AM3517 EVM WiLink™ 6.0 Getting Started Guide

Software Documentation

Logic PD // Products
Published: August 2010

Abstract

This document describes the procedures for building Texas Instruments PSP-03.00.00.05 with additional patches and software to enable the WL127x hardware on the Zoom™ AM3517 EVM Development Kit.

Revision History

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	JY, JCA	Initial release	JY	08/16/10

Table of Contents

1 Introduction

This document describes building the Linux kernel and additional supporting applications/libraries required for Bluetooth and WLAN hardware support on the Zoom AM3517 EVM Development Kit.

In order to build applications and the Linux kernel on an x86-based Linux computer for deployment on an ARM processor, a cross-compiler must be used. There are multiple methods of setting up the development environment that will remove much of the complexity involved in cross-compiling applications. This document describes the process so you may gain an understanding of the steps involved.

2 Environment Setup

This document describes building the Linux kernel with Bluetooth and WLAN hardware support using a new installation of Ubuntu 10.04 as the development environment. Adjustments to the following instructions may be necessary if you wish to use a different Linux development environment.

The base desktop installation of Ubuntu 10.04 does not include all of the development utilities needed. Install the following additional development libraries and tools:

```
$ sudo apt-get install build-essential autoconf automake \
  bison help2man texi2html m4 texinfo libtool quilt \
  libdbus-1-dev libdbus-glib-1-dev libexpat1-dev
```

Some external build scripts may rely on bash specific features that may not be present in Ubuntu's default shell (dash). To change the default shell to bash:

```
$ sudo dpkg-reconfigure dash
```

(select <**No**> then logout and login again)

Setup environment variables that will be used throughout this document:

```
$ export PSP_BUILD_DIR=${HOME}/am3517_wilink
$ export DOWNLOAD_DIR=${HOME}/downloads
$ export STAGING_DIR=${PSP_BUILD_DIR}/rfs/stage
```

3 CodeSourcery ARM 2009-q1

The CodeSourcery Lite toolchain is used for cross-compiling, which allows code targeted for the ARM processor to be compiled on an x86-based host PC.

Download and install the cross-compiler from CodeSourcery:

```
$ mkdir -p ~/downloads/com/codesourcery
$ cd ~/downloads/com/codesourcery
$ wget
http://www.codesourcery.com/sgpp/lite/arm/portal/package4571/public/arm-
none-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-
gnu.tar.bz2
```

For example, to install the toolchain in the */usr/local/ directory*:

```
$ cd /usr/local
$ sudo tar xjvf ~/downloads/com/codesourcery/arm-2009q1-203-arm-none-linux-
gnueabi-i686-pc-linux-gnu.tar.bz2
```


Set the environment variable *PATH* to contain the binaries of the CodeSourcery cross-compiler toolchain. For example, in bash:

```
$ export PATH=/usr/local/arm-2009q1/bin:$PATH
```

4 Texas Instruments PSP

Download the AM35x-OMAP35x-PSP-03.00.00.05 from Texas Instruments' (TI) website. An index of the currently available PSP versions can be found here:

http://software-dl.ti.com/dsps/dsps_public_sw/psp/LinuxPSP/OMAP_03_00/index.html

```
$ mkdir -p ${DOWNLOAD_DIR}/com/ti
$ cd ${DOWNLOAD_DIR}/com/ti
$ wget http://software-
dl.ti.com/dsps/dsps_public_sw/psp/LinuxPSP/OMAP_03_00/03_00_00_05//omap_0300
0005/AM35x-OMAP35x-PSP-SDK-03.00.00.05.tgz
```

Extract the contents of the release package with the following command:

```
$ tar zxf AM35x-OMAP35x-PSP-SDK-03.00.00.05.tgz
```

This creates a directory *OMAP35x-PSP-SDK-MM.mm.pp.bb* with the following contents:

```
\---AM35x-OMAP35x-PSP-SDK-MM.mm.pp.bb
| Software-manifest.html
| Arago-FS-Software-manifest.html
+----docs
| |----Building-RootFs-Arago.html
| |----DataSheet-MM.mm.pp.bb.pdf
| |----ReleaseNotes-MM.mm.pp.bb.pdf
| |----am3517
| | `----UserGuide-MM.mm.pp.bb.pdf
| |----omap3530
| | `----UserGuide-MM.mm.pp.bb.pdf
+----host-tools
| |----linux
| | `----signGP
| |----src
| | `----signGP.c
+----images
| |----boot-strap
| | |----am3517
| | | `----x-load.bin.ift
| | |----omap3530
| | | `----x-load.bin.ift
| |----fs
| | |----nfs-base.tar.gz
| | |----ramdisk-base.gz
| | |----rootfs-base.jffs2
| | |----am3517
| | | |----nfs.tar.gz
| | | |----ramdisk.gz
| | | `----rootfs.jffs2
| | |----omap3530
| | | |----nfs.tar.gz
| | | |----ramdisk.gz
| | | `----rootfs.jffs2
```



```
| |----kernel
| | |----am3517
| | | `----uImage
| | |----omap3530
| | | `----uImage
| |----u-boot
| | |----am3517
| | | `----u-boot.bin
| | |----omap3530
| | | `----u-boot.bin
+----scripts
| |----am3517
| | |----Readme.txt
| | |----initenv-micron.txt
| | `----reflash-micron.txt
| |----omap3530
| | |----Readme.txt
| | |----initenv-micron.txt
| | `----reflash-micron.txt
+----src
| |----boot-strap
| | |----ChangeLog-MM.mm.pp.bb
| | |----ShortLog
| | |----Unified-patch-MM.mm.pp.bb.gz
| | |----diffstat-MM.mm.pp.bb
| | |----x-loader-patches-MM.mm.pp.bb.tar.gz
| | `----x-loader-MM.mm.pp.bb.tar.gz
| |----examples
| | |----examples.tar.gz
| |----kernel
| | |----Readme.txt
| | |----ChangeLog-MM.mm.pp.bb
| | |----ShortLog
| | |----Unified-patch-MM.mm.pp.bb.gz
| | |----diffstat-MM.mm.pp.bb
| | |----kernel-patches-MM.mm.pp.bb.tar.gz
| | `----linux-MM.mm.pp.bb.tar.gz
| |----u-boot
| |----Readme.txt
| |----ChangeLog-MM.mm.pp.bb
| |----ShortLog
| |----Unified-patch-MM.mm.pp.bb.gz
| |----diffstat-MM.mm.pp.bb
| |----u-boot-patches-MM.mm.pp.bb.tar.gz
| `----u-boot-MM.mm.pp.bb.tar.gz
+----test-suite
`----lftb-MM.mm.pp.bb.tar.gz
```

IMPORTANT NOTE: The values of MM, mm, pp and bb in this illustration will vary across the releases and actually depends on individual component versions.

5 Root File System

A root file system directory structure is created in a staging directory to hold the output from the build process. As applications and libraries are built, they are deployed to this staging directory.

Applications that have dependencies on these libraries reference this staging directory during the build process.

When all the components have been built, the root file system can be deployed to an SD card, NAND flash, or to a directory on the development PC that is accessed from the target device using the Network File System (NFS) protocol. The PSP includes a base file system that will be used as a starting point.

```
$ cd ${PSP_BUILD_DIR}/build
$ mkdir -p ${PSP_BUILD_DIR}/rfs/stage/home/root/wlan
$ mkdir -p ${PSP_BUILD_DIR}/rfs/boot
$ mkdir -p ${PSP_BUILD_DIR}/rfs/rootfs
$ mkdir -p ${PSP_BUILD_DIR}/rfs/deploy
$ mkdir -p ${STAGING_DIR}/home/root/bt
```

This step is required when the root files ystem is mounted from an NFS location. Extract the contents of the NFS image (*nfs.tar.gz*) to a directory exported via NFS.

```
$ cd ${PSP_BUILD_DIR}/rfs/rootfs
$ sudo tar zxf ${PSP_BUILD_DIR}/build/nfs.tar.gz
```

IMPORTANT NOTE: Execute this command as root user. Some of the files included in this archive require root permissions for creation.

6 AM3517evm_WiLink Patches

This package contains patches to the U-Boot, the Linux kernel, BlueZ, and several supporting applications. It also contains a build script that can be run to execute the manual steps described here. Obtain the patches from the Logic PD website at the following URL:

```
http://support.logicpd.com/downloads/1349/
```

Extract the contents of the release package with the following command:

```
$ tar xvf am3517_wilink.tar.gz
```

This creates a directory *am3517_wilink* with the following structure:

```
build-am3517evm-wilink.sh
setenv
patches/PACKAGE/vVERSION
patches/PACKAGE/vVERSION/PACKAGE-vVERSION-DESCRIPTION.patch
patches/PACKAGE/vVERSION/PACKAGE-vVERSION-DESCRIPTION.patch
patches/PACKAGE/vVERSION/series
```

The series file is used by Quilt to determine the order of patches that are applied to the source files. For example, the U-Boot package has one patch file for version 2009.11 that adds support for the Logic PD product ID. The patch directory has the following structure:

```
patches/u-boot/series
patches/u-boot/2009.11-am3517-logic-01-product-id.patch
```

The series file simply lists *2009.11-am3517-logic-01-product-id.patch* as the only entry.

7 X-Loader (MLO)

X-Loader is loaded by the ROM bootloader into internal RAM. X-Loader is responsible for loading U-Boot into memory and launching it.

7.1 Compiling X-Loader

Extract X-Loader and change to the base of the X-Loader directory.

```
$ cd ${PSP_BUILD_DIR}/build
$ tar zxf ${DOWNLOAD_DIR}/com/ti/AM35x-OMAP35x-PSP-SDK-03.00.00.05/src/boot-
strap/x-loader-03.00.00.05.tar.gz

$ cd ${PSP_BUILD_DIR}/build/x-loader-03.00.00.05
```

Remove the intermediate files generated during build. This step is not necessary when building for the first time.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- distclean
```

Choose the AM3517 EVM configuration.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- am3517evm_config
```

Initiate the build.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

Upon successful completion, file *x-load.bin* will be created in the current directory.

7.2 Signing *x-load.bin*

The file *x-load.bin* needs to be signed before it can be used by the ROM bootloader. The signGP tool required for signing is available in the release package under the folder *host-tools/linux*.

To sign the X-Loader binary:

```
$ ${DOWNLOAD_DIR}/com/ti/AM35x-OMAP35x-PSP-SDK-03.00.00.05/host-
tools/linux/signGP x-load.bin
```

The signing utility creates *x-load.bin.ift* in the current directory. Copy the signed file to a directory that will be deployed later.

```
$ cp x-load.bin.ift ${PSP_BUILD_DIR}/rfs/boot/MLO
```

8 U-Boot

U-Boot (the universal bootloader) is responsible for loading the Linux kernel into memory and starting the kernel.

8.1 Compiling U-Boot

Extract the U-Boot source and change to the base of the U-Boot directory.

```
$ cd ${PSP_BUILD_DIR}/build
$ tar zxf ${DOWNLOAD_DIR}/com/ti/AM35x-OMAP35x-PSP-SDK-03.00.00.05/src/u-
boot/u-boot-03.00.00.05.tar.gz

$ cd u-boot-03.00.00.05
```

Create a symbolic link to the U-Boot patches directory and name it *patches* as required by the Quilt utility. Quilt is used to manage a set of patches described by a series file. A symbolic link is created to the patches directory so that the entire source directory may be deleted and re-extracted without losing the patches.

```
$ ln -s ../../patches/u-boot/2009.11 patches
```


Apply all the patches in the series:

```
$ quilt push -a
```

Remove the intermediate files generated during build. This step is not necessary when building for the first time.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- distclean
```

Choose the AM3517 EVM configuration.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- am3517_evm_config
```

Initiate the build.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

Upon successful completion, file *u-boot.bin* will be created in the current directory. Copy this file to a directory that will be the boot partition on the SD card.

```
$ cp u-boot.bin ${PSP_BUILD_DIR}/rfs/boot/
```

The `mkimage` tool must be in the path when building the Linux kernel in order to create a *uImage* file. Copy this tool to a location in your path. For example, in bash:

```
$ mkdir -p ${PSP_BUILD_DIR}/bin
$ cp tools/mkimage ${PSP_BUILD_DIR}/bin
$ export PATH=${PSP_BUILD_DIR}/bin:$PATH
```

9 Linux Kernel

This section describes the steps required to build and configure the Linux kernel.

9.1 Compiling Linux Kernel

Extract the Linux source and change to the base of the Linux source directory.

```
$ cd ${PSP_BUILD_DIR}/build
$ tar zxf ${DOWNLOAD_DIR}/com/ti/AM35x-OMAP35x-PSP-SDK-
03.00.00.05/src/kernel/linux-03.00.00.05.tar.gz

$ cd linux-03.00.00.05
```

Add a symbolic link to the patches and apply all the patches using Quilt.

```
$ ln -s ../../patches/kernel/2.6.32 patches
$ quilt push -a
```

Create default configuration for the AM3517 EVM with Bluetooth and WLAN enabled.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- distclean
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- \
    am3517_evm_WLAN_defconfig
```

Initiate the build.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage
```

NOTE: For the kernel image (*uImage*) to be built, `mkimage` utility must be included in the path. `mkimage` utility is generated (under tools folder) while building *u-boot.bin*.

Upon successful completion, file *uImage* will be created in the directory *./arch/arm/boot*. Copy this file to the boot directory.

```
$ cp arch/arm/boot/uImage ${PSP_BUILD_DIR}/rfs/boot/
```

Build any kernel modules and deploy them to the staging directory.

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- modules
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- \
    modules_install INSTALL_MOD_PATH=${STAGING_DIR}
```

10 WiLink

Both the Bluetooth and WLAN require installing firmware for proper operation. This firmware is available from the TI website.

10.1 Download Firmware

```
$ cd ${DOWNLOAD_DIR}/com/ti
$ wget --no-check-certificate
https://gforge.ti.com/gf/download/frsrelease/295/3359/TI_Connectivity_Packag
e_Firmware_InitScripts_RLS25.INC2.5-25.INC2.5-Linux-x86-Install.tar.bz2

$ tar xjf TI_Connectivity_Package_Firmware_InitScripts_RLS25.INC2.5-
25.INC2.5-Linux-x86-Install.tar.bz2

$ ./TI_Connectivity_Package_Firmware_InitScripts_RLS25.INC2.5-25.INC2.5-
Linux-x86-Install --mode console
```

The AM3517 Linux kernel will look for the Bluetooth firmware in the */lib/firmware directory*. Create that directory in the staging area for the root file system and copy the Bluetooth firmware there.

```
$ cd ${HOME}/TI_Connectivity_Package_Firmware_InitScripts_RLS25.INC2.5/BTFM
$ mkdir ${STAGING_DIR}/lib/firmware -p
$ cp TIInit_7.2.31.bts ${STAGING_DIR}/lib/firmware
```

10.2 Compiling WiLink Drivers

The base WiLink software is available from the omapzoom.org website. Download and extract it.

```
$ mkdir -p ${DOWNLOAD_DIR}/org/omapzoom/wilink
$ cd ${DOWNLOAD_DIR}/org/omapzoom/wilink
$ wget
"http://dev.omapzoom.org/?p=vijay/wlan.git;a=snapshot;h=ca16fa9650abcce671f2
25c290ce20f46d2b4055;sf=tgz" -O WiLink_L23.i3.6.tar.gz
```

Repackage the tar file so that it has the version embedded in the path.

```
$ tar --transform='s:wlan/:wilink-L23.i3.6/:' -xf WiLink_L23.i3.6.tar.gz
$ tar czf wilink-L23.i3.6.tar.gz wilink-L23.i3.6
$ rm -rf wilink-L23.i3.6
$ rmdir wlan
```

Extract the source and apply the patches.

```
$ cd ${PSP_BUILD_DIR}/build
$ tar xvf ${DOWNLOAD_DIR}/org/omapzoom/wilink/wilink-L23.i3.6.tar.gz
$ cd wilink-L23.i3.6
$ ln -s ../../patches/wilink/vL23.i3.6 patches
$ quilt push -a
```


Edit *am3517evm_env_noXCC.sh* to match your environment. Be sure to set the *KERNEL_DIR* to match the location of the Linux-03.00.00.05 kernel.

```
$ cd platforms/os/linux/
$ cat am3517evm_env_noXCC.sh
$ # use your favorite editor here if it isn't vi
$ vi am3517evm_env_noXCC.sh
```

Make these variables available to the WiLink build scripts.

```
$ # NOTE: non-bash users: replace source with a single period.
$ # . am3517evm_env_noXCC.sh
$ source am3517evm_env_noXCC.sh
```

Copy the WiLink firmware to the WiLink source directory and rename it.

```
$ cd ~/TI_Connectivity_Package_Firmware_InitScripts_RLS25.INC2.5/WLAN
$ cp firmware.bin \
  ${PSP_BUILD_DIR}/build/wilink-L23.i3.6/fw/Latest/Fw1273_CHIP.bin
```

Build the source without wpa_supplicant.

```
$ make clean
$ make BUILD_SUPPL=n
```

Upon successful completion, the file *am3517evmBinaries.tar* will be created in the current directory. Extract the file contents to the staging directory.

```
$ tar -C ${STAGING_DIR}/home/root/wlan/ -xf am3517evmBinaries.tar
```

Create WiLink Station start script (*sta_run*) and stop script (*sta_rm*). First the *sta_run* script.

```
$ cd ${PSP_BUILD_DIR}
$ # create sta_run
$ cat >> ${STAGING_DIR}/home/root/wlan/sta_run <<-EOF
    #! /bin/sh
    #set -e
    WLAN_DIR=/home/root/wlan
    insmod \$WLAN_DIR/sdio.ko
    insmod \$WLAN_DIR/tiwlan_drv.ko
    if [ ! -e \$WLAN_DIR/nvs_map.bin ]; then
        cp /sys/class/product_id/wifi_config_data \$WLAN_DIR/nvs_map.bin
    fi
    sleep 2
    \$WLAN_DIR/tiwlan_loader -e \$WLAN_DIR/nvs_map.bin
    ifconfig tiwlan0 10.1.14.1 netmask 255.255.255.0 up
    echo "##"
    echo "## Configure wlan using the following command:"
    echo "./wlan_cu -i/var/run/tiwlan0"
    echo "##"
    echo "##"
EOF

$ chmod 744 ${STAGING_DIR}/home/root/wlan/sta_run
```

Create *sta_rm* script

```
$ cat >> ${STAGING_DIR}/home/root/wlan/sta_rm <<-EOF
    #! /bin/sh
    set -x
```



```
    ifconfig tiwlan0 down
    rmmod tiwlan_drv
    rmmod sdio
    sleep 3
EOF

$ chmod 744 ${STAGING_DIR}/home/root/wlan/sta_rm
```

11 iperf

`iperf` is an optional package that may be used to exercise the WiLink driver and test TCP/UDP performance. Download and extract `iperf`.

11.1 Compiling iperf

```
$ mkdir -p ${DOWNLOAD_DIR}/net/sourceforge/iperf
$ cd ${DOWNLOAD_DIR}/net/sourceforge/iperf
$ wget
http://download.sourceforge.net/project/iperf/iperf/2.0.4%20source/iperf-
2.0.4.tar.gz

$ tar xvf cd ${PSP_BUILD_DIR}/build/iperf-2.0.4
```

Change to iperf directory and initiate build.

```
$ cd ${PSP_BUILD_DIR}/build/iperf-2.0.4
```

Create a configuration file that will be used by *./configure*. Change the privileges of the cache file so that it does not get overwritten.

```
$ cat > am3517.cache <<-EOF
   ac_cv_func_malloc_0_nonnull=yes
EOF
$ chmod a-w am3517.cache
```

Configure the `iperf` package for cross-compilation.

```
$ ./configure --host=arm-none-linux-gnueabi --cache-file=am3517.cache
```

Successful completion of the configure step will create a Makefile in the current directory. Initiate the build.

```
$ make clean
$ make
```

Successful completion will create a *src/iperf* file around 75K in size. Examine the file with the cross-compiler `objdump` utility to determine which library files will be needed on the target platform.

```
$ arm-none-linux-gnueabi-objdump src/iperf -x | grep NEEDED
```

The output from the command shows the necessary libraries required on the target platform at runtime.

```
  NEEDED               libstdc++.so.6
  NEEDED               libm.so.6
  NEEDED               libgcc_s.so.1
  NEEDED               libpthread.so.0
  NEEDED               libc.so.6
```


Applications may also be compiled as static linked applications so they do not have external library dependencies. For example:

```
$ make distclean

$ ./configure --host=arm-none-linux-gnueabi \
  --cache-file=am3517.cache CFLAGS="-static"
```

Successful completion of the configure step will create a Makefile in the current directory. Initiate the build.

```
$ make clean
$ make
```

The resulting program file is over 838K in size because it is static linked and contains additional symbols that may be safely stripped. This can be verified with the `file` utility.

```
$ file src/iperf

src/iperf: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically
linked, for GNU/Linux 2.6.14, not stripped
```

Use the cross-compiler `strip` utility to remove the symbols, reducing the file size to 649K.

```
$ arm-none-linux-gnueabi-strip src/iperf
```

Deploy the application and its documentation to the staging directory.

```
$ make install DESTDIR=${STAGING_DIR}
```

The man page can be viewed on the development PC with the following command:

```
$ man --local-file ${STAGING_DIR}/usr/local/share/man/man1/iperf.1
```

12 Bluetooth

Multiple packages are required for Bluetooth support. The procedure for building each of them is the same: Download and extract the source code, apply the patches (if any), configure, build, and deploy to the staging directory. The core steps involved for each package are described below. Several utilities for the local PC (host) environment are also downloaded and compiled.

12.1 expat

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://downloads.sourceforge.net/project/expat/expat/2.0.1/expat-
2.0.1.tar.gz

$ tar xvf expat-2.0.1.tar.gz
$ cd expat-2.0.1
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig
$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.2 dbus

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://dbus.freedesktop.org/releases/dbus/dbus-1.2.16.tar.gz
$ tar xvf dbus-1.2.16.tar.gz
```



```
$ cd dbus-1.2.16
$ ./configure --host=arm-none-linux-gnueabi --enable-abstract-sockets \
      --with-system-pid-file=/var/volatile/run/dbus-system.pid \
      --with-x=no --prefix='' \
   CFLAGS="-O2 -I/${STAGING_DIR}/include" \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.3 libiconv

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://ftp.gnu.org/pub/gnu/libiconv/libiconv-1.13.1.tar.gz
$ cd libiconv-1.13.1
$ ./configure --prefix='' --host=arm-none-linux-gnueabi \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.4 glib

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://ftp.gnome.org/pub/GNOME/sources/glib/2.22/glib-2.22.2.tar.gz
$ cd glib-2.22.2
$ cat > am3517.cache <<-EOF
   glib_cv_stack_grows=no
   glib_cv_uscore=no
   ac_cv_func_posix_getpwuid_r=yes
   ac_cv_func_posix_getgrgid_r=yes
EOF

$ chmod a-w am3517.cache

$ ./configure --host=arm-none-linux-gnueabi --cache-file=am3517.cache \
    --disable-selinux --prefix='' --with-libiconv=gnu \
   CFLAGS="-I/${STAGING_DIR}/include" \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.5 alsa-lib

```
$ cd ${PSP_BUILD_DIR}/build
$ wget ftp://ftp.alsa-project.org/pub/lib/alsa-lib-1.0.23.tar.bz2
$ cd alsa-lib-1.0.23
$ ./configure --host=arm-none-linux-gnueabi --prefix='' --disable-python \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
```



```
$ make
$ make install DESTDIR=${STAGING_DIR}
```

12.6 bluez

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://www.kernel.org/pub/linux/bluetooth/bluez-4.59.tar.gz
$ cd bluez-4.59
$ ln -s ../../patches/bluez/v4.59 patches
$ quilt push -a
```

Edit each of the **.la* files in *${STAGING_DIR}/${PREFIX}/lib/* to use the full path to the cross-compiled version whenever they specify a *dependency_libs*. For example:

 # dependency_libs=' -L/home/user/am3517_wilink/rfs/stage/lib /home/user/am3517_wilink/rfs/stage/lib/libiconv.la'

```
${STAGING_DIR}/lib/libgmodule-2.0.la
${STAGING_DIR}/lib/libgmodule-2.0.la
${STAGING_DIR}/lib/libgio-2.0.la
${STAGING_DIR}/lib/libgthread-2.0.la
${STAGING_DIR}/lib/libgobject-2.0.la
${STAGING_DIR}/lib/libglib-2.0.la
${STAGING_DIR}/lib/libiconv.la
```

The `pkg-config` utility can be used to return the required build flags by inspecting the relevant **.pc* files. For example, to find the CFLAGS and LIBS flags for linking with the *dbus* library:

```
$ export PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig
$ echo $(pkg-config --cflags dbus-1)
$ echo $(pkg-config --libs dbus-1)
```

Examine the output information from `pkg-config`, read the documentation for each package, and run `./configure --help` to appropriately set the configure flags.

```
$ ./configure --host=arm-none-linux-gnueabi --sysconfdir=/etc/ --prefix='' \
   --enable-tools --enable-bccmd --disable-pcmcia --enable-hid2hci \
   --enable-alsa --enable-hidd --enable-pand --enable-dund \
   --enable-cups --enable-test --enable-udevrules \
   --enable-configfiles --enable-debug \
   --disable-usb --disable-netlink --disable-gstreamer \
   ALSA_CFLAGS="-I${STAGING_DIR}/include" \
   ALSA_LIBS="-L${STAGING_DIR}/lib -lasound" \
   DBUS_CFLAGS="-I${STAGING_DIR}/lib/dbus-1.0/include -
I${STAGING_DIR}/include/dbus-1.0" \
   DBUS_LIBS="-L${STAGING_DIR}/lib -ldbus-1" \
   GLIB_CFLAGS="-I${STAGING_DIR}/lib/glib-2.0/include -
I${STAGING_DIR}/include/glib-2.0" \
   GLIB_LIBS="-L${STAGING_DIR}/lib -lglib-2.0 -lgio-2.0 -lgthread-2.0 -
lgobject-2.0 -lgmodule-2.0"

$ make clean
$ make DESTDIR=${STAGING_DIR} install
```

There is a sample Bluetooth agent application that will respond to pairing requests. Copy this to the staging area.

```
$ cp ${PSP_BUILD_DIR}/build/$PKG-$VER/test/agent \
    ${STAGING_DIR}/home/root/bt/
```


12.7 alsa-utils

```
$ cd ${PSP_BUILD_DIR}/build
$ wget ftp://ftp.alsa-project.org/pub/utils/alsa-utils-1.0.23.tar.bz2
$ tar xjvf ftp://ftp.alsa-project.org/pub/utils/alsa-utils-1.0.23.tar.bz2
$ cd alsa-utils-1.0.23
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
   --with-alsa- prefix=${STAGING_DIR}/lib \
   --with-alsa-inc-prefix=${STAGING_DIR}/include --disable-alsamixer \
   --disable-xmlto PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.8 openobex

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://www.kernel.org/pub/linux/bluetooth/openobex-1.5.tar.gz
$ tar xvf openobex-1.5.tar.gz
$ cd openobex-1.5
$ ln -s ../../patches/openobex/v1.5 patches
$ quilt push -a
$ autoreconf -v -f -i
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
    --disable-usb --enable-apps \
   BLUETOOTH_CFLAGS="-I${STAGING_DIR}/include" \
   BLUETOOTH_LIBS="-L${STAGING_DIR}/lib -lbluetooth" \
   CFLAGS="-I${STAGING_DIR}/include" \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   LIBS="-L${STAGING_DIR}/lib" \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.9 obexftp

```
$ cd ${PSP_BUILD_DIR}/build
$ wget
http://downloads.sourceforge.net/project/openobex/obexftp/0.22/obexftp-
0.22.tar.bz2

$ tar xjvf obexftp-0.22.tar.bz2
$ cd obexftp-0.22
$ ln -s ../../patches/obexftp/v0.22 patches
$ quilt push -a
$ autoreconf -v -f -i
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
   --disable-perl --disable-python --disable-ruby --disable-tcl \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig \
   OPENOBEX_CFLAGS="-I${STAGING_DIR}/include/openobex -
I${STAGING_DIR}/include" \
   OPENOBEX_LIBS="-L${STAGING_DIR}/lib -lopenobex" \
   BLUETOOTH_CFLAGS="-I${STAGING_DIR}/include/bluetooth" \
   BLUETOOTH_LIBS="-L${STAGING_DIR}/lib -lbluetooth" \
```



```
   CFLAGS="-I${STAGING_DIR}/include" \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   LIBS="-L${STAGING_DIR}/lib"

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.10 ussp-push

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://www.xmailserver.org/ussp-push-0.11.tar.gz
$ tar xvf ussp-push-0.11.tar.gz
$ cd ussp-push-0.11
$ ln -s ../../patches/ussp-push/v0.11 patches
$ quilt push -a
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
   PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   CFLAGS="-I${STAGING_DIR}/include"

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

12.11 glib (HOST TOOLS)

```
$ cd ${PSP_BUILD_DIR}/host-tools
$ wget http://ftp.gnome.org/pub/GNOME/sources/glib/2.22/glib-2.22.2.tar.gz
$ tar xvf glib-2.22.2.tar.gz
$ cd glib-2.22.2
$ ./configure --disable-selinux --prefix=${PSP_BUILD_DIR}/bin/host-tools
$ make clean
$ make install
```

12.12 dbus (HOST TOOLS)

```
$ cd ${PSP_BUILD_DIR}/host-tools
$ wget http://dbus.freedesktop.org/releases/dbus/dbus-1.2.16.tar.gz
$ tar xvf dbus-1.2.16.tar.gz
$ cd dbus-1.2.16
$ ./configure --enable-abstract-sockets \
   --with-system-pid-file=/var/volatile/run/dbus-system.pid \
   --with-x=no --prefix=${PSP_BUILD_DIR}/bin/host-tools

$ make clean
$ make
$ make install
```

12.13 dbus-glib (HOST TOOLS)

```
$ cd ${PSP_BUILD_DIR}/host-tools
$ wget http://dbus.freedesktop.org/releases/dbus-glib/dbus-glib-0.82.tar.gz
$ tar xvf dbus-glib-0.82.tar.gz
$ cd dbus-glib-0.82
$ PKG_CONFIG_PATH=${PSP_BUILD_DIR}/bin/host-tools/lib/pkgconfig \
   ./configure --prefix=${PSP_BUILD_DIR}/bin/host-tools
```



```
$ make clean
$ make
$ make install
```

12.14 dbus-glib

```
$ export OLDPATH=$PATH
$ export PATH=${PSP_BUILD_DIR}/bin/host-tools/bin:$PATH
$ cd ${PSP_BUILD_DIR}/build
$ wget http://dbus.freedesktop.org/releases/dbus-glib/dbus-glib-0.82.tar.gz
$ tar xvf dbus-glib-0.82.tar.gz
$ cd dbus-glib-0.82
$ ln -s ../../patches/dbus-glib/v0.82 patches
$ quilt push -a
$ cat > am3517.cache <<-EOF
   ac_cv_have_abstract_sockets=yes
EOF

$ chmod a-w am3517.cache
$ autoreconf -v -f -i
$ ./configure --host=arm-none-linux-gnueabi --cache-file=am3517.cache \
    --prefix='' PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig \
   LDFLAGS="-L${STAGING_DIR}/lib" \
   CFLAGS="-I${STAGING_DIR}/include" \
   DBUS_CFLAGS="-I${STAGING_DIR}/lib/dbus-1.0/include -
I${STAGING_DIR}/include/dbus-1.0" \
   DBUS_LIBS="-L${STAGING_DIR}/lib -ldbus-1" \
   DBUS_GLIB_CFLAGS="-I${STAGING_DIR}/include/glib-2.0 -
I${STAGING_DIR}/lib/glib-2.0/include" \
   DBUS_GLIB_LIBS="-L${STAGING_DIR}/lib -lglib-2.0 -lgio-2.0 -lgthread-2.0 -
lgobject-2.0 -lgmodule-2.0" \
   LIBS="-L${STAGING_DIR}/lib"

$ make clean
$ cp ${PSP_BUILD_DIR}/host-tools/dbus-glib-0.82/tools/dbus-glib-bindings.h \
   ${PSP_BUILD_DIR}/build/dbus-glib-0.82/tools/

$ cp ${PSP_BUILD_DIR}/host-tools/dbus-glib-0.82/tools/dbus-glib-bindings.h \
   ${STAGING_DIR}/include/dbus-1.0/dbus/
$ make
$ make DESTDIR=${STAGING_DIR} install
$ export PATH=$OLDPATH
```

Fix the .la file so that it references the staging directory path.

```
$  sed --in-place=bak "1,\$s: /lib/: ${STAGING_DIR}/lib/:g"
${PSP_BUILD_DIR}/rfs/stage/lib/libdbus-glib-1.la
```

12.15 obex-data-server

```
$ cd ${PSP_BUILD_DIR}/build
$ wget http://tadas.dailyda.com/software/obex-data-server-0.4.5.tar.gz
$ tar xvf obex-data-server-0.4.5.tar.gz
$ cd obex-data-server-0.4.5
$ ln -s ../../patches/obex-data-server/v0.4.5 patches
$ quilt push -a
```



```
$ ./configure --host=arm-none-linux-gnueabi --prefix='' \
   --sysconfdir=/etc --enable-bip=no --disable-usb --enable-system-config \
    PKG_CONFIG_PATH=${STAGING_DIR}/lib/pkgconfig \
    LDFLAGS="-L${STAGING_DIR}/lib" \
    CFLAGS="-I${STAGING_DIR}/include" \
    LIBS="-L${STAGING_DIR}/lib" \
    BLUEZ_CFLAGS="-I${STAGING_DIR}/include" \
    BLUEZ_LIBS="-L${STAGING_DIR}/lib -lbluetooth" \
    OPENOBEX_CFLAGS="-I${STAGING_DIR}/include/openobex" \
    OPENOBEX_LIBS="-L${STAGING_DIR}/lib -lopenobex" \
    GLIB_CFLAGS="-I${STAGING_DIR}/include/glib-2.0 -I${STAGING_DIR}/lib/glib-
2.0/include" \
    GLIB_LIBS="-L${STAGING_DIR}/lib -lglib-2.0 -lgio-2.0 -lgthread-2.0 -
lgobject-2.0 -lgmodule-2.0 -liconv" \
    DBUS_CFLAGS="-I${STAGING_DIR}/lib/dbus-1.0/include -
I${STAGING_DIR}/include/dbus-1.0" \
    DBUS_LIBS="-L${STAGING_DIR}/lib -ldbus-1 -L${STAGING_DIR}/dbus-glib-1 -
ldbus-glib-1"

$ make clean
$ make
$ make DESTDIR=${STAGING_DIR} install
```

13 Bluetooth Sample Application

This section describes creating a minimal Bluetooth application that uses D-Bus to communicate with the obex-data-server.

```
$ cd ${PSP_BUILD_DIR}/build
$ mkdir btapp-0.1
$ cd btapp-0.1
$ ln -s ../../patches/btapp/v0.1 patches
$ quilt push -a
$ cp ../obex-data-server-0.4.5/src/ods-manager.xml .
$ cp ../obex-data-server-0.4.5/src/ods-server.xml .
```

Generate header files from the obex-data-server xml using the `dbus-binding-tool`.

```
$ dbus-binding-tool --mode=glib-client \
   --output=ods-manager.h ods-manager.xml

$ dbus-binding-tool --mode=glib-client --output=ods-server.h ods-server.xml
```

Compile the *btapp.c* to produce an object file.

```
$ arm-none-linux-gnueabi-gcc -O2 \
   -I${STAGING_DIR}/include/glib-2.0 \
   -I${STAGING_DIR}/lib/glib-2.0/include \
   -I${STAGING_DIR}/lib/dbus-1.0/include \
   -I${STAGING_DIR}/include/dbus-1.0 \
   btapp.c -c -o btapp.o
```

Create the *btapp* executable from the object file.

```
$ arm-none-linux-gnueabi-gcc \
   -L${STAGING_DIR}/lib -ldbus-1 -lglib-2.0 -lgio-2.0 -lgthread-2.0 \
   -lgobject-2.0 -lgmodule-2.0 -liconv -L${STAGING_DIR}/lib/dbus-glib-1 \
   -ldbus-glib-1 btapp.o -o btapp
```


Copy the application to the staging area.

```
$ mkdir ${STAGING_DIR}/home/root/bt -p
$ cp btapp ${STAGING_DIR}/home/root/bt/
$ cp ${PSP_BUILD_DIR}/patches/toneStereo.wav ${STAGING_DIR}/home/root/bt/
```

14 Create a Bootable SD Card

For instructions on preparing a bootable SD card, please refer to the following page on the TI wiki: http://processors.wiki.ti.com/index.php?title=SD/MMC_format_for_OMAP3_boot

Assuming that the first partition is mounted on */media/boot* and the second partition is mounted on */media/disk*:

```
$ cd ${PSP_BUILD_DIR}/rfs
```

The ROM bootloader only scans initial FAT entries for the binary X-Loader named *MLO*. Therefore, ensure that *MLO* is the first file to be copied to the SD card.

```
$ cp boot/MLO /media/boot/MLO
$ sync
$ cp boot/u-boot.bin /media/boot/
$ cp boot/uImage /media/boot/
```

Change to the second partition on the SD card and extract the base root file system included in the PSP.

```
$ cd /media/disk && sudo tar zxf ${PSP_BUILD_DIR}/build/nfs.tar.gz
```

Create a tar archive with all the files from the staging area, changing the owner and group to root. Then extract that tar archive to the second partition on the SD card.

```
$ cd ${PSP_BUILD_DIR}/rfs/stage && tar --owner=root \
   --group=root -czvf ${PSP_BUILD_DIR}/rfs/deploy/apps.tar.gz *

$ cd /media/disk && sudo tar zxf ${PSP_BUILD_DIR}/rfs/deploy/apps.tar.gz
$ cd ${PSP_BUILD_DIR}
$ sync
```

Be sure to unmount the SD card before removing it.

```
$ umount /media/disk
$ umount /media/boot
```

14.1 Boot from MMC

Refer to the *Zoom AM3517 Development Kit User Manual* (available on the Logic PD website) for instructions on how to set the boot mode switches to boot from MMC1. For example:

```
( S7:4=ON, S7:2=OFF, S7:1=ON )
```

Power on the EVM and wait for U-Boot to come up.

When the kernel image and file system (ramdisk) are available on the MMC card:

```
AM3517_EVM # mmc init
AM3517_EVM # fatload mmc 0 0x80000000 uImage
AM3517_EVM # setenv bootargs 'console=ttyS0 root=/dev/mmcblk0p2 rw
rootfstype=ext3 rootwait'

AM3517_EVM # bootm 0x80000000
```


NOTE: Once the Linux kernel boots, login as "root". No password is required.

15 WLAN Operation

```
root@am3517-evm:~# cd wlan
```

To start the wireless LAN interface:

```
root@am3517-evm:~/wlan# ./sta_run
```

Refer to the TI document *CLI User Guide* (Literature Number: SPRUGT4) for details on operation of the `wlan_cu` configuration utility. To configure the driver and set up the connection information parameters, use the `wlan_cu` application. For example, to connect to an access point named *TESTAP* with the WEP encryption key *abcdef0123*:

```
root@am3517-evm:~/wlan# ./wlan_cu -i/var-run/tiwlan0
/ p r 1
/ p a 0
/ p w a abcdef0123 0 1 hex
/ c c TESTAP
/ q
```

If the WiLink driver was built with support for `wpa_supplicant` and the `wpa_supplicant` has been started with the following command:

```
wpa_supplicant -Dwext -itiwlan0 -c./wpa_supplicant.txt &
```

Then the `wlan_cu` will write an entry into the *wpa_supplicant.txt* file that looks like this:

```
network={
    ssid="TESTAP"
    key_mgmt=NONE
    pairwise=NONE
    group-WEP40
    auth_alg=OPEN
    wep_key0=abcdef0123
    wep_key1=0000000000
    wep_key2=0000000000
    wep_key3=0000000000
}
```

Ask the access point for an IP address:

```
root@am3517-evm:~/wlan# udhcpc -i tiwlan0
```

To stop the wireless LAN interface:

```
root@am3517-evm:~/wlan# ./sta_rm
```

16 Bluetooth Operation

This section covers the required steps for Bluetooth operation.

Create a unique D-Bus system ID for this device.

```
root@am3517-evm:~# dbus-uuidgen --ensure
```

Look for the *messagebus* user in the password file, create the user if it doesn't exist.

```
root@am3517-evm:~# grep messagebus /etc/passwd
root@am3517-evm:~# adduser messagebus
```


Create the *dbus* directory and start the dbus daemon.

```
root@am3517-evm:~# mkdir /var/run/dbus
root@am3517-evm:~# dbus-daemon --system --fork
```

Start the Bluetooth daemon

```
root@am3517-evm:~# bluetoothd
```

Toggle the GPIO pin to reset the Bluetooth core in the WL127x.

```
root@am3517-evm:~# echo 0 > /sys/class/gpio/gpio56/value
root@am3517-evm:~# sleep 1
root@am3517-evm:~# echo 1 > /sys/class/gpio/gpio56/value
```

Run the `hciattach` command which loads the Bluetooth firmware into the WL127x and enables the Bluetooth device (hci0).

```
root@am3517-evm:~# hciattach -t 50 /dev/ttyS1 texas 3000000
```

The Bluetooth interface hci0 is now available. Start the *obex-data-server* and the sample pairing agent application with a pin number of *0000* to run in the background.

```
root@am3517-evm:~# cd ~/bt
root@am3517-evm:~/bt# obex-data-server -s
root@am3517-evm:~/bt# agent --path /org/bluez/agent 0000 &
```

Ensure the Bluetooth interface is "**`UP RUNNING`**".

```
root@am3517-evm:~/bt# hciconfig hci0 up
root@am3517-evm:~/bt# hciconfig hci0 -a
```

Scan for nearby Bluetooth devices.

```
root@am3517-evm:~/bt# hcitool scan
```

Make the WL127x Bluetooth device discoverable to other devices.

```
root@am3517-evm:~/bt# hcitool hci0 piscan
```

From another Bluetooth enabled device, scan for nearby devices and the *am3517-evm* device should be discovered.

Using the `dbus-send` interface, the Bluetooth interface can be controlled from the command line. For example:

```
root@am3517-evm:~/bt# dbus-send --system \
   --dest=org.bluez --print-reply / org.bluez.Manager.DefaultAdapter
```

The command should return output similar to:

```
 method return sender=:1.0 -> dest=:1.4 reply_serial=2
    object path="/org/bluez/1904/hci0"
```

This is the handle that is needed to talk to the hci0 interface using `dbus`. The number *1904* in this case is the process ID of the Bluetooth daemon. Now that we have a way to identify our interface, we can send a dbus message to our device. Referring to the *doc/adapter-api.txt* in the BlueZ source code directory, we can ask for the adapter's properties by sending it a GetProperties message:

```
root@am3517-evm:~/bt# dbus-send --system --dest=org.bluez \
  --print-reply /org/bluez/1904/hci0 org.bluez.Adapter.GetProperties
```


The `dbus-send` utility can be used to control the Bluetooth interface and interact with other Bluetooth devices. For example, to play a .wav file on a Bluetooth A2DP headset, the following steps could be used. Create an alsa sound configuration file enabling Bluetooth as an output (see *audio/bluetooth.conf* in the BlueZ source directory).

```
root@am3517-evm:~/bt# cat > /etc/asound.conf <<EOF
@hooks [
    {
      func load
      files [
        "/etc/alsa/bluetooth.conf"
     ]
     errors false
     }
]
EOF
```

Put the headset into pairing mode according to the headset's documentation and scan for the Bluetooth address.

```
root@am3517-evm:~/bt# hcitool scan
```

Find the headset's address in the resulting output. Using *00:11:22:33:44:55* as an example address, send a dbus message requesting to create a device.

```
root@am3517-evm:~/bt# dbus-send --system --dest=org.bluez \
  --print-reply /org/bluez/1904/hci0 \
  org.bluez.Adapter.CreateDevice string:00:11:22:33:44:55
```

If the device has already been created, the path can be found using:

```
root@am3517-evm:~/bt# dbus-send --system --dest=org.bluez \
  --print-reply /org/bluez/1904/hci0 \
  org.bluez.Adapter.FindDevice string:00:11:22:33:44:55
```

The object path for the new device will be returned, which should be similar to:

object path="/org/bluez/1904/hci0/org/bluez/1904/hci0/dev_00_11_22_33_44_55"

Using this path, connect to the A2DP headset.

```
root@am3517-evm:~/bt# dbus-send --system --print-reply --type=method_call \
  --dest=org.bluez /org/bluez/1904/hci0/dev_00_11_22_33_44_55 \
  org.bluez.AudioSink.Connect
```

Play a .wav file, sending the output to the Bluetooth headset. The provided *toneStereo.wav* is a 20 second, stereo 44100Hz .wav file. It contains an 800Hz tone in the left channel, 400Hz tone in the right channel. The left channel fades to silence at 2 seconds and then restarts. The right channel fades to silence at 5 seconds and then restarts.

```
root@am3517-evm:~/bt# aplay -Dplug:bluetooth ./toneStereo.wav
```

17 Summary

Completing the steps in this document will build a version 2.6.32 Linux Kernel with additional software support for Bluetooth and WLAN operation on the Zoom AM3517 EVM Development Kit.

