Zigbee SOC Software Information

Details

Currently Supported Chips

- 3390A0
  - Tested on
    - BCM93390SV V11
- 7366C0
  - Tested on
    - BCM97366SV
    - BCM97366SFF
- 7364B0
  - BCM97364SFF

Software

- There are three types of software support
  - Sample Applications
    - Sample apps demonstrate Zigbee functionality (remote control and home automation) in it's simplest form.
    - The user interface is the serial terminal.
    - The sample apps will demonstrate how to call the Zigbee SW API function calls.
  - Atlas
    - Atlas support demonstrates how a Zigbee RF4CE remote control can control a GUI.
  - Stand-Alone Certification Box
    - This support provides a way for easy testing of a box, with just a USB memory stick containing test firmware.
- All three types of software support require the same BOLT, linux, kernel module driver, and server software, as described below.

Setting up BOLT & Linux

- Regardless of setting up for Atlas, sample applications, or a stand-alone certification box, you will need to flash the right version of BOLT and boot the right version of Linux.

- In addition, if you want to make sure that your bound remotes are preserved across power cycles, you will need to flash the non-initrd version of Linux and setup the Linux filesystem.

Install BOLT one of two ways

From the BOLT command line.

```
BOLT> ifconfig eth0 -auto
100 Mbps Full-Duplex
Device eth0:  hwaddr 00-10-18-76-23-C7, ipaddr 10.13.135.137, mask
255.255.254.0
        gateway 10.13.134.1, nameserver 10.10.10.10, domain broadcom.com
        DHCP server 10.9.192.28, DHCP server MAC 00-14-1B-CD-40-00
*** command status = 0
BOLT> flash stb-irva-01:bolt/release/v1.07/bin/bolt-v1.07-7366c0-bb-bfw-
2.1.0.bin flash0.bolt
Reading stb-irva-01:bolt/release/v1.07/bin/bolt-v1.07-7366c0-bb-bfw-
2.1.0.bin: .
Done. 741344 bytes read
Programming...done. 741344 bytes written
*** command status = 0
BOLT> reboot
```

From BBS.

Use the following link:

bolt_v1.07

Or, copy/paste the following:

```
\\brcm-irv\dfs\projects\bseswev_nonos\tftpboot\bolt\release\v1.07\bin
```

Flash the non-initrd version of linux and create the UBIFS file system.

```
BOLT> boot stb-irva-01:314-1.8/vmlinuz-initrd-7366c0
Loader:zimg Filesys:tftp Dev:eth0 File:stb-irva-01:314-1.8/vmlinuz-initrd-
7366c0 Options:(null)
Reading 7956992 bytes from zImage...........
Closing network 'eth0'
Starting program at 0x8000 (DTB @ 0x7821000)
.
.
.
# stbutil
```

```
stbutil v5.0
------------

Using TFTP server:     stb-irva-01
Using TFTP path:       314-1.8
Linux build target:    7366c0

Primary Linux flash:   mlc-nand

1) Install non-initrd kernel image to flash
2) Install UBIFS rootfs to flash (RW/RO)
3) Install JFFS2 rootfs to flash (RW/RO) (not available)
4) Install SQUASHFS rootfs to flash (RO) (uses UBI)
5) Format/partition entire HDD, then install rootfs (not available)
6) Update rootfs on first HDD partition (not available)
7) Install kernel+rootfs to USB thumbdrive (not available)
8) Install kernel+rootfs to eMMC (not available)
q) Exit



Selection:
```

- First, choose option 1) to install non-initrd kernel image to flash.
- Then, run stbutil again, and choose option 2) to install UBIFS file system to flash.
- When complete, you will see the "Sample boot command line". Within this line, the flash number is specified. Either flash0 or flash1.
- For example, for 97366:

```
Finished writing rootfs to flash.

Sample boot command line:

  boot stb-bld-00.broadcom.com:nightly/314/vmlinuz-7366c0
'ubi.mtd=flash1.rootfs0 rootfstype=ubifs root=ubi0:rootfs rw'
```

Set up STARTUP variable to automatically boot linux.

Choose the proper flash number from the above step when setting the STARTUP variable for the ubi.mtd parameter:

I.e.,

```
BOLT> setenv -p STARTUP "boot flash0.kernel: 'ubi.mtd=flash0.rootfs0
rootfstype=ubifs root=ubi0:rootfs rw'"
```

```
*** command status = 0
BOLT>
```

or

```
BOLT> setenv -p STARTUP "boot flash0.kernel: 'ubi.mtd=flash1.rootfs0
rootfstype=ubifs root=ubi0:rootfs rw'"
*** command status = 0
BOLT>
```

Create the \etc\zigbee directory.

- Reboot the box to the linux prompt.
- Create a zigbee subdirectory under the \etc directory.

```
# cd /etc
# mkdir zigbee
```

 From this point, you may continue to the next section below, to build the various components of software, only if you have a need to modify the source for development or debug. Or, you can select from the following choices if you just want to run the software

Setting up a Stand-alone Certification Box, with firmware residing on a USB memory stick

Setting up to run the sample apps

Linux kernel module driver

 If using Atlas, this section should be ignored, as Atlas handles building and running the kernel module driver automatically.

Building

- Change directory to BSEAV/linux/driver/zigbee
- Issue the "source plat" command, if you haven't done it yet. Or, if you do not have access to the plat script, the bare minimum environmental variables, as follows:
    - For 93390SV w/ A0:
        - If you have access to the plat script:

```
bash-4.2$ source plat 93390 a0 sv
```

        - One additional tweak, if you are building for the erouter:

```
bash-4.2$ export LINUX=/fe_lab/agin/src/git/erouter/erouter/rootfs/linux
```

-
    -
        - Bare minimum environmental variables, in case of no plat script:

```
bash-4.2$ export LINUX=/fe_lab/agin/src/git/erouter/erouter/rootfs/linux
bash-4.2$ export PATH=/opt/toolchains/stbgcc-4.8-1.0/bin:$PATH
bash-4.2$ export PLATFORM=93390
bash-4.2$ export BCHP_VER=A0
bash-4.2$ export B_REFSW_ARCH=arm-linux
```

-
    - For 97366SFF w/ C0:
        - If you have access to the plat script:

```
bash-4.2$ source plat 97366 c0 sff
```

-
    -
        - Bare minimum environmental variables, in case of no plat script:

```
bash-4.2$ export LINUX=/opt/brcm/linux-3.14-1.8/7366c0
bash-4.2$ export PATH=/opt/toolchains/stbgcc-4.8-1.2/bin:$PATH
bash-4.2$ export PLATFORM=97336
bash-4.2$ export BCHP_VER=C0
bash-4.2$ export B_REFSW_ARCH=arm-linux
```

- Finally, to build, enter the following, from BSEAV/linux/driver/zigbee:

```
bash-4.2$ make
```

- The resulting binary will be stored in a directory, based on the environmental variable PLATFORM, as shown below
    - For 93390SV w/ A0:

```
bash-4.2$ pwd
/fe_lab/agin/src/git/bcm93390/myrepo/BSEAV/linux/driver/zigbee
bash-4.2$ ls ../../../../obj.93390/BSEAV/linux/driver/zigbee/arm
linux/zigbee_drv.ko
../../../../obj.93390/BSEAV/linux/driver/zigbee/arm-linux/zigbee_drv.ko
```

-
  - For 97366SFF w/ C0:

```
bash-4.2$ pwd
/fe_lab/agin/src/git/bcm93390/myrepo/BSEAV/linux/driver/zigbee
bash-4.2$ ls ../../../../obj.97366/BSEAV/linux/driver/zigbee/arm
linux/zigbee_drv.ko
../../../../obj.97366/BSEAV/linux/driver/zigbee/arm-linux/zigbee_drv.ko
bash-4.2$
```

Running (Inserting the kernel module driver)

- For 93390SV w/ A0:

```
bash-4.2$ cd ../../../../obj.93390/BSEAV/linux/driver/zigbee/arm-linux
```

- For 97366SFF w/ C0:

```
bash-4.2$ cd ../../../../obj.97366/BSEAV/linux/driver/zigbee/arm-linux
```

- Then, do the following:

```
base-4.2$ mknod /dev/zigbee c 105 0
base-4.2$ insmod zigbee_drv.ko
```

Server process

 If using Atlas, this section should be ignored, as Atlas handles building and running the server process automatically.

Building

- The server process is used to communicate to the kernel module driver and handle requests from client processes.
- As a result, the server process will need to be built and run, regardless of whether one intends to run Atlas, sample applications, or setting up a certification box.
- Go to \BSEAV\lib\zigbee\broadbee_mailbox_host\projects\SoC_mailboxHostSide
- Issue the "source plat" command, if you haven't done it yet.
- Enter "make".

Running

- The server process needs to be run as a background process, if you intend to run your application in the same console.
- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide
- If you want to use the RF4CE remote control support, then do the following:

```
# ./SoC_mailboxHost.elf stack_binary/stack_code.bin &
```

- If you want to use home automation support, then do the following:

```
# ./SoC_mailboxHost.elf stack_binary/broadbee_zha12.bin &
```

Sample RF4CE client process

 If using Atlas, this section should be ignored, as Atlas does not use any sample apps.

Building

- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/my_rf4ce_app.
- Issue the "source plat" command, if you haven't done it yet.
- Enter "make".

Running

- Insert kernel module driver as described above
- Run server process in the background, using the appropriate firmware, as described above
- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/my_rf4ce_app
- Run ./objs/rf4ce_reset_app (Choose option 'c' cold start)

```
# ./rf4ce_reset_app
ZIGBEE_SOCKET_SERVER:  selectserver: new connection from 127.0.0.1 on socket
6
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x0
ZIGBEE_RPC_SERVER:  RPC_C2M_Open_Request received for socket 6

Press input either w(warm start) or c(cold start) to reset the stack, other
character is invalid
```

- Run ./objs/rf4ce_pair_app

```
# ./rf4ce_pair_app
ZIGBEE_SOCKET_SERVER:  selectserver: new connection from 127.0.0.1 on socket
6
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x0
ZIGBEE_RPC_SERVER:  RPC_C2M_Open_Request received for socket 6

Please press 'b' to issue binding procedure,
press any other key to use the existing pair reference.
bZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x10
ZIGBEE_RPC_API:  invoking RF4CE_ZRC_SetWakeUpActionCodeReq Callback
Set power filter key successfully
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x11
ZIGBEE_RPC_API:  invoking RF4CE_ZRC_GetWakeUpActionCodeReq Callback
Get power filter key successfully
Starting RF4CE NWK
 00  00  00  00  00  00  00  00  02  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x7
ZIGBEE_RPC_API:  invoking RF4CE_StartReq Callback
RF4CE_Start_NWK_Callback status : 0
Start NWK successfully

Now a remote control can be bound

Binding instruction for RemoteSolution remote control:
1. Press and hold the Setup button on the remote control,
   until the Indicator LED changes from red to green.
2. Press the Info button on the remote control.
Above procedure should be executed within 30 seconds.
Starting ZRC1 target Binding

ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x12
ZIGBEE_RPC_API:  invoking RF4CE_ZRC1_TargetBindReq Callback
One remote control has been bound successfully.

Press Key 'p' to send echo packet to stack, any other key to exit
In RF4CE_PairInd, found a client shows its interest.
MY_RF4CE_APP:  In My_RF4CE_ZRC_PairInd, indication->pairingRef=0x0
Key code=21 is Pressed

Key code=21 is Repeated

Key code=21 is Released
```

Sample Home Automation client process

If using Atlas, this section should be ignored, as Atlas does not use any sample apps.

Building

- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/zigbee_ha_app.
- Issue the "source plat" command, if you haven't done it yet.
- Enter "make".

Running

- Insert kernel module driver as described above
- Run server process in the background, using the appropriate firmware, as described above
- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/zigbee_ha_app/objs
- For SmartPlug[2], run the zigbee_ha_onoff_controller_app:

```
# ./zigbee_ha_onoff_controller_app
```

- For Temperature Monitor (ZBMS3), run the zigbee_ha_onoff_controller_app:

```
# # ./zigbee_ha_set_attribute_report_app
```

Sample Loopback test client process

 If using Atlas, this section should be ignored, as Atlas does not use the loopback app.

Building

- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/my_loopback_app.
- Issue the "source plat" command, if you haven't done it yet.
- Enter "make".

Running

- Insert kernel module driver as described above
- Run server process in the background as described above
- Change directory to BSEAV/lib/zigbee/broadbee_mailbox_host/projects/SoC_mailboxHostSide/my_loopback_app
- Run ./objs/my_loopback_app

Atlas

- With this setup, at power-on, the box will allow the use of a Remote Solutions RF4CE remote to control the Atlas GUI.

Building

- Change the line in \BSEAV\app\atlas\build\makefile as follows:

From:

```
ifeq ($(findstring $(NEXUS_PLATFORM), 97366), $(NEXUS_PLATFORM))
RF4CE_SUPPORT=n
endif
```

To:

```
ifeq ($(findstring $(NEXUS_PLATFORM), 97366), $(NEXUS_PLATFORM))
RF4CE_SUPPORT=y
endif
```

- Build Atlas as usual.

Running (from your mounted working directory)

- Run Atlas as usual (from obj.97366\BSEAV\bin)

```
# start install
# start atlas
```

- Proceed to "Atlas command line support for RF4CE" instructions below.

Running (as stand-alone)

- Copy the following files to \bin
  - SoC_mailboxHost.elf
- Copy the following files to \etc\zigbee
  - stack_code.bin
  - zigbee_drv.ko
- To automatically install the zigbee kernel mod driver at startup, create or update \root\rc.user to include:

```
#!/bin/sh
cd /etc/zigbee
mknod /dev/zigbee c 105 0
insmod zigbee_drv.ko
```

- To support the ability to handle a watchdog timer reset from the Zigbee core:
  - Make sure you copy the files as described above
  - update \etc\inittab to include:

```
# Zigbee stuff
::respawn:/bin/SoC_mailboxHost.elf /etc/zigbee/stack_code.bin
```

Atlas command line support for RF4CE

At power up, for the first time, no RF4CE remote is paired with the settop. You will have to add a new RF4CE remote by using the Atlas command line. Once added, the information is saved in non-volatile memory, and is preserved across power-cycles.

rf4ceRemoteAdd

```
atlas lua> atlas.rf4ceRemoteAdd("my first remote")


Now a remote control can be bound

Binding instruction for RemoteSolution remote control:
1. Press and hold the Setup button on the remote control,
   until the Indicator LED changes from red to green.
2. Press the Info button on the remote control.
Above procedure should be executed within 30 seconds.
Starting ZRC1 target Binding
waiting for rf4ce_Test_ZRC1_TargetBinding_Callback..., statusConf=255
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x10
In RF4CE_PairInd
In RF4CE_PairInd, found a client shows its interest.
MY_RF4CE_APP:  In My_RF4CE_ZRC_PairInd, indication->pairingRef=0x0
ZIGBEE_RPC_API:  invoking RF4CE_ZRC1_TargetBindReq Callback
One remote control has been bound successfully.
got it..., statusConf=0, permPairingRef=0
```

rf4ceRemotesDisplay

```
atlas lua> atlas.rf4ceRemotesDisplay()
RF4CE Remote Name                                    Pairing Reference
Number
================================================================================
```

```
===
my first remote                                          0
================================================================================
===
atlas lua>
```

rf4ceRemoteRemove

```
atlas lua> atlas.rf4ceRemoteRemove(0)
Starting ZRC1 target Unpair
ZIGBEE_RPC_SERVER:  received message from socket 6, message id=0x14
ZIGBEE_RPC_API:  invoking RF4CE_UnpairReq Callback
Unpair with pairref 0 successfully.
atlas lua>
```

Setting up a Stand-alone Certification Box, with firmware residing on a USB memory stick.

With this setup, at power-on, the box will :

- boot up to linux
- install the kernel mod driver
- copy the firmware called "stack_code.bin", residing in the USB flash memory stick at the root directory into \etc\zigbee, if present.
- run the zigbee server code (also known as mailbox_adapter_agent), loading the firmware called "stack_code.bin" residing at \etc\zigbee.

Once set up, the only thing that the user is allowed to do is change the firmware in the USB memory stick. The name of the firmware that will be run is 'stack_code.bin' from the USB FAT file system's root directory. Now, once the firmware is copied over to \etc\zigbee, it is not necessary to use the USB memory stick, as that firmware will be the version used at power-on. If the firmware needs to be updated, you can use the USB memory stick to copy the updated stack_code.bin over again.

First, make sure you follow the instructions under "Setting up BOLT & Linux", above.

Copy the Zigbee software and driver onto the file system

- Reboot board to the LINUX prompt.
    - Note, that you are now booting up with the newly installed non-initrd kernel image from flash.
- Mount to my directory to obtain the necessary zigbee software.

```
chip=7364B0 or7366C0
#
# mount stb-irva-09:/fe_lab/agin /mnt/nfs
# cd /etc
# mkdir zigbee
# cd zigbee
# cp -rp /mnt/nfs/src/zigbee/sw/{chip}/* .
# cd /root
# cp -rp /mnt/nfs/src/zigbee/init/* .
# reboot          (recommended to ensure files written)
#
```

Content of the /root/rc.user file (copied from /mnt/nfs/src/zigbee/init above):

```
#!/bin/sh
mount /dev/sda1 /mnt/usb
cp /mnt/usb/stack_code.bin /etc/zigbee/stack_code.bin
cd /etc/zigbee
mknod /dev/zigbee c 105 0
insmod zigbee_drv.ko
./SoC_mailboxHost.elf /etc/zigbee/stack_code.bin &
```

Copy the firmware into the USB memory stick

- Physically, place your USB memory stick onto one of your USB ports on your PC.
- Copy the firmware onto the USB memory stick's root directory, making sure that the filename is "stack_code.bin"
- Safely eject the USB memory stick from Windows Explorer and physically remove the stick from your PC and place into the reference board's USB port.
- Power cycle the board. You should now see activity on the UART0 serial console, and possibly, on the UART1 serial console.