的への適合性、非侵害性の保証に限定されるものではなく、明示または黙示を問わず、いかなる種類の
＃
＃AM335x EVMおよびSDK8のWL1835 COM8のイメージをビルドするスクリプト

SDK_DIR = `pwd`
FS = "sdk8-r8-6-filesystem-evm"
TAR_DIR = "tar-files-$ {FS}"
TI_SDK_PATH = $ {SDK_DIR}をエクスポートします
エクスポートCROSS_COMPILE = "arm-linux-gnueabihf-"
エクスポートTOOLCHAIN_PATH = "$ {SDK_DIR} / linux-devkit / sysroots / i686-arago-linux / usr / bin"
PATH = "$ PATH：$ {TOOLCHAIN_PATH}"をエクスポートします
INSTALL_PATH = "$ {SDK_DIR} / $ {FS}"をエクスポートします
エクスポートUSR_INCLUDE_PATH = "$ {SDK_DIR} / linux-devkit / sysroots / cortexa8t2hf-vfp-neon-oe-linux-gnueabi / usr"

もし[！-e $ {FS}]
その後
  mkdir $ {FS}
そうしないと
  cd $ {FS} || 出口
  sudo rm -r *
 CD ..
fi
もし[！-e $ {TAR_DIR}]
その後
  mkdir $ {TAR_DIR}
fi

#####ファイルシステムを抽出する
cd $ {FS}
sudo tar -xf ../filesystem/arago-base-tisdk-image-am335x-evm.tar.gz。
CD ..

#####カーネルにR8.6パッチをパッチします
もし[！-f kernel.patched]
その後
  CDボードサポート/
  ＃クリーンなtarファイルが存在しない場合は作成します
  もし[！-e linux-3.14.26-g2489c02.tar.gz]
  その後
   tar -czf linux-3.14.26-g2489c02.tar.gz linux-3.14.26-g2489c02
   ＃現在のディレクトリをバックアップに移動
   cp linux-3.14.26-g2489c02 linux-3.14.26-g2489c02-clean
 そうしないと
  ＃tarファイルが存在するため、作業中のビルドを削除し、クリーンなtarを再入力します
  echo「tarカーネルを抽出」
  sudo rm -r linux-3.14.26-g2489c02
  tar -xf linux-3.14.26-g2489c02.tar.gz
 fi
 CD ..

 ＃念のためにきれいにする
 linux_cleanを作成
 CDボードサポート/linux-3.14.26-g2489c02

####### R8.6のWifiパッチを適用する########
以下からのパッチでそれをパッチ#now http://processors.wiki.ti.com/index.php/WiLink8_Release_Notes/R8.6#List_of_patches_applied_on_top_C2_AOSitara_SDK8_k
カール--noproxy git.ti.com -k ます。http：// gitの.ti.com / wilink8-wlan / wilink8-wlan-ti-linux-kernel / commit / 9d2edaed7bbe4023dc9c58f879507d2c163b
パッチ-p1 <patch1.patch

curl --noproxy git.ti.com -k http://git.ti.com/wilink8-wlan/wilink8-wlan-ti-linux-kernel/commit/eef5c6fc3a37552c01b8eecece072085c70c3b8e?format=patch
パッチ-p1 <patch2.patch

curl --noproxy git.ti.com -k http://git.ti.com/wilink8-wlan/wilink8-wlan-ti-linux-kernel/commit/715fc23917f42891ce798a8beefc245fb440c6d2?format=patch
＃patch3にパッチを適用して適用します
パッチ-p1 <../../patches/kernel/patch3-for-sdk8-to-actually-apply.patch
パッチ-p1 <patch3.patch

curl --noproxy git.ti.com -k http://git.ti.com/wilink8-wlan/wilink8-wlan-ti-linux-kernel/commit/43c1c81c54df611896330a5fc43860447cdd3e77?format=patch >
パッチ-p1 <patch4.patch

＃標準の設定ファイルにパッチを適用
パッチ-p1 <../../patches/kernel/0001-update-sdk8-config-for-r8-6.patch

＃パッチを追加してカーネルを変更し、最大ドリフトを緩和して補償する
＃500ppmから2000ppm
パッチ-p1 <../../patches/kernel/1001-change-kernel-to-handle-2000ppm-drift.patch

＃GPIO1_5のBBBデバイスツリーにパッチを適用して、Time-Sync GPIOとして接続する
パッチ-p1 <../../patches/kernel/0001-add-gpio1-5-to-bone-dtsi-for-time-sync.patch || 出口

cd ../..
＃このファイルを作成して、カーネルにパッチが適用されたことを伝えます。最初から再作成するには、このファイルを削除してください
kernel.patchedに触れます
fi

#####カーネルを作る
Linuxを作成MAKE_JOBS = 8 || 出口

＃メインのlinux_installスクリプトは、Rules.makを使用してファイルシステムを設定します。ここではそのディレクトリを使用していません。
＃$ {FS}を渡すことができるように、インストールルーチンを明示的に呼び出します
#sudo make linux_install
sudo install -d $ {FS} / boot
sudo install board-support / linux-3.14.26-g2489c02 / arch / arm / boot / zImage $ {FS} / boot
```

```
#vmlinux（109Mb）をインストールせず、元のzImageを削除して、ファイルシステムのサイズを小さくする
sudo rm $ {FS} /boot/zImage-3.14.26-g2489c02
#この行を削除して、より小さなファイルシステムを作成します（vmlinuxは109Mb）
#sudo install board-support / linux-3.12.10-ti2013.12.01 / vmlinux $ {FS} / boot

sudo install board-support / linux-3.14.26-g2489c02 / System.map $ {FS} / boot
sudo cp -f board-support / linux-3.14.26-g2489c02 / arch / arm / boot / dts / *。dtb $ {FS} / boot /
sudo make -C board-support / linux-3.14.26-g2489c02 ARCH = arm CROSS_COMPILE = arm-linux-gnueabihf- INSTALL_MOD_PATH = $ {SDK_DIR} / $ {FS} modules_ins

##### R8.6ドライバーをダウンロードし、パッチを適用してntpバージョンの時刻同期サポートを追加します
もし[！-e wifiダウンロード]
その後
  mkdir -p wifi-wl8
  cd wifi-wl8
  git clone git：//git.ti.com/wilink8-wlan/build-utilites.git || 出口
  cd build-utilites || 出口
  gitチェックアウトマスター
  cp ../../patches/wifi-build-patches/setup-env-evm setup-env || 出口

  #ここでは、ディレクトリを作成できないことを示すエラーはここで問題ありません。ファイルシステムはルートによって所有されています
  #したがって、ユーザーは一部のディレクトリを作成できません。実際のビルドはsudoで実行されるため、これは問題ではありません
  #rootがすべてのソースファイルの所有者になるため、ステップをrootとして実行しません。
  ./build_wl18xx.shアップデートR8.6

  #wl8ドライバーソースにパッチを適用して、64ビットTSF同期とgpio時間測定のサポートを追加し、データを書き込む
  #読み取るntp基準クロックの実装のdebugfs
  cd src /ドライバー
  git am ../../../../patches/time_sync/0001-initial-commit-to-calculate-difference-in-time-measu.patch || 出口
  git am ../../../../patches/time_sync/0002-export-gpio-and-tsf-timers-to-debugfs-so-ntp-can-pic.patch || 出口
  git am ../../../../patches/time_sync/0003-temp-hack-to-align-reported-TSF-to-system-clock.patch || 出口
  git am ../../../../patches/time_sync/0004-first-working-version-with-ntp-refclock_wl8-driver.patch || 出口
  git am ../../../../patches/time_sync/0005-add-64-bit-TSF-support-from-Yaniv.patch || 出口
  git am ../../../../patches/time_sync/0006-now-adjust-gpio-time-with-offset-so-it-has-a-real-ti.patch || 出口
  git am ../../../../patches/time_sync/0007-move-all-debugfs-data-into-a-single-file-to-ensure-t.patch || 出口
  git am ../../../../patches/time_sync/0008-add-support-in-debugfs-for-TSFOffset-and-initial-dri.patch || 出口
  git am ../../../../patches/time_sync/0009-ensure-estimatedDriftPPM-is-zero-ed-on-startup.patch || 出口
 ## git am ../../../../patches/time_sync/0010-change-time-sync-gpio-for-BBB-and-cape.patch || 出口
  git am ../../../../patches/time_sync/0011-add-master-slave-concept-so-that-master-calculates-T.patch || 出口
  git am ../../../../patches/time_sync/0012-fix-type-cast-error-in-debugfs.patch || 出口
  git am ../../../../patches/time_sync/0013-change-drift-estimate-window-to-be-between-3-and-4-s.patch || 出口
  cd ../ ..

  ../../wifi-downloadedをタッチします
  cd ../ ..
fi

##### NTPをダウンロードしてパッチを追加し、WL8のリファレンスクロックドライバー47を追加します。
ROOTFS = $ {SDK_DIR} / $ {FS}をエクスポート

もし[！-e ntp-downloaded]
 その後
    mkdir -p ntp
    #ntpをダウンロード
   cd ntp
    rm ntp-4.2.8p3.tar.gz
    wget http://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.2/ntp-4.2.8p3.tar.gz

    tar -xzf ntp-4.2.8p3.tar.gz
    cd ntp-4.2.8p3
    パッチ-p1 <../../patches/ntp/0001-add-wl8-reference-clock.patch || 出口
    パッチ-p1 <../../patches/ntp/0002-update-for-drift-and-offset-now-in-debugfs.patch || 出口
    パッチ-p1 <../../patches/ntp/0003-remove-tsfoffset-from-debugfs-structure.patch || 出口
   cd ../ ..
    ntp-downloadedをタッチします
fi

#今それを設定
cd ntp / ntp-4.2.8p3 || 出口
./configure --host = arm-linux-gnueabihf --with-yielding-select = yes CC = $ {TOOLCHAIN_PATH} / $ {CROSS_COMPILE} gcc AR = $ {TOOLCHAIN_PATH} / $ {CROS

#そしてそれを構築する
きれいにする
作る
sudo make install DESTDIR = $ {ROOTFS}
cd ../ ..

######サーバーの待機機能へのアクセスを許可するnetcatをダウンロード
もし[！-e nc-downloaded]
その後
    mkdir -p nc
   #ncをダウンロード
    CD nc
    rm netcat-0.7.1.tar.gz
    wget http://sourceforge.net/projects/netcat/files/netcat/0.7.1/netcat-0.7.1.tar.gz

   tar -xzf netcat-0.7.1.tar.gz

   CD ..
   nc-downloadedをタッチします
fi

#今それを設定
cd nc / netcat-0.7.1 || 出口
./configure --host = arm-linux-gnueabihf CC = $ {TOOLCHAIN_PATH} / $ {CROSS_COMPILE} gcc AR = $ {TOOLCHAIN_PATH} / $ {CROSS_COMPILE} ar LD = $ {TOOLCHA

#そしてそれを構築する
きれいにする
作る
sudo make install DESTDIR = $ {ROOTFS}
cd ../ ..

##### wifiドライバーをビルドしてインストールする
cd wifi-wl8 / build-utilites
#クリーンアップしてビルドする
。./sudo_build_wl18xx.sh clean
#。./sudo_build_wl18xx.shモジュール
cd ../ ..

##### NTPをサポートするすべてのランタイムファイルを作成します
#WL8ドライバーを使用するためのconfファイルを作成します。
#WL8ドライバーから派生したntp.driftを初期推定値として使用するように指示
#リファレンスクロックドライバー47（つまりwl8）を使用するように指示する
   sudo sh -c "エコー"
 ドリフトファイル/etc/ntp.drift
サーバー127.127.47.0 '>> $ {FS} /etc/ntp.conf "

#wlconf-toggle-set.shスクリプトを更新して、時刻同期ビットを正しく設定する
#バイナリを事前に入力しているため、これは使用されませんが、変更できます
sudo cp wifi-wl8 / build-utilites / src / scripts_download / testing / wlconf-toggle-set.sh $ {FS} / usr / sbin / wlconf /
```

```
#ステーションとして同期を有効にしてwl1837 confを事前に入力する
sudo cp patch / time_sync / wl18xx-conf-1837-sync1.bin $ {FS} /lib/firmware/ti-connectivity/wl18xx-conf.bin

#64ビットTSFで更新されたfwを使用
sudo cp patch / time_sync / wl18xx-fw-4.bin.64bitTSF $ {FS} /lib/firmware/ti-connectivity/wl18xx-fw-4.bin

#timesyncを有効にするスクリプトを作成
sudo sh -c "echo '#!/ bin / sh
#tsf_masterに引数を渡します。0はスレーブ、1はマスター
echo ¥ $ 1> / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_master
エコー0> / sys / kernel / debug / ieee80211 / phy0 / wlcore / sleep_auth
エコー100> / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / time_sync

#有効な初期ドリフトが推定されるまでループする
drift = ¥ `head -1 / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_read ¥`

while [¥ $ drift -eq 0]
行う
   睡眠1

   drift = ¥ `head -1 / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_read ¥`
終わった

#これで、ゼロ以外の値が/etc/ntp.driftに書き込まれました
echo ¥ $ drift> /etc/ntp.drift
'>> $ {FS} /home/root/enable-sync.sh "
  sudo chmod a + x $ {FS} /home/root/enable-sync.sh

#ntpを起動してMASTERで同期を有効にするスクリプトを作成します。つまり、同期を開始する最初のデバイス
#これはユーザーが呼び出すスクリプトです
sudo sh -c "echo '#!/ bin / sh

#イーサネット経由でシステム時刻を更新
ntpdate uk.pool.ntp.org

#wifiインターフェースを立ち上げる
ifconfig wlan0 up
#APに接続
wpa_supplicant -Dnl80211 -i wlan0 -c / etc / wpa-supplicant-ap.conf -B

#接続が確立されるまで待機します。関連付けには10秒で十分です
睡眠10

#IPアドレスを取得
udhcpc -i wlan0

#時刻同期機能をマスターとして起動
./enable-sync.sh 1

#tsf_offsetがすべてのスレーブデバイスで利用できるようになりました
./serve-up-offset.sh&

#そして同期を開始します。-dは基本的なデバッグログ情報を提供します。ntp収束を追跡するのに十分です。
ntpd -d&
'>> $ {FS} /home/root/start-ntp-master.sh "
sudo chmod a + x $ {FS} /home/root/start-ntp-master.sh

#ntpを起動してスレーブで同期を有効にするスクリプトを作成
#これは、useが呼び出すスクリプトです
sudo sh -c "echo '#!/ bin / sh

#イーサネット経由でシステム時刻を更新
ntpdate uk.pool.ntp.org

#wifiインターフェースを立ち上げる
ifconfig wlan0 up
#APに接続
wpa_supplicant -Dnl80211 -i wlan0 -c / etc / wpa-supplicant-ap.conf -B

#接続が確立されるまで待機します。関連付けには10秒で十分です
睡眠10

#IPアドレスを取得
udhcpc -i wlan0

#続行する前に、有効なオフセット（つまり、ゼロ以外）があることを確認します
offsetNs = ¥ `cat / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_offset ¥`

while [¥ $ offsetNs -eq 0]
行う
  #これで、マスターからのIPアドレス要求オフセットが取得されました
  #私たちが作成したbusybox ncを強制的に使用すると、ARMにバグがあるようです!!!!
  / usr / bin / nc 192.168.0.100 49152> / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_offset

  offsetNs = ¥ `cat / sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_offset ¥`
  echo ¥ "offsetNs = ¥ $ offsetNs ¥"
  寝る5
終わった

#時刻同期機能をスレーブとして起動
./enable-sync.sh 0

#そして同期を開始します。-dは基本的なデバッグログ情報を提供します。ntp収束を追跡するのに十分です。
ntpd -d&
'>> $ {FS} /home/root/start-ntp-slave.sh "
sudo chmod a + x $ {FS} /home/root/start-ntp-slave.sh

#他のデバイスがオフセットを照会できるように、マスターでnetcatサーバーを起動するスクリプトを作成します
sudo sh -c "echo '#!/ bin / sh

ながら[1]
行う
  #netcatサーバーがポート49152で接続を受信したときの応答
  #debugfsからのtsf_offset。読み取り後、接続を閉じます（-c）。
  #busyboxはリスニングモードをサポートしていないため、ncビルドされたバージョンの使用を強制します
  猫/ sys / kernel / debug / ieee80211 / phy0 / wlcore / wl18xx / tsf_offset | / usr / local / bin / nc -l -c -p 49152
  echo ¥ "1つのクライアントがtsf_offsetを読み取り、リッスンモードに戻りました¥"
終わった
'>> $ {FS} /home/root/serve-up-offset.sh "
sudo chmod a + x $ {FS} /home/root/serve-up-offset.sh

#接続するAPのサプリカントファイルを作成する
sudo sh -c "echo 'ctrl_interface = / var / run / wpa_supplicant

network = {
      key_mgmt = NONE
      ssid = ¥ "IAIN841 ¥"
}
```

```
network = {
        psk = ¥ "simplelink ¥"
        ssid = ¥ "CC-2.4GHz ¥"
}
' >> $ {FS} /etc/wpa-supplicant-ap.conf "

#####次に、SDK /bin/create-sdcard.shを実行して読み込まれるrootfsとブートパーティションtarballを作成します。
cd $ {FS}
sudo tar -czf ../${TAR_DIR}/${FS}_rootfs_partition.tar.gz *

＃ブートに変更し、「cd」コマンドが正しく実行された場合のみファイルを削除
＃ディレクトリから既存のmlo、u-bootを削除します。シンボリックリンクをFATにコピーできません
＃ホスト/ bootではなく、ターゲットファイルシステムのブートであることを確認します!!!
cd ../${FS}/boot && sudo rm -r *

＃FATに書き込んでいるため、最終的にシンボリックリンクは機能しないため、ターゲット名に書き込んでそれらを削除します
#MLOとu-bootのデフォルトは変更していません。
sudo cp ../../board-support/prebuilt-images/MLO-am335x-evm MLO
sudo cp ../../board-support/prebuilt-images/u-boot-am335x-evm.img u-boot.img

sudo tar -czf ../../${TAR_DIR}/${FS}_boot_partition.tar.gz *
cd ../ ..
```

これにより、標準のSDKユーティリティbin / create-sd.shを使用してSDカードに書き込むことができるtar-file-sdk8-r8-6-filesystemにブートパーティションとrootfsパーティションが作成されます

デモを実行する

マスターステーション

ログインし、最初に起動したステーションでスクリプトstart-ntp-master.shを実行して、マスターを実行します。これは、次の一連のアクションを実行します

- イーサネットインターフェイスでntpdateを実行して、コマンド「date」またはAPI gettimeofday（）で指定されたLinux時間が正確であることを確認します
- Wi-Fiを起動し、wpa_supplicantで目的のアクセスポイントに接続します
- IPアドレスを取得する
- システムクロックとAPクロック間のPPMの初期ドリフトを推定するWL18xxで時刻同期機能を開始します。このドリフト推定値を/etc/ntp.driftに書き込んで、ntpdの起動時にこの値を初期推定値として使用できるようにします。このスクリプトは、/ etc / ntp.driftがゼロ以外になるまでループします
- ポート49152でnetcat待機サーバーを起動し、そのポートで要求するすべてのデバイスにTSFオフセットを返します。各読み取り後、netcatは閉じて再起動し、要求する他のスレーブにサービスを提供します。
- レベル1デバッグを有効にしてNTPデーモン（ntpd）を開始（-d）

スレーブ局

スレーブを実行するには、ログインして、開始された2番目以降のステーションでスクリプトstart-ntp-slave.shを実行します。これは、次の一連のアクションを実行します

- イーサネットインターフェイスでntpdateを実行して、コマンド「date」またはAPI gettimeofday（）で指定されたLinux時間が正確であることを確認します
- Wi-Fiを起動し、wpa_supplicantで目的のアクセスポイントに接続します
- IPアドレスを取得する
- TSFオフセットポート49152でサーバー（ここでは192.168.0.100にハードコード化）からのnetcat要求を使用します。非血清値が受信されるまでループします。debugfsに値を書き込みます。
- システムクロックとAPクロック間のPPMの初期ドリフトを推定するWL18xxで時刻同期機能を開始します。このドリフト推定値を/etc/ntp.driftに書き込んで、ntpdの起動時にこの値を初期推定値として使用できるようにします。
- レベル1デバッグを有効にしてNTPデーモン（ntpd）を開始（-d）

デバッグについて

NTP（レベル1デバッグ）によって生成されるログの主要な行は次のとおりです。

- 1秒ごとに収集されますが、ログ出力を最小限に抑えるために8秒ごとにログに記録されるタイムスタンプ

wl8_query rcv 1441634552.3935886**クロック**1441634552.5166418

最初の値は「参照」時間を示すTSFから導出された時間で、2番目の値はそのインスタンスの「Linux」時間です。この場合、Linux時間は参照より123ミリ秒進んでいます。ntpクロック調整の目的は、この差を最小限に抑え、そのデバイスの「linux」時間を「参照」クロックに確実に同期させることです。

- 「ドリフト」NTPの調整はローカルクロックに適用されます

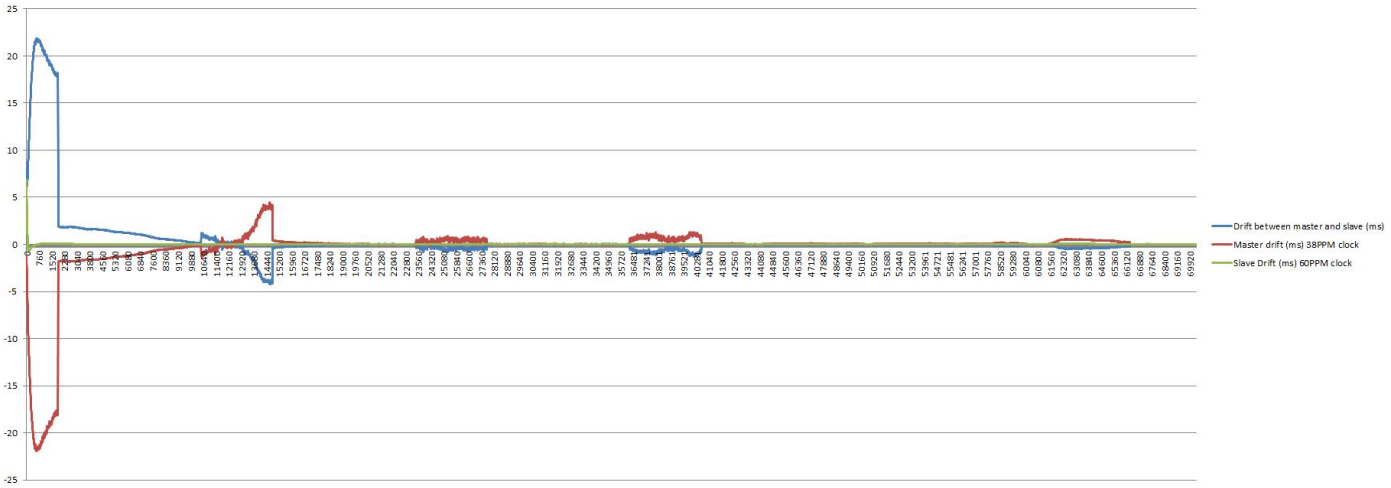
local_clock：**オフセット**0.002095232 jit 0.000314658 freq 1172.813 stab 0.094 poll 6

freq値は、「linux」クロックに適用される現在のドリフトであり、デバイスクロックをAM335xに駆動する実際の水晶とAPとの間の速度の違いを補正します。この場合、1171.813PPMのドリフトがあります。この値は毎分更新されます。

WL18xx NTPソリューションのパフォーマンス

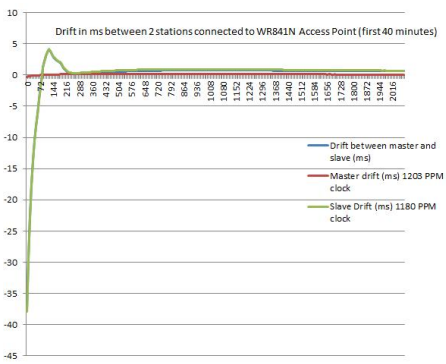
APに接続された2つのステーションの「参照」時間と「Linux」時間の差を記録するテストが1日以上実行されました。1つのステーションはBBBで実行されていてマスターであり、もう1つのステーションはEVMで実行されていてスレーブです。さらに、各ステーション間のLinux時間の差が計算されます。これは、ステーション間で重要な同期の実際の測定であるためです。これはすべて、各ステーションのntpログから計算されます。このテストは、TP-Linkの2つのAP（WR841NとN900）に対して行われました。結果を以下に示します

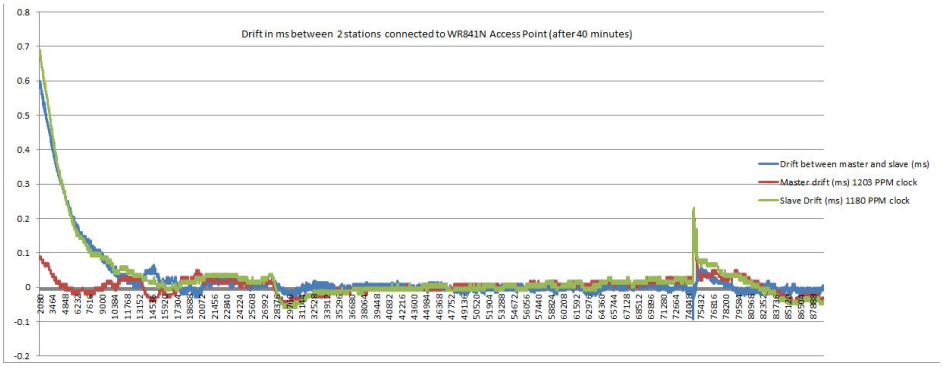
N900



このAPには、ステーションのクロックに近い周波数のクロックがありました。それとBBBの間には約-38PPMのドリフトがあり、それとEVMの間には-60PPMのドリフトがありました。つまり、1秒ごとに38または60usのドリフトがあり、NTPによって修正する必要がありました。この場合、EVMは+/- 20usにゆっくりと収束する前に、AP時間に数分かけてすばやく収束しました。ただし、BBBクロックの変動はさらに大きく、数msのエラーが発生する短期間でした。

WR841N





このAPには、どちらのステーションからも離れたクロックがありました。それとBBBの間には約1203 PPM、それとEVMの間には1180 PPMのドリフトがありました。プロットは、異なるスケールでの同期の2つのフェーズを示しています。クロックが収束する最初の40分間は、40ミリ秒の初期オフセットがあり、3分で2ミリ秒未満に修正されました。40分後、クロックは、1スパイクから200マイクロ秒を除いて50マイクロ秒の範囲で密接に同期されました。

結論

WiLink8時間同期方法をソースとして使用してNTP基準クロックの実装を駆動すると、ユーザーはアクセスポイントに接続されたワイヤレスステーションを数ミリ秒の精度で同期できます。この方法は、クロック制御情報を生成するためにデータトラフィックに依存しないため、精度に影響を与えることなく、トラフィックが少ないまたは散発的なシステムで使用できます。NTPなどのよく知られたユーティリティを使用すると、システムへの使用と統合が容易になります。遅い収束はNTPの設計機能であり、調整の速度よりも長期的な精度を優先します。したがって、この方法は、継続的に電源がオンになっているシステムに適しています。

カテゴリー：

- 壊れたファイルリンクのあるページ
- WiLinkコンボソリューション

ナビゲーションメニュー

Personal tools

- ログインする
- アカウントをリクエスト

Namespaces

- ページ
- 討論

Variants

Views

- 読んだ
- ソースを表示
- 履歴を見る

More

探す

Texas Instruments Wikiを検索 Go

Navigation

- メインページ
- すべてのページ
- すべてのカテゴリ
- 最近の変化
- ランダムページ
- 助けて

ツールボックス

- ここのリンク
- 関連する変更
- 特別ページ
- 印刷可能バージョン
- パーマリンク
- ページ情報