Differences Between Provided Host Driver and My Working Version

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **cc3000\_common.c**.c - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup common\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Include files

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "cc3000\_common.h"

#include "socket.h"

#include "wlan.h"

#include "evnt\_handler.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \_\_error\_\_

//!

//! @param pcFilename - file name, where error occurred

//! @param ulLine - line number, where error occurred

//!

//! @return none

//!

//! @brief stub function for ASSERT macro

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void \_\_error\_\_(**~~CHAR \*~~char \***pcFilename, **~~UINT32~~unsigned long** ulLine)

{

//TODO full up function

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! UINT32\_TO\_STREAM\_f

//!

//! @param p pointer to the new stream

//! @param u32 pointer to the 32 bit

//!

//! @return pointer to the new stream

//!

//! @brief This function is used for copying 32 bit to stream

//! while converting to little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~UINT8\*~~ unsigned char\*** UINT32\_TO\_STREAM\_f (**~~UINT8 \*p, UINT32~~unsigned char \*p, unsigned long** u32)

{

\*(p)++ = (**~~UINT8~~unsigned char**)(u32);

\*(p)++ = (**~~UINT8~~unsigned char**)((u32) >> 8);

\*(p)++ = (**~~UINT8~~unsigned char**)((u32) >> 16);

\*(p)++ = (**~~UINT8~~unsigned char**)((u32) >> 24);

return p;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! UINT16\_TO\_STREAM\_f

//!

//! @param p pointer to the new stream

//! @param u32 pointer to the 16 bit

//!

//! @return pointer to the new stream

//!

//! @brief This function is used for copying 16 bit to stream

//! while converting to little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~UINT8\*~~ unsigned char\*** UINT16\_TO\_STREAM\_f (**~~UINT8 \*p, UINT16~~unsigned char \*p, unsigned short** u16)

{

\*(p)++ = (**~~UINT8~~unsigned char**)(u16);

\*(p)++ = (**~~UINT8~~unsigned char**)((u16) >> 8);

return p;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! STREAM\_TO\_UINT16\_f

//!

//! @param p pointer to the stream

//! @param offset offset in the stream

//!

//! @return pointer to the new 16 bit

//!

//! @brief This function is used for copying received stream to

//! 16 bit in little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This caused an incorrect response opcode to be returned and so the message wasn’t handled

**~~UINT16~~ unsigned short** STREAM\_TO\_UINT16\_f(**~~CHAR\* p, UINT16~~char\* p, unsigned short** offset)

{

**~~return (UINT16)((UINT16)((UINT16)~~**

**unsigned char\* ptr = (unsigned char\*)p;**

**return (unsigned short)((unsigned short)((unsigned short)**

**~~(\*(p +~~  (\*(ptr +** offset + 1)) << 8) + (**~~UINT16)(\*(p~~unsigned short)(\*(ptr** + offset)));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! STREAM\_TO\_UINT32\_f

//!

//! @param p pointer to the stream

//! @param offset offset in the stream

//!

//! @return pointer to the new 32 bit

//!

//! @brief This function is used for copying received stream to

//! 32 bit in little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~UINT32~~ unsigned long** STREAM\_TO\_UINT32\_f(**~~CHAR\* p, UINT16~~char\* p, unsigned short** offset)

{

**~~return (UINT32)((UINT32)((UINT32)~~**

**return (unsigned long)((unsigned long)((unsigned long)**

(\*(p + offset + 3)) << 24) + (**~~UINT32)((UINT32~~unsigned long)((unsigned long**)

(\*(p + offset + 2)) << 16) + (**~~UINT32)((UINT32~~unsigned long)((unsigned long**)

(\*(p + offset + 1)) << 8) + (**~~UINT32~~unsigned long**)(\*(p + offset)));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **cc3000\_common.h** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_COMMON\_H\_\_

#define \_\_COMMON\_H\_\_

#include "data\_types.h"

#include "error\_codes.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Include files

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdlib.h>

#include <errno.h>

#include <stdint.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// If building with a C++ compiler, make all of the definitions in this header

// have a C binding.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

extern "C" {

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ERROR CODES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define ESUCCESS 0

#define EFAIL -1

#define EERROR EFAIL

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COMMON DEFINES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define WLAN\_ENABLE (1)

#define WLAN\_DISABLE (0)

#define MAC\_ADDR\_LEN (6)

#define SP\_PORTION\_SIZE (32)

/\*Defines for minimal and maximal RX buffer size. This size includes the spi

header and hci header.

The maximal buffer size derives from:

MTU + HCI header + SPI header + sendto() agrs size

The minimum buffer size derives from:

HCI header + SPI header + max args size

This buffer is used for receiving events and data.

The packet can not be longer than MTU size and CC3000 does not support

fragmentation. Note that the same buffer is used for reception of the data

and events from CC3000. That is why the minimum is defined.

The calculation for the actual size of buffer for reception is:

Given the maximal data size MAX\_DATA that is expected to be received by

application, the required buffer is:

Using recv() or recvfrom():

max(CC3000\_MINIMAL\_RX\_SIZE, MAX\_DATA + HEADERS\_SIZE\_DATA + fromlen

+ ucArgsize + 1)

Using gethostbyname() with minimal buffer size will limit the host name

returned to 99 bytes only.

The 1 is used for the overrun detection

Buffer size increased to 130 following the add\_profile() with WEP security

which requires TX buffer size of 130 bytes:

HEADERS\_SIZE\_EVNT + WLAN\_ADD\_PROFILE\_WEP\_PARAM\_LEN + MAX SSID LEN + 4 \* MAX KEY LEN = 130

MAX SSID LEN = 32

MAX SSID LEN = 13 (with add\_profile only ascii key setting is supported,

therfore maximum key size is 13)

\*/

#define CC3000\_MINIMAL\_RX\_SIZE (130 + 1)

#define CC3000\_MAXIMAL\_RX\_SIZE (511 + 1)

/\*Defines for minimal and maximal TX buffer size.

This buffer is used for sending events and data.

The packet can not be longer than MTU size and CC3000 does not support

fragmentation. Note that the same buffer is used for transmission of the data

and commands. That is why the minimum is defined.

The calculation for the actual size of buffer for transmission is:

Given the maximal data size MAX\_DATA, the required buffer is:

Using Sendto():

max(CC3000\_MINIMAL\_TX\_SIZE, MAX\_DATA + SPI\_HEADER\_SIZE

+ SOCKET\_SENDTO\_PARAMS\_LEN + SIMPLE\_LINK\_HCI\_DATA\_HEADER\_SIZE + 1)

Using Send():

max(CC3000\_MINIMAL\_TX\_SIZE, MAX\_DATA + SPI\_HEADER\_SIZE

+ HCI\_CMND\_SEND\_ARG\_LENGTH + SIMPLE\_LINK\_HCI\_DATA\_HEADER\_SIZE + 1)

The 1 is used for the overrun detection \*/

#define CC3000\_MINIMAL\_TX\_SIZE (130 + 1)

#define CC3000\_MAXIMAL\_TX\_SIZE (511 + 1)

//TX and RX buffer sizes, allow to receive and transmit maximum data at length 8.

#ifdef CC3000\_TINY\_DRIVER

#define TINY\_CC3000\_MAXIMAL\_RX\_SIZE 44

#define TINY\_CC3000\_MAXIMAL\_TX\_SIZE 59

#endif

/\*In order to determine your preferred buffer size,

change CC3000\_MAXIMAL\_RX\_SIZE and CC3000\_MAXIMAL\_TX\_SIZE to a value between

the minimal and maximal specified above.

Note that the buffers are allocated by SPI.

In case you change the size of those buffers, you might need also to change

the linker file, since for example on MSP430 FRAM devices the buffers are

allocated in the FRAM section that is allocated manually and not by IDE.

\*/

#ifndef CC3000\_TINY\_DRIVER

#ifdef MDNS\_ADVERTISE\_HOST

#define CC3000\_RX\_BUFFER\_SIZE (CC3000\_MAXIMAL\_RX\_SIZE)

#define CC3000\_TX\_BUFFER\_SIZE (CC3000\_MAXIMAL\_TX\_SIZE)

#else

#define CC3000\_RX\_BUFFER\_SIZE (CC3000\_MINIMAL\_RX\_SIZE)

#define CC3000\_TX\_BUFFER\_SIZE (CC3000\_MINIMAL\_TX\_SIZE)

#endif

//if defined TINY DRIVER we use smaller RX and TX buffer in order to minimize RAM consumption

#else

#define CC3000\_RX\_BUFFER\_SIZE (TINY\_CC3000\_MAXIMAL\_RX\_SIZE)

#define CC3000\_TX\_BUFFER\_SIZE (TINY\_CC3000\_MAXIMAL\_TX\_SIZE)

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Compound Types

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef **~~INT32~~signed long** time\_t;

typedef **~~UINT32~~unsigned long** clock\_t;

typedef **~~INT32~~signed long** suseconds\_t;

typedef struct timeval timeval;

struct timeval

{

time\_t tv\_sec; /\* seconds \*/

suseconds\_t tv\_usec; /\* microseconds \*/

};

typedef **~~CHAR \*(\*~~char \*(\***tFWPatches)(**~~UINT32~~unsigned long** \*usLength);

typedef **~~CHAR \*(\*~~char \*(\***tDriverPatches)(**~~UINT32~~unsigned long** \*usLength);

typedef **~~CHAR \*(\*~~char \*(\***tBootLoaderPatches)(**~~UINT32~~unsigned long** \*usLength);

typedef void (\*tWlanCB)(**~~INT32~~ signed long** event\_type, **~~CHAR \* data, UINT8~~char \* data, unsigned char** length );

typedef **~~INT32~~signed long** (\*tWlanReadInteruptPin)(void);

typedef void (\*tWlanInterruptEnable)(void);

typedef void (\*tWlanInterruptDisable)(void);

typedef void (\*tWriteWlanPin)(**~~UINT8~~unsigned char** val);

typedef struct

{

**~~UINT16~~unsigned short** usRxEventOpcode;

**~~UINT16~~unsigned short** usEventOrDataReceived;

**~~UINT8~~unsigned char** \*pucReceivedData;

**~~UINT8~~unsigned char** \*pucTxCommandBuffer;

tFWPatches sFWPatches;

tDriverPatches sDriverPatches;

tBootLoaderPatches sBootLoaderPatches;

tWlanCB sWlanCB;

tWlanReadInteruptPin ReadWlanInterruptPin;

tWlanInterruptEnable WlanInterruptEnable;

tWlanInterruptDisable WlanInterruptDisable;

tWriteWlanPin WriteWlanPin;

**~~INT32~~signed long** slTransmitDataError;

**~~UINT16~~unsigned short** usNumberOfFreeBuffers;

**~~UINT16~~unsigned short** usSlBufferLength;

**~~UINT16~~unsigned short** usBufferSize;

**~~UINT16~~unsigned short** usRxDataPending;

**~~UINT32~~unsigned long** NumberOfSentPackets;

**~~UINT32~~unsigned long** NumberOfReleasedPackets;

**~~UINT8~~unsigned char** InformHostOnTxComplete;

}sSimplLinkInformation;

extern volatile sSimplLinkInformation tSLInformation;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Prototypes for the APIs.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! SimpleLinkWaitEvent

//!

//! @param usOpcode command operation code

//! @param pRetParams command return parameters

//!

//! @return none

//!

//! @brief Wait for event, pass it to the hci\_event\_handler and

//! update the event opcode in a global variable.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern void SimpleLinkWaitEvent(**~~UINT16~~unsigned short** usOpcode, void \*pRetParams);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! SimpleLinkWaitData

//!

//! @param pBuf data buffer

//! @param from from information

//! @param fromlen from information length

//!

//! @return none

//!

//! @brief Wait for data, pass it to the hci\_event\_handler

//! and update in a global variable that there is

//! data to read.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern void SimpleLinkWaitData(**~~UINT8 \*pBuf, UINT8 \*from, UINT8~~unsigned char \*pBuf, unsigned char \*from, unsigned char** \*fromlen);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! UINT32\_TO\_STREAM\_f

//!

//! \param p pointer to the new stream

//! \param u32 pointer to the 32 bit

//!

//! \return pointer to the new stream

//!

//! \brief This function is used for copying 32 bit to stream

//! while converting to little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~UINT8\*~~ unsigned char\*** UINT32\_TO\_STREAM\_f (**~~UINT8 \*p, UINT32~~unsigned char \*p, unsigned long** u32);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! UINT16\_TO\_STREAM\_f

//!

//! \param p pointer to the new stream

//! \param u32 pointer to the 16 bit

//!

//! \return pointer to the new stream

//!

//! \brief This function is used for copying 16 bit to stream

//! while converting to little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~UINT8\*~~ unsigned char\*** UINT16\_TO\_STREAM\_f (**~~UINT8 \*p, UINT16~~unsigned char \*p, unsigned short** u16);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! STREAM\_TO\_UINT16\_f

//!

//! \param p pointer to the stream

//! \param offset offset in the stream

//!

//! \return pointer to the new 16 bit

//!

//! \brief This function is used for copying received stream to

//! 16 bit in little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~UINT16~~ unsigned short** STREAM\_TO\_UINT16\_f(**~~CHAR\* p, UINT16~~char\* p, unsigned short** offset);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! STREAM\_TO\_UINT32\_f

//!

//! \param p pointer to the stream

//! \param offset offset in the stream

//!

//! \return pointer to the new 32 bit

//!

//! \brief This function is used for copying received stream to

//! 32 bit in little endian format.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~UINT32~~ unsigned long** STREAM\_TO\_UINT32\_f(**~~CHAR\* p, UINT16~~char\* p, unsigned short** offset);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COMMON MACROs

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//This macro is used for copying 8 bit to stream while converting to little endian format.

#define UINT8\_TO\_STREAM(\_p, \_val) {\*(\_p)++ = (\_val);}

//This macro is used for copying 16 bit to stream while converting to little endian format.

#define UINT16\_TO\_STREAM(\_p, \_u16) (UINT16\_TO\_STREAM\_f(\_p, \_u16))

//This macro is used for copying 32 bit to stream while converting to little endian format.

#define UINT32\_TO\_STREAM(\_p, \_u32) (UINT32\_TO\_STREAM\_f(\_p, \_u32))

//This macro is used for copying a specified value length bits (l) to stream while converting to little endian format.

#define ARRAY\_TO\_STREAM(p, a, l) {register INT16 \_i; for (\_i = 0; \_i < l; \_i++) \*(p)++ = ((**~~UINT8~~unsigned char** \*) a)[\_i];}

//This macro is used for copying received stream to 8 bit in little endian format.

#define STREAM\_TO\_UINT8(\_p, \_offset, \_u8) {\_u8 = (**~~UINT8~~unsigned char**)(\*(\_p + \_offset));}

//This macro is used for copying received stream to 16 bit in little endian format.

#define STREAM\_TO\_UINT16(\_p, \_offset, \_u16) {\_u16 = STREAM\_TO\_UINT16\_f(\_p, \_offset);}

//This macro is used for copying received stream to 32 bit in little endian format.

#define STREAM\_TO\_UINT32(\_p, \_offset, \_u32) {\_u32 = STREAM\_TO\_UINT32\_f(\_p, \_offset);}

#define STREAM\_TO\_STREAM(p, a, l) {register INT16 \_i; for (\_i = 0; \_i < l; \_i++) \*(a)++= ((**~~UINT8~~unsigned char** \*) p)[\_i];}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Mark the end of the C bindings section for C++ compilers.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

}

#endif // \_\_cplusplus

#endif // \_\_COMMON\_H\_\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **evnt\_handler.c** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup evnt\_handler\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// INCLUDE FILES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#include <inttypes.h>**

**#include <string.h>**

**#include <avr/io.h>**

**#include <avr/pgmspace.h>**

**#include <avr/interrupt.h>**

#include "cc3000\_common.h"

#include "string.h"

#include "hci.h"

#include "evnt\_handler.h"

#include "wlan.h"

#include "socket.h"

#include "netapp.h"

#include "spi.h"

**#include "ui.h"**

**#include "..\libnerdkits\lcd.h"**

**#include "../libnerdkits/delay.h"**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COMMON DEFINES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define FLOW\_CONTROL\_EVENT\_HANDLE\_OFFSET (0)

#define FLOW\_CONTROL\_EVENT\_BLOCK\_MODE\_OFFSET (1)

#define FLOW\_CONTROL\_EVENT\_FREE\_BUFFS\_OFFSET (2)

#define FLOW\_CONTROL\_EVENT\_SIZE (4)

#define BSD\_RSP\_PARAMS\_SOCKET\_OFFSET (0)

#define BSD\_RSP\_PARAMS\_STATUS\_OFFSET (4)

#define GET\_HOST\_BY\_NAME\_RETVAL\_OFFSET (0)

#define GET\_HOST\_BY\_NAME\_ADDR\_OFFSET (4)

#define ACCEPT\_SD\_OFFSET (0)

#define ACCEPT\_RETURN\_STATUS\_OFFSET (4)

#define ACCEPT\_ADDRESS\_\_OFFSET (8)

#define SL\_RECEIVE\_SD\_OFFSET (0)

#define SL\_RECEIVE\_NUM\_BYTES\_OFFSET (4)

#define SL\_RECEIVE\_\_FLAGS\_\_OFFSET (8)

#define SELECT\_STATUS\_OFFSET (0)

#define SELECT\_READFD\_OFFSET (4)

#define SELECT\_WRITEFD\_OFFSET (8)

#define SELECT\_EXFD\_OFFSET (12)

#define NETAPP\_IPCONFIG\_IP\_OFFSET (0)

#define NETAPP\_IPCONFIG\_SUBNET\_OFFSET (4)

#define NETAPP\_IPCONFIG\_GW\_OFFSET (8)

#define NETAPP\_IPCONFIG\_DHCP\_OFFSET (12)

#define NETAPP\_IPCONFIG\_DNS\_OFFSET (16)

#define NETAPP\_IPCONFIG\_MAC\_OFFSET (20)

#define NETAPP\_IPCONFIG\_SSID\_OFFSET (26)

#define NETAPP\_IPCONFIG\_IP\_LENGTH (4)

#define NETAPP\_IPCONFIG\_MAC\_LENGTH (6)

#define NETAPP\_IPCONFIG\_SSID\_LENGTH (32)

#define NETAPP\_PING\_PACKETS\_SENT\_OFFSET (0)

#define NETAPP\_PING\_PACKETS\_RCVD\_OFFSET (4)

#define NETAPP\_PING\_MIN\_RTT\_OFFSET (8)

#define NETAPP\_PING\_MAX\_RTT\_OFFSET (12)

#define NETAPP\_PING\_AVG\_RTT\_OFFSET (16)

#define GET\_SCAN\_RESULTS\_TABlE\_COUNT\_OFFSET (0)

#define GET\_SCAN\_RESULTS\_SCANRESULT\_STATUS\_OFFSET (4)

#define GET\_SCAN\_RESULTS\_ISVALID\_TO\_SSIDLEN\_OFFSET (8)

#define GET\_SCAN\_RESULTS\_FRAME\_TIME\_OFFSET (10)

#define GET\_SCAN\_RESULTS\_SSID\_MAC\_LENGTH (38)

#define GET\_MSS\_VAL\_RETVAL\_OFFSET (0)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GLOBAL VARAIABLES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~UINT32~~ unsigned long** socket\_active\_status = SOCKET\_STATUS\_INIT\_VAL;

#ifdef MDNS\_ADVERTISE\_HOST

**~~UINT8~~unsigned char** localIP[NETAPP\_IPCONFIG\_IP\_LENGTH];

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Prototypes for the static functions

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static **~~INT32~~ long** hci\_event\_unsol\_flowcontrol\_handler(**~~CHAR~~char** \*pEvent);

static void update\_socket\_active\_status(**~~CHAR~~char** \*resp\_params);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_unsol\_handle\_patch\_request

//!

//! @param event\_hdr event header

//!

//! @return none

//!

//! @brief Handle unsolicited event from type patch request

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void hci\_unsol\_handle\_patch\_request(**~~CHAR~~char** \*event\_hdr)

{

**~~CHAR \*~~char \***params = (**~~CHAR~~char** \*)(event\_hdr) + HCI\_EVENT\_HEADER\_SIZE;

**~~UINT32~~unsigned long** ucLength = 0;

**~~CHAR~~char** \*patch;

switch (\*params)

{

case HCI\_EVENT\_PATCHES\_DRV\_REQ:

if (tSLInformation.sDriverPatches)

{

patch = tSLInformation.sDriverPatches(&ucLength);

if (patch)

{

hci\_patch\_send(HCI\_EVENT\_PATCHES\_DRV\_REQ,

tSLInformation.pucTxCommandBuffer, patch, ucLength);

return;

}

}

// Send 0 length Patches response event

hci\_patch\_send(HCI\_EVENT\_PATCHES\_DRV\_REQ,

tSLInformation.pucTxCommandBuffer, 0, 0);

break;

case HCI\_EVENT\_PATCHES\_FW\_REQ:

if (tSLInformation.sFWPatches)

{

patch = tSLInformation.sFWPatches(&ucLength);

// Build and send a patch

if (patch)

{

hci\_patch\_send(HCI\_EVENT\_PATCHES\_FW\_REQ,

tSLInformation.pucTxCommandBuffer, patch, ucLength);

return;

}

}

// Send 0 length Patches response event

hci\_patch\_send(HCI\_EVENT\_PATCHES\_FW\_REQ,

tSLInformation.pucTxCommandBuffer, 0, 0);

break;

case HCI\_EVENT\_PATCHES\_BOOTLOAD\_REQ:

if (tSLInformation.sBootLoaderPatches)

{

patch = tSLInformation.sBootLoaderPatches(&ucLength);

if (patch)

{

hci\_patch\_send(HCI\_EVENT\_PATCHES\_BOOTLOAD\_REQ,

tSLInformation.pucTxCommandBuffer, patch, ucLength);

return;

}

}

// Send 0 length Patches response event

hci\_patch\_send(HCI\_EVENT\_PATCHES\_BOOTLOAD\_REQ,

tSLInformation.pucTxCommandBuffer, 0, 0);

break;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_event\_handler

//!

//! @param pRetParams incoming data buffer

//! @param from from information (in case of data received)

//! @param fromlen from information length (in case of data received)

//!

//! @return none

//!

//! @brief Parse the incoming events packets and issues corresponding

//! event handler from global array of handlers pointers

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~UINT8 \*~~ unsigned char \*** hci\_event\_handler(void \*pRetParams, **~~UINT8 \*from, UINT8~~unsigned char \*from, unsigned char** \*fromlen)

{

**~~UINT8~~unsigned char** \*pucReceivedData, ucArgsize;

**~~UINT16~~unsigned short** usLength;

**~~UINT8~~unsigned char** \*pucReceivedParams;

**~~UINT16~~unsigned short** usReceivedEventOpcode = 0;

**~~UINT32~~unsigned long** retValue32;

**~~UINT8~~unsigned char** \* RecvParams;

**~~UINT8~~unsigned char** \*RetParams;

while (1)

{

if (tSLInformation.usEventOrDataReceived != 0)

{

**// this is pointing 5 bytes into the message (past the SPI header)**

pucReceivedData = (tSLInformation.pucReceivedData);

**// if first byte indicates HCI\_TYPE\_EVENT (it's a 4)...**

if (\*pucReceivedData == HCI\_TYPE\_EVNT)

{

**~~// Event Received~~**

**// Event Opcode starts 1 byte past past HCI hdr and is two bytes, LSB (so flip)**

STREAM\_TO\_UINT16((**~~CHAR~~char** \*)pucReceivedData, HCI\_EVENT\_OPCODE\_OFFSET,

usReceivedEventOpcode);

pucReceivedParams = pucReceivedData + HCI\_EVENT\_HEADER\_SIZE; **//, +5 past HCI hdr start**

RecvParams = pucReceivedParams;

RetParams = pRetParams;

**// TODO: TI SPI doc say this HCI param length is 2 bytes, not one!**

// In case unsolicited event received - here the handling finished

if (hci\_unsol\_event\_handler((**~~CHAR~~char** \*)pucReceivedData) == 0)

{

**// + 3 past HCI hdr start (4th byte is 8-bit length)**

STREAM\_TO\_UINT8(pucReceivedData, HCI\_DATA\_LENGTH\_OFFSET, usLength);

switch(usReceivedEventOpcode)

{

case HCI\_CMND\_READ\_BUFFER\_SIZE:

{

STREAM\_TO\_UINT8((**~~CHAR \*)~~char \*)**pucReceivedParams, 0,

tSLInformation.usNumberOfFreeBuffers);

STREAM\_TO\_UINT16((**~~CHAR \*)~~char \*)**pucReceivedParams, 1,

tSLInformation.usSlBufferLength);

}

break;

case HCI\_CMND\_WLAN\_CONFIGURE\_PATCH:

case HCI\_NETAPP\_DHCP:

case HCI\_NETAPP\_PING\_SEND:

case HCI\_NETAPP\_PING\_STOP:

case HCI\_NETAPP\_ARP\_FLUSH:

case HCI\_NETAPP\_SET\_DEBUG\_LEVEL:

case HCI\_NETAPP\_SET\_TIMERS:

case HCI\_EVNT\_NVMEM\_READ:

case HCI\_EVNT\_NVMEM\_CREATE\_ENTRY:

case HCI\_CMND\_NVMEM\_WRITE\_PATCH:

case HCI\_NETAPP\_PING\_REPORT:

case HCI\_EVNT\_MDNS\_ADVERTISE:

STREAM\_TO\_UINT8(pucReceivedData, HCI\_EVENT\_STATUS\_OFFSET

,\*(**~~UINT8~~unsigned char** \*)pRetParams);

break;

case HCI\_CMND\_SETSOCKOPT:

case HCI\_CMND\_WLAN\_CONNECT:

case HCI\_CMND\_WLAN\_IOCTL\_STATUSGET:

case HCI\_EVNT\_WLAN\_IOCTL\_ADD\_PROFILE:

case HCI\_CMND\_WLAN\_IOCTL\_DEL\_PROFILE:

case HCI\_CMND\_WLAN\_IOCTL\_SET\_CONNECTION\_POLICY:

case HCI\_CMND\_WLAN\_IOCTL\_SET\_SCANPARAM:

case HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_START:

case HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_STOP:

case HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_SET\_PREFIX:

case HCI\_CMND\_EVENT\_MASK:

case HCI\_EVNT\_WLAN\_DISCONNECT:

case HCI\_EVNT\_SOCKET:

case HCI\_EVNT\_BIND:

case HCI\_CMND\_LISTEN:

case HCI\_EVNT\_CLOSE\_SOCKET:

case HCI\_EVNT\_CONNECT:

case HCI\_EVNT\_NVMEM\_WRITE:

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams,0

,\*(**~~UINT32~~unsigned long** \*)pRetParams);

break;

case HCI\_EVNT\_READ\_SP\_VERSION:

STREAM\_TO\_UINT8(pucReceivedData, HCI\_EVENT\_STATUS\_OFFSET

,\*(**~~UINT8~~unsigned char** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 1;

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams, 0, retValue32);

UINT32\_TO\_STREAM((**~~UINT8 \*)~~unsigned char \*)**pRetParams, retValue32);

break;

case HCI\_EVNT\_BSD\_GETHOSTBYNAME:

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams

,GET\_HOST\_BY\_NAME\_RETVAL\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams

,GET\_HOST\_BY\_NAME\_ADDR\_OFFSET,\*(**~~UINT32 \*)~~unsigned long \*)**pRetParams);

break;

case HCI\_EVNT\_GETMSSVALUE:

STREAM\_TO\_UINT16((**~~CHAR~~char** \*)pucReceivedParams

,GET\_MSS\_VAL\_RETVAL\_OFFSET,\*(**~~UINT16 \*)~~unsigned short \*)**pRetParams);

break;

case HCI\_EVNT\_ACCEPT:

{

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams,ACCEPT\_SD\_OFFSET

,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR~~char** \*)pucReceivedParams

,ACCEPT\_RETURN\_STATUS\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR \*)~~char \*)**pRetParams) + 4;

//This argument returns in network order

memcpy((**~~UINT8 \*)~~unsigned char \*)**pRetParams,

pucReceivedParams + ACCEPT\_ADDRESS\_\_OFFSET, sizeof(sockaddr));

break;

}

case HCI\_EVNT\_RECV:

case HCI\_EVNT\_RECVFROM:

{

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SL\_RECEIVE\_SD\_OFFSET ,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SL\_RECEIVE\_NUM\_BYTES\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SL\_RECEIVE\_\_FLAGS\_\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

if(((tBsdReadReturnParams \*)pRetParams)->iNumberOfBytes == ERROR\_SOCKET\_INACTIVE)

{

set\_socket\_active\_status(((tBsdReadReturnParams \*)pRetParams)->iSocketDescriptor,SOCKET\_STATUS\_INACTIVE);

}

break;

}

case HCI\_EVNT\_SEND:

case HCI\_EVNT\_SENDTO:

{

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SL\_RECEIVE\_SD\_OFFSET ,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SL\_RECEIVE\_NUM\_BYTES\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

break;

}

case HCI\_EVNT\_SELECT:

{

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SELECT\_STATUS\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SELECT\_READFD\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SELECT\_WRITEFD\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR~~char** \*)pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,SELECT\_EXFD\_OFFSET,\*(**~~UINT32 \*)~~unsigned long \*)**pRetParams);

break;

}

case HCI\_CMND\_GETSOCKOPT:

STREAM\_TO\_UINT8(pucReceivedData, HCI\_EVENT\_STATUS\_OFFSET,((tBsdGetSockOptReturnParams \*)pRetParams)->iStatus);

//This argument returns in network order

memcpy((**~~UINT8~~unsigned char** \*)pRetParams, pucReceivedParams, 4);

break;

case HCI\_CMND\_WLAN\_IOCTL\_GET\_SCAN\_RESULTS:

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,GET\_SCAN\_RESULTS\_TABlE\_COUNT\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR \*)~~char \*)**pRetParams) + 4;

STREAM\_TO\_UINT32((**~~CHAR \*)~~char \*)**pucReceivedParams,GET\_SCAN\_RESULTS\_SCANRESULT\_STATUS\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR \*)~~char \*)**pRetParams) + 4;

STREAM\_TO\_UINT16((**~~CHAR \*)~~char \*)**pucReceivedParams,GET\_SCAN\_RESULTS\_ISVALID\_TO\_SSIDLEN\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR \*)~~char \*)**pRetParams) + 2;

STREAM\_TO\_UINT16((**~~CHAR \*)~~char \*)**pucReceivedParams,GET\_SCAN\_RESULTS\_FRAME\_TIME\_OFFSET,\*(**~~UINT32~~unsigned long** \*)pRetParams);

pRetParams = ((**~~CHAR \*)~~char \*)**pRetParams) + 2;

memcpy((**~~UINT8 \*)~~unsigned char \*)**pRetParams, (**~~CHAR \*)(~~char \*)(**pucReceivedParams + GET\_SCAN\_RESULTS\_FRAME\_TIME\_OFFSET + 2), GET\_SCAN\_RESULTS\_SSID\_MAC\_LENGTH);

break;

case HCI\_CMND\_SIMPLE\_LINK\_START:

break;

case HCI\_NETAPP\_IPCONFIG:

//Read IP address

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

RecvParams += 4;

//Read subnet

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

RecvParams += 4;

//Read default GW

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

RecvParams += 4;

//Read DHCP server

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

RecvParams += 4;

//Read DNS server

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

RecvParams += 4;

//Read Mac address

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_MAC\_LENGTH);

RecvParams += 6;

//Read SSID

STREAM\_TO\_STREAM(RecvParams,RetParams,NETAPP\_IPCONFIG\_SSID\_LENGTH);

}

}

**// if we received the one we're waiting for, then indicate that we're not looking for one any longer**

if (usReceivedEventOpcode == tSLInformation.usRxEventOpcode)

{

tSLInformation.usRxEventOpcode = 0;

}

}

else

{

pucReceivedParams = pucReceivedData;

STREAM\_TO\_UINT8((**~~CHAR~~char** \*)pucReceivedData, HCI\_PACKET\_ARGSIZE\_OFFSET, ucArgsize);

STREAM\_TO\_UINT16((**~~CHAR~~char** \*)pucReceivedData, HCI\_PACKET\_LENGTH\_OFFSET, usLength);

// Data received: note that the only case where from and from length

// are not null is in recv from, so fill the args accordingly

if (from)

{

STREAM\_TO\_UINT32((**~~CHAR \*)(~~char \*)(**pucReceivedData + HCI\_DATA\_HEADER\_SIZE), BSD\_RECV\_FROM\_FROMLEN\_OFFSET, \*(**~~UINT32~~unsigned long** \*)fromlen);

memcpy(from, (pucReceivedData + HCI\_DATA\_HEADER\_SIZE + BSD\_RECV\_FROM\_FROM\_OFFSET) ,\*fromlen);

}

memcpy(pRetParams, pucReceivedParams + HCI\_DATA\_HEADER\_SIZE + ucArgsize,

usLength - ucArgsize);

tSLInformation.usRxDataPending = 0;

}

tSLInformation.usEventOrDataReceived = 0;

SpiResumeSpi();

// Since we are going to TX - we need to handle this event after the

// ResumeSPi since we need interrupts

if ((\*pucReceivedData == HCI\_TYPE\_EVNT) &&

(usReceivedEventOpcode == HCI\_EVNT\_PATCHES\_REQ))

{

hci\_unsol\_handle\_patch\_request((**~~CHAR~~char** \*)pucReceivedData);

}

if ((tSLInformation.usRxEventOpcode == 0) && (tSLInformation.usRxDataPending == 0))

{

return NULL;

}

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_unsol\_event\_handler

//!

//! @param event\_hdr event header

//!

//! @return 1 if event supported and handled

//! 0 if event is not supported

//!

//! @brief Handle unsolicited events

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~INT32~~ long** hci\_unsol\_event\_handler(**~~CHAR~~char** \*event\_hdr)

{

**~~CHAR~~char** \* data = NULL;

**~~INT32~~unsigned short** event\_type; // not sure why they had this as long

**~~UINT32~~unsigned long** NumberOfReleasedPackets;

**~~UINT32~~unsigned long** NumberOfSentPackets;

STREAM\_TO\_UINT16(event\_hdr, HCI\_EVENT\_OPCODE\_OFFSET,event\_type);

if (event\_type & HCI\_EVNT\_UNSOL\_BASE)

{

switch(event\_type)

{

case HCI\_EVNT\_DATA\_UNSOL\_FREE\_BUFF:

{

hci\_event\_unsol\_flowcontrol\_handler(event\_hdr);

NumberOfReleasedPackets = tSLInformation.NumberOfReleasedPackets;

NumberOfSentPackets = tSLInformation.NumberOfSentPackets;

if (NumberOfReleasedPackets == NumberOfSentPackets)

{

if (tSLInformation.InformHostOnTxComplete)

{

tSLInformation.sWlanCB(HCI\_EVENT\_CC3000\_CAN\_SHUT\_DOWN, NULL, 0);

}

}

return 1;

}

}

}

if(event\_type & HCI\_EVNT\_WLAN\_UNSOL\_BASE)

{

switch(event\_type)

{

case HCI\_EVNT\_WLAN\_KEEPALIVE:

case HCI\_EVNT\_WLAN\_UNSOL\_CONNECT:

case HCI\_EVNT\_WLAN\_UNSOL\_DISCONNECT:

case HCI\_EVNT\_WLAN\_UNSOL\_INIT:

case HCI\_EVNT\_WLAN\_ASYNC\_SIMPLE\_CONFIG\_DONE:

if( tSLInformation.sWlanCB )

{

tSLInformation.sWlanCB(event\_type, 0, 0);

}

break;

case HCI\_EVNT\_WLAN\_UNSOL\_DHCP:

{

**~~UINT8~~unsigned char** params[NETAPP\_IPCONFIG\_MAC\_OFFSET + 1]; // extra byte is for the status

**~~UINT8~~unsigned char** \*recParams = params;

data = (**~~CHAR~~char**\*)(event\_hdr) + HCI\_EVENT\_HEADER\_SIZE;

//Read IP address

STREAM\_TO\_STREAM(data,recParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

data += 4;

#ifdef MDNS\_ADVERTISE\_HOST

localIP[0] = \*(recParams-NETAPP\_IPCONFIG\_IP\_LENGTH);

localIP[1] = \*(recParams-NETAPP\_IPCONFIG\_IP\_LENGTH + 1);

localIP[2] = \*(recParams-NETAPP\_IPCONFIG\_IP\_LENGTH + 2);

localIP[3] = \*(recParams-NETAPP\_IPCONFIG\_IP\_LENGTH + 3);

#endif

//Read subnet

STREAM\_TO\_STREAM(data,recParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

data += 4;

//Read default GW

STREAM\_TO\_STREAM(data,recParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

data += 4;

//Read DHCP server

STREAM\_TO\_STREAM(data,recParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

data += 4;

//Read DNS server

STREAM\_TO\_STREAM(data,recParams,NETAPP\_IPCONFIG\_IP\_LENGTH);

// read the status

STREAM\_TO\_UINT8(event\_hdr, HCI\_EVENT\_STATUS\_OFFSET, \*recParams);

if( tSLInformation.sWlanCB )

{

tSLInformation.sWlanCB(event\_type, (**~~CHAR~~char** \*)params, sizeof(params));

}

}

break;

case HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT:

{

netapp\_pingreport\_args\_t params;

data = (**~~CHAR\*)(~~char\*)(**event\_hdr) + HCI\_EVENT\_HEADER\_SIZE;

STREAM\_TO\_UINT32(data, NETAPP\_PING\_PACKETS\_SENT\_OFFSET, params.packets\_sent);

STREAM\_TO\_UINT32(data, NETAPP\_PING\_PACKETS\_RCVD\_OFFSET, params.packets\_received);

STREAM\_TO\_UINT32(data, NETAPP\_PING\_MIN\_RTT\_OFFSET, params.min\_round\_time);

STREAM\_TO\_UINT32(data, NETAPP\_PING\_MAX\_RTT\_OFFSET, params.max\_round\_time);

STREAM\_TO\_UINT32(data, NETAPP\_PING\_AVG\_RTT\_OFFSET, params.avg\_round\_time);

if( tSLInformation.sWlanCB )

{

tSLInformation.sWlanCB(event\_type, (**~~CHAR~~char** \*)&params, sizeof(params));

}

}

break;

case HCI\_EVNT\_BSD\_TCP\_CLOSE\_WAIT:

{

data = (**~~CHAR~~char** \*)(event\_hdr) + HCI\_EVENT\_HEADER\_SIZE;

if( tSLInformation.sWlanCB )

{

//data[0] represents the socket id, for which FIN was received by remote.

//Upon receiving this event, the user can close the socket, or else the

//socket will be closded after inacvitity timeout (by default 60 seconds)

tSLInformation.sWlanCB(event\_type, data, 1);

}

}

break;

case HCI\_EVNT\_ASYNC\_ARP\_DONE:

case HCI\_EVNT\_ASYNC\_ARP\_WAITING:

if( tSLInformation.sWlanCB )

{

tSLInformation.sWlanCB(event\_type, 0, 0);

}

break;

//'default' case which means "event not supported"

default:

return (0);

}

return(1);

}

if ((event\_type == HCI\_EVNT\_SEND) || (event\_type == HCI\_EVNT\_SENDTO)

|| (event\_type == HCI\_EVNT\_WRITE))

{

**~~CHAR~~char** \*pArg;

**~~INT32~~long** status;

pArg = M\_BSD\_RESP\_PARAMS\_OFFSET(event\_hdr);

STREAM\_TO\_UINT32(pArg, BSD\_RSP\_PARAMS\_STATUS\_OFFSET,status);

if (ERROR\_SOCKET\_INACTIVE == status)

{

// The only synchronous event that can come from SL device in form of

// command complete is "Command Complete" on data sent, in case SL device

// was unable to transmit

STREAM\_TO\_UINT8(event\_hdr, HCI\_EVENT\_STATUS\_OFFSET, tSLInformation.slTransmitDataError);

update\_socket\_active\_status(M\_BSD\_RESP\_PARAMS\_OFFSET(event\_hdr));

return (1);

}

else

return (0);

}

//handle a case where unsolicited event arrived, but was not handled by any of the cases above

if ((event\_type != tSLInformation.usRxEventOpcode) && (event\_type != HCI\_EVNT\_PATCHES\_REQ))

{

**lcd\_line\_three();**

**FILE lcd\_stream = FDEV\_SETUP\_STREAM(lcd\_putchar, 0, \_FDEV\_SETUP\_WRITE);**

**fprintf\_P(&lcd\_stream, PSTR(" NO MATCH: %2x:%2x "), event\_type, tSLInformation.usRxEventOpcode);**

return(1);

**} else {**

**lcd\_line\_four();**

**FILE lcd\_stream = FDEV\_SETUP\_STREAM(lcd\_putchar, 0, \_FDEV\_SETUP\_WRITE);**

**fprintf\_P(&lcd\_stream, PSTR(" MATCH: %2x "), event\_type, tSLInformation.usRxEventOpcode);**

}

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_unsolicited\_event\_handler

//!

//! @param None

//!

//! @return ESUCCESS if successful, EFAIL if an error occurred

//!

//! @brief Parse the incoming unsolicited event packets and issues

//! corresponding event handler.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~INT32~~long** hci\_unsolicited\_event\_handler(void)

{

**~~UINT32~~unsigned long** res = 0;

**~~UINT8~~unsigned char** \*pucReceivedData;

if (tSLInformation.usEventOrDataReceived != 0)

{

pucReceivedData = (tSLInformation.pucReceivedData);

if (\*pucReceivedData == HCI\_TYPE\_EVNT)

{

// In case unsolicited event received - here the handling finished

if (hci\_unsol\_event\_handler((**~~CHAR~~char** \*)pucReceivedData) == 1)

{

// There was an unsolicited event received - we can release the buffer

// and clean the event received

tSLInformation.usEventOrDataReceived = 0;

res = 1;

SpiResumeSpi();

}

}

}

return res;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! set\_socket\_active\_status

//!

//! @param Sd

//! @param Status

//! @return none

//!

//! @brief Check if the socket ID and status are valid and set

//! accordingly the global socket status

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void set\_socket\_active\_status(**~~INT32 Sd, INT32~~long Sd, long** Status)

{

if(M\_IS\_VALID\_SD(Sd) && M\_IS\_VALID\_STATUS(Status))

{

socket\_active\_status &= ~(1 << Sd); /\* clean socket's mask \*/

socket\_active\_status |= (Status << Sd); /\* set new socket's mask \*/

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_event\_unsol\_flowcontrol\_handler

//!

//! @param pEvent pointer to the string contains parameters for IPERF

//! @return ESUCCESS if successful, EFAIL if an error occurred

//!

//! @brief Called in case unsolicited event from type

//! HCI\_EVNT\_DATA\_UNSOL\_FREE\_BUFF has received.

//! Keep track on the number of packets transmitted and update the

//! number of free buffer in the SL device.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~INT32~~ long** hci\_event\_unsol\_flowcontrol\_handler(**~~CHAR~~char** \*pEvent)

{

**~~INT32~~long** temp, value;

**~~UINT16~~unsigned short** i;

**~~UINT16~~unsigned short** pusNumberOfHandles=0;

**~~CHAR~~char** \*pReadPayload;

STREAM\_TO\_UINT16((**~~CHAR~~char** \*)pEvent,HCI\_EVENT\_HEADER\_SIZE,pusNumberOfHandles);

pReadPayload = ((**~~CHAR~~char** \*)pEvent +

HCI\_EVENT\_HEADER\_SIZE + sizeof(pusNumberOfHandles));

temp = 0;

for(i = 0; i < pusNumberOfHandles ; i++)

{

STREAM\_TO\_UINT16(pReadPayload, FLOW\_CONTROL\_EVENT\_FREE\_BUFFS\_OFFSET, value);

temp += value;

pReadPayload += FLOW\_CONTROL\_EVENT\_SIZE;

}

tSLInformation.usNumberOfFreeBuffers += temp;

tSLInformation.NumberOfReleasedPackets += temp;

return(ESUCCESS);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! get\_socket\_active\_status

//!

//! @param Sd Socket IS

//! @return Current status of the socket.

//!

//! @brief Retrieve socket status

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**~~INT32~~ long** get\_socket\_active\_status(**~~INT32~~long** Sd)

{

if(M\_IS\_VALID\_SD(Sd))

{

return (socket\_active\_status & (1 << Sd)) ? SOCKET\_STATUS\_INACTIVE : SOCKET\_STATUS\_ACTIVE;

}

return SOCKET\_STATUS\_INACTIVE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! update\_socket\_active\_status

//!

//! @param resp\_params Socket IS

//! @return Current status of the socket.

//!

//! @brief Retrieve socket status

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void update\_socket\_active\_status(**~~CHAR~~char** \*resp\_params)

{

**~~INT32~~long** status, sd;

STREAM\_TO\_UINT32(resp\_params, BSD\_RSP\_PARAMS\_SOCKET\_OFFSET,sd);

STREAM\_TO\_UINT32(resp\_params, BSD\_RSP\_PARAMS\_STATUS\_OFFSET,status);

if(ERROR\_SOCKET\_INACTIVE == status)

{

set\_socket\_active\_status(sd, SOCKET\_STATUS\_INACTIVE);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

**~~//! SimpleLinkWaitEvent~~**

**//!**

//!

//! @param usOpcode command operation code

//! @param pRetParams command return parameters

//!

//! @return none

//!

//! @brief Wait for event, pass it to the hci\_event\_handler and

//! update the event opcode in a global variable.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void SimpleLinkWaitEvent(**~~UINT16~~unsigned short** usOpcode, void \*pRetParams)

{

// In the blocking implementation the control to caller will be returned only

// after the end of current transaction

**//** tSLInformation.usRxEventOpcode = usOpcode;

hci\_event\_handler(pRetParams, 0, 0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! SimpleLinkWaitData

//!

//! @param pBuf data buffer

//! @param from from information

//! @param fromlen from information length

//!

//! @return none

//!

//! @brief Wait for data, pass it to the hci\_event\_handler

//! and update in a global variable that there is

//! data to read.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void SimpleLinkWaitData(**~~UINT8 \*pBuf, UINT8 \*from, UINT8~~unsigned char \*pBuf, unsigned char \*from, unsigned char** \*fromlen)

{

// In the blocking implementation the control to caller will be returned only

// after the end of current transaction, i.e. only after data will be received

tSLInformation.usRxDataPending = 1;

hci\_event\_handler(pBuf, from, fromlen);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **evnt\_handler.h** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_EVENT\_HANDLER\_H\_\_

#define \_\_EVENT\_HANDLER\_H\_\_

#include "hci.h"

#include "socket.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// If building with a C++ compiler, make all of the definitions in this header

// have a C binding.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

extern "C" {

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Prototypes for the APIs.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_event\_handler

//!

//! @param pRetParams incoming data buffer

//! @param from from information (in case of data received)

//! @param fromlen from information length (in case of data received)

//!

//! @return none

//!

//! @brief Parse the incoming events packets and issues corresponding

//! event handler from global array of handlers pointers

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~UINT8 \*~~unsigned char \***hci\_event\_handler(void \*pRetParams, **~~UINT8 \*from, UINT8~~unsigned char \*from, unsigned char** \*fromlen);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_unsol\_event\_handler

//!

//! @param event\_hdr event header

//!

//! @return 1 if event supported and handled

//! 0 if event is not supported

//!

//! @brief Handle unsolicited events

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~INT32~~ long** hci\_unsol\_event\_handler(**~~CHAR~~char** \*event\_hdr);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_unsolicited\_event\_handler

//!

//! @param None

//!

//! @return ESUCCESS if successful, EFAIL if an error occurred

//!

//! @brief Parse the incoming unsolicited event packets and issues

//! corresponding event handler.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern **~~INT32~~long** hci\_unsolicited\_event\_handler(void);

#define M\_BSD\_RESP\_PARAMS\_OFFSET(hci\_event\_hdr)((**~~CHAR~~char** \*)(hci\_event\_hdr) + HCI\_EVENT\_HEADER\_SIZE)

#define SOCKET\_STATUS\_ACTIVE 0

#define SOCKET\_STATUS\_INACTIVE 1

/\* Init socket\_active\_status = 'all ones': init all sockets with SOCKET\_STATUS\_INACTIVE.

Will be changed by 'set\_socket\_active\_status' upon 'connect' and 'accept' calls \*/

#define SOCKET\_STATUS\_INIT\_VAL 0xFFFF

#define M\_IS\_VALID\_SD(sd) ((0 <= (sd)) && ((sd) <= 7))

#define M\_IS\_VALID\_STATUS(status) (((status) == SOCKET\_STATUS\_ACTIVE)||((status) == SOCKET\_STATUS\_INACTIVE))

extern **~~UINT32~~unsigned long** socket\_active\_status;

extern void set\_socket\_active\_status(**~~INT32 Sd, INT32~~long Sd, long** Status);

extern **~~INT32~~ long** get\_socket\_active\_status(**~~INT32~~long** Sd);

typedef struct \_bsd\_accept\_return\_t

{

**~~INT32~~long** iSocketDescriptor;

**~~INT32~~long** iStatus;

sockaddr tSocketAddress;

} tBsdReturnParams;

typedef struct \_bsd\_read\_return\_t

{

**~~INT32~~long** iSocketDescriptor;

**~~INT32~~long** iNumberOfBytes;

**~~UINT32~~unsigned long** uiFlags;

} tBsdReadReturnParams;

#define BSD\_RECV\_FROM\_FROMLEN\_OFFSET (4)

#define BSD\_RECV\_FROM\_FROM\_OFFSET (16)

typedef struct \_bsd\_select\_return\_t

{

**~~INT32~~long** iStatus;

**~~UINT32~~unsigned long** uiRdfd;

**~~UINT32~~unsigned long** uiWrfd;

**~~UINT32~~unsigned long** uiExfd;

} tBsdSelectRecvParams;

typedef struct \_bsd\_getsockopt\_return\_t

{

**~~UINT8~~unsigned char** ucOptValue[4];

**~~CHAR~~char** iStatus;

} tBsdGetSockOptReturnParams;

typedef struct \_bsd\_gethostbyname\_return\_t

{

**~~INT32~~long** retVal;

**~~INT32~~long** outputAddress;

} tBsdGethostbynameParams;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Mark the end of the C bindings section for C++ compilers.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

}

#endif // \_\_cplusplus

#endif // \_\_EVENT\_HANDLER\_H\_\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **hci.c** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup hci\_app

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "cc3000\_common.h"

#include "hci.h"

#include "spi.h"

#include "evnt\_handler.h"

#include "wlan.h"

#define SL\_PATCH\_PORTION\_SIZE (1000)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_command\_send

//!

//! @param usOpcode command operation code

//! @param pucBuff pointer to the command's arguments buffer

//! @param ucArgsLength length of the arguments

//!

//! @return none

//!

//! @brief Initiate an HCI command.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UINT16 hci\_command\_send(UINT16 usOpcode, UINT8 \*pucBuff, UINT8 ucArgsLength)

{

// have to set this here (not after this is called, otherwise an IRQ could come before it’s set and the IRQ handler won’t know it’s the one we’re waiting on

**tSLInformation.usRxEventOpcode = usOpcode;**

UINT8 \*stream;

stream = (pucBuff + SPI\_HEADER\_SIZE);

UINT8\_TO\_STREAM(stream, HCI\_TYPE\_CMND);

stream = UINT16\_TO\_STREAM(stream, usOpcode);

UINT8\_TO\_STREAM(stream, ucArgsLength);

//Update the opcode of the event we will be waiting for

SpiWrite(pucBuff, ucArgsLength + SIMPLE\_LINK\_HCI\_CMND\_HEADER\_SIZE);

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_data\_send

//!

//! @param usOpcode command operation code

//! @param ucArgs pointer to the command's arguments buffer

//! @param usArgsLength length of the arguments

//! @param ucTail pointer to the data buffer

//! @param usTailLength buffer length

//!

//! @return none

//!

//! @brief Initiate an HCI data write operation

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 hci\_data\_send(UINT8 ucOpcode,

UINT8 \*ucArgs,

UINT16 usArgsLength,

UINT16 usDataLength,

const UINT8 \*ucTail,

UINT16 usTailLength)

{

UINT8 \*stream;

stream = ((ucArgs) + SPI\_HEADER\_SIZE);

UINT8\_TO\_STREAM(stream, HCI\_TYPE\_DATA);

UINT8\_TO\_STREAM(stream, ucOpcode);

UINT8\_TO\_STREAM(stream, usArgsLength);

stream = UINT16\_TO\_STREAM(stream, usArgsLength + usDataLength + usTailLength);

// Send the packet over the SPI

SpiWrite(ucArgs, SIMPLE\_LINK\_HCI\_DATA\_HEADER\_SIZE + usArgsLength + usDataLength + usTailLength);

return(ESUCCESS);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_data\_command\_send

//!

//! @param usOpcode command operation code

//! @param pucBuff pointer to the data buffer

//! @param ucArgsLength arguments length

//! @param ucDataLength data length

//!

//! @return none

//!

//! @brief Prepeare HCI header and initiate an HCI data write operation

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void hci\_data\_command\_send(UINT16 usOpcode, UINT8 \*pucBuff, UINT8 ucArgsLength,UINT16 ucDataLength)

{

UINT8 \*stream = (pucBuff + SPI\_HEADER\_SIZE);

UINT8\_TO\_STREAM(stream, HCI\_TYPE\_DATA);

UINT8\_TO\_STREAM(stream, usOpcode);

UINT8\_TO\_STREAM(stream, ucArgsLength);

stream = UINT16\_TO\_STREAM(stream, ucArgsLength + ucDataLength);

// Send the command over SPI on data channel

SpiWrite(pucBuff, ucArgsLength + ucDataLength + SIMPLE\_LINK\_HCI\_DATA\_CMND\_HEADER\_SIZE);

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! hci\_patch\_send

//!

//! @param usOpcode command operation code

//! @param pucBuff pointer to the command's arguments buffer

//! @param patch pointer to patch content buffer

//! @param usDataLength data length

//!

//! @return none

//!

//! @brief Prepeare HCI header and initiate an HCI patch write operation

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void hci\_patch\_send(UINT8 ucOpcode, UINT8 \*pucBuff, CHAR \*patch, UINT16 usDataLength)

{

UINT8 \*data\_ptr = (pucBuff + SPI\_HEADER\_SIZE);

UINT16 usTransLength;

UINT8 \*stream = (pucBuff + SPI\_HEADER\_SIZE);

UINT8\_TO\_STREAM(stream, HCI\_TYPE\_PATCH);

UINT8\_TO\_STREAM(stream, ucOpcode);

stream = UINT16\_TO\_STREAM(stream, usDataLength + SIMPLE\_LINK\_HCI\_PATCH\_HEADER\_SIZE);

if (usDataLength <= SL\_PATCH\_PORTION\_SIZE)

{

UINT16\_TO\_STREAM(stream, usDataLength);

stream = UINT16\_TO\_STREAM(stream, usDataLength);

memcpy((pucBuff + SPI\_HEADER\_SIZE) + HCI\_PATCH\_HEADER\_SIZE, patch, usDataLength);

// Update the opcode of the event we will be waiting for

SpiWrite(pucBuff, usDataLength + HCI\_PATCH\_HEADER\_SIZE);

}

else

{

usTransLength = (usDataLength/SL\_PATCH\_PORTION\_SIZE);

UINT16\_TO\_STREAM(stream, usDataLength + SIMPLE\_LINK\_HCI\_PATCH\_HEADER\_SIZE + usTransLength\*SIMPLE\_LINK\_HCI\_PATCH\_HEADER\_SIZE);

stream = UINT16\_TO\_STREAM(stream, SL\_PATCH\_PORTION\_SIZE);

memcpy(pucBuff + SPI\_HEADER\_SIZE + HCI\_PATCH\_HEADER\_SIZE, patch, SL\_PATCH\_PORTION\_SIZE);

usDataLength -= SL\_PATCH\_PORTION\_SIZE;

patch += SL\_PATCH\_PORTION\_SIZE;

// Update the opcode of the event we will be waiting for

SpiWrite(pucBuff, SL\_PATCH\_PORTION\_SIZE + HCI\_PATCH\_HEADER\_SIZE);

while (usDataLength)

{

if (usDataLength <= SL\_PATCH\_PORTION\_SIZE)

{

usTransLength = usDataLength;

usDataLength = 0;

}

else

{

usTransLength = SL\_PATCH\_PORTION\_SIZE;

usDataLength -= usTransLength;

}

\*(UINT16 \*)data\_ptr = usTransLength;

memcpy(data\_ptr + SIMPLE\_LINK\_HCI\_PATCH\_HEADER\_SIZE, patch, usTransLength);

patch += usTransLength;

// Update the opcode of the event we will be waiting for

SpiWrite((UINT8 \*)data\_ptr, usTransLength + sizeof(usTransLength));

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **nvmem.c** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup nvmem\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <string.h>

#include "nvmem.h"

#include "hci.h"

#include "socket.h"

#include "evnt\_handler.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Prototypes for the structures for APIs.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define NVMEM\_READ\_PARAMS\_LEN (12)

#define NVMEM\_CREATE\_PARAMS\_LEN (8)

#define NVMEM\_WRITE\_PARAMS\_LEN (16)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_read

//!

//! @param ulFileId nvmem file id:\n

//! NVMEM\_NVS\_FILEID, NVMEM\_NVS\_SHADOW\_FILEID,

//! NVMEM\_WLAN\_CONFIG\_FILEID, NVMEM\_WLAN\_CONFIG\_SHADOW\_FILEID,

//! NVMEM\_WLAN\_DRIVER\_SP\_FILEID, NVMEM\_WLAN\_FW\_SP\_FILEID,

//! NVMEM\_MAC\_FILEID, NVMEM\_FRONTEND\_VARS\_FILEID,

//! NVMEM\_IP\_CONFIG\_FILEID, NVMEM\_IP\_CONFIG\_SHADOW\_FILEID,

//! NVMEM\_BOOTLOADER\_SP\_FILEID, NVMEM\_RM\_FILEID,

//! and user files 12-15.

//! @param ulLength number of bytes to read

//! @param ulOffset ulOffset in file from where to read

//! @param buff output buffer pointer

//!

//! @return on success 0, error otherwise.

//!

//! @brief Reads data from the file referred by the ulFileId parameter.

//! Reads data from file ulOffset till length. Err if the file can't

//! be used, is invalid, or if the read is out of bounds.

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 nvmem\_read(UINT32 ulFileId, UINT32 ulLength, UINT32 ulOffset, UINT8 \*buff)

{

UINT8 ucStatus = 0xFF;

UINT8 \*ptr;

UINT8 \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, ulFileId);

args = UINT32\_TO\_STREAM(args, ulLength);

args = UINT32\_TO\_STREAM(args, ulOffset);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_NVMEM\_READ, ptr, NVMEM\_READ\_PARAMS\_LEN);

SimpleLinkWaitEvent(HCI\_CMND\_NVMEM\_READ, &ucStatus);

// In case there is data - read it - even if an error code is returned

// Note: It is the user responsibility to ignore the data in case of an error code

// Wait for the data in a synchronous way. Here we assume that the buffer is

// big enough to store also parameters of nvmem

SimpleLinkWaitData(buff, 0, 0);

return(ucStatus);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_write

//!

//! @param ulFileId nvmem file id:\n

//! NVMEM\_WLAN\_DRIVER\_SP\_FILEID, NVMEM\_WLAN\_FW\_SP\_FILEID,

//! NVMEM\_MAC\_FILEID, NVMEM\_BOOTLOADER\_SP\_FILEID,

//! and user files 12-15.

//! @param ulLength number of bytes to write

//! @param ulEntryOffset offset in file to start write operation from

//! @param buff data to write

//!

//! @return on success 0, error otherwise.

//!

//! @brief Write data to nvmem.

//! writes data to file referred by the ulFileId parameter.

//! Writes data to file ulOffset till ulLength.The file id will be

//! marked invalid till the write is done. The file entry doesn't

//! need to be valid - only allocated.

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 nvmem\_write(UINT32 ulFileId, UINT32 ulLength, UINT32 ulEntryOffset, UINT8 \*buff)

{

INT32 iRes;

UINT8 \*ptr;

UINT8 \*args;

iRes = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + SPI\_HEADER\_SIZE + HCI\_DATA\_CMD\_HEADER\_SIZE);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, ulFileId);

args = UINT32\_TO\_STREAM(args, 12);

args = UINT32\_TO\_STREAM(args, ulLength);

args = UINT32\_TO\_STREAM(args, ulEntryOffset);

memcpy((ptr + SPI\_HEADER\_SIZE + HCI\_DATA\_CMD\_HEADER\_SIZE +

NVMEM\_WRITE\_PARAMS\_LEN),buff,ulLength);

// have to set this here (not after this is called, otherwise an IRQ could come before it’s set and the IRQ handler won’t know it’s the one we’re waiting on

**tSLInformation.usRxEventOpcode = HCI\_EVNT\_NVMEM\_WRITE;**

// Initiate a HCI command but it will come on data channel

hci\_data\_command\_send(HCI\_CMND\_NVMEM\_WRITE, ptr, NVMEM\_WRITE\_PARAMS\_LEN,

ulLength);

SimpleLinkWaitEvent(HCI\_EVNT\_NVMEM\_WRITE, &iRes);

return(iRes);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_set\_mac\_address

//!

//! @param mac mac address to be set

//!

//! @return on success 0, error otherwise.

//!

//! @brief Write MAC address to EEPROM.

//! mac address as appears over the air (OUI first)

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UINT8 nvmem\_set\_mac\_address(UINT8 \*mac)

{

return nvmem\_write(NVMEM\_MAC\_FILEID, MAC\_ADDR\_LEN, 0, mac);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_get\_mac\_address

//!

//! @param[out] mac mac address

//!

//! @return on success 0, error otherwise.

//!

//! @brief Read MAC address from EEPROM.

//! mac address as appears over the air (OUI first)

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UINT8 nvmem\_get\_mac\_address(UINT8 \*mac)

{

return nvmem\_read(NVMEM\_MAC\_FILEID, MAC\_ADDR\_LEN, 0, mac);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_write\_patch

//!

//! @param ulFileId nvmem file id:\n

//! NVMEM\_WLAN\_DRIVER\_SP\_FILEID, NVMEM\_WLAN\_FW\_SP\_FILEID,

//! @param spLength number of bytes to write

//! @param spData SP data to write

//!

//! @return on success 0, error otherwise.

//!

//! @brief program a patch to a specific file ID.

//! The SP data is assumed to be organized in 2-dimensional.

//! Each line is SP\_PORTION\_SIZE bytes long. Actual programming is

//! applied in SP\_PORTION\_SIZE bytes portions.

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UINT8 nvmem\_write\_patch(UINT32 ulFileId, UINT32 spLength, const UINT8 \*spData)

{

UINT8 status = 0;

UINT16 offset = 0;

UINT8\* spDataPtr = (UINT8\*)spData;

while ((status == 0) && (spLength >= SP\_PORTION\_SIZE))

{

status = nvmem\_write(ulFileId, SP\_PORTION\_SIZE, offset, spDataPtr);

offset += SP\_PORTION\_SIZE;

spLength -= SP\_PORTION\_SIZE;

spDataPtr += SP\_PORTION\_SIZE;

}

if (status !=0)

{

// NVMEM error occurred

return status;

}

if (spLength != 0)

{

// if reached here, a reminder is left

status = nvmem\_write(ulFileId, spLength, offset, spDataPtr);

}

return status;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_read\_sp\_version

//!

//! @param[out] patchVer first number indicates package ID and the second

//! number indicates package build number

//!

//! @return on success 0, error otherwise.

//!

//! @brief Read patch version. read package version (WiFi FW patch,

//! driver-supplicant-NS patch, bootloader patch)

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

UINT8 nvmem\_read\_sp\_version(UINT8\* patchVer)

{

UINT8 \*ptr;

// 1st byte is the status and the rest is the SP version

UINT8 retBuf[5];

ptr = tSLInformation.pucTxCommandBuffer;

// Initiate a HCI command, no args are required

hci\_command\_send(HCI\_CMND\_READ\_SP\_VERSION, ptr, 0);

SimpleLinkWaitEvent(HCI\_CMND\_READ\_SP\_VERSION, retBuf);

// package ID

\*patchVer = retBuf[3];

// package build number

\*(patchVer+1) = retBuf[4];

return(retBuf[0]);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! nvmem\_create\_entry

//!

//! @param ulFileId nvmem file Id:\n

//! \* NVMEM\_AES128\_KEY\_FILEID: 12

//! \* NVMEM\_SHARED\_MEM\_FILEID: 13

//! \* and fileIDs 14 and 15

//! @param ulNewLen entry ulLength

//!

//! @return on success 0, error otherwise.

//!

//! @brief Create new file entry and allocate space on the NVMEM.

//! Applies only to user files.

//! Modify the size of file.

//! If the entry is unallocated - allocate it to size

//! ulNewLen (marked invalid).

//! If it is allocated then deallocate it first.

//! To just mark the file as invalid without resizing -

//! set ulNewLen=0.

//!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 nvmem\_create\_entry(UINT32 ulFileId, UINT32 ulNewLen)

{

UINT8 \*ptr;

UINT8 \*args;

UINT8 retval;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, ulFileId);

args = UINT32\_TO\_STREAM(args, ulNewLen);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_NVMEM\_CREATE\_ENTRY,ptr, NVMEM\_CREATE\_PARAMS\_LEN);

SimpleLinkWaitEvent(HCI\_CMND\_NVMEM\_CREATE\_ENTRY, &retval);

return(retval);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **socket.c** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup socket\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "hci.h"

#include "socket.h"

#include "evnt\_handler.h"

#include "netapp.h"

//Enable this flag if and only if you must comply with BSD socket

//close() function

#ifdef \_API\_USE\_BSD\_CLOSE

#define close(sd) closesocket(sd)

#endif

//Enable this flag if and only if you must comply with BSD socket read() and

//write() functions

#ifdef \_API\_USE\_BSD\_READ\_WRITE

#define read(sd, buf, len, flags) recv(sd, buf, len, flags)

#define write(sd, buf, len, flags) send(sd, buf, len, flags)

#endif

#define SOCKET\_OPEN\_PARAMS\_LEN (12)

#define SOCKET\_CLOSE\_PARAMS\_LEN (4)

#define SOCKET\_ACCEPT\_PARAMS\_LEN (4)

#define SOCKET\_BIND\_PARAMS\_LEN (20)

#define SOCKET\_LISTEN\_PARAMS\_LEN (8)

#define SOCKET\_GET\_HOST\_BY\_NAME\_PARAMS\_LEN (9)

#define SOCKET\_CONNECT\_PARAMS\_LEN (20)

#define SOCKET\_SELECT\_PARAMS\_LEN (44)

#define SOCKET\_SET\_SOCK\_OPT\_PARAMS\_LEN (20)

#define SOCKET\_GET\_SOCK\_OPT\_PARAMS\_LEN (12)

#define SOCKET\_RECV\_FROM\_PARAMS\_LEN (12)

#define SOCKET\_SENDTO\_PARAMS\_LEN (24)

#define SOCKET\_MDNS\_ADVERTISE\_PARAMS\_LEN (12)

#define SOCKET\_GET\_MSS\_VALUE\_PARAMS\_LEN (4)

// The legnth of arguments for the SEND command: sd + buff\_offset + len + flags,

// while size of each parameter is 32 bit - so the total length is 16 bytes;

#define HCI\_CMND\_SEND\_ARG\_LENGTH (16)

#define SELECT\_TIMEOUT\_MIN\_MICRO\_SECONDS 5000

#define HEADERS\_SIZE\_DATA (SPI\_HEADER\_SIZE + 5)

#define SIMPLE\_LINK\_HCI\_CMND\_TRANSPORT\_HEADER\_SIZE (SPI\_HEADER\_SIZE + SIMPLE\_LINK\_HCI\_CMND\_HEADER\_SIZE)

#define MDNS\_DEVICE\_SERVICE\_MAX\_LENGTH (32)

#ifdef MDNS\_ADVERTISE\_HOST

extern UINT8 localIP[4];

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! HostFlowControlConsumeBuff

//!

//! @param sd socket descriptor

//!

//! @return 0 in case there are buffers available,

//! -1 in case of bad socket

//! -2 if there are no free buffers present (only when

//! SEND\_NON\_BLOCKING is enabled)

//!

//! @brief if SEND\_NON\_BLOCKING not define - block until have free buffer

//! becomes available, else return immediately with correct status

//! regarding the buffers available.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 HostFlowControlConsumeBuff(INT16 sd)

{

#ifndef SEND\_NON\_BLOCKING

/\* wait in busy loop \*/

do

{

// In case last transmission failed then we will return the last failure

// reason here.

// Note that the buffer will not be allocated in this case

if (tSLInformation.slTransmitDataError != 0)

{

errno = tSLInformation.slTransmitDataError;

tSLInformation.slTransmitDataError = 0;

return errno;

}

if(SOCKET\_STATUS\_ACTIVE != get\_socket\_active\_status(sd))

return -1;

} while(0 == tSLInformation.usNumberOfFreeBuffers);

tSLInformation.usNumberOfFreeBuffers--;

return 0;

#else

// In case last transmission failed then we will return the last failure

// reason here.

// Note that the buffer will not be allocated in this case

if (tSLInformation.slTransmitDataError != 0)

{

errno = tSLInformation.slTransmitDataError;

tSLInformation.slTransmitDataError = 0;

return errno;

}

if(SOCKET\_STATUS\_ACTIVE != get\_socket\_active\_status(sd))

return -1;

//If there are no available buffers, return -2. It is recommended to use

// select or receive to see if there is any buffer occupied with received data

// If so, call receive() to release the buffer.

if(0 == tSLInformation.usNumberOfFreeBuffers)

{

return -2;

}

else

{

tSLInformation.usNumberOfFreeBuffers--;

return 0;

}

#endif

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! socket

//!

//! @param domain selects the protocol family which will be used for

//! communication. On this version only AF\_INET is supported

//! @param type specifies the communication semantics. On this version

//! only SOCK\_STREAM, SOCK\_DGRAM, SOCK\_RAW are supported

//! @param protocol specifies a particular protocol to be used with the

//! socket IPPROTO\_TCP, IPPROTO\_UDP or IPPROTO\_RAW are

//! supported.

//!

//! @return On success, socket handle that is used for consequent socket

//! operations. On error, -1 is returned.

//!

//! @brief create an endpoint for communication

//! The socket function creates a socket that is bound to a specific

//! transport service provider. This function is called by the

//! application layer to obtain a socket handle.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 socket(INT32 domain, INT32 type, INT32 protocol)

{

INT32 ret;

UINT8 \*ptr, \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, domain);

args = UINT32\_TO\_STREAM(args, type);

args = UINT32\_TO\_STREAM(args, protocol);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_SOCKET, ptr, SOCKET\_OPEN\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_SOCKET, &ret);

// Process the event

errno = ret;

set\_socket\_active\_status(ret, SOCKET\_STATUS\_ACTIVE);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! closesocket

//!

//! @param sd socket handle.

//!

//! @return On success, zero is returned. On error, -1 is returned.

//!

//! @brief The socket function closes a created socket.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 closesocket(INT32 sd)

{

INT32 ret;

UINT8 \*ptr, \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, sd);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_CLOSE\_SOCKET,

ptr, SOCKET\_CLOSE\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_CLOSE\_SOCKET, &ret);

errno = ret;

// since 'close' call may result in either OK (and then it closed) or error

// mark this socket as invalid

set\_socket\_active\_status(sd, SOCKET\_STATUS\_INACTIVE);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! accept

//!

//! @param[in] sd socket descriptor (handle)

//! @param[out] addr the argument addr is a pointer to a sockaddr structure

//! This structure is filled in with the address of the

//! peer socket, as known to the communications layer.

//! determined. The exact format of the address returned

//! addr is by the socket's address sockaddr.

//! On this version only AF\_INET is supported.

//! This argument returns in network order.

//! @param[out] addrlen the addrlen argument is a value-result argument:

//! it should initially contain the size of the structure

//! pointed to by addr.

//!

//! @return For socket in blocking mode:

//! On success, socket handle. on failure negative

//! For socket in non-blocking mode:

//! - On connection establishment, socket handle

//! - On connection pending, SOC\_IN\_PROGRESS (-2)

//! - On failure, SOC\_ERROR (-1)

//!

//! @brief accept a connection on a socket:

//! This function is used with connection-based socket types

//! (SOCK\_STREAM). It extracts the first connection request on the

//! queue of pending connections, creates a new connected socket, and

//! returns a new file descriptor referring to that socket.

//! The newly created socket is not in the listening state.

//! The original socket sd is unaffected by this call.

//! The argument sd is a socket that has been created with socket(),

//! bound to a local address with bind(), and is listening for

//! connections after a listen(). The argument addr is a pointer

//! to a sockaddr structure. This structure is filled in with the

//! address of the peer socket, as known to the communications layer.

//! The exact format of the address returned addr is determined by the

//! socket's address family. The addrlen argument is a value-result

//! argument: it should initially contain the size of the structure

//! pointed to by addr, on return it will contain the actual

//! length (in bytes) of the address returned.

//!

//! @sa socket ; bind ; listen

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 accept(INT32 sd, sockaddr \*addr, socklen\_t \*addrlen)

{

INT32 ret;

UINT8 \*ptr, \*args;

tBsdReturnParams tAcceptReturnArguments;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_ACCEPT,

ptr, SOCKET\_ACCEPT\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_ACCEPT, &tAcceptReturnArguments);

// need specify return parameters!!!

memcpy(addr, &tAcceptReturnArguments.tSocketAddress, ASIC\_ADDR\_LEN);

\*addrlen = ASIC\_ADDR\_LEN;

errno = tAcceptReturnArguments.iStatus;

ret = errno;

// if succeeded, iStatus = new socket descriptor. otherwise - error number

if(M\_IS\_VALID\_SD(ret))

{

set\_socket\_active\_status(ret, SOCKET\_STATUS\_ACTIVE);

}

else

{

set\_socket\_active\_status(sd, SOCKET\_STATUS\_INACTIVE);

}

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! bind

//!

//! @param[in] sd socket descriptor (handle)

//! @param[out] addr specifies the destination address. On this version

//! only AF\_INET is supported.

//! @param[out] addrlen contains the size of the structure pointed to by addr.

//!

//! @return On success, zero is returned. On error, -1 is returned.

//!

//! @brief assign a name to a socket

//! This function gives the socket the local address addr.

//! addr is addrlen bytes long. Traditionally, this is called when a

//! socket is created with socket, it exists in a name space (address

//! family) but has no name assigned.

//! It is necessary to assign a local address before a SOCK\_STREAM

//! socket may receive connections.

//!

//! @sa socket ; accept ; listen

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 bind(INT32 sd, const sockaddr \*addr, INT32 addrlen)

{

INT32 ret;

UINT8 \*ptr, \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

addrlen = ASIC\_ADDR\_LEN;

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, 0x00000008);

args = UINT32\_TO\_STREAM(args, addrlen);

ARRAY\_TO\_STREAM(args, ((UINT8 \*)addr), addrlen);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_BIND,

ptr, SOCKET\_BIND\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_BIND, &ret);

errno = ret;

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! listen

//!

//! @param[in] sd socket descriptor (handle)

//! @param[in] backlog specifies the listen queue depth. On this version

//! backlog is not supported.

//! @return On success, zero is returned. On error, -1 is returned.

//!

//! @brief listen for connections on a socket

//! The willingness to accept incoming connections and a queue

//! limit for incoming connections are specified with listen(),

//! and then the connections are accepted with accept.

//! The listen() call applies only to sockets of type SOCK\_STREAM

//! The backlog parameter defines the maximum length the queue of

//! pending connections may grow to.

//!

//! @sa socket ; accept ; bind

//!

//! @note On this version, backlog is not supported

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 listen(INT32 sd, INT32 backlog)

{

INT32 ret;

UINT8 \*ptr, \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, backlog);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_LISTEN,

ptr, SOCKET\_LISTEN\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_LISTEN, &ret);

errno = ret;

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! gethostbyname

//!

//! @param[in] hostname host name

//! @param[in] usNameLen name length

//! @param[out] out\_ip\_addr This parameter is filled in with host IP address.

//! In case that host name is not resolved,

//! out\_ip\_addr is zero.

//! @return On success, positive is returned. On error, negative is returned

//!

//! @brief Get host IP by name. Obtain the IP Address of machine on network,

//! by its name.

//!

//! @note On this version, only blocking mode is supported. Also note that

//! the function requires DNS server to be configured prior to its usage.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT16 gethostbyname(CHAR \* hostname, UINT16 usNameLen,

UINT32\* out\_ip\_addr)

{

tBsdGethostbynameParams ret;

UINT8 \*ptr, \*args;

errno = EFAIL;

if (usNameLen > HOSTNAME\_MAX\_LENGTH)

{

return errno;

}

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + SIMPLE\_LINK\_HCI\_CMND\_TRANSPORT\_HEADER\_SIZE);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, 8);

args = UINT32\_TO\_STREAM(args, usNameLen);

ARRAY\_TO\_STREAM(args, hostname, usNameLen);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_GETHOSTNAME, ptr, SOCKET\_GET\_HOST\_BY\_NAME\_PARAMS\_LEN

+ usNameLen - 1);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_EVNT\_BSD\_GETHOSTBYNAME, &ret);

errno = ret.retVal;

(\*((INT32\*)out\_ip\_addr)) = ret.outputAddress;

return (errno);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! connect

//!

//! @param[in] sd socket descriptor (handle)

//! @param[in] addr specifies the destination addr. On this version

//! only AF\_INET is supported.

//! @param[out] addrlen contains the size of the structure pointed to by addr

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief initiate a connection on a socket

//! Function connects the socket referred to by the socket descriptor

//! sd, to the address specified by addr. The addrlen argument

//! specifies the size of addr. The format of the address in addr is

//! determined by the address space of the socket. If it is of type

//! SOCK\_DGRAM, this call specifies the peer with which the socket is

//! to be associated; this address is that to which datagrams are to be

//! sent, and the only address from which datagrams are to be received.

//! If the socket is of type SOCK\_STREAM, this call attempts to make a

//! connection to another socket. The other socket is specified by

//! address, which is an address in the communications space of the

//! socket. Note that the function implements only blocking behavior

//! thus the caller will be waiting either for the connection

//! establishment or for the connection establishment failure.

//!

//! @sa socket

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 connect(INT32 sd, const sockaddr \*addr, INT32 addrlen)

{

INT32 ret;

UINT8 \*ptr, \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + SIMPLE\_LINK\_HCI\_CMND\_TRANSPORT\_HEADER\_SIZE);

addrlen = 8;

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, 0x00000008);

args = UINT32\_TO\_STREAM(args, addrlen);

ARRAY\_TO\_STREAM(args, ((UINT8 \*)addr), addrlen);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_CONNECT,

ptr, SOCKET\_CONNECT\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_CONNECT, &ret);

errno = ret;

return((INT32)ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! select

//!

//! @param[in] nfds the highest-numbered file descriptor in any of the

//! three sets, plus 1.

//! @param[out] writesds socket descriptors list for write monitoring

//! @param[out] readsds socket descriptors list for read monitoring

//! @param[out] exceptsds socket descriptors list for exception monitoring

//! @param[in] timeout is an upper bound on the amount of time elapsed

//! before select() returns. Null means infinity

//! timeout. The minimum timeout is 5 milliseconds,

//! less than 5 milliseconds will be set

//! automatically to 5 milliseconds.

//! @return On success, select() returns the number of file descriptors

//! contained in the three returned descriptor sets (that is, the

//! total number of bits that are set in readfds, writefds,

//! exceptfds) which may be zero if the timeout expires before

//! anything interesting happens.

//! On error, -1 is returned.

//! \*readsds - return the sockets on which Read request will

//! return without delay with valid data.

//! \*writesds - return the sockets on which Write request

//! will return without delay.

//! \*exceptsds - return the sockets which closed recently.

//!

//! @brief Monitor socket activity

//! Select allow a program to monitor multiple file descriptors,

//! waiting until one or more of the file descriptors become

//! "ready" for some class of I/O operation

//!

//! @Note If the timeout value set to less than 5ms it will automatically set

//! to 5ms to prevent overload of the system

//!

//! @sa socket

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 select(INT32 nfds, fd\_set \*readsds, fd\_set \*writesds, fd\_set \*exceptsds,

struct timeval \*timeout)

{

UINT8 \*ptr, \*args;

tBsdSelectRecvParams tParams;

UINT32 is\_blocking;

if( timeout == NULL)

{

is\_blocking = 1; /\* blocking , infinity timeout \*/

}

else

{

is\_blocking = 0; /\* no blocking, timeout \*/

}

// Fill in HCI packet structure

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, nfds);

args = UINT32\_TO\_STREAM(args, 0x00000014);

args = UINT32\_TO\_STREAM(args, 0x00000014);

args = UINT32\_TO\_STREAM(args, 0x00000014);

args = UINT32\_TO\_STREAM(args, 0x00000014);

args = UINT32\_TO\_STREAM(args, is\_blocking);

args = UINT32\_TO\_STREAM(args, ((readsds) ? \*(UINT32\*)readsds : 0));

args = UINT32\_TO\_STREAM(args, ((writesds) ? \*(UINT32\*)writesds : 0));

args = UINT32\_TO\_STREAM(args, ((exceptsds) ? \*(UINT32\*)exceptsds : 0));

if (timeout)

{

if ( 0 == timeout->tv\_sec && timeout->tv\_usec <

SELECT\_TIMEOUT\_MIN\_MICRO\_SECONDS)

{

timeout->tv\_usec = SELECT\_TIMEOUT\_MIN\_MICRO\_SECONDS;

}

args = UINT32\_TO\_STREAM(args, timeout->tv\_sec);

args = UINT32\_TO\_STREAM(args, timeout->tv\_usec);

}

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_BSD\_SELECT, ptr, SOCKET\_SELECT\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_EVNT\_SELECT, &tParams);

// Update actually read FD

if (tParams.iStatus >= 0)

{

if (readsds)

{

memcpy(readsds, &tParams.uiRdfd, sizeof(tParams.uiRdfd));

}

if (writesds)

{

memcpy(writesds, &tParams.uiWrfd, sizeof(tParams.uiWrfd));

}

if (exceptsds)

{

memcpy(exceptsds, &tParams.uiExfd, sizeof(tParams.uiExfd));

}

return(tParams.iStatus);

}

else

{

errno = tParams.iStatus;

return(-1);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! setsockopt

//!

//! @param[in] sd socket handle

//! @param[in] level defines the protocol level for this option

//! @param[in] optname defines the option name to Interrogate

//! @param[in] optval specifies a value for the option

//! @param[in] optlen specifies the length of the option value

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief set socket options

//! This function manipulate the options associated with a socket.

//! Options may exist at multiple protocol levels; they are always

//! present at the uppermost socket level.

//! When manipulating socket options the level at which the option

//! resides and the name of the option must be specified.

//! To manipulate options at the socket level, level is specified as

//! SOL\_SOCKET. To manipulate options at any other level the protocol

//! number of the appropriate protocol controlling the option is

//! supplied. For example, to indicate that an option is to be

//! interpreted by the TCP protocol, level should be set to the

//! protocol number of TCP;

//! The parameters optval and optlen are used to access optval -

//! use for setsockopt(). For getsockopt() they identify a buffer

//! in which the value for the requested option(s) are to

//! be returned. For getsockopt(), optlen is a value-result

//! parameter, initially containing the size of the buffer

//! pointed to by option\_value, and modified on return to

//! indicate the actual size of the value returned. If no option

//! value is to be supplied or returned, option\_value may be NULL.

//!

//! @Note On this version the following two socket options are enabled:

//! The only protocol level supported in this version

//! is SOL\_SOCKET (level).

//! 1. SOCKOPT\_RECV\_NONBLOCK (optname)

//! SOCKOPT\_RECV\_NONBLOCK sets the recv and recvfrom

//! non-blocking modes on or off.

//! In that case optval should be SOCK\_ON or SOCK\_OFF (optval).

//!

//! 2. SOCKOPT\_RECV\_TIMEOUT (optname)

//! SOCKOPT\_RECV\_TIMEOUT configures recv and recvfrom timeout

//! in milliseconds.

//! In that case optval should be pointer to UINT32.

//!

//! 3. SOCKOPT\_ACCEPT\_NONBLOCK (optname). sets the socket accept

//! non-blocking mode on or off.

//! In that case optval should be SOCK\_ON or SOCK\_OFF (optval).

//!

//! @sa getsockopt

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT16 setsockopt(INT32 sd, INT32 level, INT32 optname, const void \*optval,

socklen\_t optlen)

{

INT32 ret;

UINT8 \*ptr, \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, level);

args = UINT32\_TO\_STREAM(args, optname);

args = UINT32\_TO\_STREAM(args, 0x00000008);

args = UINT32\_TO\_STREAM(args, optlen);

ARRAY\_TO\_STREAM(args, ((UINT8 \*)optval), optlen);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_SETSOCKOPT,

ptr, SOCKET\_SET\_SOCK\_OPT\_PARAMS\_LEN + optlen);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_SETSOCKOPT, &ret);

if (ret >= 0)

{

return (0);

}

else

{

errno = ret;

return ret;

}

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! getsockopt

//!

//! @param[in] sd socket handle

//! @param[in] level defines the protocol level for this option

//! @param[in] optname defines the option name to Interrogate

//! @param[out] optval specifies a value for the option

//! @param[out] optlen specifies the length of the option value

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief set socket options

//! This function manipulate the options associated with a socket.

//! Options may exist at multiple protocol levels; they are always

//! present at the uppermost socket level.

//! When manipulating socket options the level at which the option

//! resides and the name of the option must be specified.

//! To manipulate options at the socket level, level is specified as

//! SOL\_SOCKET. To manipulate options at any other level the protocol

//! number of the appropriate protocol controlling the option is

//! supplied. For example, to indicate that an option is to be

//! interpreted by the TCP protocol, level should be set to the

//! protocol number of TCP;

//! The parameters optval and optlen are used to access optval -

//! use for setsockopt(). For getsockopt() they identify a buffer

//! in which the value for the requested option(s) are to

//! be returned. For getsockopt(), optlen is a value-result

//! parameter, initially containing the size of the buffer

//! pointed to by option\_value, and modified on return to

//! indicate the actual size of the value returned. If no option

//! value is to be supplied or returned, option\_value may be NULL.

//!

//! @Note On this version the following two socket options are enabled:

//! The only protocol level supported in this version

//! is SOL\_SOCKET (level).

//! 1. SOCKOPT\_RECV\_NONBLOCK (optname)

//! SOCKOPT\_RECV\_NONBLOCK sets the recv and recvfrom

//! non-blocking modes on or off.

//! In that case optval should be SOCK\_ON or SOCK\_OFF (optval).

//!

//! 2. SOCKOPT\_RECV\_TIMEOUT (optname)

//! SOCKOPT\_RECV\_TIMEOUT configures recv and recvfrom timeout

//! in milliseconds.

//! In that case optval should be pointer to UINT32.

//!

//! 3. SOCKOPT\_ACCEPT\_NONBLOCK (optname). sets the socket accept

//! non-blocking mode on or off.

//! In that case optval should be SOCK\_ON or SOCK\_OFF (optval).

//!

//! @sa setsockopt

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 getsockopt (INT32 sd, INT32 level, INT32 optname, void \*optval, socklen\_t \*optlen)

{

UINT8 \*ptr, \*args;

tBsdGetSockOptReturnParams tRetParams;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, level);

args = UINT32\_TO\_STREAM(args, optname);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_GETSOCKOPT,

ptr, SOCKET\_GET\_SOCK\_OPT\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_CMND\_GETSOCKOPT, &tRetParams);

if (((INT8)tRetParams.iStatus) >= 0)

{

\*optlen = 4;

memcpy(optval, tRetParams.ucOptValue, 4);

return (0);

}

else

{

errno = tRetParams.iStatus;

return errno;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! simple\_link\_recv

//!

//! @param sd socket handle

//! @param buf read buffer

//! @param len buffer length

//! @param flags indicates blocking or non-blocking operation

//! @param from pointer to an address structure indicating source address

//! @param fromlen source address structure size

//!

//! @return Return the number of bytes received, or -1 if an error

//! occurred

//!

//! @brief Read data from socket

//! Return the length of the message on successful completion.

//! If a message is too long to fit in the supplied buffer,

//! excess bytes may be discarded depending on the type of

//! socket the message is received from

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 simple\_link\_recv(INT32 sd, void \*buf, INT32 len, INT32 flags, sockaddr \*from,

socklen\_t \*fromlen, INT32 opcode)

{

UINT8 \*ptr, \*args;

tBsdReadReturnParams tSocketReadEvent;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, len);

args = UINT32\_TO\_STREAM(args, flags);

// Generate the read command, and wait for the

hci\_command\_send(opcode, ptr, SOCKET\_RECV\_FROM\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(opcode, &tSocketReadEvent);

// In case the number of bytes is more then zero - read data

if (tSocketReadEvent.iNumberOfBytes > 0)

{

// Wait for the data in a synchronous way. Here we assume that the bug is

// big enough to store also parameters of receive from too....

SimpleLinkWaitData(buf, (UINT8 \*)from, (UINT8 \*)fromlen);

}

errno = tSocketReadEvent.iNumberOfBytes;

return(tSocketReadEvent.iNumberOfBytes);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! recv

//!

//! @param[in] sd socket handle

//! @param[out] buf Points to the buffer where the message should be stored

//! @param[in] len Specifies the length in bytes of the buffer pointed to

//! by the buffer argument.

//! @param[in] flags Specifies the type of message reception.

//! On this version, this parameter is not supported.

//!

//! @return Return the number of bytes received, or -1 if an error

//! occurred

//!

//! @brief function receives a message from a connection-mode socket

//!

//! @sa recvfrom

//!

//! @Note On this version, only blocking mode is supported.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 recv(INT32 sd, void \*buf, INT32 len, INT32 flags)

{

return(simple\_link\_recv(sd, buf, len, flags, NULL, NULL, HCI\_CMND\_RECV));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! recvfrom

//!

//! @param[in] sd socket handle

//! @param[out] buf Points to the buffer where the message should be stored

//! @param[in] len Specifies the length in bytes of the buffer pointed to

//! by the buffer argument.

//! @param[in] flags Specifies the type of message reception.

//! On this version, this parameter is not supported.

//! @param[in] from pointer to an address structure indicating the source

//! address: sockaddr. On this version only AF\_INET is

//! supported.

//! @param[in] fromlen source address tructure size

//!

//! @return Return the number of bytes received, or -1 if an error

//! occurred

//!

//! @brief read data from socket

//! function receives a message from a connection-mode or

//! connectionless-mode socket. Note that raw sockets are not

//! supported.

//!

//! @sa recv

//!

//! @Note On this version, only blocking mode is supported.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 recvfrom(INT32 sd, void \*buf, INT32 len, INT32 flags, sockaddr \*from,

socklen\_t \*fromlen)

{

return(simple\_link\_recv(sd, buf, len, flags, from, fromlen,

HCI\_CMND\_RECVFROM));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! simple\_link\_send

//!

//! @param sd socket handle

//! @param buf write buffer

//! @param len buffer length

//! @param flags On this version, this parameter is not supported

//! @param to pointer to an address structure indicating destination

//! address

//! @param tolen destination address structure size

//!

//! @return Return the number of bytes transmitted, or -1 if an error

//! occurred, or -2 in case there are no free buffers available

//! (only when SEND\_NON\_BLOCKING is enabled)

//!

//! @brief This function is used to transmit a message to another

//! socket

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 simple\_link\_send(INT32 sd, const void \*buf, INT32 len, INT32 flags,

const sockaddr \*to, INT32 tolen, INT32 opcode)

{

UINT8 uArgSize, addrlen;

UINT8 \*ptr, \*pDataPtr, \*args;

UINT32 addr\_offset;

INT16 res;

tBsdReadReturnParams tSocketSendEvent;

// Check the bsd\_arguments

if (0 != (res = HostFlowControlConsumeBuff(sd)))

{

return res;

}

//Update the number of sent packets

tSLInformation.NumberOfSentPackets++;

// Allocate a buffer and construct a packet and send it over spi

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_DATA);

// Update the offset of data and parameters according to the command

switch(opcode)

{

case HCI\_CMND\_SENDTO:

{

addr\_offset = len + sizeof(len) + sizeof(len);

addrlen = 8;

uArgSize = SOCKET\_SENDTO\_PARAMS\_LEN;

pDataPtr = ptr + HEADERS\_SIZE\_DATA + SOCKET\_SENDTO\_PARAMS\_LEN;

break;

}

case HCI\_CMND\_SEND:

{

tolen = 0;

to = NULL;

uArgSize = HCI\_CMND\_SEND\_ARG\_LENGTH;

pDataPtr = ptr + HEADERS\_SIZE\_DATA + HCI\_CMND\_SEND\_ARG\_LENGTH;

break;

}

default:

{

break;

}

}

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

args = UINT32\_TO\_STREAM(args, uArgSize - sizeof(sd));

args = UINT32\_TO\_STREAM(args, len);

args = UINT32\_TO\_STREAM(args, flags);

if (opcode == HCI\_CMND\_SENDTO)

{

args = UINT32\_TO\_STREAM(args, addr\_offset);

args = UINT32\_TO\_STREAM(args, addrlen);

}

// Copy the data received from user into the TX Buffer

ARRAY\_TO\_STREAM(pDataPtr, ((UINT8 \*)buf), len);

// In case we are using SendTo, copy the to parameters

if (opcode == HCI\_CMND\_SENDTO)

{

ARRAY\_TO\_STREAM(pDataPtr, ((UINT8 \*)to), tolen);

}

// have to set this here (not after this is called, otherwise an IRQ could come before it’s set and the IRQ handler won’t know it’s the one we’re waiting on

**~~// Initiate a HCI command~~**

**if (opcode == HCI\_CMND\_SENDTO) {**

**tSLInformation.usRxEventOpcode = HCI\_EVNT\_SENDTO;**

hci\_data\_send(opcode, ptr, uArgSize, len,(UINT8\*)to, tolen);

**~~if (opcode == HCI\_CMND\_SENDTO)~~**

SimpleLinkWaitEvent(HCI\_EVNT\_SENDTO, &tSocketSendEvent);

**~~else~~} else {**

**tSLInformation.usRxEventOpcode = HCI\_EVNT\_SEND;**

**hci\_data\_send(opcode, ptr, uArgSize, len,(UINT8\*)to, tolen);**

SimpleLinkWaitEvent(HCI\_EVNT\_SEND, &tSocketSendEvent);

**}**

return (len);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! send

//!

//! @param sd socket handle

//! @param buf Points to a buffer containing the message to be sent

//! @param len message size in bytes

//! @param flags On this version, this parameter is not supported

//!

//! @return Return the number of bytes transmitted, or -1 if an

//! error occurred

//!

//! @brief Write data to TCP socket

//! This function is used to transmit a message to another

//! socket.

//!

//! @Note On this version, only blocking mode is supported.

//!

//! @sa sendto

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 send(INT32 sd, const void \*buf, INT32 len, INT32 flags)

{

return(simple\_link\_send(sd, buf, len, flags, NULL, 0, HCI\_CMND\_SEND));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! sendto

//!

//! @param sd socket handle

//! @param buf Points to a buffer containing the message to be sent

//! @param len message size in bytes

//! @param flags On this version, this parameter is not supported

//! @param to pointer to an address structure indicating the destination

//! address: sockaddr. On this version only AF\_INET is

//! supported.

//! @param tolen destination address structure size

//!

//! @return Return the number of bytes transmitted, or -1 if an

//! error occurred

//!

//! @brief Write data to TCP socket

//! This function is used to transmit a message to another

//! socket.

//!

//! @Note On this version, only blocking mode is supported.

//!

//! @sa send

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 sendto(INT32 sd, const void \*buf, INT32 len, INT32 flags, const sockaddr \*to,

socklen\_t tolen)

{

return(simple\_link\_send(sd, buf, len, flags, to, tolen, HCI\_CMND\_SENDTO));

}

#ifndef MDNS\_ADVERTISE\_HOST

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! mdnsAdvertiser

//!

//! @param[in] mdnsEnabled flag to enable/disable the mDNS feature

//! @param[in] deviceServiceName Service name as part of the published

//! canonical domain name

//! @param[in] deviceServiceNameLength Length of the service name - up to 32 chars

//!

//!

//! @return On success, zero is returned, return SOC\_ERROR if socket was not

//! opened successfully, or if an error occurred.

//!

//! @brief Set CC3000 in mDNS advertiser mode in order to advertise itself.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT16 mdnsAdvertiser(UINT16 mdnsEnabled, CHAR \* deviceServiceName, UINT16 deviceServiceNameLength)

{

INT8 ret;

UINT8 \*pTxBuffer, \*pArgs;

if (deviceServiceNameLength > MDNS\_DEVICE\_SERVICE\_MAX\_LENGTH)

{

return EFAIL;

}

pTxBuffer = tSLInformation.pucTxCommandBuffer;

pArgs = (pTxBuffer + SIMPLE\_LINK\_HCI\_CMND\_TRANSPORT\_HEADER\_SIZE);

// Fill in HCI packet structure

pArgs = UINT32\_TO\_STREAM(pArgs, mdnsEnabled);

pArgs = UINT32\_TO\_STREAM(pArgs, 8);

pArgs = UINT32\_TO\_STREAM(pArgs, deviceServiceNameLength);

ARRAY\_TO\_STREAM(pArgs, deviceServiceName, deviceServiceNameLength);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_MDNS\_ADVERTISE, pTxBuffer, SOCKET\_MDNS\_ADVERTISE\_PARAMS\_LEN + deviceServiceNameLength);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_EVNT\_MDNS\_ADVERTISE, &ret);

return ret;

}

#else

INT16 mdnsAdvertiser(UINT16 mdnsEnabled, CHAR \* deviceServiceName, UINT16 deviceServiceNameLength)

{

sockaddr tSocketAddr;

INT32 mdnsSocket = -1;

INT device\_name\_len;

CHAR mdnsResponse[220];

INT16 mdnsResponseLength;

CHAR \*mdnsResponsePtr;

if(deviceServiceName != NULL)

{

device\_name\_len = strlen(deviceServiceName);

}

else

{

return EFAIL;

}

if (deviceServiceNameLength > MDNS\_DEVICE\_SERVICE\_MAX\_LENGTH)

{

return EFAIL;

}

mdnsSocket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if(mdnsSocket < 0)

{

return -1;

}

//Send mDNS data to 224.0.0.251

tSocketAddr.sa\_family = AF\_INET;

// the destination port 5353

tSocketAddr.sa\_data[0] = 0x14;

tSocketAddr.sa\_data[1] = 0xe9;

tSocketAddr.sa\_data[2] = 0xe0;

tSocketAddr.sa\_data[3] = 0x00;

tSocketAddr.sa\_data[4] = 0x00;

tSocketAddr.sa\_data[5] = 0xfb;

memset(mdnsResponse, 0, sizeof(mdnsResponse));

mdnsResponsePtr = mdnsResponse;

// mDNS header

mdnsResponse[2] = 0x84; // DNS flags

mdnsResponse[7] = 0x5; // number of answers

mdnsResponsePtr += 12;

// answer 1 - the device service name

\*mdnsResponsePtr++ = 12; // size of \_device-info

memcpy(mdnsResponsePtr, "\_device-info", 12); // \_device-info

mdnsResponsePtr += 12;

\*mdnsResponsePtr++ = 4; // size of \_udp

memcpy(mdnsResponsePtr, "\_udp", 4); // \_udp

mdnsResponsePtr += 4;

\*mdnsResponsePtr++ = 5; // size of local

memcpy(mdnsResponsePtr, "local", 5); // local

mdnsResponsePtr += 7;

\*mdnsResponsePtr = 0xc; // PTR type

mdnsResponsePtr += 2;

\*mdnsResponsePtr = 0x1; // class IN

mdnsResponsePtr += 3;

\*mdnsResponsePtr++ = 0x11; // TTL = 4500 seconds

\*mdnsResponsePtr = 0x94; // TTL = 4500 seconds

mdnsResponsePtr += 4; // domain and its length - filled during invoke of mDNS advertiser

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr++ = 0x0c; // points to rest of the domain

// answer 2 - the device-info service

\*mdnsResponsePtr++ = 9; // size of \_services

memcpy(mdnsResponsePtr, "\_services", 9); // \_services

mdnsResponsePtr += 9;

\*mdnsResponsePtr++ = 7; // size of \_dns-sd

memcpy(mdnsResponsePtr, "\_dns-sd", 7); // \_dns-sd

mdnsResponsePtr += 7;

\*mdnsResponsePtr++ = 4; // size of \_udp

memcpy(mdnsResponsePtr, "\_udp", 4); // \_udp

mdnsResponsePtr += 4;

\*mdnsResponsePtr++ = 5; // size of local

memcpy(mdnsResponsePtr, "local", 5); // local

mdnsResponsePtr += 7;

\*mdnsResponsePtr = 0xc; // PTR type

mdnsResponsePtr += 2;

\*mdnsResponsePtr = 0x1; // class IN

mdnsResponsePtr += 3;

\*mdnsResponsePtr++ = 0x11; // TTL = 4500 seconds

\*mdnsResponsePtr = 0x94; // TTL = 4500 seconds

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 2; // size of PTR

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr++ = 0x0c; // points to rest of the domain

// answer 3 - TXT record of the service

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr = 0x2f; // points to device service name

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 0x10; // TXT type

\*mdnsResponsePtr++ = 0x80; // class UNICAST

\*mdnsResponsePtr = 0x1; // class IN

mdnsResponsePtr += 3;

\*mdnsResponsePtr++ = 0x11; // TTL = 4500 seconds

\*mdnsResponsePtr = 0x94; // TTL = 4500 seconds

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 36; // size of TXT

\*mdnsResponsePtr++ = 10; // size of dev=CC3000

memcpy(mdnsResponsePtr, "dev=CC3000", 10); // \_device-info

mdnsResponsePtr += 10;

\*mdnsResponsePtr++ = 24; // size of vendor=Texas-Instruments

memcpy(mdnsResponsePtr, "vendor=Texas-Instruments", 24); // \_udp

mdnsResponsePtr += 24;

// answer 4 - SRV record of the service

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr = 0x2f; // points to device service name

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 0x21; // SRV type

\*mdnsResponsePtr++ = 0x80; // class UNICAST

\*mdnsResponsePtr = 0x1; // class IN

mdnsResponsePtr += 3;

\*mdnsResponsePtr++ = 0x11; // TTL = 4500 seconds

\*mdnsResponsePtr = 0x94; // TTL = 4500 seconds

mdnsResponsePtr += 2;

//data length to be filled later in hook\_sl\_cmd\_parser function

mdnsResponsePtr += 5;

\*mdnsResponsePtr++ = 0x4; // high portion of port 1234

\*mdnsResponsePtr++ = 0xd2; // low portion of port 1234

//size should be according to device\_name (input parameter from API)

mdnsResponsePtr += 1; //leave free slot for device\_name length

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr++ = 0x1e; // points to local

// answer 5 - ADDRESS record of the service

\*mdnsResponsePtr++ = 0xc0;

\*mdnsResponsePtr =

(UINT16)(mdnsResponsePtr - mdnsResponse) - 4; //adding the required offset in hook\_sl\_cmd\_parser function

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 0x1; // Address type

\*mdnsResponsePtr++ = 0x80; // class UNICAST

\*mdnsResponsePtr = 0x1; // class IN

mdnsResponsePtr += 3;

\*mdnsResponsePtr++ = 0x11; // TTL = 4500 seconds

\*mdnsResponsePtr = 0x94; // TTL = 4500 seconds

mdnsResponsePtr += 2;

\*mdnsResponsePtr++ = 4; // size of Address

mdnsResponseLength = (UINT16)(mdnsResponsePtr - mdnsResponse);

//

// Move to the domain and its length

//

mdnsResponsePtr = &mdnsResponse[46];

//

// Domain length

//

\*mdnsResponsePtr++ = 3 + device\_name\_len;

//

// Size of device service name

//

\*mdnsResponsePtr++ = device\_name\_len;

//

// Now we need to insert the device service name here

// (so push the rest accordingly).

//

memmove(mdnsResponsePtr + device\_name\_len,

mdnsResponsePtr,

mdnsResponseLength - 48);

//

// Device service name.

//

memcpy(mdnsResponsePtr, deviceServiceName, device\_name\_len);

//

// Start handling Host Domain Name (Type = 1)

// DNS IE starts at constant offset: 62

// First answer starts at constant offset: 74

// Second answer starts at offset which depends on device\_name length:

// 112 + device\_name length

// Third answer starts at offset which depends on device\_name length:

// 154 + device\_name length

// Forth answer starts at offset which depends on device\_name length:

// 202 + device\_name length

// Fifth answer starts at offset which depends on device\_name length

// +SRV target: 223 + device\_name length \* 2 \*/

//

// Fill SRV Data Length -> Fourth answer,

// 10 bytes offset (Domain Name, Type, Class, TTL) + 1 byte (Fill LSB bits)

// => (202 + device\_name length + 11) - 62 (base offset) = 151 +

// device\_name length.

//

// Move to data length

//

mdnsResponsePtr = &mdnsResponse[151 + device\_name\_len];

//

//Data Length: Priority (2 bytes) + Weight (2 bytes) + Port (2 bytes) +

//Target size (1 byte) + 2 bytes (PTR + Offset of .local) = 9 bytes

//

\*mdnsResponsePtr = 9 + device\_name\_len;

//

//Fill SRV Target -> 7 bytes offset from Data Length.

//Derived from: (Priority 2 bytes, Weight 2 bytes, Port 2 bytes)

//

mdnsResponsePtr =

&mdnsResponse[158 + device\_name\_len];// Move to the domain and its length

\*mdnsResponsePtr++ = device\_name\_len; // Domain length

/\*now we need to insert the device service name here

(so push the rest accordingly)\*/

memmove(mdnsResponsePtr + device\_name\_len,

mdnsResponsePtr,

mdnsResponseLength - 158);

//

// Device service name

//

memcpy(mdnsResponsePtr,

((char \*)deviceServiceName),

device\_name\_len);

//

// Move to the end of the packet

//

mdnsResponsePtr =

&mdnsResponse[mdnsResponseLength + device\_name\_len + device\_name\_len];

//

//End handling Host Domain Name (Type = 1)

//

\*mdnsResponsePtr++ = localIP[3];

\*mdnsResponsePtr++ = localIP[2];

\*mdnsResponsePtr++ = localIP[1];

\*mdnsResponsePtr++ = localIP[0];

//

// Add the length of the device name to the ADDRESS record

//

\*(mdnsResponsePtr - 15) += device\_name\_len;

mdnsResponseLength = (UINT16)(mdnsResponsePtr - mdnsResponse);

//

// Send the mDNS response packet.

//

sendto(mdnsSocket,

mdnsResponse,

sizeof(mdnsResponse),

0,

(const sockaddr\*)&tSocketAddr,

mdnsResponseLength);

closesocket(mdnsSocket);

mdnsSocket = 0xFFFFFFFF;

return mdnsSocket;

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! getmssvalue

//!

//! @param[in] sd socket descriptor

//!

//! @return On success, returns the MSS value of a TCP connection

//!

//! @brief Returns the MSS value of a TCP connection according to the socket descriptor

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UINT16 getmssvalue (INT32 sd)

{

UINT8 \*ptr, \*args;

UINT16 ret;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, sd);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_GETMSSVALUE, ptr, SOCKET\_GET\_MSS\_VALUE\_PARAMS\_LEN);

// Since we are in blocking state - wait for event complete

SimpleLinkWaitEvent(HCI\_EVNT\_GETMSSVALUE, &ret);

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **wlan.c** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup wlan\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <string.h>

#include "wlan.h"

#include "hci.h"

#include "spi.h"

#include "socket.h"

#include "nvmem.h"

#include "security.h"

#include "evnt\_handler.h"

volatile sSimplLinkInformation tSLInformation;

#define SMART\_CONFIG\_PROFILE\_SIZE 67 // 67 = 32 (max ssid) + 32 (max key) + 1 (SSID length) + 1 (security type) + 1 (key length)

#ifndef CC3000\_UNENCRYPTED\_SMART\_CONFIG

UINT8 key[AES128\_KEY\_SIZE];

UINT8 profileArray[SMART\_CONFIG\_PROFILE\_SIZE];

#endif //CC3000\_UNENCRYPTED\_SMART\_CONFIG

/\* patches type \*/

#define PATCHES\_HOST\_TYPE\_WLAN\_DRIVER 0x01

#define PATCHES\_HOST\_TYPE\_WLAN\_FW 0x02

#define PATCHES\_HOST\_TYPE\_BOOTLOADER 0x03

#define SL\_SET\_SCAN\_PARAMS\_INTERVAL\_LIST\_SIZE (16)

#define SL\_SIMPLE\_CONFIG\_PREFIX\_LENGTH (3)

#define ETH\_ALEN (6)

#define MAXIMAL\_SSID\_LENGTH (32)

#define SL\_PATCHES\_REQUEST\_DEFAULT (0)

#define SL\_PATCHES\_REQUEST\_FORCE\_HOST (1)

#define SL\_PATCHES\_REQUEST\_FORCE\_NONE (2)

#define WLAN\_SEC\_UNSEC (0)

#define WLAN\_SEC\_WEP (1)

#define WLAN\_SEC\_WPA (2)

#define WLAN\_SEC\_WPA2 (3)

#define WLAN\_SL\_INIT\_START\_PARAMS\_LEN (1)

#define WLAN\_PATCH\_PARAMS\_LENGTH (8)

#define WLAN\_SET\_CONNECTION\_POLICY\_PARAMS\_LEN (12)

#define WLAN\_DEL\_PROFILE\_PARAMS\_LEN (4)

#define WLAN\_SET\_MASK\_PARAMS\_LEN (4)

#define WLAN\_SET\_SCAN\_PARAMS\_LEN (100)

#define WLAN\_GET\_SCAN\_RESULTS\_PARAMS\_LEN (4)

#define WLAN\_ADD\_PROFILE\_NOSEC\_PARAM\_LEN (24)

#define WLAN\_ADD\_PROFILE\_WEP\_PARAM\_LEN (36)

#define WLAN\_ADD\_PROFILE\_WPA\_PARAM\_LEN (44)

#define WLAN\_CONNECT\_PARAM\_LEN (29)

#define WLAN\_SMART\_CONFIG\_START\_PARAMS\_LEN (4)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! SimpleLink\_Init\_Start

//!

//! @param usPatchesAvailableAtHost flag to indicate if patches available

//! from host or from EEPROM. Due to the

//! fact the patches are burn to the EEPROM

//! using the patch programmer utility, the

//! patches will be available from the EEPROM

//! and not from the host.

//!

//! @return none

//!

//! @brief Send HCI\_CMND\_SIMPLE\_LINK\_START to CC3000

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void SimpleLink\_Init\_Start(UINT16 usPatchesAvailableAtHost)

{

UINT8 \*ptr;

UINT8 \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (UINT8 \*)(ptr + HEADERS\_SIZE\_CMD);

// If I recall, Martin Mauer had me change this…not sure why

UINT8\_TO\_STREAM(args, ((usPatchesAvailableAtHost) ? SL\_PATCHES\_REQUEST\_FORCE\_**~~NONE~~HOST** : SL\_PATCHES\_REQUEST\_DEFAULT));

// IRQ Line asserted - send HCI\_CMND\_SIMPLE\_LINK\_START to CC3000

hci\_command\_send(HCI\_CMND\_SIMPLE\_LINK\_START, ptr, WLAN\_SL\_INIT\_START\_PARAMS\_LEN);

SimpleLinkWaitEvent(HCI\_CMND\_SIMPLE\_LINK\_START, 0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_init

//!

//! @param sWlanCB Asynchronous events callback.

//! 0 no event call back.

//! -call back parameters:

//! 1) event\_type: HCI\_EVNT\_WLAN\_UNSOL\_CONNECT connect event,

//! HCI\_EVNT\_WLAN\_UNSOL\_DISCONNECT disconnect event,

//! HCI\_EVNT\_WLAN\_ASYNC\_SIMPLE\_CONFIG\_DONE config done,

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP dhcp report,

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT ping report OR

//! HCI\_EVNT\_WLAN\_KEEPALIVE keepalive.

//! 2) data: pointer to extra data that received by the event

//! (NULL no data).

//! 3) length: data length.

//! -Events with extra data:

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP: 4 bytes IP, 4 bytes Mask,

//! 4 bytes default gateway, 4 bytes DHCP server and 4 bytes

//! for DNS server.

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT: 4 bytes Packets sent,

//! 4 bytes Packets received, 4 bytes Min round time,

//! 4 bytes Max round time and 4 bytes for Avg round time.

//!

//! @param sFWPatches 0 no patch or pointer to FW patches

//! @param sDriverPatches 0 no patch or pointer to driver patches

//! @param sBootLoaderPatches 0 no patch or pointer to bootloader patches

//! @param sReadWlanInterruptPin init callback. the callback read wlan

//! interrupt status.

//! @param sWlanInterruptEnable init callback. the callback enable wlan

//! interrupt.

//! @param sWlanInterruptDisable init callback. the callback disable wlan

//! interrupt.

//! @param sWriteWlanPin init callback. the callback write value

//! to device pin.

//!

//! @return none

//!

//! @sa wlan\_set\_event\_mask , wlan\_start , wlan\_stop

//!

//! @brief Initialize wlan driver

//!

//! @warning This function must be called before ANY other wlan driver function

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void wlan\_init( tWlanCB sWlanCB,

tFWPatches sFWPatches,

tDriverPatches sDriverPatches,

tBootLoaderPatches sBootLoaderPatches,

tWlanReadInteruptPin sReadWlanInterruptPin,

tWlanInterruptEnable sWlanInterruptEnable,

tWlanInterruptDisable sWlanInterruptDisable,

tWriteWlanPin sWriteWlanPin)

{

tSLInformation.sFWPatches = sFWPatches;

tSLInformation.sDriverPatches = sDriverPatches;

tSLInformation.sBootLoaderPatches = sBootLoaderPatches;

// init io callback

tSLInformation.ReadWlanInterruptPin = sReadWlanInterruptPin;

tSLInformation.WlanInterruptEnable = sWlanInterruptEnable;

tSLInformation.WlanInterruptDisable = sWlanInterruptDisable;

tSLInformation.WriteWlanPin = sWriteWlanPin;

//init asynchronous events callback

tSLInformation.sWlanCB= sWlanCB;

// By default TX Complete events are routed to host too

tSLInformation.InformHostOnTxComplete = 1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! SpiReceiveHandler

//!

//! @param pvBuffer - pointer to the received data buffer

//! The function triggers Received event/data processing

//!

//! @param Pointer to the received data

//! @return none

//!

//! @brief The function triggers Received event/data processing. It is

//! called from the SPI library to receive the data

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void SpiReceiveHandler(void \*pvBuffer)

{

tSLInformation.usEventOrDataReceived = 1;

tSLInformation.pucReceivedData = (UINT8 \*)pvBuffer;

hci\_unsolicited\_event\_handler();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_start

//!

//! @param usPatchesAvailableAtHost - flag to indicate if patches available

//! from host or from EEPROM. Due to the

//! fact the patches are burn to the EEPROM

//! using the patch programmer utility, the

//! patches will be available from the EEPROM

//! and not from the host.

//!

//! @return none

//!

//! @brief Start WLAN device. This function asserts the enable pin of

//! the device (WLAN\_EN), starting the HW initialization process.

//! The function blocked until device Initialization is completed.

//! Function also configure patches (FW, driver or bootloader)

//! and calls appropriate device callbacks.

//!

//! @Note Prior calling the function wlan\_init shall be called.

//! @Warning This function must be called after wlan\_init and before any

//! other wlan API

//! @sa wlan\_init , wlan\_stop

//!

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// can’t remember why I pass a buffer instead of using the one originally specified

void wlan\_start(UINT16 usPatchesAvailableAtHost**, unsigned char \*transmitBuffer**)

{

UINT32 ulSpiIRQState;

tSLInformation.NumberOfSentPackets = 0;

tSLInformation.NumberOfReleasedPackets = 0;

tSLInformation.usRxEventOpcode = 0;

tSLInformation.usNumberOfFreeBuffers = 0;

tSLInformation.usSlBufferLength = 0;

tSLInformation.usBufferSize = 0;

tSLInformation.usRxDataPending = 0;

tSLInformation.slTransmitDataError = 0;

tSLInformation.usEventOrDataReceived = 0;

tSLInformation.pucReceivedData = 0;

// Allocate the memory for the RX/TX data transactions

tSLInformation.pucTxCommandBuffer = (UINT8 \*)**~~wlan\_tx\_b~~transmitB**uffer;

// init spi**, doesn't really "open" anything**

SpiOpen(SpiReceiveHandler);

// Check the IRQ line **before power-enabling the TiWi-SL (should be low at this point)**

ulSpiIRQState = tSLInformation.ReadWlanInterruptPin();

// Chip enable: toggle WLAN EN line

tSLInformation.WriteWlanPin( WLAN\_ENABLE );

**// if for some strange reason the IRQ line was already high before power-enable**

if (ulSpiIRQState)

{

// wait till the IRQ line goes low**, indicating readiness to receive first write**

while(tSLInformation.ReadWlanInterruptPin() != 0)

{

}

}

else

{

// wait till the IRQ line goes high and than low **(indicating no interrupt)**

while(tSLInformation.ReadWlanInterruptPin() == 0)

{

}

**// wait till the IRQ line goes low, indicating readiness to receive first write**

while(tSLInformation.ReadWlanInterruptPin() != 0)

{

}

}

SimpleLink\_Init\_Start(usPatchesAvailableAtHost);

// Read Buffer's size and finish

hci\_command\_send(HCI\_CMND\_READ\_BUFFER\_SIZE, tSLInformation.pucTxCommandBuffer, 0);

SimpleLinkWaitEvent(HCI\_CMND\_READ\_BUFFER\_SIZE, 0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_stop

//!

//! @param none

//!

//! @return none

//!

//! @brief Stop WLAN device by putting it into reset state.

//!

//! @sa wlan\_start

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void wlan\_stop(void)

{

// Chip disable

tSLInformation.WriteWlanPin( WLAN\_DISABLE );

// Wait till IRQ line goes high...

while(tSLInformation.ReadWlanInterruptPin() == 0)

{

}

// Free the used by WLAN Driver memory

if (tSLInformation.pucTxCommandBuffer)

{

tSLInformation.pucTxCommandBuffer = 0;

}

SpiClose();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_connect

//!

//! @param sec\_type security options:

//! WLAN\_SEC\_UNSEC,

//! WLAN\_SEC\_WEP (ASCII support only),

//! WLAN\_SEC\_WPA or WLAN\_SEC\_WPA2

//! @param ssid up to 32 bytes and is ASCII SSID of the AP

//! @param ssid\_len length of the SSID

//! @param bssid 6 bytes specified the AP bssid

//! @param key up to 32 bytes specified the AP security key

//! @param key\_len key length

//!

//! @return On success, zero is returned. On error, negative is returned.

//! Note that even though a zero is returned on success to trigger

//! connection operation, it does not mean that CCC3000 is already

//! connected. An asynchronous "Connected" event is generated when

//! actual association process finishes and CC3000 is connected to

//! the AP. If DHCP is set, An asynchronous "DHCP" event is

//! generated when DHCP process is finish.

//!

//!

//! @brief Connect to AP

//! @warning Please Note that when connection to AP configured with security

//! type WEP, please confirm that the key is set as ASCII and not

//! as HEX.

//! @sa wlan\_disconnect

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT32 wlan\_connect(UINT32 ulSecType, CHAR \*ssid, INT32 ssid\_len,

UINT8 \*bssid, UINT8 \*key, INT32 key\_len)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

UINT8 bssid\_zero[] = {0, 0, 0, 0, 0, 0};

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in command buffer

args = UINT32\_TO\_STREAM(args, 0x0000001c);

args = UINT32\_TO\_STREAM(args, ssid\_len);

args = UINT32\_TO\_STREAM(args, ulSecType);

args = UINT32\_TO\_STREAM(args, 0x00000010 + ssid\_len);

args = UINT32\_TO\_STREAM(args, key\_len);

args = UINT16\_TO\_STREAM(args, 0);

// padding shall be zeroed

if(bssid)

{

ARRAY\_TO\_STREAM(args, bssid, ETH\_ALEN);

}

else

{

ARRAY\_TO\_STREAM(args, bssid\_zero, ETH\_ALEN);

}

ARRAY\_TO\_STREAM(args, ssid, ssid\_len);

if(key\_len && key)

{

ARRAY\_TO\_STREAM(args, key, key\_len);

}

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_CONNECT, ptr, WLAN\_CONNECT\_PARAM\_LEN +

ssid\_len + key\_len - 1);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_CONNECT, &ret);

errno = ret;

return(ret);

}

#else

INT32 wlan\_connect(CHAR \*ssid, INT32 ssid\_len)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

UINT8 bssid\_zero[] = {0, 0, 0, 0, 0, 0};

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in command buffer

args = UINT32\_TO\_STREAM(args, 0x0000001c);

args = UINT32\_TO\_STREAM(args, ssid\_len);

args = UINT32\_TO\_STREAM(args, 0);

args = UINT32\_TO\_STREAM(args, 0x00000010 + ssid\_len);

args = UINT32\_TO\_STREAM(args, 0);

args = UINT16\_TO\_STREAM(args, 0);

// padding shall be zeroed

ARRAY\_TO\_STREAM(args, bssid\_zero, ETH\_ALEN);

ARRAY\_TO\_STREAM(args, ssid, ssid\_len);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_CONNECT, ptr, WLAN\_CONNECT\_PARAM\_LEN +

ssid\_len - 1);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_CONNECT, &ret);

errno = ret;

return(ret);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_disconnect

//!

//! @return 0 disconnected done, other CC3000 already disconnected

//!

//! @brief Disconnect connection from AP.

//!

//! @sa wlan\_connect

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_disconnect()

{

INT32 ret;

UINT8 \*ptr;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

hci\_command\_send(HCI\_CMND\_WLAN\_DISCONNECT, ptr, 0);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_DISCONNECT, &ret);

errno = ret;

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_set\_connection\_policy

//!

//! @param should\_connect\_to\_open\_ap enable(1), disable(0) connect to any

//! available AP. This parameter corresponds to the configuration of

//! item # 3 in the brief description.

//! @param should\_use\_fast\_connect enable(1), disable(0). if enabled, tries

//! to connect to the last connected AP. This parameter corresponds

//! to the configuration of item # 1 in the brief description.

//! @param auto\_start enable(1), disable(0) auto connect

//! after reset and periodically reconnect if needed. This

//! configuration configures option 2 in the above description.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief When auto is enabled, the device tries to connect according

//! the following policy:

//! 1) If fast connect is enabled and last connection is valid,

//! the device will try to connect to it without the scanning

//! procedure (fast). The last connection will be marked as

//! invalid, due to adding/removing profile.

//! 2) If profile exists, the device will try to connect it

//! (Up to seven profiles).

//! 3) If fast and profiles are not found, and open mode is

//! enabled, the device will try to connect to any AP.

//! \* Note that the policy settings are stored in the CC3000 NVMEM.

//!

//! @sa wlan\_add\_profile , wlan\_ioctl\_del\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_ioctl\_set\_connection\_policy(UINT32 should\_connect\_to\_open\_ap,

UINT32 ulShouldUseFastConnect,

UINT32 ulUseProfiles)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (UINT8 \*)(ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, should\_connect\_to\_open\_ap);

args = UINT32\_TO\_STREAM(args, ulShouldUseFastConnect);

args = UINT32\_TO\_STREAM(args, ulUseProfiles);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_SET\_CONNECTION\_POLICY,

ptr, WLAN\_SET\_CONNECTION\_POLICY\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_SET\_CONNECTION\_POLICY, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_add\_profile

//!

//! @param ulSecType WLAN\_SEC\_UNSEC,WLAN\_SEC\_WEP,WLAN\_SEC\_WPA,WLAN\_SEC\_WPA2

//! @param ucSsid ssid SSID up to 32 bytes

//! @param ulSsidLen ssid length

//! @param ucBssid bssid 6 bytes

//! @param ulPriority ulPriority profile priority. Lowest priority:0.

//! Important Note: Smartconfig process (in unencrypted mode)

//! stores the profile internally with priority 1, so changing

//! priorities when adding new profiles should be done with extra care

//! @param ulPairwiseCipher\_Or\_TxKeyLen key length for WEP security

//! @param ulGroupCipher\_TxKeyIndex key index

//! @param ulKeyMgmt KEY management

//! @param ucPf\_OrKey security key

//! @param ulPassPhraseLen security key length for WPA\WPA2

//!

//! @return On success, index (1-7) of the stored profile is returned.

//! On error, -1 is returned.

//!

//! @brief When auto start is enabled, the device connects to

//! station from the profiles table. Up to 7 profiles are supported.

//! If several profiles configured the device choose the highest

//! priority profile, within each priority group, device will choose

//! profile based on security policy, signal strength, etc

//! parameters. All the profiles are stored in CC3000 NVMEM.

//!

//! @sa wlan\_ioctl\_del\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT32 wlan\_add\_profile(UINT32 ulSecType,

UINT8\* ucSsid,

UINT32 ulSsidLen,

UINT8 \*ucBssid,

UINT32 ulPriority,

UINT32 ulPairwiseCipher\_Or\_TxKeyLen,

UINT32 ulGroupCipher\_TxKeyIndex,

UINT32 ulKeyMgmt,

UINT8\* ucPf\_OrKey,

UINT32 ulPassPhraseLen)

{

UINT16 arg\_len;

INT32 ret;

UINT8 \*ptr;

INT32 i = 0;

UINT8 \*args;

UINT8 bssid\_zero[] = {0, 0, 0, 0, 0, 0};

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

args = UINT32\_TO\_STREAM(args, ulSecType);

// Setup arguments in accordance with the security type

switch (ulSecType)

{

//OPEN

case WLAN\_SEC\_UNSEC:

{

args = UINT32\_TO\_STREAM(args, 0x00000014);

args = UINT32\_TO\_STREAM(args, ulSsidLen);

args = UINT16\_TO\_STREAM(args, 0);

if(ucBssid)

{

ARRAY\_TO\_STREAM(args, ucBssid, ETH\_ALEN);

}

else

{

ARRAY\_TO\_STREAM(args, bssid\_zero, ETH\_ALEN);

}

args = UINT32\_TO\_STREAM(args, ulPriority);

ARRAY\_TO\_STREAM(args, ucSsid, ulSsidLen);

arg\_len = WLAN\_ADD\_PROFILE\_NOSEC\_PARAM\_LEN + ulSsidLen;

}

break;

//WEP

case WLAN\_SEC\_WEP:

{

args = UINT32\_TO\_STREAM(args, 0x00000020);

args = UINT32\_TO\_STREAM(args, ulSsidLen);

args = UINT16\_TO\_STREAM(args, 0);

if(ucBssid)

{

ARRAY\_TO\_STREAM(args, ucBssid, ETH\_ALEN);

}

else

{

ARRAY\_TO\_STREAM(args, bssid\_zero, ETH\_ALEN);

}

args = UINT32\_TO\_STREAM(args, ulPriority);

args = UINT32\_TO\_STREAM(args, 0x0000000C + ulSsidLen);

args = UINT32\_TO\_STREAM(args, ulPairwiseCipher\_Or\_TxKeyLen);

args = UINT32\_TO\_STREAM(args, ulGroupCipher\_TxKeyIndex);

ARRAY\_TO\_STREAM(args, ucSsid, ulSsidLen);

for(i = 0; i < 4; i++)

{

UINT8 \*p = &ucPf\_OrKey[i \* ulPairwiseCipher\_Or\_TxKeyLen];

ARRAY\_TO\_STREAM(args, p, ulPairwiseCipher\_Or\_TxKeyLen);

}

arg\_len = WLAN\_ADD\_PROFILE\_WEP\_PARAM\_LEN + ulSsidLen +

ulPairwiseCipher\_Or\_TxKeyLen \* 4;

}

break;

//WPA

//WPA2

case WLAN\_SEC\_WPA:

case WLAN\_SEC\_WPA2:

{

args = UINT32\_TO\_STREAM(args, 0x00000028);

args = UINT32\_TO\_STREAM(args, ulSsidLen);

args = UINT16\_TO\_STREAM(args, 0);

if(ucBssid)

{

ARRAY\_TO\_STREAM(args, ucBssid, ETH\_ALEN);

}

else

{

ARRAY\_TO\_STREAM(args, bssid\_zero, ETH\_ALEN);

}

args = UINT32\_TO\_STREAM(args, ulPriority);

args = UINT32\_TO\_STREAM(args, ulPairwiseCipher\_Or\_TxKeyLen);

args = UINT32\_TO\_STREAM(args, ulGroupCipher\_TxKeyIndex);

args = UINT32\_TO\_STREAM(args, ulKeyMgmt);

args = UINT32\_TO\_STREAM(args, 0x00000008 + ulSsidLen);

args = UINT32\_TO\_STREAM(args, ulPassPhraseLen);

ARRAY\_TO\_STREAM(args, ucSsid, ulSsidLen);

ARRAY\_TO\_STREAM(args, ucPf\_OrKey, ulPassPhraseLen);

arg\_len = WLAN\_ADD\_PROFILE\_WPA\_PARAM\_LEN + ulSsidLen + ulPassPhraseLen;

}

break;

}

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_ADD\_PROFILE,

ptr, arg\_len);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_ADD\_PROFILE, &ret);

return(ret);

}

#else

INT32 wlan\_add\_profile(UINT32 ulSecType,

UINT8\* ucSsid,

UINT32 ulSsidLen,

UINT8 \*ucBssid,

UINT32 ulPriority,

UINT32 ulPairwiseCipher\_Or\_TxKeyLen,

UINT32 ulGroupCipher\_TxKeyIndex,

UINT32 ulKeyMgmt,

UINT8\* ucPf\_OrKey,

UINT32 ulPassPhraseLen)

{

return -1;

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_del\_profile

//!

//! @param index number of profile to delete

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Delete WLAN profile

//!

//! @Note In order to delete all stored profile, set index to 255.

//!

//! @sa wlan\_add\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_ioctl\_del\_profile(UINT32 ulIndex)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (UINT8 \*)(ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, ulIndex);

ret = EFAIL;

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_DEL\_PROFILE,

ptr, WLAN\_DEL\_PROFILE\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_DEL\_PROFILE, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_get\_scan\_results

//!

//! @param[in] scan\_timeout parameter not supported

//! @param[out] ucResults scan results (\_wlan\_full\_scan\_results\_args\_t)

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Gets entry from scan result table.

//! The scan results are returned one by one, and each entry

//! represents a single AP found in the area. The following is a

//! format of the scan result:

//! - 4 Bytes: number of networks found

//! - 4 Bytes: The status of the scan: 0 - aged results,

//! 1 - results valid, 2 - no results

//! - 42 bytes: Result entry, where the bytes are arranged as follows:

//!

//! - 1 bit isValid - is result valid or not

//! - 7 bits rssi - RSSI value;

//! - 2 bits: securityMode - security mode of the AP:

//! 0 - Open, 1 - WEP, 2 WPA, 3 WPA2

//! - 6 bits: SSID name length

//! - 2 bytes: the time at which the entry has entered into

//! scans result table

//! - 32 bytes: SSID name

//! - 6 bytes: BSSID

//!

//! @Note scan\_timeout, is not supported on this version.

//!

//! @sa wlan\_ioctl\_set\_scan\_params

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT32 wlan\_ioctl\_get\_scan\_results(UINT32 ulScanTimeout,

UINT8 \*ucResults)

{

UINT8 \*ptr;

UINT8 \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, ulScanTimeout);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_GET\_SCAN\_RESULTS,

ptr, WLAN\_GET\_SCAN\_RESULTS\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_GET\_SCAN\_RESULTS, ucResults);

return(0);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_set\_scan\_params

//!

//! @param uiEnable - start/stop application scan:

//! 1 = start scan with default interval value of 10 min.

//! in order to set a different scan interval value apply the value

//! in milliseconds. minimum 1 second. 0=stop). Wlan reset

//! (wlan\_stop() wlan\_start()) is needed when changing scan interval

//! value. Saved: No

//! @param uiMinDwellTime minimum dwell time value to be used for each

//! channel, in milliseconds. Saved: yes

//! Recommended Value: 100 (Default: 20)

//! @param uiMaxDwellTime maximum dwell time value to be used for each

//! channel, in milliseconds. Saved: yes

//! Recommended Value: 100 (Default: 30)

//! @param uiNumOfProbeRequests max probe request between dwell time.

//! Saved: yes. Recommended Value: 5 (Default:2)

//! @param uiChannelMask bitwise, up to 13 channels (0x1fff).

//! Saved: yes. Default: 0x7ff

//! @param uiRSSIThreshold RSSI threshold. Saved: yes (Default: -80)

//! @param uiSNRThreshold NSR threshold. Saved: yes (Default: 0)

//! @param uiDefaultTxPower probe Tx power. Saved: yes (Default: 205)

//! @param aiIntervalList pointer to array with 16 entries (16 channels)

//! each entry (UINT32) holds timeout between periodic scan

//! (connection scan) - in millisecond. Saved: yes. Default 2000ms.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief start and stop scan procedure. Set scan parameters.

//!

//! @Note uiDefaultTxPower, is not supported on this version.

//!

//! @sa wlan\_ioctl\_get\_scan\_results

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT32 wlan\_ioctl\_set\_scan\_params(UINT32 uiEnable, UINT32 uiMinDwellTime,

UINT32 uiMaxDwellTime,

UINT32 uiNumOfProbeRequests,

UINT32 uiChannelMask,INT32 iRSSIThreshold,

UINT32 uiSNRThreshold,

UINT32 uiDefaultTxPower,

UINT32 \*aiIntervalList)

{

UINT32 uiRes;

UINT8 \*ptr;

UINT8 \*args;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

// Fill in temporary command buffer

args = UINT32\_TO\_STREAM(args, 36);

args = UINT32\_TO\_STREAM(args, uiEnable);

args = UINT32\_TO\_STREAM(args, uiMinDwellTime);

args = UINT32\_TO\_STREAM(args, uiMaxDwellTime);

args = UINT32\_TO\_STREAM(args, uiNumOfProbeRequests);

args = UINT32\_TO\_STREAM(args, uiChannelMask);

args = UINT32\_TO\_STREAM(args, iRSSIThreshold);

args = UINT32\_TO\_STREAM(args, uiSNRThreshold);

args = UINT32\_TO\_STREAM(args, uiDefaultTxPower);

ARRAY\_TO\_STREAM(args, aiIntervalList, sizeof(UINT32) \*

SL\_SET\_SCAN\_PARAMS\_INTERVAL\_LIST\_SIZE);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_SET\_SCANPARAM,

ptr, WLAN\_SET\_SCAN\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_SET\_SCANPARAM, &uiRes);

return(uiRes);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_set\_event\_mask

//!

//! @param mask mask option:

//! HCI\_EVNT\_WLAN\_UNSOL\_CONNECT connect event

//! HCI\_EVNT\_WLAN\_UNSOL\_DISCONNECT disconnect event

//! HCI\_EVNT\_WLAN\_ASYNC\_SIMPLE\_CONFIG\_DONE smart config done

//! HCI\_EVNT\_WLAN\_UNSOL\_INIT init done

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP dhcp event report

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT ping report

//! HCI\_EVNT\_WLAN\_KEEPALIVE keepalive

//! HCI\_EVNT\_WLAN\_TX\_COMPLETE - disable information on end of transmission

//! Saved: no.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Mask event according to bit mask. In case that event is

//! masked (1), the device will not send the masked event to host.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_set\_event\_mask(UINT32 ulMask)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

if ((ulMask & HCI\_EVNT\_WLAN\_TX\_COMPLETE) == HCI\_EVNT\_WLAN\_TX\_COMPLETE)

{

tSLInformation.InformHostOnTxComplete = 0;

// Since an event is a virtual event - i.e. it is not coming from CC3000

// there is no need to send anything to the device if it was an only event

if (ulMask == HCI\_EVNT\_WLAN\_TX\_COMPLETE)

{

return 0;

}

ulMask &= ~HCI\_EVNT\_WLAN\_TX\_COMPLETE;

ulMask |= HCI\_EVNT\_WLAN\_UNSOL\_BASE;

}

else

{

tSLInformation.InformHostOnTxComplete = 1;

}

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (UINT8 \*)(ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, ulMask);

// Initiate a HCI command

hci\_command\_send(HCI\_CMND\_EVENT\_MASK,

ptr, WLAN\_SET\_MASK\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_EVENT\_MASK, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_statusget

//!

//! @param none

//!

//! @return WLAN\_STATUS\_DISCONNECTED, WLAN\_STATUS\_SCANING,

//! STATUS\_CONNECTING or WLAN\_STATUS\_CONNECTED

//!

//! @brief get wlan status: disconnected, scanning, connecting or connected

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

INT32 wlan\_ioctl\_statusget(void)

{

INT32 ret;

UINT8 \*ptr;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_STATUSGET,

ptr, 0);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_STATUSGET, &ret);

return(ret);

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_start

//!

//! @param algoEncryptedFlag indicates whether the information is encrypted

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Start to acquire device profile. The device acquire its own

//! profile, if profile message is found. The acquired AP information

//! is stored in CC3000 EEPROM only in case AES128 encryption is used.

//! In case AES128 encryption is not used, a profile is created by

//! CC3000 internally.

//!

//! @Note An asynchronous event - Smart Config Done will be generated as soon

//! as the process finishes successfully.

//!

//! @sa wlan\_smart\_config\_set\_prefix , wlan\_smart\_config\_stop

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_smart\_config\_start(UINT32 algoEncryptedFlag)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (UINT8 \*)(ptr + HEADERS\_SIZE\_CMD);

// Fill in HCI packet structure

args = UINT32\_TO\_STREAM(args, algoEncryptedFlag);

ret = EFAIL;

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_START, ptr,

WLAN\_SMART\_CONFIG\_START\_PARAMS\_LEN);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_START, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_stop

//!

//! @param algoEncryptedFlag indicates whether the information is encrypted

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Stop the acquire profile procedure

//!

//! @sa wlan\_smart\_config\_start , wlan\_smart\_config\_set\_prefix

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_smart\_config\_stop(void)

{

INT32 ret;

UINT8 \*ptr;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_STOP, ptr, 0);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_STOP, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_set\_prefix

//!

//! @param newPrefix 3 bytes identify the SSID prefix for the Smart Config.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Configure station ssid prefix. The prefix is used internally

//! in CC3000. It should always be TTT.

//!

//! @Note The prefix is stored in CC3000 NVMEM

//!

//! @sa wlan\_smart\_config\_start , wlan\_smart\_config\_stop

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INT32 wlan\_smart\_config\_set\_prefix(CHAR\* cNewPrefix)

{

INT32 ret;

UINT8 \*ptr;

UINT8 \*args;

ret = EFAIL;

ptr = tSLInformation.pucTxCommandBuffer;

args = (ptr + HEADERS\_SIZE\_CMD);

if (cNewPrefix == NULL)

return ret;

else // with the new Smart Config, prefix must be TTT

{

\*cNewPrefix = 'T';

\*(cNewPrefix + 1) = 'T';

\*(cNewPrefix + 2) = 'T';

}

ARRAY\_TO\_STREAM(args, cNewPrefix, SL\_SIMPLE\_CONFIG\_PREFIX\_LENGTH);

hci\_command\_send(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_SET\_PREFIX, ptr,

SL\_SIMPLE\_CONFIG\_PREFIX\_LENGTH);

// Wait for command complete event

SimpleLinkWaitEvent(HCI\_CMND\_WLAN\_IOCTL\_SIMPLE\_CONFIG\_SET\_PREFIX, &ret);

return(ret);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_process

//!

//! @param none

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief process the acquired data and store it as a profile. The acquired

//! AP information is stored in CC3000 EEPROM encrypted.

//! The encrypted data is decrypted and stored as a profile.

//! behavior is as defined by connection policy.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_UNENCRYPTED\_SMART\_CONFIG

INT32 wlan\_smart\_config\_process()

{

INT32 returnValue;

UINT32 ssidLen, keyLen;

UINT8 \*decKeyPtr;

UINT8 \*ssidPtr;

// read the key from EEPROM - fileID 12

returnValue = aes\_read\_key(key);

if (returnValue != 0)

return returnValue;

// read the received data from fileID #13 and parse it according to the followings:

// 1) SSID LEN - not encrypted

// 2) SSID - not encrypted

// 3) KEY LEN - not encrypted. always 32 bytes long

// 4) Security type - not encrypted

// 5) KEY - encrypted together with true key length as the first byte in KEY

// to elaborate, there are two corner cases:

// 1) the KEY is 32 bytes long. In this case, the first byte does not represent KEY length

// 2) the KEY is 31 bytes long. In this case, the first byte represent KEY length and equals 31

returnValue = nvmem\_read(NVMEM\_SHARED\_MEM\_FILEID, SMART\_CONFIG\_PROFILE\_SIZE, 0, profileArray);

if (returnValue != 0)

return returnValue;

ssidPtr = &profileArray[1];

ssidLen = profileArray[0];

decKeyPtr = &profileArray[profileArray[0] + 3];

aes\_decrypt(decKeyPtr, key);

if (profileArray[profileArray[0] + 1] > 16)

aes\_decrypt((UINT8 \*)(decKeyPtr + 16), key);

if (\*(UINT8 \*)(decKeyPtr +31) != 0)

{

if (\*decKeyPtr == 31)

{

keyLen = 31;

decKeyPtr++;

}

else

{

keyLen = 32;

}

}

else

{

keyLen = \*decKeyPtr;

decKeyPtr++;

}

// add a profile

switch (profileArray[profileArray[0] + 2])

{

case WLAN\_SEC\_UNSEC://None

{

returnValue = wlan\_add\_profile(profileArray[profileArray[0] + 2], // security type

ssidPtr, // SSID

ssidLen, // SSID length

NULL, // BSSID

1, // Priority

0, 0, 0, 0, 0);

break;

}

case WLAN\_SEC\_WEP://WEP

{

returnValue = wlan\_add\_profile(profileArray[profileArray[0] + 2], // security type

ssidPtr, // SSID

ssidLen, // SSID length

NULL, // BSSID

1, // Priority

keyLen, // KEY length

0, // KEY index

0,

decKeyPtr, // KEY

0);

break;

}

case WLAN\_SEC\_WPA://WPA

case WLAN\_SEC\_WPA2://WPA2

{

returnValue = wlan\_add\_profile(WLAN\_SEC\_WPA2, // security type

ssidPtr,

ssidLen,

NULL, // BSSID

1, // Priority

0x18, // PairwiseCipher

0x1e, // GroupCipher

2, // KEY management

decKeyPtr, // KEY

keyLen); // KEY length

break;

}

}

return returnValue;

}

#endif //CC3000\_UNENCRYPTED\_SMART\_CONFIG

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* **wlan.h** - CC3000 Host Driver Implementation.

\* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_WLAN\_H\_\_

#define \_\_WLAN\_H\_\_

#include "cc3000\_common.h"

**#include "ui.h"**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// If building with a C++ compiler, make all of the definitions in this header

// have a C binding.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

extern "C" {

#endif

#define WLAN\_SEC\_UNSEC (0)

#define WLAN\_SEC\_WEP (1)

#define WLAN\_SEC\_WPA (2)

#define WLAN\_SEC\_WPA2 (3)

#define WLAN\_STATUS\_DISCONNECTED (0)

#define WLAN\_STATUS\_SCANNING (1)

#define WLAN\_STATUS\_CONNECTING (2)

#define WLAN\_STATUS\_CONNECTED (3)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup wlan\_api

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_init

//!

//! @param sWlanCB Asynchronous events callback.

//! 0 no event call back.

//! -call back parameters:

//! 1) event\_type: HCI\_EVNT\_WLAN\_UNSOL\_CONNECT connect event,

//! HCI\_EVNT\_WLAN\_UNSOL\_DISCONNECT disconnect event,

//! HCI\_EVNT\_WLAN\_ASYNC\_SIMPLE\_CONFIG\_DONE config done,

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP dhcp report,

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT ping report OR

//! HCI\_EVNT\_WLAN\_KEEPALIVE keepalive.

//! 2) data: pointer to extra data that received by the event

//! (NULL no data).

//! 3) length: data length.

//! -Events with extra data:

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP: 4 bytes IP, 4 bytes Mask,

//! 4 bytes default gateway, 4 bytes DHCP server and 4 bytes

//! for DNS server.

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT: 4 bytes Packets sent,

//! 4 bytes Packets received, 4 bytes Min round time,

//! 4 bytes Max round time and 4 bytes for Avg round time.

//!

//! @param sFWPatches 0 no patch or pointer to FW patches

//! @param sDriverPatches 0 no patch or pointer to driver patches

//! @param sBootLoaderPatches 0 no patch or pointer to bootloader patches

//! @param sReadWlanInterruptPin init callback. the callback read wlan

//! interrupt status.

//! @param sWlanInterruptEnable init callback. the callback enable wlan

//! interrupt.

//! @param sWlanInterruptDisable init callback. the callback disable wlan

//! interrupt.

//! @param sWriteWlanPin init callback. the callback write value

//! to device pin.

//!

//! @return none

//!

//! @sa wlan\_set\_event\_mask , wlan\_start , wlan\_stop

//!

//! @brief Initialize wlan driver

//!

//! @warning This function must be called before ANY other wlan driver function

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern void wlan\_init( tWlanCB sWlanCB,

tFWPatches sFWPatches,

tDriverPatches sDriverPatches,

tBootLoaderPatches sBootLoaderPatches,

tWlanReadInteruptPin sReadWlanInterruptPin,

tWlanInterruptEnable sWlanInterruptEnable,

tWlanInterruptDisable sWlanInterruptDisable,

tWriteWlanPin sWriteWlanPin);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_start

//!

//! @param usPatchesAvailableAtHost - flag to indicate if patches available

//! from host or from EEPROM. Due to the

//! fact the patches are burn to the EEPROM

//! using the patch programmer utility, the

//! patches will be available from the EEPROM

//! and not from the host.

//!

//! @return none

//!

//! @brief Start WLAN device. This function asserts the enable pin of

//! the device (WLAN\_EN), starting the HW initialization process.

//! The function blocked until device Initialization is completed.

//! Function also configure patches (FW, driver or bootloader)

//! and calls appropriate device callbacks.

//!

//! @Note Prior calling the function wlan\_init shall be called.

//! @Warning This function must be called after wlan\_init and before any

//! other wlan API

//! @sa wlan\_init , wlan\_stop

//!

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern void wlan\_start(UINT16 usPatchesAvailableAtHost**~~);~~, unsigned char \*transmitBuffer);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_stop

//!

//! @param none

//!

//! @return none

//!

//! @brief Stop WLAN device by putting it into reset state.

//!

//! @sa wlan\_start

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern void wlan\_stop(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_connect

//!

//! @param sec\_type security options:

//! WLAN\_SEC\_UNSEC,

//! WLAN\_SEC\_WEP (ASCII support only),

//! WLAN\_SEC\_WPA or WLAN\_SEC\_WPA2

//! @param ssid up to 32 bytes and is ASCII SSID of the AP

//! @param ssid\_len length of the SSID

//! @param bssid 6 bytes specified the AP bssid

//! @param key up to 32 bytes specified the AP security key

//! @param key\_len key length

//!

//! @return On success, zero is returned. On error, negative is returned.

//! Note that even though a zero is returned on success to trigger

//! connection operation, it does not mean that CCC3000 is already

//! connected. An asynchronous "Connected" event is generated when

//! actual association process finishes and CC3000 is connected to

//! the AP. If DHCP is set, An asynchronous "DHCP" event is

//! generated when DHCP process is finish.

//!

//!

//! @brief Connect to AP

//! @warning Please Note that when connection to AP configured with security

//! type WEP, please confirm that the key is set as ASCII and not

//! as HEX.

//! @sa wlan\_disconnect

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CC3000\_TINY\_DRIVER

extern INT32 wlan\_connect(UINT32 ulSecType, CHAR \*ssid, INT32 ssid\_len,

UINT8 \*bssid, UINT8 \*key, INT32 key\_len);

#else

extern INT32 wlan\_connect(CHAR \*ssid, INT32 ssid\_len);

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_disconnect

//!

//! @return 0 disconnected done, other CC3000 already disconnected

//!

//! @brief Disconnect connection from AP.

//!

//! @sa wlan\_connect

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_disconnect(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_add\_profile

//!

//! @param ulSecType WLAN\_SEC\_UNSEC,WLAN\_SEC\_WEP,WLAN\_SEC\_WPA,WLAN\_SEC\_WPA2

//! @param ucSsid ssid SSID up to 32 bytes

//! @param ulSsidLen ssid length

//! @param ucBssid bssid 6 bytes

//! @param ulPriority ulPriority profile priority. Lowest priority:0.

//! @param ulPairwiseCipher\_Or\_TxKeyLen key length for WEP security

//! @param ulGroupCipher\_TxKeyIndex key index

//! @param ulKeyMgmt KEY management

//! @param ucPf\_OrKey security key

//! @param ulPassPhraseLen security key length for WPA\WPA2

//!

//! @return On success, index (1-7) of the stored profile is returned.

//! On error, -1 is returned.

//!

//! @brief When auto start is enabled, the device connects to

//! station from the profiles table. Up to 7 profiles are supported.

//! If several profiles configured the device choose the highest

//! priority profile, within each priority group, device will choose

//! profile based on security policy, signal strength, etc

//! parameters. All the profiles are stored in CC3000 NVMEM.

//!

//! @sa wlan\_ioctl\_del\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_add\_profile(UINT32 ulSecType, UINT8\* ucSsid,

UINT32 ulSsidLen,

UINT8 \*ucBssid,

UINT32 ulPriority,

UINT32 ulPairwiseCipher\_Or\_Key,

UINT32 ulGroupCipher\_TxKeyLen,

UINT32 ulKeyMgmt,

UINT8\* ucPf\_OrKey,

UINT32 ulPassPhraseLen);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_del\_profile

//!

//! @param index number of profile to delete

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Delete WLAN profile

//!

//! @Note In order to delete all stored profile, set index to 255.

//!

//! @sa wlan\_add\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_ioctl\_del\_profile(UINT32 ulIndex);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_set\_event\_mask

//!

//! @param mask mask option:

//! HCI\_EVNT\_WLAN\_UNSOL\_CONNECT connect event

//! HCI\_EVNT\_WLAN\_UNSOL\_DISCONNECT disconnect event

//! HCI\_EVNT\_WLAN\_ASYNC\_SIMPLE\_CONFIG\_DONE smart config done

//! HCI\_EVNT\_WLAN\_UNSOL\_INIT init done

//! HCI\_EVNT\_WLAN\_UNSOL\_DHCP dhcp event report

//! HCI\_EVNT\_WLAN\_ASYNC\_PING\_REPORT ping report

//! HCI\_EVNT\_WLAN\_KEEPALIVE keepalive

//! HCI\_EVNT\_WLAN\_TX\_COMPLETE - disable information on end of transmission

//! Saved: no.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Mask event according to bit mask. In case that event is

//! masked (1), the device will not send the masked event to host.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_set\_event\_mask(UINT32 ulMask);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_statusget

//!

//! @param none

//!

//! @return WLAN\_STATUS\_DISCONNECTED, WLAN\_STATUS\_SCANING,

//! STATUS\_CONNECTING or WLAN\_STATUS\_CONNECTED

//!

//! @brief get wlan status: disconnected, scanning, connecting or connected

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_ioctl\_statusget(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_set\_connection\_policy

//!

//! @param should\_connect\_to\_open\_ap enable(1), disable(0) connect to any

//! available AP. This parameter corresponds to the configuration of

//! item # 3 in the brief description.

//! @param should\_use\_fast\_connect enable(1), disable(0). if enabled, tries

//! to connect to the last connected AP. This parameter corresponds

//! to the configuration of item # 1 in the brief description.

//! @param auto\_start enable(1), disable(0) auto connect

//! after reset and periodically reconnect if needed. This

//! configuration configures option 2 in the above description.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief When auto is enabled, the device tries to connect according

//! the following policy:

//! 1) If fast connect is enabled and last connection is valid,

//! the device will try to connect to it without the scanning

//! procedure (fast). The last connection will be marked as

//! invalid, due to adding/removing profile.

//! 2) If profile exists, the device will try to connect it

//! (Up to seven profiles).

//! 3) If fast and profiles are not found, and open mode is

//! enabled, the device will try to connect to any AP.

//! \* Note that the policy settings are stored in the CC3000 NVMEM.

//!

//! @sa wlan\_add\_profile , wlan\_ioctl\_del\_profile

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_ioctl\_set\_connection\_policy(

UINT32 should\_connect\_to\_open\_ap,

UINT32 should\_use\_fast\_connect,

UINT32 ulUseProfiles);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_get\_scan\_results

//!

//! @param[in] scan\_timeout parameter not supported

//! @param[out] ucResults scan result (\_wlan\_full\_scan\_results\_args\_t)

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Gets entry from scan result table.

//! The scan results are returned one by one, and each entry

//! represents a single AP found in the area. The following is a

//! format of the scan result:

//! - 4 Bytes: number of networks found

//! - 4 Bytes: The status of the scan: 0 - aged results,

//! 1 - results valid, 2 - no results

//! - 42 bytes: Result entry, where the bytes are arranged as follows:

//!

//! - 1 bit isValid - is result valid or not

//! - 7 bits rssi - RSSI value;

//! - 2 bits: securityMode - security mode of the AP:

//! 0 - Open, 1 - WEP, 2 WPA, 3 WPA2

//! - 6 bits: SSID name length

//! - 2 bytes: the time at which the entry has entered into

//! scans result table

//! - 32 bytes: SSID name

//! - 6 bytes: BSSID

//!

//! @Note scan\_timeout, is not supported on this version.

//!

//! @sa wlan\_ioctl\_set\_scan\_params

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_ioctl\_get\_scan\_results(UINT32 ulScanTimeout,

UINT8 \*ucResults);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_ioctl\_set\_scan\_params

//!

//! @param uiEnable - start/stop application scan:

//! 1 = start scan with default interval value of 10 min.

//! in order to set a different scan interval value apply the value

//! in milliseconds. minimum 1 second. 0=stop). Wlan reset

//! (wlan\_stop() wlan\_start()) is needed when changing scan interval

//! value. Saved: No

//! @param uiMinDwellTime minimum dwell time value to be used for each

//! channel, in milliseconds. Saved: yes

//! Recommended Value: 100 (Default: 20)

//! @param uiMaxDwellTime maximum dwell time value to be used for each

//! channel, in milliseconds. Saved: yes

//! Recommended Value: 100 (Default: 30)

//! @param uiNumOfProbeRequests max probe request between dwell time.

//! Saved: yes. Recommended Value: 5 (Default:2)

//! @param uiChannelMask bitwise, up to 13 channels (0x1fff).

//! Saved: yes. Default: 0x7ff

//! @param uiRSSIThreshold RSSI threshold. Saved: yes (Default: -80)

//! @param uiSNRThreshold NSR threshold. Saved: yes (Default: 0)

//! @param uiDefaultTxPower probe Tx power. Saved: yes (Default: 205)

//! @param aiIntervalList pointer to array with 16 entries (16 channels)

//! each entry (UINT32) holds timeout between periodic scan

//! (connection scan) - in milliseconds. Saved: yes. Default 2000ms.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief start and stop scan procedure. Set scan parameters.

//!

//! @Note uiDefaultTxPower, is not supported on this version.

//!

//! @sa wlan\_ioctl\_get\_scan\_results

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_ioctl\_set\_scan\_params(UINT32 uiEnable, UINT32

uiMinDwellTime,UINT32 uiMaxDwellTime,

UINT32 uiNumOfProbeRequests,

UINT32 uiChannelMask,

INT32 iRSSIThreshold,UINT32 uiSNRThreshold,

UINT32 uiDefaultTxPower,

UINT32 \*aiIntervalList);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_start

//!

//! @param algoEncryptedFlag indicates whether the information is encrypted

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Start to acquire device profile. The device acquire its own

//! profile, if profile message is found. The acquired AP information

//! is stored in CC3000 EEPROM only in case AES128 encryption is used.

//! In case AES128 encryption is not used, a profile is created by

//! CC3000 internally.

//!

//! @Note An asynchronous event - Smart Config Done will be generated as soon

//! as the process finishes successfully.

//!

//! @sa wlan\_smart\_config\_set\_prefix , wlan\_smart\_config\_stop

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_smart\_config\_start(UINT32 algoEncryptedFlag);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_stop

//!

//! @param algoEncryptedFlag indicates whether the information is encrypted

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Stop the acquire profile procedure

//!

//! @sa wlan\_smart\_config\_start , wlan\_smart\_config\_set\_prefix

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_smart\_config\_stop(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_set\_prefix

//!

//! @param newPrefix 3 bytes identify the SSID prefix for the Smart Config.

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief Configure station ssid prefix. The prefix is used internally

//! in CC3000. It should always be TTT.

//!

//! @Note The prefix is stored in CC3000 NVMEM

//!

//! @sa wlan\_smart\_config\_start , wlan\_smart\_config\_stop

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_smart\_config\_set\_prefix(CHAR\* cNewPrefix);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! wlan\_smart\_config\_process

//!

//! @param none

//!

//! @return On success, zero is returned. On error, -1 is returned

//!

//! @brief process the acquired data and store it as a profile. The acquired

//! AP information is stored in CC3000 EEPROM encrypted.

//! The encrypted data is decrypted and stored as a profile.

//! behavior is as defined by connection policy.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

extern INT32 wlan\_smart\_config\_process(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Mark the end of the C bindings section for C++ compilers.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef \_\_cplusplus

}

#endif // \_\_cplusplus

#endif // \_\_WLAN\_H\_\_