SIMPLELINK GEN1 TO GEN2 SDK Migration GUIDE

User's Guide

4/1/17

Table of Contents

Contents

List of Figures

List of Tables

1 Introduction

CC3120 SimpleLink Wi-Fi Network Processor and CC3220 SimpleLink Wireless MCU are the second generation devices in the SimpleLink Wi-Fi and Internet-of-Things family. These devices introduce some new features and capabilities that further simplify connectivity of things to the internet.

Among of the new capabilities are:

- AP supporting up to 4 stations
- Improved provisioning
- Improved power consumption
- Increased number of Secure sockets with a new ability to upgrade TCP socket to secure during connection
- IPv6 addressing scheme
- Return to factory defaults or Factory image
- Bundle protection to allow keeping the system integrity during OTA
- Security aspects:
  - Secure File System
  - Cloning protection
  - Software tamper detection
  - Trusted root certificate catalog for authentication
  - Access control using tokens
  - Other utilities such as Secure Content delivery

The document intends to guide users who already have experience with the CC3100/CC3200 devices and need to port their software to the new CC3120/CC3220 family of devices. The document describes the new SDK structure and highlights the software components that require attention including host driver, OS layers, board drivers, and external libraries.

2 SDK structure

2.1 Overview

The new SDK structure for CC3120/CC3220 intends to create a common software development platform across all TI chipsets. Its main target is to emphasize the ease of use, and specifically highlight:

- New folder structure which is easy to access and follow through
- Same look and feel for all documents and modules
- Simple migration between TI MCUs
- Simple migration between operating systems

2.2 High level view of SDK

Figure 1 illustrates the top level SDK structure. The SDK software abstracts the specific hardware (in this case: CC3120 / CC3220) to the application.

The hardware abstraction layer main target is to simplify access to device modules. Each device or family of devices implements its device specific implementation. In the previous SDK, it was also known as the driverlib module.

The RTOS layer implements the operating system. SDK includes implementations for SYS/BIOS (TIRTOS) and FreeRTOS. In case a different OS is desired, its implementation goes under the RTOS block. POSIX layer is used for OS abstraction. If a new OS is implemented, it needs to fit the POSIX layer. DPL layer (Driver Porting Layer) is used by the Drivers layer whenever OS services are required. The DPL layer abstracts the drivers similarly to how the POSIX layer abstracts the RTOS kernel functionality used by the application.

The Drivers layer is platform independent. This layer bridges between the platform dependent HAL layer and the application layer. This layer is common to all TI family of devices. In case OS services are required, the Drivers layer uses the DPL block.

The Middleware layer implements common libraries that may be used by the application. These components add additional functionality on top of the hardware-specific drivers. Examples of such common libraries include the graphic grlib, file system fatfs, and so forth.

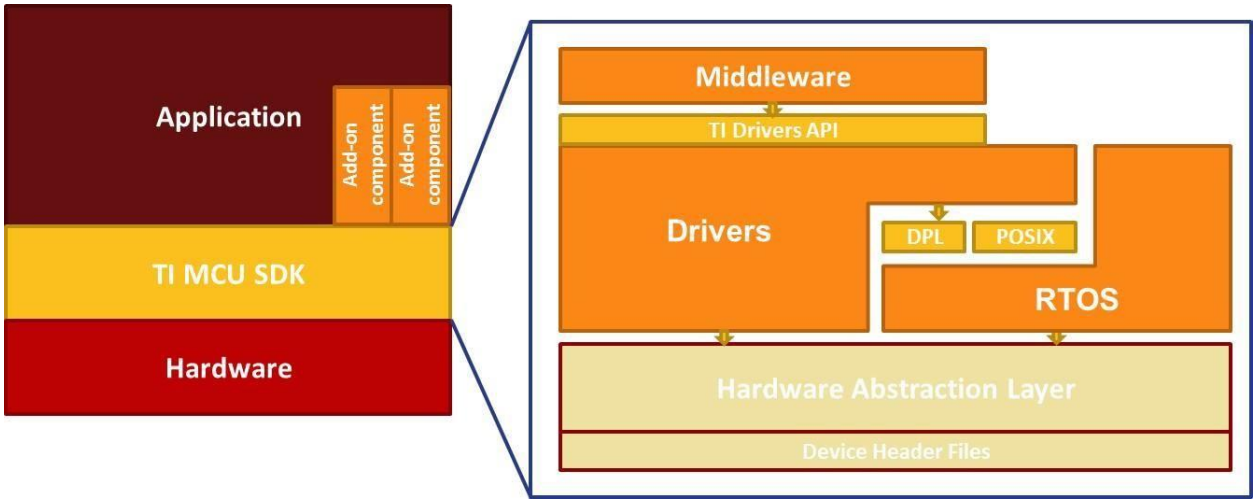

Figure 1: Top Level SDK Structure

2.3 Top level folder structure

Figure 2 illustrates the top level folder structure. *OS* directory includes SYS/BIOS (TIRTOS) implementation as well as FreeRTOS implementation. *Source* directory includes the *third_party* directory for middleware implementation and *ti* directory mainly for Drivers (*/source/ti/drivers* directory) and HAL layers (*/source/ti/devices/cc32xx/driverlib* directory). It also includes *net* directory for external libraries. Additional libraries may be added in future releases. For reference, the Wi-Fi host driver is located as one of the many drivers under /source/ti/drivers/net/wifi directory. The *examples* directory is detailed in section 2.4.

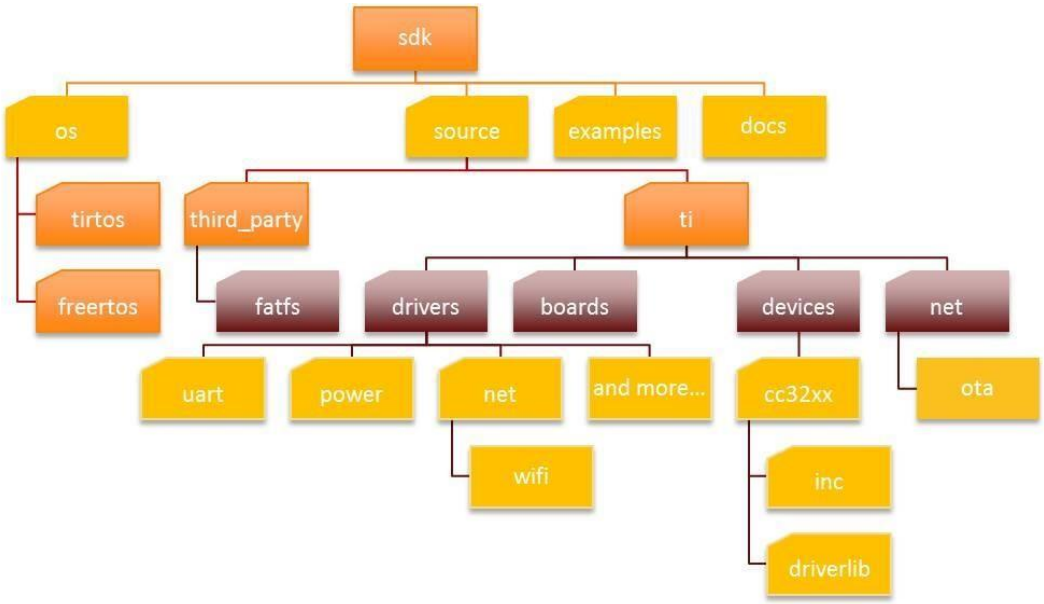

Figure 2: Top level folder structure

2.4 Examples folder structure

Figure 3 illustrates the *examples* folder structure. Every board flavor has a corresponding directory. As illustrated, CC3220S and CC3220SF implementations for the CC3220 secured and CC3220 secured flash devices are included. Each board has a *sysbios* directory for TIRTOS examples, *drivers* directory for drivers examples, and *demos* directory for a few full demos examples.

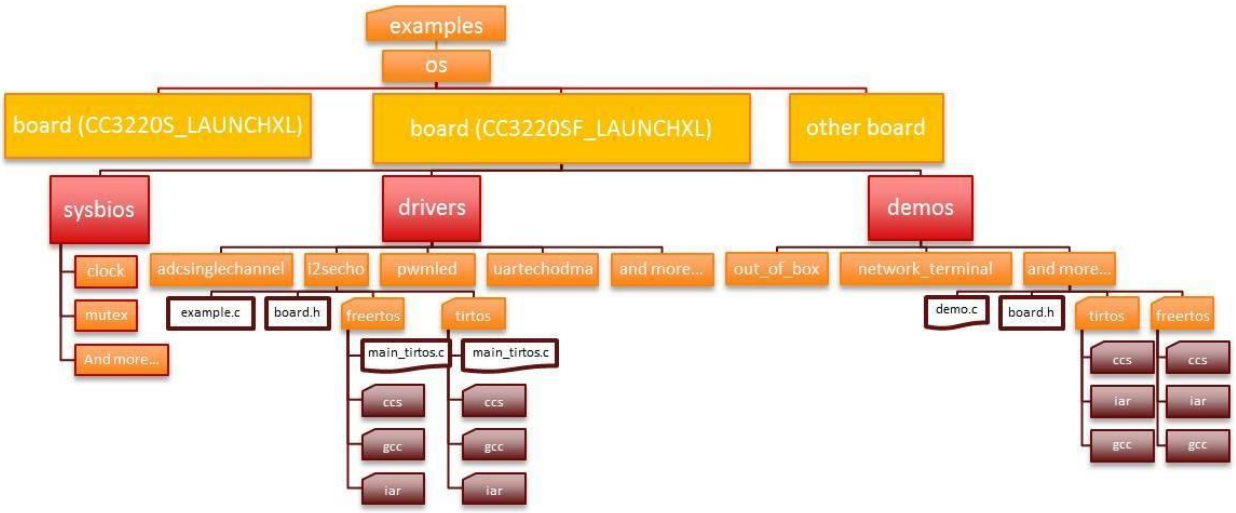

Figure 3: Examples folder structure

3 Host Driver

3.1 Overview

The new Host Driver adds support for the new features and capabilities. Most of the APIs remain similar but not all of them are identical to those used in CC3100/CC3200 host driver. The changes include modifications to function parameters (type, meaning or values), return code and types and new naming conventions.

Additionally, there are a few changes in device behavior or defaults that also require attention when moving to CC3120/CC3220 devices.

3.2 SimpleLink™ Device Behavior and Defaults Changes

The following list outlines key changes to CC3120/CC3220 SimpleLink™ devices that require attention. CC3100/CC3200 devices are marked as R1 denoting 1st generation devices while CC3120/CC3220 devices are marked as R2 denoting 2nd generation devices:

- **Device Default**
  - **Wi-Fi Mode**
    - **R1**: STA
    - **R2**: AP
  - **Regulatory Domain**
    - **R1**: US (channel s 1-11)
    - **R2**: EU (channels 1-13)
- **Security related**
  - **Behavior while the SFLASH is empty:**
    - **R1**: Start to run with some default settings
    - **R2**: Device enters into a “lock” state and waits to be programmed. In this state all APIs are blocked excluding those APIs intended for programming the device or return to factory image or default
  - **SFLASH Corruption**
    - **R1**: No clear indication, system behavior is not predicted
    - **R2**: System is locked. The host gets an event that allows it to be able only to reprogram the system or try to return to factory image
  - **Certificate files**
    - **R1**: Certificates were located under /sys/cert/. Only der format supported
    - **R2**: Certificates could be stored anywhere. Both der and pem formats supported. Enterprise certificates still need to be stored under predefined locations:
      - "/sys/cert/ca.pem"
      - "/sys/cert/client.pem"
      - "/sys/cert/private.key"
- **HTTP Web Server**

  - **R1**: HTML format
  - **R2**: Improved web pages based on rest APIs
- **IPv6**
  - **R1**: Not supported
  - **R2:** IPv6 supported, disabled by default
    - Supported only in STA mode
    - In Host driver under user.h header file, ensure to #define SL_SUPPORT_IPV6
    - Enable at least local address (global is optional) – see netcfg.h header file for more details

    IfBitmap = SL_NETCFG_IF_IPV6_STA_LOCAL | SL_NETCFG_IF_IPV6_STA_GLOBAL;
    sl_NetCfgSet(SL_NETCFG_IF,SL_NEFCFG_IF_STATE,sizeof(IfBitmap),&IfBitmap);
- **File System**
  - **R1**: Support up to 100 user files
  - **R2**: Support up to 180 user files
- **CC3100 ForceAP GPIO (GPIO13):**
  - **R1**: Force the system to wake-up in AP mode regardless other settings
  - **R2**: Removed
- **Trusted root-certificate catalog:**
  - **R1**: NA
  - **R2**: All certificate authorities should reside in the list of valid CAs as part of the certificate catalog file. Some of the file system security options require the CA to be part of the list otherwise it might be interpreted as a security alert. Secure sockets could be used without it but consequently, a warning might be sent to the Host. The Image Creator installs the certificate catalog by default.
- **IPv4 Addressing**
  - **LLA**
    - **R1**: When DHCP client is enabled and there is no DHCP server available, the device does not have a valid IPv4 address, not being able to use the network
    - **R2**: On timeout of trying to acquire an IPv4 address from a DHCP server, the system self-acquires an IP address using LLA protocol. This IP is limited to be used in a local network only. The host should be aware of this possibility.
  - **Default IP settings in AP mode:**
    - **R1**: 192.168.1.1
    - **R2**: 10.123.45.1
- **API Errors codes**
  - **R1:** Each function implements its own error codes. Same error codes could repeat in different functions
  - **R2:** Unique errors per function. All error definition available in a new header file

3.3 API changes

API modifications are required when porting a working application from CC3100/CC3200 to CC3120/CC3220. The type of change is either API addition, API removal, API change, parameters extension or return value change. It can also be a combination of changes. New naming conventions for functions and variables are also used in order to align with unified coding conventions across all product families. Table 1 lists all API changes.

API name	Silo	Change type	Change description
sl_start	device	Return code	Extended return codes in case of error. For example, error of calibration, corrupted FS, device get locked, and so forth
sl_DeviceSet	device	API change Return code	API was named *sl_DevSet* *SL_DEVICE_GENERAL_PERSISTENT* option added. persistent system configuration option saves configuration in the FS (valid after reset) The *SlDateTime_t* members renamed *ConfigLen* value was changed from _i8 to _i16 Return value was changed from _i32 to _i16
sl_DeviceGet	device	API change Return code	API was named *sl_DevGet* The *SlDateTime_t* members renamed *pConfigLen* value was changed from _u8* to _u16* Return value was changed from _i32 to _i16
sl_DeviceEventMaskGet	device	API change	In R1 the API was named *sl_EventMaskGet* Event class removed: *SL_EVENT_CLASS_GLOBAL* Event class added: *SL_DEVICE_EVENT_CLASS_NETUTIL* Event mask bitmap changed for WLAN and NETAPP classes
sl_DeviceEventMaskSet	device	API change	API was named *sl_EventMaskSet* Event class removed: *SL_EVENT_CLASS_GLOBAL* Event class added: *SL_DEVICE_EVENT_CLASS_NETUTIL* Event mask bitmap changed for WLAN and NETAPP classes
sl_DeviceUartSetMode	device	API change	API was named *sl_UartSetMode* *pUartParams* type changes from *SlDeviceUartIfParams_t** to *SlUartIfParams_t**
sl_FsClose	fs	API change	Parameters: *pCeritificateFileName*, *pSignature* and *SignatureLen* are supported
sl_FsOpen	fs	API change Return code	pFileHandle parameter removed file handle returns as part of return code (together with error codes in case of failure) pToken parameter is supported
sl_FsGetInfo	fs	API change	Token parameter is supported
sl_FsDel	fs	API change	Token parameter is now supported
sl_FsGetFileList	fs	New API	New API for file system listing
sl_FsProgram	fs	New API	New API for device programming
sl_FsCtl	fs	New API	New API to control various file system operations. For example, commit or rollback a file or a bundle, return to factory image or default, get storage info, and so forth
sl_NetAppDnsGetHostByService	netapp	Return code	Return value was changed from _i32 to _i16
sl_NetAppGetServiceList	netapp	Parameters extended	IPv6 options added: *SL_NET_APP_FULL_SERVICE_WITH_TEXT_IPV6_TYPE* , *SL_NET_APP_FULL_SERVICE_IPV6_TYPE*, *SL_NET_APP_SHORT_SERVICE_IPV6_TYPE*
sl_NetAppMDNSUnRegisterService	netapp	API change	Parameter *Options* added to the API
sl_NetAppMDNSRegisterService	netapp	Parameters extended	Parameter *Options* is extended: *SL_NETAPP_MDNS_IPV6_IPV4_SERVICE* *SL_NETAPP_MDNS_IPV6_ONLY_SERVICE* *SL_NETAPP_MDNS_OPTION_UPDATE_TEXT* *SL_NETAPP_MDNS_OPTIONS_IS_NOT_PERSISTENT*
sl_NetAppPing	netapp	API change	API was named *sl_NetAppPingStart* Structures renamed to *SlNetAppPingReport _t* and *SlNetAppPingCommand_t*

sl_NetAppGet	netapp	Parameters extended Return code	Return value was changed from _i32 to _i16 Option names have been changed DNS Option added: *SL_NETAPP_DNS_CLIENT_TIME* Http options added: *SL_NETAPP_HTTP_SECONDARY_PORT_NUMBER* *SL_NETAPP_HTTP_SECONDARY_PORT_ENABLE* *SL_NETAPP_HTTP_PRIMARY_PORT_SECURITY_MODE*
sl_NetAppSet	netapp	Parameters extended Return code	Return value was changed from _i32 to _i16 Option names have been changed DNS Option added: *SL_NETAPP_DNS_CLIENT_TIME* Http options added: *SL_NETAPP_HTTP_SECONDARY_PORT_NUMBER* *SL_NETAPP_HTTP_SECONDARY_PORT_ENABLE* *SL_NETAPP_HTTP_PRIMARY_PORT_SECURITY_MODE* *SL_NETAPP_HTTP_PRIVATE_KEY_FILENAME* *SL_NETAPP_HTTP_DEVICE_CERTIFICATE_FILE_NAME* *SL_NETAPP_HTTP_CA_CERTIFICATE_FILENAME*
sl_NetAppRecv	netapp	New API	New API for retrieving data from the network processor following a Netapp request event
sl_NetAppSend	netapp	New API	New API for sending Netapp response or data following a Netapp request event
sl_NetAppGet	netapp	API change	Return value was changed from _i32 to _i16
sl_NetCfgGet	netcfg	API change Parameters extended Return code	Parameter ConfigId changed from _u8 to _u16 Parameter pConfigOpt changed from _u8* to _u16* Parameter pConfigLen changed from _u8* to _u16* Return value was changed from _i32 to _i16
sl_NetCfgSet	netcfg	API change Parameters extended Return code	Parameter ConfigId changed from _u8 to _u16 Parameter ConfigOpt changed from _u8 to _u16 Parameter ConfigLen changed from _u8 to _u16 Return value was changed from _i32 to _i16 The ConfigId & ConfigOpt have been extended
sl_SetSockOpt	socket	Parameters extended	Socket options added: *SL_SO_KEEPALIVETIME* *SL_SO_RX_NO_IP_BOUNDARY* *SL_SO_SECURE_ALPN* *SL_SO_PHY_TX_INHIBIT_THRESHOLD* *SL_SO_PHY_TX_TIMEOUT* *SL_SO_PHY_ALLOW_ACKS* *SL_SO_LINGER* *SL_SO_SECURE_EXT_CLIENT_CHLNG_RESP* *SL_SO_SECURE_DOMAIN_NAME_VERIFICATION* Socket options changed: *SL_SO_SECURE_MASK*
sl_NetUtilCmd	netutil	New API	New API for performing utilities-related commands
sl_NetUtilGet	netutil	New API	New API for getting configurations of utilities
sl_NetUtilSet	netutil	New API	New API for setting configurations of utilities
sl_WlanConnect	wlan	Parameters extended	*pSecParams* changed to *SlWlanSecParams_t* type, its members changed and SL_WLAN_SEC_TYPE_WEP_SHARED added *pSecExtParams* changed to *SlWlanSecParamsExt_t*
sl_WlanProfileAdd	wlan	Parameters extended	*pSecParams* changed to *SlWlanSecParams_t* type, its members changed and SL_WLAN_SEC_TYPE_WEP_SHARED added *pSecExtParams* changed to *SlWlanSecParamsExt_t*
sl_WlanProfileGet	wlan	Parameters extended	*pSecParams* changed to *SlWlanSecParams_t* type, its members changed and SL_WLAN_SEC_TYPE_WEP_SHARED added *pSecExtParams* changed to *SlWlanSecParamsExt_t*
sl_WlanPolicySet	wlan	Parameters extended	Policy types renamed. For connection policy, o*pen* policy removed and *autoSmartConfig* renamed to *autoProvisioning*
sl_WlanPolicyGet	wlan	Parameters extended API change	Policy types renamed. For connection policy, o*pen* policy removed and *autoSmartConfig* renamed to *autoProvisioning* *Policy* argument has changed to pointer type from _u8

sl_WlanGetNetworkList	wlan	Parameters extended	*Sl_WlanNetworkEntry_t* instead of *SlWlanNetworkEntry_t.* *Sl_WlanNetworkEntry_t* members renamed and channel member added
sl_WlanProvisioning	wlan	New API	New API added in R2 instead of *sl_WlanSmartConfigStart* and *sl_WlanSmartConfigStop*
sl_WlanSmartConfigStart	wlan	API replaced	Replaced by *sl_WlanProvisioning*
sl_WlanSmartConfigStop	wlan	API replaced	Replaced by *sl_WlanProvisioning*
sl_WlanGet	wlan	Parameters extended	*SL_WLAN_RX_FILTERS_ID* added *SL_WLAN_CFG_AP_ACCESS_LIST_ID* added *SL_WLAN_CFG_AP_ID* options added: *SL_WLAN_AP_OPT_MAX_STATIONS* *SL_WLAN_AP_OPT_MAX_STA_AGING* *SL_WLAN_AP_ACCESS_LIST_NUM_ENTRIES* *SL_WLAN_CFG_AP_ACCESS_LIST_ID*
sl_WlanSet	**wlan**	**Parameters extended**	*SL_WLAN_RX_FILTERS_ID* added *SL_WLAN_CFG_AP_ID* options added: *SL_WLAN_AP_OPT_MAX_STATIONS* *SL_WLAN_AP_OPT_MAX_STA_AGING* *SL_WLAN_AP_ACCESS_LIST_MODE* *SL_WLAN_AP_ACCESS_LIST_ADD_MAC* *SL_WLAN_AP_ACCESS_LIST_DEL_MAC* *SL_WLAN_AP_ACCESS_LIST_DEL_IDX* *SL_WLAN_CFG_GENERAL_PARAM_ID* options added: *SL_WLAN_GENERAL_PARAM_OPT_SUSPEND_PROFILES* *SL_WLAN_GENERAL_PARAM_DISABLE_ENT_SERVER_AUTH*

Table 1: Host Driver API Changes

NOTE: To reduce the initial porting effort, you may refer to the sample code provided below (see I-1. Simplelink API Porting Code) that converts the old CC3X00 API to the updated CC3X20 interface (it mostly includes the updates as defined by the table above).
For future compatibility, it is recommended to update the original application code so it directly uses the latest Simplelink API. Although the provided code can be used as is, it should be referred as an example for the required code adaptation.
Note also that the code cover most of the API changes but not all of them. Some of the changes would still cause compilation errors and would need to be handled manually. The code only cover backward compatibilities, there are new features and parameters values in the new release that must be handled specifically by the application (example of usage can be found in the CC3X20 SDK demo code).

3.4 Miscellaneous

In additional to the changes above, the following additional changes were made to the new Host Driver:

- *netutil.c*: new module providing network utilities such as secure content delivery.
- *errors.h*: new header file added to concentrate all errors types in the driver.
- *user.h*: the following optional async event user's callbacks were added:
  - *slcb_GetTimestamp*: enables the host driver to detect timeouts and errors during its communication with the network processor.
  - *slcb_NetAppRequestHdlr*: enables the application to respond to resource requests by the network processor during http requests.
  - *slcb_NetAppRequestMemFree*: enables the host driver to free resources which were dynamically allocated by the application. If not implemented, static allocation is assumed.
  - *slcb_DeviceFatalErrorEvtHdlr*: enables the application to receive fatal error events which occur in the network processor or Host Driver, and require device restart. In previous version, these kinds of events have been notified by the slcb_DeviceGeneralEvtHdlr callback.

4 Operating System

4.1 Overview

The SDK for CC3100/C3200 included an *osi.h* header file which declared all the supported OS related utilities as well as *osi_tirtos.c* and *osi_freertos.c* for specific SYS/BIOS (TIRTOS) and FreeRTOS implementations.

The SDK for CC3120/CC3220 includes a POSIX layer which implements the operating system. The SDK includes implementations for SYS/BIOS (TIRTOS) and FreeRTOS. In case a different OS is desired, it needs to be implemented such that its API matches the POSIX layer for OS abstraction.

POSIX implementation for FreeRTOS is located under:

/os/freertos/posix

POSIX implementation for SYS/BIOS (TIRTOS) is located under:

/os/tirtos/packages/ti/sysbios/posix

NOTE: To reduce the initial porting effort, you may refer to the abstraction code provided here (see I-2. OS API Porting Code) that converts the obsolete OSI interface to POSIX calls. For future compatibility (and to eliminate redundant code), it is recommended to update the original application code so it directly uses the POSIX API. Although the provided code can be used as is, it should be referred as an example for the required code adaptation.

The following paragraphs describe the typical configuration of all OS related elements using the POSIX APIs. Browse the POSIX documentation in case a special OS configuration is desired.

4.2 Threads

Threads creation is composed of 2-steps procedure, setting the thread attributes and creating the thread. Thread attributes are configured using the *pthread_attr* APIs and thread creation is configured using the *pthread* APIs.

```
int pthread_attr_init(pthread_attr_t *attr)
int pthread_attr_destroy(pthread_attr_t *attr)
```

pthread_attr_init() initializes the *pthread_attr_t* object pointed to by *attr* with default values. The stack size is initialized to *Task_defaultStackSize* and the priority to 1. The other *pthread_attr_* functions can be used to change the defaults.

pthread_attr_destroy() destroys the *pthread_attr_t* object pointed to by *attr.*

To modify the stack size, *pthread_attr_setstack()* needs to be used.

```
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize)
```

To modify the thread priority, *pthread_attr_setstack()* needs to be used.

```
int pthread_attr_setschedparam(pthread_attr_t *attr,  const struct sched_param *schedparam)
```

To create a thread, *pthread_create()* needs to be used.

```
int pthread_create(pthread_t *newthread, const pthread_attr_t *attr,
void * (*startroutine)(void *), void *arg)
```

This function starts a new thread running the *startroutine* function, with the parameter *arg* passed to *startroutine*. If *attr* is NULL, the thread is created with default pthread attributes.

4.3 Semaphores

The following semaphore APIs are supported:

```
int   sem_destroy(sem_t *sem);
int   sem_getvalue(sem_t *sem, int *value);
int   sem_init(sem_t *sem, int pshared, unsigned value);
int   sem_post(sem_t *sem);
int   sem_timedwait(sem_t *sem, const struct timespec *abstime);
int   sem_trywait(sem_t *sem);
int   sem_wait(sem_t *sem);
```

sem_wait() waits forever on a semaphore where as *sem_timedwait()* waits for a specific timeout. *sem_timedwait()* API blocks only until the specified time has been reached or the semaphore becomes available, whichever happens first. Note that the time passed to the API is an absolute time, not a relative timeout. The *timespec* structure is defined in *types.h*:

```
struct timespec {
time_t  tv_sec;   /* Seconds */
long   tv_nsec;  /* Nanoseconds */
};
```

To fill in the *timespec* object, *clock_gettime()* API needs to be used first.

```
clock_gettime(0, &abstime);
```

Add the relative timeout value to *abstime*. For example:

```
abstime.tv_sec++;  /* A one second timeout */
or
abstime.tv_nsec = 1000000;  /* A one msec timeout */
```

4.4 Mutex

A mutex (for "mutual exclusion") is a data structure used to allow multiple threads to share a resource without accessing it simultaneously. Upon mutex creation, its attributes can be created. The

*pthread_mutexattr_* APIs are used for setting and getting the attributes to be used when a mutex is created. They do not modify the attributes of a mutex that has already been created.

The default attributes if not set indicate that subsequent calls to lock the mutex will block, until the owner releases it. If the owner holds the mutex and tries to lock it again without first releasing it, a deadlock situation will occur.

To initialize or de-initialize a mutex:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex)
int pthread_mutex_init(pthread_mutex_t *mutex,
const pthread_mutexattr_t *attr)
```

attr can be NULL for default attributes.

To lock a mutex:

```
int pthread_mutex_lock(pthread_mutex_t *mutex)
int pthread_mutex_timedlock(pthread_mutex_t *mutex,
const struct timespec *abstime)
int pthread_mutex_trylock(pthread_mutex_t *mutex)
```

The function *pthread_mutex_lock()* blocks until the mutex is acquired by the calling thread. The non-blocking function, *pthread_mutex_trylock()*, returns immediately, and if the mutex is available, the calling thread will now own it.

The *pthread_mutex_timedlock()* API blocks only until the specified time has been reached or the mutex becomes available, whichever happens first. Note that the time passed to *pthread_mutex_timedlock()* is an absolute time, not a relative timeout. The *timespec* structure is defined in *types.h*:

```
struct timespec {
time_t  tv_sec;   /* Seconds */
long   tv_nsec;  /* Nanoseconds */
};
```

To fill in the *timespec* object, *clock_gettime()* API needs to be used first.

```
clock_gettime(0, &abstime);
```

Add the relative timeout value to *abstime*. For example:

```
abstime.tv_sec++;  /* A one second timeout */
or
abstime.tv_nsec = 1000000;  /* A one msec timeout */
```

To unlock a mutex:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex)
```

4.5 Message Queues

To create or open an existing message queue:

```
mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr);
```

mq_attr structure includes flags (as for non-blocking mode), message size and maximum number of messages in the queue.

To close an existing message queue:

```
int mq_close(mqd_t mqdes)
```

To send an element to or receive an element from message queue:

```
long mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio)

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio)
```

To send an element to or receive an element from message queue and wait on timeout:

```
long mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
unsigned int *msg_prio, const struct timespec *abstime)

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio, const struct
timespec *abstime)
```

These APIs block only until the specified time has been reached or a queue element becomes available, whichever happens first. Note that the time passed to *pthread_mutex_timedlock()* is an absolute time, not a relative timeout. The *timespec* structure is defined in *types.h*:

```
struct timespec {
time_t  tv_sec;   /* Seconds */
long   tv_nsec;  /* Nanoseconds */
};
```

To fill in the *timespec* object, *clock_gettime()* API needs to be used first.

```
clock_gettime(0, &abstime);
```

Add the relative timeout value to *abstime*. For example:

```
abstime.tv_sec++;  /* A one second timeout */
or
abstime.tv_nsec = 1000000;  /* A one msec timeout */
```

4.6 Timers

The following timer APIs are supported:

```
int timer_create(clockid_t clockid, struct sigevent *evp, timer_t *timerid);
int timer_delete(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *its);
int timer_settime(timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue);
```

The *timer_create()* function creates a new timer object. A handle to the newly created timer is stored in the *timerid* parameter. *clockid* parameter can be set to *CLOCK_MONOTONIC*.

```
typedef struct sigevent {
int sigev_notify; int sigev_signo; sigval sigev_value;
void (*sigev_notify_function)(sigval val);
pthread_attr_t *sigev_notify_attributes;
} sigevent;
```

The *evp* parameter points to a *sigevent* structure, and must not be NULL, since it contains the notification function that is called when the timer expires. For SYS/BIOS only, since it does not have support for signals, the *sigev_signo* parameter in the structure is not used. However, the SYS/BIOS implementation of timer allows the application to specify whether the notification function is called from Swi context or from Task context. If the *sigev_notify* field is set to *SIGEV_SIGNAL*, the notification will be called from a Swi. In this case, the notification function must not make any blocking calls.

The following APIs get and set a timer's expiration.

```
int timer_gettime(timer_t timerid, struct itimerspec *its);
int timer_settime(timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue);
```

Itimerspec is as follows:

```
struct itimerspec {
struct timespec it_interval; /* Timer interval */
struct timespec it_value;    /* Timer expiration */
};
```

The *timer_settime()* API can be passed either a relative time or an absolute time when the timer expires. When this API is called, the timer will be armed if the *it_value* parameter specifies a non-zero time. If *it_value* specifies a time of 0, the timer will be dis-armed. A non-zero *it_interval* will cause the timer to fire periodically at the specified interval.

4.7 Memory management

Implementation for *malloc()* or *free()* for memory management is taken from the OS flavor used. For SYS/BIOS, it is taken from TIRTOS package. For FreeRTOS, the package needs to be installed separately. The package includes the type of the heap used. For example, *heap_4.c* can be used and all memory implementations should reside in this file.

5 Board Drivers

5.1 Overview

CC3200 SDK included *driverlib* as the hardware abstraction layer. Calling to drivers was applied directly to one of the *driverlib* APIs. This design requires the developer to know the details of every peripheral, its registers, the flow and procedures of how to configure it, and so forth.

CC3220 SDK is designed differently. As described in Figure 1, the original *driverlib* still exists and is located under */ti/devices/cc32xx/driverlib*. On top of the *driverlib*, there is a new abstraction layer that is referred to TI Drivers and is located under */ti/drivers.* This layer is common to all TI family of devices. Application layer should call the TI Drivers layer instead of directly calling the *driverlib*.

In case OS services are required, the TI Drivers layer uses the DPL block (Driver Porting Layer). The DPL block for SYS/BIOS is located under */os/tirtos/packages/ti/dpl*. The DPL block for FreeRTOS is located under */os/freertos/dpl*. In case a different RTOS is desired, its corresponding DPL block needs to be implemented.

5.2 Programming model

Next paragraphs describe a set of discrepancies between drivers' implementation and usage on CC3200 SDK and CC3220 SDK.

5.2.1 Driver Configuration and initialization

5.2.1.1 CC3200 SDK

Driver configuration is implemented in dedicated APIs according to the peripheral and is located in interface files under */example/common* directory. The name convention is <*peripheral*>*_if.c/h*; usually it is part of the initialization function.

5.2.1.2 CC3220 SDK

Driver configuration is implemented in a board specific file according to the board flavor. For CC3220S device it is *CC3220S_LAUNCHXL.c/h* and *board.h*, and can be found under */source/ti/boards/CC3220S_LAUNCHXL* directory. For CC3220SF device it is *CC3220SF_LAUNCHXL.c/h* and *board.h*, and can be found under */source/ti/boards/CC3220SF_LAUNCHXL* directory.

These files include configuration and setup for all peripherals. Each peripheral includes structures which include an object for TI drivers internal usage (application layer should not alter it), HW attributes configuration and functions table initialized to peripheral specific operations.

For example:

```
/*
*  ============================== Watchdog ==============================
*/
#include <ti/drivers/Watchdog.h>

#include <ti/drivers/watchdog/WatchdogCC3200.h>

WatchdogCC3200_Object watchdogCC3220SObjects[CC3220S_LAUNCHXL_WATCHDOGCOUNT];

const WatchdogCC3200_HWAttrs watchdogCC3220SHWAttrs[CC3220S_LAUNCHXL_WATCHDOGCOUNT]
=
{
  {
    .baseAddr = WDT_BASE,
    .intNum = INT_WDT,
    .intPriority = (~0),
    .reloadValue = 80000000 // 1 second period at default CPU clock freq
  }
};

const Watchdog_Config Watchdog_config[CC3220S_LAUNCHXL_WATCHDOGCOUNT] = {
  {
    .fxnTablePtr = &WatchdogCC3200_fxnTable,
    .object = &watchdogCC3220SObjects[CC3220S_LAUNCHXL_WATCHDOG0],
   .hwAttrs = &watchdogCC3220SHWAttrs[CC3220S_LAUNCHXL_WATCHDOG0]
  }
};

const uint8_t Watchdog_count = CC3220S_LAUNCHXL_WATCHDOGCOUNT;
```

Functions for peripherals initialization is located in the *board.h* header file and should be invoked from application layer.

5.2.2 Set of APIs

5.2.2.1 CC3200 SDK

The set of drivers APIs pointing to the *driversLib* is located under *<u>/driverlib</u>* directory. Another layer is implemented according to the peripheral and is located in interface files under *<u>/example/common</u>* directory. The name convention is <*peripheral*>*_if.c/h*.

Application layer may directly call the *driverLib* APIs or use the interface files.

5.2.2.2 CC3220 SDK

The set of drivers APIs pointing to TI drivers is located under *<u>/source/ti/drivers</u>* directory. Each peripheral has a C function following its name which implements the supported operations. Using this model, some

device-specific features that are less typical, are not available. The motivation is that TI Drivers abstract the common features rather than providing feature-completeness.

In a different programming model, it is possible to directly interact with the *driverLib* layer. The goal of this model is to provide most (if not all) of the features of the device or peripheral at an abstracted API level. Since APIs and features are closely tied to the hardware, it may not be easy to migrate the application code from one MCU to another.

The TI Drivers include files per peripheral in its root directory as well as directory per peripheral. All APIs used by the application layer can be found in the root directory files. These APIs further call similar APIs that are implemented under the directory per peripheral.

For example, watchdog includes the following APIs in *watchdog.c*:

```
void Watchdog_clear(Watchdog_Handle handle)
void Watchdog_close(Watchdog_Handle handle)
int Watchdog_control(Watchdog_Handle handle, unsigned int cmd, void *arg)
void Watchdog_init(void)
Watchdog_Handle Watchdog_open(unsigned int index, Watchdog_Params *params)
void Watchdog_Params_init(Watchdog_Params *params)
void Watchdog_setReload(Watchdog_Handle handle, uint32_t ticks)
uint32_t Watchdog_convertMsToTicks(Watchdog_Handle handle, uint32_t milliseconds)
```

6 External Libraries

External libraries refer to pre-compiled libraries (with sources) implementing functionalities that are required by the application.

CC3200 SDK included the following libraries:

- Under */simplelink_extlib*
  - Ota and flc for over the air update
- Under */netapps*
  - http client and server
  - json parser
  - smtp client
  - tftp client
  - xmpp client
  - mqtt client and server

CC3220 SDK currently includes Ota library only and it is located under */source/ti/net/ota*. Further libraries are planned in future releases.

6.1 OTA library

OTA design is different in CC3220 compared to CC3200 so it is not possible to port it to SimpleLink Gen2 devices. The main reasons for redesigning OTA include the new secured file system and the new boot loader.

In SimpleLink Gen2 devices, the file system enables bundle support and rollback/commit options in case new image found to be non-valid. SimpleLink Gen1 devices do not have this feature and this is why a new entity called Secondary boot loader was introduced. Additionally, SimpleLink Gen2 devices boot loader has full support for OTA procedure including watchdog.

Although Gen1 OTA implementation can be used as is on Gen2 devices, it is highly recommended to move to the new OTA design and make advantage of the new features supported by the boot loader and the file system.

Appendix I: Reference Porting Code

This chapter contains a reference C code (header and source files) that can be used to reduce the porting effort. The reference code provided below handles both the Simplelink driver's API changes (see Appendix I-1. Simplelink API Changes) and OS API changes (see Appendix I-1. Simplelink API Changes).

IMPORTANT: This code is provided to ease the initial migration efforts. It is recommended that a customer will update the original code (based on the given reference) so it directly uses the new API.

NOTE: As driverlib (board specific peripheral drivers) still part of the new CC3220 SDK (see Board Drivers), the direct calls to driverlib API would work as is. For future compatibility, it is also recommended to replace these calls with the equivalent TI drivers' ones.

Appendix I-1. Simplelink API Porting Code (sl_compat.h)

The following header file contains most of the definitions required for the Simplelink driver API migration. Few other updates must be fixed manually to resolve specific compilation error (see API changes for details on required fix). This header should be included by any source file that uses the R1 driver's API (include "sl_compat.h" just before the "simplelink.h" is included).

```
/*
 *  Copyright (C) 2016 Texas Instruments Incorporated
 *
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 *  The program may not be used without the written permission of
 *  Texas Instruments Incorporated or against the terms and conditions
 *  stipulated in the agreement under which this program has been supplied,
 *  and under no circumstances can it be used with non-TI connectivity device.
 *
 */


/*****************************************************************************/
/* Include files                                                             */
/*****************************************************************************/

#ifndef __SL_COMPAT_H__
#define __SL_COMPAT_H__


#ifdef    __cplusplus
extern "C" {
#endif

#ifdef SUPPORT_SL_R1_API

/* Updated Macro Definitions */
#define SL_DEVICE_GENERAL_CONFIGURATION                 SL_DEVICE_GENERAL
#define SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME       SL_DEVICE_GENERAL_DATE_TIME

#define SL_SCAN_SEC_TYPE_OPEN                           SL_WLAN_SEC_TYPE_OPEN
#define SL_SCAN_SEC_TYPE_WEP                            SL_WLAN_SEC_TYPE_WEP
#define SL_SCAN_SEC_TYPE_WPA                            SL_WLAN_SEC_TYPE_WPA
#define SL_SCAN_SEC_TYPE_WPA2                           SL_WLAN_SEC_TYPE_WPA_WPA2

#define FS_MODE_OPEN_WRITE                              SL_FS_WRITE
#define FS_MODE_OPEN_READ                               SL_FS_READ
```

#define SL_POLICY_CONNECTION	SL_WLAN_POLICY_CONNECTION
#define SL_POLICY_SCAN	SL_WLAN_POLICY_SCAN
#define SL_POLICY_PM	SL_WLAN_POLICY_PM
#define SL_POLICY_P2P	SL_WLAN_POLICY_P2P
#define SL_NORMAL_POLICY	SL_WLAN_NORMAL_POLICY
#define SL_BSSID_LENGTH	SL_WLAN_BSSID_LENGTH
#define SL_WLAN_CONNECT_EVENT	SL_WLAN_EVENT_CONNECT
#define SL_WLAN_DISCONNECT_EVENT	SL_WLAN_EVENT_DISCONNECT
#define SL_WLAN_SMART_CONFIG_COMPLETE_EVENT	SL_WLAN_EVENT_PROVISIONING_STATUS
#define SL_WLAN_STA_CONNECTED_EVENT	SL_WLAN_EVENT_P2P_CLIENT_ADDED
#define SL_WLAN_STA_DISCONNECTED_EVENT	SL_WLAN_EVENT_P2P_CLIENT_REMOVED
#define SL_USER_INITIATED_DISCONNECTION	SL_WLAN_DISCONNECT_USER_INITIATED
#define WLAN_AP_OPT_SSID	SL_WLAN_AP_OPT_SSID
#define WLAN_AP_OPT_CHANNEL	SL_WLAN_AP_OPT_CHANNEL
#define WLAN_AP_OPT_HIDDEN_SSID	SL_WLAN_AP_OPT_HIDDEN_SSID
#define WLAN_AP_OPT_SECURITY_TYPE	SL_WLAN_AP_OPT_SECURITY_TYPE
#define WLAN_AP_OPT_PASSWORD	SL_WLAN_AP_OPT_PASSWORD
#define WLAN_AP_OPT_MAX_STATIONS	SL_WLAN_AP_OPT_MAX_STATIONS
#define SL_SEC_TYPE_OPEN	SL_WLAN_SEC_TYPE_OPEN
#define SL_SEC_TYPE_WEP	SL_WLAN_SEC_TYPE_WEP
#define SL_SEC_TYPE_WPA	SL_WLAN_SEC_TYPE_WPA
#define SL_SEC_TYPE_WPA_WPA2	SL_WLAN_SEC_TYPE_WPA_WPA2
#define WLAN_GENERAL_PARAM_OPT_STA_TX_POWER	SL_WLAN_GENERAL_PARAM_OPT_STA_TX_POWER
#define WLAN_GENERAL_PARAM_OPT_AP_TX_POWER	SL_WLAN_GENERAL_PARAM_OPT_AP_TX_POWER
#define SL_REMOVE_RX_FILTER	SL_WLAN_RX_FILTER_REMOVE
#define SL_NETAPP_IP_LEASED_EVENT	SL_NETAPP_EVENT_DHCPV4_LEASED
#define SL_NETAPP_IP_RELEASED_EVENT	SL_NETAPP_EVENT_DHCPV4_RELEASED
#define SL_NETAPP_IPV4_IPACQUIRED_EVENT	SL_NETAPP_EVENT_IPV4_ACQUIRED
#define SL_NETAPP_HTTPGETTOKENVALUE_EVENT	SL_NETAPP_EVENT_HTTP_TOKEN_GET
#define SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT	SL_NETAPP_EVENT_HTTP_TOKEN_POST
#define SL_NET_APP_HTTP_SERVER_ID	SL_NETAPP_HTTP_SERVER_ID
#define SL_IPV4_DNS_CLIENT	SL_NETCFG_IPV4_DNS_CLIENT
#define SL_MAC_ADDRESS_GET	SL_NETCFG_MAC_ADDRESS_GET
#define SL_MAC_ADDRESS_SET	SL_NETCFG_MAC_ADDRESS_SET
#define SL_IPV4_STA_P2P_CL_DHCP_ENABLE	SL_NETCFG_IPV4_DHCP_CLIENT
#define SL_SO_SECURE_FILES_DH_KEY_FILE_NAME	SL_SO_SECURE_FILES_PEER_CERT_OR_DH_KEY_FILE_NAME
#define SO_SECURE_DOMAIN_NAME_VERIFICATION	SL_SO_SECURE_DOMAIN_NAME_VERIFICATION
/*** Updatetd Structure Names ***/	
#define Sl_WlanNetworkEntry_t	SlWlanNetworkEntry_t
#define SlHttpServerEvent_t	SlNetAppHttpServerEvent_t
#define SlHttpServerResponse_t	SlNetAppHttpServerResponse_t
#define APModeStaConnected	P2PClientAdded
#define APModestaDisconnected	P2PClientRemoved
#define STAandP2PModeWlanConnected	Connect
#define STAandP2PModeDisconnected	Disconnect
#define SlSecParams_t	SlWlanSecParams_t
#define SlSecParamsExt_t	SlWlanSecParamsExt_t
#define SlGetSecParamsExt_t	SlWlanGetSecParamsExt_t
#define SlWlanProtocolInfoElement_t	SlWlanInfoElement_t
#define SlWlanProtocolSetInfoElement_t	SlWlanSetInfoElement_t
#define slWlanConnectAsyncResponse_t	SlWlanEventDisconnect_t /* ??? */
#define _WlanRxFilterOperationCommandBuff_t	SlWlanRxFilterOperationCommandBuff_t
#define slPeerInfoAsyncResponse_t	SlWlanEventP2PClientAdded_t
#define SlVersionFull	SlDeviceVersion_t
#define SlSockSecureMethod	SlSockSecureMethod_t
#define SlSockSecureMask	SlSockSecureMask_t
/*** Updatetd Structure Members Names ***/	
#define AllocatedLen	StorageSize
#define FileLen	Len
#define NonblockingEnabled	NonBlockingEnabled
#define reason	Reason

```
#define reason_code                                     ReasonCode
#define sd                                              Sd
#define type                                            Type
#define val                                             Val
#define mac                                             Mac
#define secureMask                                      SecureMask
#define secureMethod                                    SecureMethod
#define sl_tm_year                                      tm_year
#define sl_tm_mon                                       tm_mon
#define sl_tm_day                                       tm_day
#define sl_tm_hour                                      tm_hour
#define sl_tm_min                                       tm_min
#define sl_tm_sec                                       tm_sec
#define socketAsyncEvent                                SocketAsyncEvent
#define deviceDriverReport                              DeviceDriverReport
#define deviceEvent                                     DeviceEvent
#define deviceReport                                    DeviceReport
#define ipAcquiredV4                                    IpAcquiredV4
#define ipLeased                                        IpLeased
#define ipReleased                                      IpReleased
#define status                                          Status
#define rssi                                            Rssi
#define bssid                                           Bssid
#define ssid                                            Ssid
#define ssid_len                                        SsidLen
#define sec_type                                        SecurityInfo
#define ssid_name                                       SsidName
#define FilterIdMask                                    FilterBitmap

/*** Updatetd Error Names ***/
#define SL_FS_ERR_FILE_NOT_EXISTS                       SL_ERROR_FS_FILE_NOT_EXISTS
#define SL_EAGAIN                                       SL_ERROR_BSD_EAGAIN
#define SL_ENSOCK                                       SL_ERROR_BSD_ENSOCK
#define SL_INEXE                                        SL_ERROR_BSD_INEXE
#define SL_ESECCLOSED                                   SL_ERROR_BSD_ESECCLOSED
#define SL_ECLOSE                                       SL_ERROR_BSD_ECLOSE
#define SL_ESECT00MANYSSLOPENED                         SL_ERROR_BSD_ESECT00MANYSSLOPENED
#define SL_FS_FILE_HAS_NOT_BEEN_CLOSE_CORRECTLY         SL_ERROR_FS_FILE_HAS_NOT_BEEN_CLOSE_CORRECTLY
#define SL_ESEC_RSA_WRONG_TYPE_E
#define SL_ESEC_ASN_SIG_CONFIRM_E                       SL_ERROR_BSD_ESEC_ASN_SIG_CONFIRM_E
#define ROLE_STA_ERR                                    SL_ERROR_ROLE_STA_ERR
#define ROLE_AP_ERR                                     SL_ERROR_ROLE_AP_ERR
#define ROLE_P2P_ERR                                    SL_ERROR_ROLE_P2P_ERR

/*** Update Macro Functions ***/
#define SL_CONNECTION_POLICY(Auto,Fast,Open,anyP2P,autoSmartConfig)  \
                                                        SL_WLAN_CONNECTION_POLICY(Auto,Fast,anyP2P,autoSmartConfig)
#define SL_SCAN_POLICY(enable)                          SL_WLAN_SCAN_POLICY(enable, 0)
#define FS_MODE_OPEN_CREATE(_maxSize_, _flag_)          (SL_FS_CREATE | SL_FS_CREATE_MAX_SIZE( _maxSize_ ))

/*** Updatetd Function ***/
#define sl_WlanRxFilterSet(opcode, maskeSize, mask)     sl_WlanSet(SL_WLAN_RX_FILTERS_ID, opcode, mask, maskeSize)
#define sl_DevSet                                       sl_DeviceSet
#define sl_DevGet                                       sl_DeviceGet

#endif // SUPPORT_SL_R1_API

#ifdef  __cplusplus
}
#endif /*  __cplusplus */

#endif   /*  __SL_COMPAT_H__ */
```

Appendix I-2. OS API Porting Code

The following code (header and source file) contains implementation of POSIX . Most of the porting is achieved by using direct (MACRO) calls to POSIX API (within the "osi.h"), but some updates required specific handling in a source file ("osi_posix") mainly due to the following:

1. OSI API requires maintaining a context (e.g. for storing the Message queue element's size)
2. To avoid long/complicated macros (e.g. for Task creation which prepares the thread attributes through several pthread system calls).

Appendix I-2.1. OS Adaptation Header (osi.h)

This header should replace the "osi.h" used in CC3200 SDK. This supposed to be included by any source that uses OS resources.

```
//*****************************************************************************
// osi.h
//
// MACRO and Function prototypes for TI-RTOS and Free-RTOS API calls
//
// Copyright (C) 2014 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 zof 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 __OSI_H__
#define     __OSI_H__


#include <pthread.h>
#include <time.h>
#include <unistd.h>
#include <ti/sysbios/BIOS.h>

#ifdef       __cplusplus
extern "C" {
#endif

#define OSI_WAIT_FOREVER                                (0xFFFFFFFF)
```

```
#define OSI_NO_WAIT                                     (0)

#ifndef SPAWN_TASK_STACK
#define STACK_LEN                                       (2048) /*Stack Size*/
#else
#define STACK_LEN                                        (SPAWN_TASK_STACK)
#endif


#define OSI_OK                                          0
#define OSI_FAILURE                                     SL_ERROR_BSD_SOC_ERROR
#define OSI_OPERATION_FAILED                            SL_ERROR_BSD_SOC_ERROR
#define OSI_ABORTED                                     SL_ERROR_BSD_SOC_ERROR
#define OSI_INVALID_PARAMS                              SL_ERROR_BSD_EINVAL
#define OSI_MEMORY_ALLOCATION_FAILURE                   SL_ERROR_BSD_ENOMEM
#define OSI_TIMEOUT                                     SL_ERROR_BSD_ETIMEDOUT
#define OSI_EVENTS_IN_USE                               -16 //SL_ERROR_BSD_EBUSY
#define OSI_EVENT_OPEARTION_FAILURE    SL_ERROR_BSD_SOC_ERROR

typedef int OsiReturnVal_e;

typedef void* OsiMsgQ_t;

 /*!
            \brief type definition for a time value

            \note       On each porting or platform the type could be whatever is needed - integer, pointer to structure etc.
*/
//typedef unsigned int OsiTime_t;
typedef unsigned int OsiTime_t;
/*!
            \brief      type definition for a sync object container

            Sync object is object used to synchronize between two threads or thread and interrupt handler.
            One thread is waiting on the object and the other thread send a signal, which then
            release the waiting thread.
            The signal must be able to be sent from interrupt context.
            This object is generally implemented by binary semaphore or events.

            \note       On each porting or platform the type could be whatever is needed - integer, structure etc.
*/
//typedef unsigned int OsiSyncObj_t;
typedef sem_t OsiSyncObj_t;

/*!
            \brief      type definition for a locking object container

            Locking object are used to protect a resource from mutual accesses of two or more threads.
            The locking object should support re-entrant locks by a signal thread.
            This object is generally implemented by mutex semaphore

            \note       On each porting or platform the type could be whatever is needed - integer, structure etc.
*/
//typedef unsigned int OsiLockObj_t;
typedef pthread_mutex_t OsiLockObj_t;

/*!
            \brief      type definition for a spawn entry callback

            the spawn mechanism enable to run a function on different context.
            This mechanism allow to transfer the execution context from interrupt context to thread context
            or changing the context from an unknown user thread to general context.
            The implementation of the spawn mechanism depends on the user's system requirements and could varies
            from implementation of serialized execution using single thread to creating thread per call

            \note       The stack size of the execution thread must be at least of TBD bytes!
*/
typedef short (*P_OSI_SPAWN_ENTRY)(void* pValue);

typedef void (*P_OSI_EVENT_HANDLER)(void* pValue);
```

```
typedef void* (*P_OSI_TASK_ENTRY)(void* pValue);

typedef void (*P_OSI_INTR_ENTRY)(void);

typedef pthread_t OsiTaskHandle;

int osi_ThreadCreate(void *(*pEntry)(void *),unsigned short stackDepth,void *pvParameters,unsigned long uxPriority,pthread_t *pThread);

/*!
        \brief      This function registers an interrupt in NVIC table

        The sync object is used for synchronization between different thread or ISR and
        a thread.

        \param      iIntrNum   -        Interrupt number to register
        \param      pEntry     -        Pointer to the interrupt handler

        \return upon successful creation the function should return 0
                            Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
OsiReturnVal_e osi_InterruptRegister(int iIntrNum,P_OSI_INTR_ENTRY pEntry,unsigned char ucPriority);

/*!
        \brief      This function De-registers an interrupt in NVIC table

        \param      iIntrNum   -        Interrupt number to register
        \param      pEntry     -        Pointer to the interrupt handler

        \return upon successful creation the function should return Positive number
                            Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
void osi_InterruptDeRegister(int iIntrNum);


/*!
        \brief      This function creates a sync object

        The sync object is used for synchronization between different thread or ISR and
        a thread.

        \param      pSyncObj   -        pointer to the sync object control block

        \return upon successful creation the function should return 0
                            Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_SyncObjCreate(OsiSyncObj_t* pSyncObj);
#define osi_SyncObjCreate(pSyncObj)                     sem_init(pSyncObj, 0 , 0)

/*!
        \brief      This function deletes a sync object

        \param      pSyncObj   -        pointer to the sync object control block

        \return upon successful deletion the function should return 0
                            Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_SyncObjDelete(OsiSyncObj_t* pSyncObj);
#define osi_SyncObjDelete(pSyncObj)                     sem_destroy(pSyncObj)

/*!
        \brief              This function generates a sync signal for the object.
```

```
        All suspended threads waiting on this sync object are resumed

        \param      pSyncObj  -     pointer to the sync object control block

        \return     upon successful signalling the function should return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note       the function could be called from ISR context
        \warning
*/
//OsiReturnVal_e osi_SyncObjSignal(OsiSyncObj_t* pSyncObj);
#define osi_SyncObjSignal(pSyncObj)                     sem_post(pSyncObj)

/*!
        \brief      This function generates a sync signal for the object.
                        from ISR context.

        All suspended threads waiting on this sync object are resumed

        \param      pSyncObj  -     pointer to the sync object control block

        \return     upon successful signalling the function should return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note       the function is called from ISR context
        \warning
*/
//OsiReturnVal_e osi_SyncObjSignalFromISR(OsiSyncObj_t* pSyncObj);
#define osi_SyncObjSignalFromISR(pSyncObj)              sem_post(pSyncObj)

/*!
        \brief      This function waits for a sync signal of the specific sync object

        \param      pSyncObj  -     pointer to the sync object control block
        \param      Timeout   -     numeric value specifies the maximum number of mSec to
                                    stay suspended while waiting for the sync signal
                                    Currently, the simple link driver uses only two values:
                                        - OSI_WAIT_FOREVER
                                        - OSI_NO_WAIT

        \return upon successful reception of the signal within the timeout window return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_SyncObjWait(OsiSyncObj_t* pSyncObj , OsiTime_t Timeout);
#define osi_SyncObjWait(pSyncObj, Timeout)              sem_wait_timed(pSyncObj, Timeout)

/*!
        \brief      This function clears a sync object

        \param      pSyncObj  -     pointer to the sync object control block

        \return upon successful clearing the function should return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_SyncObjClear(OsiSyncObj_t* pSyncObj);
#define osi_SyncObjClear(pSyncObj)                      sem_wait_timed(pSyncObj, 0)

/*!
        \brief      This function creates a locking object.

        The locking object is used for protecting a shared resources between different
        threads.

        \param      pLockObj  -     pointer to the locking object control block

        \return upon successful creation the function should return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
```

```
*/
//OsiReturnVal_e osi_LockObjCreate(OsiLockObj_t* pLockObj);
#define osi_LockObjCreate(pLockObj)                    pthread_mutex_init(pLockObj, NULL)

/*!
        \brief      This function deletes a locking object.

        \param      pLockObj   -        pointer to the locking object control block

        \return upon successful deletion the function should return 0
                                Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_LockObjDelete(OsiLockObj_t* pLockObj);
#define osi_LockObjDelete(pLockObj)                    pthread_mutex_destroy(pLockObj, NULL)

/*!
        \brief      This function locks a locking object.

        All other threads that call this function before this thread calls
        the osi_LockObjUnlock would be suspended

        \param      pLockObj   -        pointer to the locking object control block
        \param      Timeout             -       numeric value specifies the maximum number of mSec to
                                                                stay suspended while waiting for the locking object
                                                                Currently, the simple link driver uses only two values:
                                                                        - OSI_WAIT_FOREVER
                                                                        - OSI_NO_WAIT


        \return upon successful reception of the locking object the function should return 0
                                Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_LockObjLock(OsiLockObj_t* pLockObj , OsiTime_t Timeout);
#define osi_LockObjLock(pLockObj, Timeout)                      mutex_lock_timed(pLockObj, Timeout)

/*!
        \brief      This function unlock a locking object.

        \param      pLockObj   -        pointer to the locking object control block

        \return upon successful unlocking the function should return 0
                                Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
//OsiReturnVal_e osi_LockObjUnlock(OsiLockObj_t* pLockObj);
#define osi_LockObjUnlock(pLockObj)                     pthread_mutex_unlock(pLockObj)


/*!
        \brief      This function call the pEntry callback from a different context

        \param      pEntry              -       pointer to the entry callback function

        \param      pValue              -       pointer to any type of memory structure that would be
                                                                passed to pEntry callback from the execution thread.

        \param      flags               -       execution flags - reserved for future usage

        \return upon successful registration of the spawn the function should return 0
                                (the function is not blocked till the end of the execution of the function
                                and could be returned before the execution is actually completed)
                                Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
/*!
```

```
            \brief      This function creates a Task.

            Creates a new Task and add it to the last of tasks that are ready to run

            \param      pEntry      -           pointer to the Task Function
            \param      pcName      -           Task Name String
            \param      usStackDepth            -           Stack Size Stack Size in 32-bit long words
            \param      pvParameters            -           pointer to structure to be passed to the Task Function
            \param      uxPriority  -           Task Priority

            \return upon successful unlocking the function should return 0
                                    Otherwise, a negative value indicating the error code shall be returned
            \note
            \warning
*/


//int osi_TaskCreate(P_OSI_TASK_ENTRY pEntry,const signed char * const pcName,unsigned short usStackDepth,void *pvParameters,unsigned
long uxPriority,OsiTaskHandle *pTaskHandle);
#define osi_TaskCreate(pEntry, pcName, usStackDepth, pvParameters, uxPriority, pTaskHandle) \
            osi_ThreadCreate(pEntry, usStackDepth, pvParameters, uxPriority, pTaskHandle)


/*!
            \brief      This function Deletes a Task.

            Deletes a  Task and remove it from list of running task

            \param      pTaskHandle             -           Task Handle

            \note
            \warning
*/
//void osi_TaskDelete(OsiTaskHandle* pTaskHandle);
#define osi_TaskDelete(pTaskHandle)                     pthread_detach(pTaskHandle)

/*!
            \brief      This function used to disable the tasks
            \param      - void
            \return - Key with the suspended tasks
            \note
            \warning
*/
//unsigned long osi_TaskDisable(void);
#define osi_TaskDisable()                                       SL_ERROR_BSD_EOPNOTSUPP // NOT SUPPORTED
/*!
            \brief      This function used to enable all tasks
            \param      unsigned long
            \return - void
            \note
            \warning
*/
//void osi_TaskEnable(unsigned long);
#define osi_TaskEnable()                                        SL_ERROR_BSD_EOPNOTSUPP // NOT SUPPORTED


/*!
            \brief      This function call the pEntry callback from a different context

            \param      pEntry                  -           pointer to the entry callback function

            \param      pValue                  -           pointer to any type of memory structure that would be
                                                                    passed to pEntry callback from the execution thread.

            \param      flags                   -           execution flags - reserved for future usage

            \return upon successful registration of the spawn the function should return 0
                                    (the function is not blocked till the end of the execution of the function
                                    and could be returned before the execution is actually completed)
                                    Otherwise, a negative value indicating the error code shall be returned
            \note
```

```
            \warning
*/
//OsiReturnVal_e osi_Spawn(P_OSI_SPAWN_ENTRY pEntry , void* pValue , unsigned long flags);
#define osi_Spawn(pEntry, pValue, flags)

/*****************************************************************************

This function creates a message queue that is typically used for inter thread
communication.

Parameters:

            pMsgQ               -           pointer to the message queue control block
            pMsgQName           -           pointer to the name of the message queue
            MsgSize             -           the size of the message.

                                NOTICE: THE MESSGAE SIZE MUST BE SMALLER THAN 16

            MaxMsgs             -           maximum number of messages.

Please note that this function allocates the entire memory required
for the maximum number of messages (MsgSize * MaxMsgs).

*****************************************************************************/
OsiReturnVal_e osi_MsgQCreate(OsiMsgQ_t*                    pMsgQ ,
                              char*                         pMsgQName,
                              unsigned long                 MsgSize,
                              unsigned long                 MaxMsgs);

/*****************************************************************************

This function deletes a specific message queue.
All threads suspended waiting for a message from this queue are resumed with
an error return value.

Parameters:

            pMsgQ               -           pointer to the message queue control block

*****************************************************************************/
OsiReturnVal_e osi_MsgQDelete(OsiMsgQ_t*                    pMsgQ);


/*****************************************************************************

This function writes a message to a specific message queue.

Notice that the message is copied to the queue from the memory area specified
by pMsg pointer.

------------------------------------------------------------------------
THIS FUNCTION COULD BE CALLED FROM ISR AS LONG AS THE TIMEOUT PARAMETER IS
SET TO "OSI_NO_WAIT"
------------------------------------------------------------------------

Parameters:

            pMsgQ               -           pointer to the message queue control block
            pMsg                -           pointer to the message
            Timeout             -           numeric value specifies the maximum number of mSec to stay
                                                suspended while waiting for available space for the message

*****************************************************************************/
OsiReturnVal_e osi_MsgQWrite(OsiMsgQ_t* pMsgQ, void* pMsg , OsiTime_t Timeout);


/*****************************************************************************

This function retrieves a message from the specified message queue. The
retrieved message is copied from the queue into the memory area specified by
the pMsg pointer
```

```
Parameters:

        pMsgQ       -   pointer to the message queue control block
        pMsg        -   pointer that specify the location where to copy the message
        Timeout     -   numeric value specifies the maximum number of mSec to stay
                        suspended while waiting for a message to be available

*****************************************************************************/
OsiReturnVal_e osi_MsgQRead(OsiMsgQ_t* pMsgQ, void* pMsg , OsiTime_t Timeout);

/*!
        \brief      This function starts the OS Scheduler
        \param      - void
        \return - void
        \note
        \warning
*/
//void osi_start();
#ifdef USE_FREERTOS
#define osi_start()                         vTaskStartScheduler(); /* Start the FreeRTOS scheduler */
#else
#define osi_start()                         BIOS_start()
#endif


/*!
  \brief                                    Allocates Memory on Heap
        \param      Size            -       Size of the Buffer to be allocated
  \sa
  \note
  \warning
*/
//void * mem_Malloc(unsigned long Size);
#define mem_Malloc(Size)                    malloc(Size)

/*!
  \brief                                    Deallocates Memory
        \param      pMem            -       Pointer to the Buffer to be freed
        \return void
  \sa
  \note
  \warning
*/
//void mem_Free(void *pMem);
#define mem_Free(pMem)                      free(pMem)


/*!
  \brief                                    Set Memory
        \param      pBuf            -       Pointer to the Buffer
        \param      Val                     -       Value to be set
        \param      Size            -       Size of the memory to be set
  \sa
  \note
  \warning
*/
//void  mem_set(void *pBuf,int Val, int Size);
#define  mem_set(pBuf, Val, Size)           memset(pBuf, Val, Size)

/*!
  \brief                                    Copy Memory
        \param      pDst            -       Pointer to the Destination Buffer
        \param      pSrc            -       Pointer to the Source Buffer
        \param      Size            -       Size of the memory to be copied
        \return void
  \note
  \warning
*/
//void  mem_copy(void *pDst, void *pSrc, int Size);
#define mem_copy(pDst, pSrc, Size)          memcpy(pDst, pSrc, Size)
```

```
/*!
   \brief                          Enter Critical Section
   \sa
   \note
   \warning
*/
//unsigned long osi_EnterCritical(void);
#define osi_EnterCritical()                             HwiP_disable()


/*!
   \brief                          Exit Critical Section
   \sa
   \note
   \warning
*/
//void osi_ExitCritical(unsigned long ulKey);
#define osi_ExitCritical(ulKey)                         HwiP_restore(ulKey)

/*!
            \brief      This function used to save the os context before sleep
            \param      void
            \return void
            \note
            \warning
*/
//void osi_ContextSave();
#define osi_ContextSave()                               SL_ERROR_BSD_EOPNOTSUPP // NOT SUPPORTED
/*!
            \brief      This function used to retrieve the context after sleep
            \param      void
            \return void
            \note
            \warning
*/
//void osi_ContextRestore();
#define osi_ContextRestore()                            SL_ERROR_BSD_EOPNOTSUPP // NOT SUPPORTED

/*!
            \brief      This function used to suspend the task for the specified number of milli secs
            \param      MilliSecs   -           Time in millisecs to suspend the task
            \return void
            \note
            \warning
*/
//void osi_Sleep(unsigned int MilliSecs);
#define osi_Sleep(MilliSecs)        {if(MilliSecs >= 1000) { sleep((MilliSecs+500)/1000); } else { usleep(MilliSecs * 1000); } }




/* API for SL Task*/
//OsiReturnVal_e VStartSimpleLinkSpawnTask(unsigned long uxPriority);
extern pthread_t slSpawnThread;
#define SPAWN_TASK_STACK_SIZE           2048
#define VStartSimpleLinkSpawnTask(priority)             osi_ThreadCreate(sl_Task, SPAWN_TASK_STACK_SIZE, NULL, priority, &slSpawnThread)


//void VDeleteSimpleLinkSpawnTask( void );
#define VDeleteSimpleLinkSpawnTask()                    pthread_detach(&slSpawnThread)


#ifdef  __cplusplus
}
#endif // __cplusplus

#endif
```

Appendix I-2.1. OS Adaptation Source (osi_posix.c)

The following C source file should be compiled and linked with the application code. It includes implementation of specific OS system calls that could not be ported through MACROs.

```
//*****************************************************************************
// osi_posix.c
//
// OS Adaptation implementation converting the obsolete CC3x00 OSI interface
//  CC3X20 (MCPI) POSIX implemtation (supporting both TI-RTOS and FreeRTOS).
//
// Copyright (C) 2014 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.
//
//*****************************************************************************

#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include <signal.h>
#include <unistd.h>
#include <semaphore.h>
#include <mqueue.h>
#include "osi.h"
#include "uart_term.h"
#include <ti/drivers/net/wifi/simplelink.h>  /* sl_Task */
/* RTOS header files */

#ifdef USE_FREERTOS
#else
#define SPAWN_TASK_PRIORITY           9
#endif

pthread_t slSpawnThread;


static void st_GetAbsoluteTime(OsiTime_t msecToAdd, struct timespec *pOutTM)
{
        clock_gettime(0, pOutTM);
        pOutTM->tv_sec += msecToAdd / 1000 ;
```

```
        pOutTM->tv_nsec += (msecToAdd % 1000)*1000000;

        pOutTM->tv_sec += (pOutTM->tv_nsec / 1000000000);
        pOutTM->tv_nsec = pOutTM->tv_nsec % 1000000000;
}

unsigned long GetAbsoluteTimeMS()
{
        struct timespec tm;
        clock_gettime(0, &tm);
        return (tm.tv_sec * 1000) + (tm.tv_nsec / 1000000);
}

/*!
        \brief      This function registers an interrupt in NVIC table

        The sync object is used for synchronization between different thread or ISR and
        a thread.

        \param      iIntrNum  -         Interrupt number to register
        \param      pEntry        -     Pointer to the interrupt handler e.g. INT_PRIORITY_LVL_1
        \param ucPriority  -  Priority of the interrupt

        \return upon successful creation the function should return 0
                        Otherwise, a negative value indicating the error code shall be returned
        \note
        \warning
*/
typedef struct _osiIntContext_t
{
        int                                     intNum;
        HwiP_Handle                     intHandle;
        struct _osiIntContext_t *pNext;
} osiIntContext_t;
osiIntContext_t *g_osiInterrupts = NULL;

OsiReturnVal_e osi_InterruptRegister(int iIntrNum, P_OSI_INTR_ENTRY fInterruptHdl, unsigned char ucPriority)
{
#ifdef SL_PLATFORM_MULTI_THREADED

        HwiP_Params intParams;
        HwiP_Handle intHandle;
        OsiReturnVal_e ret = OSI_FAILURE;

        //OSI_HEADER_NL();
        HwiP_clearInterrupt(iIntrNum);

        if(!fInterruptHdl)
        {
                ret = OSI_INVALID_PARAMS;
        }
        else
        {
                intParams.priority = ucPriority; //

                intHandle = HwiP_create(iIntrNum , (HwiP_Fxn)(fInterruptHdl) , &intParams);

                if(intHandle)
                {
                        osiIntContext_t *pIntContext = (osiIntContext_t*)malloc(sizeof(osiIntContext_t));
                        if(pIntContext)
                        {
                                pIntContext->intNum = iIntrNum;
                                pIntContext->intHandle = intHandle;
                                pIntContext->pNext = g_osiInterrupts;
                                g_osiInterrupts = pIntContext;

                                ret = OSI_OK;
                        }
                }
        }
        return ret;

#else
```

```
        if(!InterruptHdl)
        {
                MAP_IntDisable(INT_NWPIC);
                MAP_IntUnregister(INT_NWPIC);
                MAP_IntPendClear(INT_NWPIC);
        }
        else
        {
                MAP_IntRegister(INT_NWPIC, InterruptHdl);
                MAP_IntPrioritySet(INT_NWPIC, INT_PRIORITY_LVL_1);
                MAP_IntPendClear(INT_NWPIC);
                MAP_IntEnable(INT_NWPIC);
        }

        return 0 ;
#endif //SL_PLATFORM_MULTI_THREADED
}

/*!
        \brief      This function De registers an interrupt in NVIC table

        \param      iIntrNum   -        Interrupt number to register

        \return none
        \note
        \warning
*/
void osi_InterruptDeRegister(int iIntrNum)
{
        osiIntContext_t *pIntContext, *pPrev;

        //OSI_HEADER_NL();
        for(pIntContext=g_osiInterrupts;
                        pIntContext && pIntContext->intNum!=iIntrNum;
                        pIntContext=pIntContext->pNext)
                pPrev = pIntContext;

        if(pIntContext)
        {
                HwiP_delete(pIntContext->intHandle);
                if(pPrev)
                        pPrev->pNext = pIntContext->pNext;
                else if(pIntContext == g_osiInterrupts)
                        g_osiInterrupts = NULL;
                free(pIntContext);
        }
}


int osi_ThreadCreate(void *(*pEntry)(void *),unsigned short stackDepth,void *pvParameters,unsigned long uxPriority,pthread_t *pThread)
{
        int ret;
        pthread_attr_t attr;
        struct sched_param priParam;
        int detachState;
        ret = pthread_attr_init(&attr);
        priParam.sched_priority = uxPriority;
        ret |= pthread_attr_setschedparam(&attr, &priParam);
        detachState = PTHREAD_CREATE_DETACHED;
        ret |= pthread_attr_setdetachstate(&attr, detachState);
        ret |= pthread_attr_setstacksize(&attr, stackDepth);
        if (ret == 0)
                ret = pthread_create(pThread, &attr, pEntry, pvParameters);
        return ret;
}


typedef struct
{
        mqd_t                   mq;
        unsigned long           MsgSize;
} osiMSGQContext_t;

/*!
        \brief      This function used to create a message queue
```

```
            \param      pMsgQ               - pointer to the message queue
            \param      pMsgQName           - Name of the Message queue
            \param      MsgSize             - Size of msg on the message queue
            \param      MaxMsgs             - Max number of msgs on the queue
            \return - OsiReturnVal_e
            \note
            \warning
*/
OsiReturnVal_e osi_MsgQCreate(OsiMsgQ_t     *pMsgQ,
                        char        *pMsgQName,
                        unsigned long  MsgSize,
                        unsigned long  MaxMsgs)
{
            int ret = OSI_FAILURE;
            mq_attr attr;
            osiMSGQContext_t *pMqContext;

            attr.mq_maxmsg = MaxMsgs;       // queue size
            attr.mq_msgsize = MsgSize;     // Size of message

            pMqContext = (osiMSGQContext_t*) malloc (sizeof (osiMSGQContext_t));
            if(pMqContext)
            {
                        pMqContext->mq = mq_open(pMsgQName, O_CREAT, 0, &attr);
                        //OSI_PRINT(("%p->%p %d %d\r\n",  pMqContext, pMqContext->mq, MsgSize, MaxMsgs));
                        if(pMqContext->mq == 0)
                        {
                                    free (pMqContext);
                        }
                        else
                        {
                                    pMqContext->MsgSize = MsgSize;
                                    *pMsgQ = (void *)pMqContext;
                                    ret = OSI_OK;
                        }
            }
            return (OsiReturnVal_e)ret;
}
/*!
            \brief      This function used to delete a message queue
            \param      pMsgQ     -           pointer to the msg queue
            \return - OsiReturnVal_e
            \note
            \warning
*/
OsiReturnVal_e osi_MsgQDelete(OsiMsgQ_t *pMsgQ)
{
            osiMSGQContext_t *pMqContext = (osiMSGQContext_t*)*pMsgQ;
            //OSI_HEADER_NL();
            mq_close(pMqContext->mq);
            free(pMqContext);

            return OSI_OK;
}

/*!
            \brief      This function is used to write data to the MsgQ

            \param      pMsgQ     -           pointer to the message queue
            \param      pMsg      -           pointer to the Msg strut to read into
            \param      Timeout   -           timeout to wait for the Msg to be available

            \return - OsiReturnVal_e
            \note
            \warning
*/
OsiReturnVal_e osi_MsgQWrite(OsiMsgQ_t *pMsgQ, void *pMsg, OsiTime_t Timeout)
{
            osiMSGQContext_t *pMqContext = (osiMSGQContext_t*)*pMsgQ;
   int ret;

            //OSI_PRINT(("%p\r\n",  pMqContext));
   //
   //signal provisioning task about SL Event
   //
```

```
	if(Timeout == OSI_WAIT_FOREVER)
	{
		ret = mq_send(pMqContext->mq, (char *)pMsg, pMqContext->MsgSize, 0);

	}
	else
	{
		struct  timespec tm;
		st_GetAbsoluteTime(Timeout, &tm);
		ret = mq_timedsend(pMqContext->mq, (char *)pMsg, pMqContext->MsgSize, 0, &tm);
	}
	if(ret < 0)
		ret = OSI_FAILURE;
	else
		ret = OSI_OK;

	return (OsiReturnVal_e)ret;
}

/*!
	\brief	This function is used to read data from the MsgQ

	\param	pMsgQ	-	pointer to the message queue
	\param	pMsg	-	pointer to the Msg strut to read into
	\param	Timeout	-	timeout to wait for the Msg to be available

	\return - OsiReturnVal_e
	\note
	\warning
*/
OsiReturnVal_e osi_MsgQRead(OsiMsgQ_t* pMsgQ, void* pMsg , OsiTime_t Timeout)
{
	osiMSGQContext_t *pMqContext = (osiMSGQContext_t*)*pMsgQ;
   int ret;
	unsigned int prio;


	//OSI_PRINT(("%p\r\n",  pMqContext));
   //
   //signal provisioning task about SL Event
   //
	if(Timeout == OSI_WAIT_FOREVER)
	{
		ret = mq_receive(pMqContext->mq, (char *)pMsg, pMqContext->MsgSize, &prio);
	}
	else
	{
		struct  timespec tm;
		st_GetAbsoluteTime(Timeout, &tm);
		ret = mq_timedreceive(pMqContext->mq, (char *)pMsg, pMqContext->MsgSize, &prio, &tm);
	}
	if(ret == ETIMEDOUT)
		ret = OSI_TIMEOUT;
	else if (ret < 0)
		ret = OSI_FAILURE;
	else
		ret = OSI_OK;

	return (OsiReturnVal_e)ret;
}
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE–SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, the customer to minimize inherent or procedural hazards must provide adequate design and operating safeguards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.